Eurographics Symposium on Geometry Processing 2010
Bruno Levy and Olga Sorkine
(Guest Editors)

Volume 29 (2010), Number 5

Polygonal Boundary Evaluation of Minkowski Sums and Swept Volumes

Marcel Campen and Leif Kobbelt

Computer Graphics Group, RWTH Aachen University, Germany

Abstract

We present a novel technique for the efficient boundary evaluation of sweep operations applied to objects in polygonal boundary representation. These sweep operations include Minkowski addition, offsetting, and sweeping along a discrete rigid motion trajectory. Many previous methods focus on the construction of a polygonal superset (containing self-intersections and spurious internal geometry) of the boundary of the volumes which are swept. Only few are able to determine a clean representation of the actual boundary, most of them in a discrete volumetric setting. We unify such superset constructions into a succinct common formulation and present a technique for the robust extraction of a polygonal mesh representing the outer boundary, i.e. it makes no general position assumptions and always yields a manifold, watertight mesh. It is exact for Minkowski sums and approximates swept volumes polygonally. By using plane-based geometry in conjunction with hierarchical arrangement computations we avoid the necessity of arbitrary precision arithmetics and extensive special case handling. By restricting operations to regions containing pieces of the boundary, we significantly enhance the performance of the algorithm.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

1. Introduction

Computing the volume occupied by an object being swept along a path, surface, or solid, possibly undergoing rotation during that process, is of interest in various fields of Computer Graphics. The case of translationally sweeping a solid by another one is known as *Minkowski addition*, the special case of one of them being a sphere is referred to as *offsetting*. When a solid is swept along a path while undergoing rotation, the occupied volume is called *swept volume*. Such volumes are for instance used in solid modeling, simulation systems, path and assembly planning, working volume examination, collision detection, NC machining, and many others.

Of special interest is the application of such sweeping operations to solids represented by a polygonal boundary, since this is the most commonly used representations for three-dimensional objects. Several methods that have been proposed to handle problems of this kind for general polyhedra so far have problems with one important aspect: the efficient and precise determination of a boundary representation of the result. Some produce a (non-manifold) superset of the boundary, containing self-intersections and spurious internal geometry (cf. Figure 1), and those that yield the actual boundary often have issues concerning robustness (due to numerics and degeneracies), accuracy (due to discrete volumetric resampling), or efficiency (due to arbitrary precision arithmetics). While some applications can work with boundary supersets, others indeed require a clean boundary representation of the result for further use and processing.

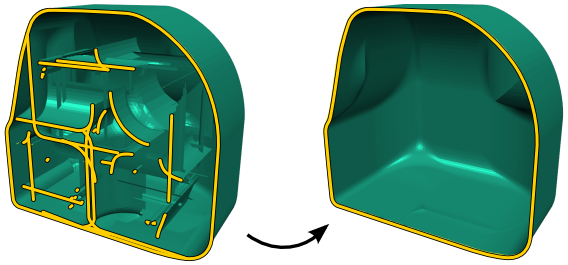

Figure 1: *Left: Non-manifold polygonal superset of the boundary of a Minkowski sum (as in Figure 2, but cut open for illustration; cut curves yellow). Right: consistent manifold polygon mesh of the boundary obtained by our method.*

We revise and unify approaches to generate tight supersets of the (polygonally approximated) boundaries of objects resulting from operations like Minkowski addition, sweeping, or offsetting. We then propose a novel, efficient technique to robustly and precisely convert these supersets into clean, manifold polygonal boundary representations. To achieve robustness and exactness efficiently, we make use of the paradigm of plane-based geometry representation in conjunction with binary space partition tree operations that has recently been brought into focus [BF09]. It elegantly handles all kinds of intersection constellations and degenerate contact situations, and completely avoids the demand of slow arbitrary precision arithmetics. Still, it is able to guarantee exact computations in the sense that starting from input of a given precision all computations are performed without introducing any round-off errors. Furthermore, for efficiency we do not process the whole volume globally but apply a novel localization scheme that disregards the interior space that usually contains lots of irrelevant geometry.

We build upon previous results concerning boundary superset constructions and arrangement computations and specifically contribute the following:

- Unification of boundary superset constructions for Minkowski sums and swept volumes into a succinct common formulation.
- Intuitive, easy-to-implement and cheap-to-evaluate culling rules that heavily reduce the excess of the supersets in a unified way.
- Exact and robust extraction of manifold and watertight polygonal boundaries of such supersets, resolving and removing all intersections and redundant internal geometry.

2. Related Work

2.1. Minkowski Sums

Kaul and Rossignac [KR92] constructed supersets of boundaries of Minkowski sums for visualization purposes. They introduced the notion of *locally supporting tentative faces* that describes if a face of the superset could possibly be visible from the outside. Similar notions, called *contributing vertices*, *elevated vertices*, or *horizon edges*, have been employed by Wu et al. [WSD03] and Barki et al. [BDD09a, BDD09b]. When restricting to convex polyhedra these concepts can be used to obtain the actual boundary; otherwise a superset, suitable for visualization, is constructed.

Ghosh [Gho93] introduced the slope diagram, inspired by Guibas and Seidel [GS87]. Overlaying the diagrams of two polyhedra yields a diagram that can be transformed into a superset of the polygonal boundary of the Minkowski sum. Further researchers followed this path [BGRR96, BR01, FH05, FH07, GXG08], but again only for convex input the actual boundary can be obtained by these methods.

Recently, Chazal et al. [CLM09] presented a further approach. It is based on critical points of the projection of higher-dimensional objects constructed from the operands. Again, a superset of the boundary, possibly containing self-intersections and internal geometry, is generated.

Hachenberger [Hac07] presented a well-working solution to the problem based on convex decomposition, constructing pairwise convex Minkowski sums, and finally forming the Boolean union of these elementary results. A robust implementation is available in the CGAL library. Since it is based on general Nef-polyhedra and uses arbitrary precision arithmetics, it does not scale very well for larger input models. In order to overcome the efficiency and/or robustness problems of the Boolean union step of such convex decomposition based algorithms, volumetric approaches [VM06, PS07] have been proposed that perform this step in a discrete setting. This, of course, limits the accuracy of the result.

2.2. Swept Volumes

The exact boundary of a swept volume consists of patches of ruled surfaces and developable surfaces in general. Abrams and Allen [AA95, AA00] constructed polyhedral approximations thereof. Polygonal arrangement computations are applied to an initial superset to extract the outer boundary. They state that these computations are often not robust enough to handle practical instances of the problem.

Rossignac et al. [RK01, RKS*07] constructed supersets of swept volume boundaries of polyhedra and free-form solids. Extraction of the actual boundary was performed approximately using a *helix-shooting* approach. This approach has later also been applied for polyhedral sweeps [Kim03]. Blackmore et al. [BSL99] propose an analytical method for trimming of supersets constructed based on the sweep-envelope differential equation [LBW97]. Due to the employed hyperbolic tangent functions, obtaining a robust implementation appears to be rather involved. Abdel-Malek et al. [AMYO98] also presented such analytical formulations of boundary identification criteria.

The exact swept volume of a polyhedron, consisting of ruled surface patches, has been considered by Weld and Leu [WL90]. They give a point-wise criterion for boundary identification but an explicit representation is not extracted.

Schroeder et al. [SLL94], Schwanecke and Kobbelt [SK01], and Kim et al. [KVLM03] followed the path of implicit modeling to simplify the issue by volumetric discretization. While this yields robust algorithms, accuracy is limited due to the spatial (and temporal) sampling.

2.3. Offsets

Specialized methods have been proposed for the particular case of offset surfaces, e.g. by Jung et al. [JSC04] where the faces of the input are shifted and the introduced intersections are detected and resolved. To overcome robustness problems, again volumetric methods have been proposed, e.g. by Chen et al. [CWRR05] or Pavić and Kobbelt [PK08].

While they augment their voxel structures by explicit geometry in form of points and polygons to enhance output quality, the overall accuracy of the results is still limited by the underlying voxel resolution.

2.4. Outer Boundary Extraction

Computing the outer boundary of a polygonal arrangement in space has proven to be a complicated task. The consistent determination of all intersection loci is numerically hard to handle, and the piecewise nature of a polygonal surface introduces further challenges. Usually, descriptions of intersection determination and subsequent topology modification [Par04, JSC04, Lie08, Elb97, dBGH96, SH02] rely on general position assumptions or require arbitrary precision arithmetics, which makes it hard to provide implementations that are efficient and reliable. In application scenarios where robustness is of higher importance than accuracy, again methods that rely on volumetric representations can be employed [BPK05, WTGT09, NT99, Ju04].

Recently, Campen and Kobbelt presented a method [CK10] to robustly and exactly extract the outer boundary of a self-intersecting polygon mesh. We build upon this work and develop a novel method able of handling loose arrangements of polygons as they are commonly constructed as boundary supersets for Minkowski sums or swept volumes.

3. Tight Superset Construction

We begin by giving definitions of the operations *Minkowski addition* and *sweeping*, and then describe how supersets of the boundary of the volumes they produce can be constructed in a unified fashion. If this construction is done in a naïve way, only a very small subset of the polygon set that is generated actually contributes to the boundary – most polygons lie in its interior. We therefore also describe intuitive rules to efficiently cull most of these interior polygons, which greatly improves the performance of the subsequent boundary extraction step that produces the exact surface.

3.1. Minkowski Addition

The Minkowski sum of two sets, A and B, is defined as $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$. We are interested in the case of A and B being three-dimensional (possibly degenerate) solids specified by a polygonal boundary representation. Their Minkowski sum is again a polygonal solid (cf. Figure 2) and we want to obtain an exact representation of its boundary. Offsetting is a special case, with one operand being an origin-centered sphere. Note that the actual offset surface of a polygonal surface is not polygonal: it contains cylindrical and spherical parts. However, we can use polygonally approximated spheres to obtain (arbitrarily precise) polygonal approximations of an offset surface.

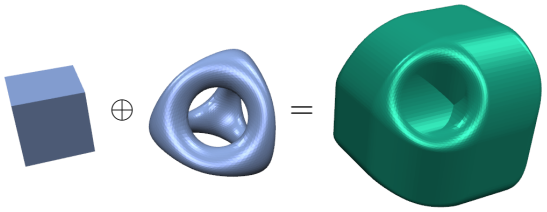

Figure 2: *Minkowski sum of solids* CUBE *and* TETRATHING

3.2. Sweeping

Stating a concise definition of swept volume computation for polygonal solids, however, is not that easy. The actual result of the operation is not polygonal and various approximations are used in practice. Furthermore, various characterizations of the sweep motion are common. In general, our boundary extraction (cf. Section 4) can be applied for any definition yielding a polygonal boundary. We here exemplarily present our method on the following problem characterization:

Let B be a polygonal solid in $\mathbb{R}^3$, called *generator* in this context. Further, let $\tau(t) = (T(t), R(t))$ be the *sweep trajectory*, a time-varying affine transformation, where $T(t)$ is a differentiable vector in $\mathbb{R}^3$ and $R(t)$ is an orthonormal matrix in the rotation group $SO(3)$. Both depend on the time parameter $t \in [0,1]$. Let $B(t) = T(t) + R(t)B$, then the swept volume of the moving generator $B(t)$ is defined as $B \otimes \tau = \bigcup_{t\in[0,1]} B(t) = \{T(t) + R(t)b \mid t \in [0,1], b \in B\}$. We strive to obtain a polygonal approximation of its boundary $\partial(B \otimes \tau)$ (cf. Figure 3), that actually consists of patches of ruled and developable surfaces that are generated by the sweeping edges and faces of B [WL90]. Such approximation can be found by choosing a discrete set of consecutive time samples $\{t_0 = 0, t_1, \ldots, t_{n-1}, t_n = 1\}$, $t_i < t_{i+1}$ and replacing the sweep patches by polygonal facets, corresponding to faces of an instance $B(t_i)$ or spanned by pairs of edges from consecutive instances $B(t_i)$ and $B(t_{i+1})$ as done in [AA00]. The quality of this approximation is determined by the sampling density of τ on the one hand and the edge lengths of B on the other hand – it can hence be controlled arbitrarily by the sampling strategy and by edge subdivision on the generator. Due to the wide spectrum of possible trajectory specifications and precision requirements, in this paper we assume the input to be given in a form that has already been suitably discretized in the context of a given application scenario.

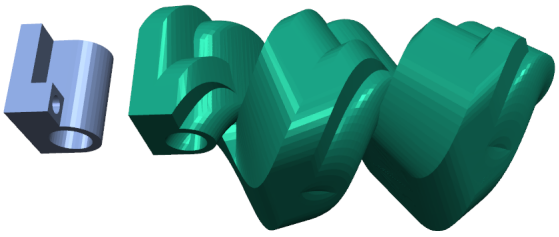

Figure 3: *Swept volume of generator object* JOINT *moving along a helical trajectory while rotating.*

3.3. Unified Superset Construction

We now unify Minkowski addition and sweeping as defined above into a generalized sweeping operation $\odot$ that subsumes both operations as special cases. This allows us to streamline the description of the superset construction and the culling rules in the following section.

We take as input two polygonal complexes, A and B, and a map $R : V_A \to SO(3)$, V_A being the set of vertices of A. In order to ease presentation we, w.l.o.g., assume all polygons to be triangles. A and B do not need to be closed meshes; they may also have holes or be consisting of only vertices and edges representing a path. Conceptually, object B is the generator and A together with R is a generalized discrete sweep trajectory, i.e. B is swept over A, while being rotated according to R that specifies a rotation at each vertex of A – in between the surface is linearly approximated as mentioned before. Note that if A is a path, consisting only of vertices and edges, this generalized sweeping reduces to the sweeping operation as defined above; on the other hand, if R is the identity, it directly corresponds to the Minkowski addition of A and B. The remaining cases, i.e. sweeping over a surface or a solid while rotating, are basically academic byproducts.

Matheron [Mat75] showed that the boundary of a Minkowski sum $A \oplus B$ is a subset of the boundary of the Minkowski sum of the boundaries of the operands, i.e. $\partial(A \oplus B) \subseteq \partial(\partial A \oplus \partial B)$. Similarly, it is well known from envelope theory, that the boundary of the volume swept by a moving object $A(t)$ is a subset of the boundary of the volume swept by its boundary, i.e. $\partial(A \otimes \tau) \subseteq \partial(\partial A \otimes \tau)$ [WL90]. Furthermore, these operations distribute the set union, i.e. $A \odot (B_1 \cup B_2) = (A \odot B_1) \cup (A \odot B_2)$. This means that a superset of the boundary can be constructed incrementally by uniting the boundaries of the elementary results of applying the operation to all face-face pairs of the operands, one face taken from A, one taken from B. The construction of the boundary of the volume swept by a triangle sweeping over another one can furthermore be reduced to considering vertex-face and edge-edge pairs [KR92, BDD09b, AA00].

Let $v_x \in V_x$, $e_x \in E_x$, $f_x \in F_x$, $x \in \{A,B\}$, with V_x, E_x, F_x being the sets of vertices, edges, and faces of object x, respectively. Furthermore, let e^0 and e^1 denote the two vertices incident to edge e, and f^i, $i \in \{0,1,2\}$ the three vertices of triangular face f. (In the following, vertex symbols are also used to refer to the respective spatial positions.) We construct the following polygons, called *facets* hereafter to distinguish between the polygons of the input and those of the constructed superset. For each vertex-face pair (v_A, f_B) we construct a so-called *VF facet*, for each face-vertex pair (f_A, v_B) we construct an *FV facet*, and for each edge-edge pair (e_A, e_B) two *EE facets* in the following way:

VF facet (v,f)**:** $\triangle\left(v+R(v)f^0, v+R(v)f^1, v+R(v)f^2\right)$

FV facet (f,v)**:** $\triangle\left(f^0+R(f^0)v, f^1+R(f^1)v, f^2+R(f^2)v\right)$

EE facets (d,e)**:** Two $\triangle$ by inserting a diagonal into $\square\left(d^0+R(d^0)e^0, d^0+R(d^0)e^1, d^1+R(d^1)e^1, d^1+R(d^1)e^0\right)$

For the EE facets the diagonal with lower dihedral angle is chosen. This also prevents fold-overs in cases where the quadrangle is flat and non-convex.

The set of all these facets forms a (highly excessive) superset of the desired boundary. Figure 4 shows an example. If the rotation map R is constant, as it is the case for Minkowski sums, this boundary is exact, otherwise it is a piecewise linear approximation. In order to avoid the actual construction of this excessive superset the culling rules presented in the next section can be applied beforehand to determine if a facet needs to be constructed at all.

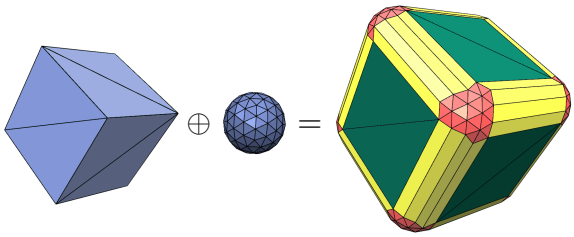

Figure 4: *Collection of facets constructed for a Minkowski sum: VF (red), EE (yellow, depicted without diagonal), and FV facets (green). Only 288 of the 12264 facets contribute to the boundary, i.e. are visible from the outside.*

3.4. Culling

We now strive to cull as many *irrelevant* facets as possible to increase efficiency of the subsequent processing steps. A facet is called irrelevant if no non-degenerate part of it lies on the boundary. This culling is performed using rules that conservatively estimate the relevance of a facet from the local mesh neighborhood of the generating vertices, edges, and faces. Note that the determination of the actual boundary is a global problem; it cannot be solved completely by local investigation of the objects. The exact determination on a global scale is thus subsequently performed implicitly during the actual boundary extraction (cf. Section 4).

The culling rules base on the intuitive property of *local coveredness*: if we are able to determine that the elementary volumes swept by neighboring entities (vertices, edges, or faces) of a mesh entity completely cover both sides of a facet generated from it, this facet consequently cannot appear on the boundary. Conceptually the elementary volumes we consider are prisms (or deformed prisms if rotations occur). Such a prism basically is the volume swept by a face along an edge, thus is part of the whole volume. Its boundary is made from the two VF or FV facets emerging from the face and the two vertices of the edge, and the three EE facets generated from the edge and the three edges of the face. A side of a facet that completely points to the inside of a prism it bounds is covered – it cannot be visible from the outside.

Beforehand, if a facet is generated from a face of a solid, we can already conclude that its backside is covered since it points to the inside. Similarly, if an EE facet is generated from edges d of B and e of A with $R(e^0) = R(e^1)$ a side of it is already covered if the corresponding normal points to the inside of A at e or to the inside of B at d.

Now let $N_1(v)$ denote the set of 1-ring vertices of vertex v, and $O(e)$ the set of (zero, one, or two) opposite vertices of edge e, i.e. the vertices incident to faces incident to e that are not incident to e. Then (cf. Figure 5 for an illustration):

- **a side of VF facet (v,f) is covered** if $\exists u \in N_1(v)$ such that facet (u,f) lies completely on that side of the supporting plane of facet (v,f). This is due to the fact that the EE facets generated from the edge connecting v and u and the edges of f form a "prism" together with (v,f) and (u,f) that completely covers that side of (v,f).
- **a side of EE facet g generated from (d,e) is covered** if $\exists u \in O(e)$ such that $d^0 + R(d^0)u$ and $d^1 + R(d^1)u$ lie on this side of the supporting plane of g or if $\exists v \in O(d)$ such that $v + R(v)e^0$ and $v + R(v)e^1$ lie on this side: the "prism" swept by the triangle incident to e and u along d resp. d and v along e covers that side of g in this case.
- **a side of FV facet (f,v) is covered** if there exists a vertex $u \in N_1(v)$ such that facet (f,u) lies completely on that side of the supporting plane of facet (f,v). The argument is analogous to the VF facet case.

A facet can safely be culled if both of its sides are found to be covered. Note that this local coveredness property is general enough to handle all possible special cases including rotational sweeps of surfaces or solids along trajectories and translational sweeps, i.e. Minkowski sums, along solids, surfaces, or paths. In fact, for the case of translational sweeping, it directly reduces to the property of non-matching normal directions employed for Minkowski addition using convolution or by inspecting Gauss maps [Gho93, BGRR96, BR01, FH05, FH07, GXG08], to the concepts of locally supporting directions, contributing vertices, elevated vertices, and horizon edges [KR92, WSD03, BDD09a, BDD09b], and is closely related to *grazing points* used for swept volumes [LBW97]. In contrast to these notions, in our formulation the evaluation relies on shallower arithmetic trees since all rules only require the determination of orientation of a plane with respect to a point. In fact, exact evaluation induces no further overhead, posing the same precision requirements as the exact plane-based processing we employ (cf. Section 4.1.2).

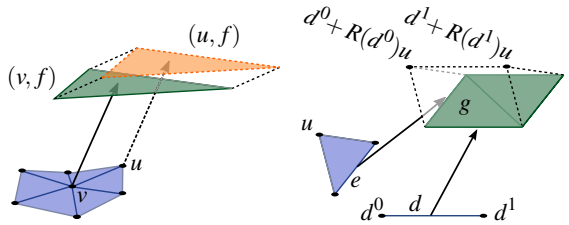

Figure 5: *Illustration of the rules for determining coveredness. On the left VF facet (v,f) (green) is covered by VF facet (u,f) (orange) and on the right the EE facets (green) of edges d and e are covered due to opposite vertex u.*

By applying these rules to determine if a facet is potentially relevant and culling it otherwise we are able to greatly reduce the amount of irrelevant interior facets. In fact, in all of the Minkowski sum and offsetting examples presented throughout this paper the culling performance has been between 98.80% and 99.99%, i.e. at most 1.2% of the irrelevant facets remained after culling. For the swept volume examples culling performance has been at least 91.5%.

4. Boundary Extraction

The constructions presented in the last sections yield supersets of the (polygonally approximated) boundary of the volume resulting from the applied sweep operation. We now strive to extract the outer boundary of this spatial facet arrangement. The outer boundary is defined as those parts of the boundary of the volume enclosed by the facets that can be reached from the outside, i.e. if inner voids do exist, these are omitted. In Section 4.3 we discuss this limitation and provide viable solutions. Note that this extraction of a manifold outer boundary can also be applied to other kinds of polygonal arrangements that in some way enclose volumes.

Bernstein and Fussell [BF09] showed how a plane-based geometry representation combined with binary space partition (BSP) techniques can be used to deal with polygon intersections in the evaluation of Boolean operations. Campen and Kobbelt [CK10] recently accelerated and extended that technique to handle intersections and also self-intersections in polygonal meshes in a fully robust and exact way. Their approach, however, relies on and draws its efficiency from two essential assumptions that do not hold here:

- **Closed meshes:** The input needs to be a fully connected, closed mesh. In contrast, here we have to handle a polygon soup without connectivity information.
- **Local self-intersections:** Their localization scheme is tailored to handle local self-intersections in the input mesh and derived its performance from their spatial coherence. In contrast, here we usually have to deal with a lot of intersections that are distributed over the whole surface. Especially, many of them lie in the interior and we neither need nor want to actually process or even just detect them.

In order to account for these differences we present a novel approach that marches along the outer boundary while extracting. It also enables the handling of multiple non-connected boundary parts, even if their number and constellation is not known in advance. The approach builds upon the basics concerning representation conversions and BSP techniques presented in [BF09, CK10]. Here we give just a brief overview and afterwards describe the marching approach.

4.1. Plane-Based BSP Processing

4.1.1. BSP Techniques

Binary space partition trees can be used to represent polyhedra: by recursively splitting space by the supporting planes of the faces of a polyhedron a *compatible* BSP-tree can be constructed, i.e. each leaf cell can be labeled 'inside' or 'outside' such that the boundary between inside and outside cells defines the polyhedron. However, since our polygonal supersets do not form closed polyhedra, inside and outside is not defined inherently. But one can employ flood-filling on the BSP [CK10] to conquer the whole outside region. Afterwards, a polygon mesh representation of the boundary between inside and outside cells can be extracted [CK10], which represents the outer boundary of the polygonal arrangement. However, this approach would not be very efficient since the global BSP constructed from the entire polygonal boundary superset is often very imbalanced and unnecessarily considers all redundant internal geometry. Although we already cull large portions of this internal geometry, still a considerable amount might remain. For instance, in example CASTING-KNOT (cf. Figure 9) after culling still 87.6% of the facets are irrelevant interior facets. Hence it is beneficial to work with *local* BSPs and restrict operations to the actual outer boundary region. We achieve this by the process of *octree marching* that is introduced in Section 4.2.

4.1.2. Plane-Based Geometry Representation

The BSP techniques reduce the amount of special cases that need to be treated, easily handle degenerate contact situations, and allow for an efficient spatial flood-filling. The only geometric operation that is required, whether during BSP construction, processing, or extraction, is that of polygon clipping. So to finally achieve full robustness and accuracy, we only need to perform this in an appropriate manner.

While polygonal objects are usually represented by vertex coordinates, this setting is disadvantageous for the clipping operation: new vertices have to be constructed and their coordinates need to be computed explicitly at the intersections. Since these constructions cause increasing precision requirements with every step, exact arithmetics which are able to handle arbitrary precision need to be employed. In contrast, if we choose to explicitly represent the geometry of the faces by plane equations and let vertices be defined implicitly by plane intersections [BF09, BR96] this clipping operation reduces to a purely combinatorial operation – no intersections need to be computed, only geometric decision predicates are required. Due to the absence of increasing precision requirements we can use efficient predicates, specifically optimized for a fixed precision known a priori which guarantees correct evaluation. Details on determining the precision requirements for error-free conversion of the point-based polygons of our superset into plane-based representation can be found in [CK10]. Essentially, about three times as many bits are required for the plane coefficients and the exact predicates are tailored to that precision to guarantee correct decisions.

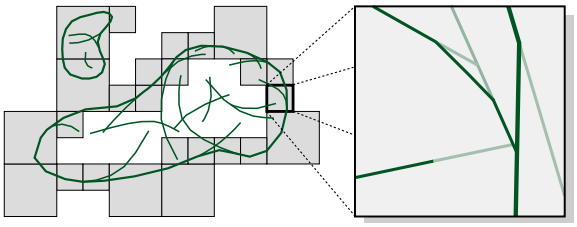

Figure 6: *2D illustration of the localization scheme using an adaptive octree. Only the grey cells that are intersected by the outer boundary are actually processed by embedded BSPs. The zoomed inset exemplarily shows such a local BSP.*

4.2. Octree Marching

The fundamental idea of the localization scheme is to use an octree that is adaptively refined such that all leaf cells intersected by the (yet unknown) outer boundary contain an approximately constant number of facets, and then apply the BSP-processing in these cells individually. Figure 6 illustrates the concept. In this way we lower complexity due to the application of locally restricted BSPs and increase efficiency by only processing the spatial region occupied by the outer boundary we are interested in. This confronts us with two major challenges: first of all, the location of the outer boundary is yet unknown such that we cannot directly construct the desired octree in a straight-forward fashion. Secondly, the information about outside and inside regions has to be exchanged between the local BSPs that are embedded into the individual octree leaf cells.

Suppose we are given an octree leaf cell that contains part of the outer boundary. Conceptually, we can create a polygonal mesh patch that represents this part of the boundary by applying the local BSP techniques presented above. Starting from that cell and its boundary patch we march through the octree along the outer boundary while extracting it, thereby successively enlarging the initial patch. During that process the octree is adaptively refined on demand. In order to be able to handle cases where the outer boundary consists of multiple non-connected parts, we need to make sure that such marching is started for every part. To achieve that, we also march through empty cells on the outside and use cell corners as representative points to propagate the information about the locality of the outside.

We start by adaptively refining the octree (initially consisting of the root cell encompassing the whole set of facets) such that a leaf cell c_{corner} in one of the outer corners meets the criterion "*intersected by no more than m facets*". The value m is a constant that bounds the complexity per cell. In our experiments $m = 60$ proved to be a good choice, gaining near-minimal runtime in general. This cell c_{corner} incident to the corner with coordinates p_{corner} defines our starting position, and corner point p_{corner} is our first representative point known to be on the outside. The following pseudo-code out-

lines the further processing steps – a detailed explanation follows and the whole process is illustrated in Figure 7.

```
Insert (c_corner, {p_corner}) into queue
while queue not empty:
  take (c, P) from queue
  build local BSP in cell c
  determine BSP components and extract boundary patches
  mark components containing a point of P as outside
  establish connectivity and propagate outside marks
  for all newly marked outside components:
    enqueue adjacent cells touched by component's patch
    enqueue incident cells of corners lying on the outside
```

- **Build local BSP in cell c** A BSP-tree restricted to cell c is built from the facets intersecting c by successive insertion as described in section 4.1.1.
- **Determine BSP components and extract boundary patches** The volume of cell c is partitioned into components (i.e. sets of BSP cells) by the facets. We determine these components by flood-filling and per component extract its polygonal boundary – note that this component boundary might consist of several patches. In order to determine which of these components are on the outside (which is equivalent to their patches being part of the outer boundary) the following two steps are performed:
- **Mark components containing a point of P as outside** Point set P is either empty or contains a point in c (more precisely: a cell corner on c's cell border) that is known to be on the outside. If it contains such point it is sent down the BSP, eventually ending up in a BSP-cell. The component this BSP-cell belongs to is thus found to be on the outside and marked accordingly.
- **Establish connectivity and propagate outside marks** Connectivity between boundary patches extracted in c and adjoining patches extracted previously in neighbor cells is established [CK10]. Afterwards, components marked outside recursively propagate this marking to all components whose patches has been connected to their patches.

After a cell has been processed this way, a number of components (obtained from this or other octree cells) might have newly been marked outside. These are the anchor points for adding new, yet unprocessed octree cells to the queue – we want to achieve that all cells that are either partially or fully on the outside are processed eventually:

- **Enqueue adjacent cells touched by component's patch** Suppose component o with boundary patch a was obtained from cell c and has newly been marked outside. We enqueue those unprocessed and unqueued cells adjacent to c that are touched by a. This basically accomplishes the marching process along the outer boundary (or one connected component of it). These cells are enqueued without a representative outside point ($P = \varnothing$): the outside information is implicitly provided by the open boundary of patch a that patches of neighbor cells will connect to.
- **Enqueue incident cells of corners lying on the outside** Suppose component o (with or without a patch) was obtained from cell c and has newly been marked outside. In order to not only march along one part of the outer boundary but also conquer the whole outer space to find all parts, we enqueue further cells that are known to be at least partly outside: for each corner r of c that lies in component o we enqueue the unprocessed, unqueued incident cells together with r as outside representative ($P = \{r\}$).

As stated before, octree cells are always refined in an on-demand fashion before entering the queue, such that all enqueued cells meet the "*no more than m facets*" criterion.

When the algorithm terminates, all cells that are fully or partially on the outside have been considered, all outside components are marked as such, and their connected boundary patches form the desired outer boundary.

4.3. Inner Boundaries

While being able to extract the outer boundary as defined in Section 4 the presented method is oblivious to inner voids: if a Minkowski sum or swept volume encloses void space, the boundary towards these voids is not extracted. While many applications are indeed only interested in this *outer* boundary, others explicitly require these voids to be represented in the result. Under the assumption that one witness point somewhere inside a void is known, we can easily extend our method to extract also the *inner* boundary towards it. For a given set of void witness points we recursively refine leaf cells of the octree that contain a witness point until the abovementioned refinement criterion is met. Then we enqueue these leaf cells together with the contained witness points as outside representatives. By then proceeding in exactly the same manner as described in the last section, not only the outer boundary but also the inner boundary towards each void that contained a witness point is extracted. This also, for instance, allows us to construct Minkowski sums of a solid and the complement of another one as it is required, e.g., for motion planning tasks or for the computation of inner, shrunk offsets (cf. Figure 8 for an example).

This leaves us with the task of determining witness points inside voids that shall be extracted. In some application scenarios they might already be known as part of the problem specification; otherwise we would have to find them automatically. A simple approach to determine such witness points at least for all voids down to a certain minimum size is the use of a standard discrete volumetric method to (conservatively) perform the desired sweep operation. Voids can be detected in the discrete result, witness points be chosen within those, and then these can be used as input for our method to perform the operation accurately. Tiny voids not resolved by the discrete method are of course missed. General detection and extraction of voids will thus be part of our future research. This will probably entail the construction not of an arbitrary boundary superset but a complete 2-cycle as done, e.g., by Ghosh [Gho93] and Chazal et al. [CLM09], which should then allow to distinguish between void regions and swept regions, e.g. by some notion of a winding number.

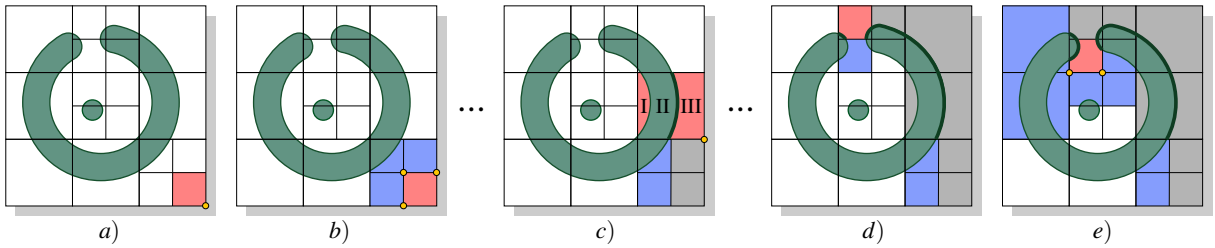

Figure 7: *2D illustration of the octree marching process. The octree is depicted in its final state (i.e. without on demand refinement) and only the outer boundary of the object (green) is shown for clarity. a) First, a corner cell (red) is processed: There is only one space component and since it contains the outside point (yellow), it is marked as outside. b) The adjacent cells (blue) of outside corners are enqueued (with the respective corner as outside point). c) A few steps later, the first non-empty cell is processed by building a local BSP and extracting the component boundaries per component. One of the three components (III) is found to be outside since it contains the outside point – component I will be marked outside later on when marks are propagated from adjacent boundary patches. d) A few steps later a cell (blue) is enqueued solely because it is touched by the previously extracted outside patch – this shows the necessity of the first enqueue-rule. e) Now further cells are enqueued, some solely because of the second enqueue-rule – facilitating the conquering of multiple non-connected boundary parts.*

5. Results

To evaluate the efficiency of our method we applied it to compute several Minkowski sums, offsets, and swept volumes (cf. Figures 2, 3, 8, 9, and 10 – some of the models used have been obtained from the AIM@SHAPE repository). All computations were done on a system with 2.67GHz CPU and 8GB RAM; timings (of a prototype implementation tailored to 17 bits input vertex coordinate precision) are presented in Table 1. We compare our method to two other approaches that are widely used in areas where a robust boundary evaluation of such operations is required: the first one is a routine provided by the CGAL library – the de-facto reference for exact and robust geometric computations – that can be used to perform polyhedral Minkowski addition and offsetting; the second one is the approach of volumetric processing in the style of Varadhan et al. [VM06, KVLM03] or Pavić and Kobbelt [PK08]: for the Minkowski sums and swept volumes a volumetric approach has been used that voxelizes the boundary superset using an adaptive octree and extracts an outer boundary mesh using Manifold Dual Contouring. This approach is quite simple and does not provide guarantees regarding topological correctness – if we would have used a more sophisticated algorithm [VM06] that does, further computational cost would have been introduced by the additional operations (e.g. convex decompositions, convex hulls, star-shaped cell tests). For the offset surfaces a specialized volumetric method [PK08] that avoids the polygonal sphere approximation and explicit superset construction for accuracy and efficiency, has been used for comparison.

As can be seen easily (cf. Table 1), our method performs significantly faster than the Minkowski addition routine of CGAL while still providing accuracy and robustness. On the other hand volumetric approaches introduce a considerable amount of geometric error (due to aliasing) at comparable runtimes and are not able to reliably reproduce sharp features in the output. Another problem of these methods is the fact that the output is highly overtesselated (cf. last column of Table 1) due to the high resolution if tight error bounds shall be fulfilled. In some application areas – especially when working with very complex objects – it might nevertheless be beneficial to use such volumetric methods, possibly with topological guarantees, to reasonably balance between performance and accuracy.

In the examples the construction of the supersets took between 0.05% and 19.4% of the total time, the extraction of the clean outer boundary mesh the remaining time. This variation is due to the fact that the superset construction time depends on the product of the input sizes, whereas the boundary extraction time depends roughly on the output size.

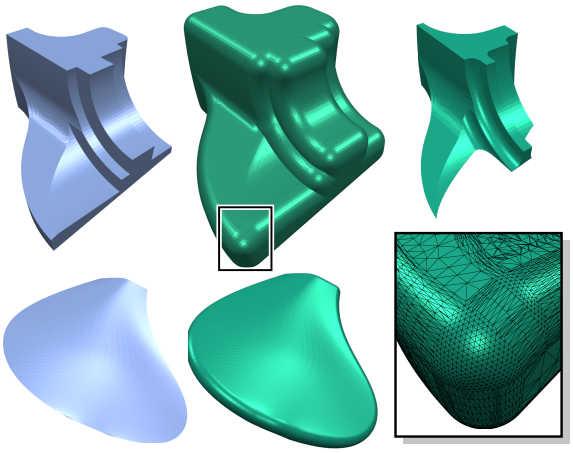

Figure 8: *Polygonal models* FANDISK *(solid) and* BLADE *(surface) and their offset surfaces generated by Minkowski addition of an approximated sphere. The zoomed inset shows the output mesh structure as it is generated by our method when a polygonal sphere with 5K faces is used. The inner offset has been generated by complementing the operand and applying inner boundary extraction (cf. Section 4.3).*

Operation	Object(s)	Input Complexity	Runtime Ours	Runtime CGAL	Runtime Volum.	Error Bound ($\cdot 10^{-4}$) Ours	Error Bound ($\cdot 10^{-4}$) CGAL	Error Bound ($\cdot 10^{-4}$) Volum.	Output Complexity Ours	Output Complexity CGAL	Output Complexity Volum.
Minkowski	TORUS-JOINT (Fig. 9)	460 × 100	1.7s	729s	2.8s	0	0	68	1.4K	1.2K	151K
	TETRA-CUBE (Fig. 2)	13K × 6	10s	1560s	21s	0	0	17	9.0K	9.0K	2688K
	KNOT-CAST (Fig. 9)	10K × 8K	43s	>1h**	31s	0	0	34	34K	**	488K
	ROLL-BAR (Fig. 9)	108K × 162	223s	>2h**	110s	0	0	8	130K	**	5610K
Sweeping	JOINT (Fig. 3)	460 × 100	49s	–	32s	0*	–	8*	10K	–	9998K
	MOUNT (Fig. 10)	1.2K × 12	59s	–	39s	0*	–	4*	8K	–	9405K
	LEVER (Fig. 10)	5.2K × 100	152s	–	106s	0*	–	2*	55K	–	30182K
Offsetting	BLADE (Fig. 8)	6K (× 540)	10s	>1h**	12s	50	50	545	14K	**	1289K
	FANDISK (Fig. 8)	13K (× 540)	19s	>2h**	16s	50	50	551	24K	**	479K
	FANDISK (Fig. 8)	13K (×1620)	26s	>2h**	42s	16	16	275	36K	**	1960K
	FANDISK (Fig. 8)	13K (× 15K)	85s	>2h**	96s	2	2	138	121K	**	7929K

Table 1: *Performance evaluation of our method and comparison with* CGAL *and volumetric approaches. The volumetric resolution has generally been chosen such that runtime is close to that of our method. We clearly see that our method performs significantly faster than* CGAL *for equal accuracy and significantly more accurate than volumetric approaches at comparable runtimes. Error bounds are specified relative to output bounding box size resp. relative to offset distance for the offsetting case. *) For the case of sweeping (along a path, including rotation) the error to the polygonally approximated sweep surface is stated. **) Operation has not been finished, since memory requirements exceeded the available main memory after the specified time.*

Compared to CGAL (that for most of the examples required much more memory than available) and the volumetric approaches (that for some examples nearly exhausted it) memory requirements of our method are more moderate – even in our current non-optimized implementation: the ROLL-BAR example peaked at 2.1GB, the LEVER example at 1.3GB, and all others required less than 700MB. This allows us to handle even quite complex examples (e.g. Minkowski addition of refined versions of KNOT-CAST with 128K×92K faces: 6.9GB) without problems.

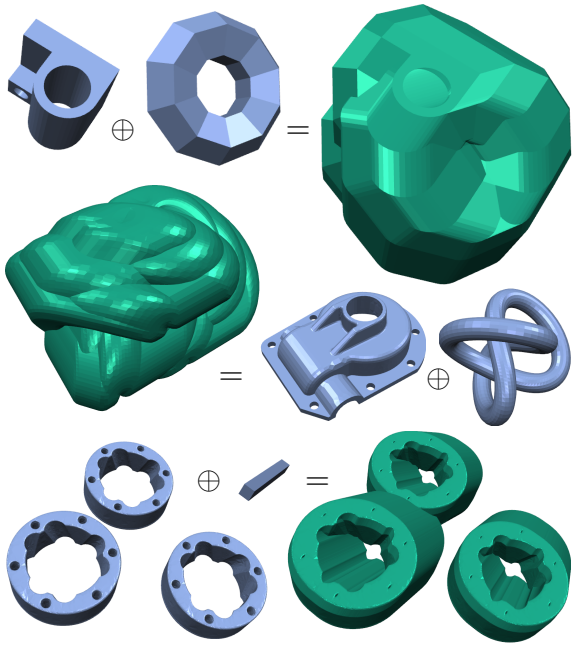

Figure 9: *Minkowski sums (green) of various objects (blue) computed by our method:* JOINT *and* TORUS, CASTING *and* KNOT, ROLLINGSTAGES *and* BAR. *All results are manifold and watertight polygon meshes. The third example demonstrates that multi-component input is handled appropriately.*

6. Conclusion

We revised and unified boundary superset constructions for Minkowski sums and swept volumes and streamlined them with intuitive local culling rules. These efficiently reduce the overhead of the constructed supersets of the actual boundaries of the volumes resulting from the sweeping operations. Using a plane-based geometry representation in conjunction with BSP-based spatial arrangement computations, we then showed how to extract the actual outer boundary robustly and precisely. By applying a localization structure we restricted computational efforts to the regions of interest, thereby enhancing the performance of the proposed method.

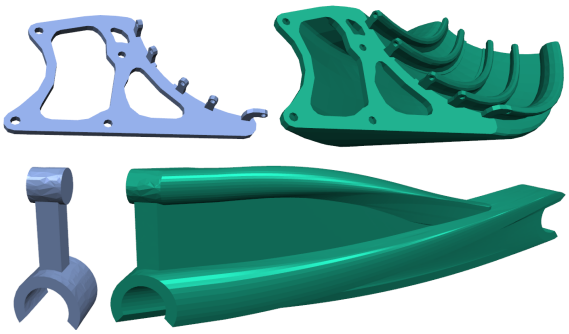

Figure 10: *Manifold and watertight boundary polygon meshes (green) of the volumes swept by* MOUNT *and* LEVER *(blue) moving and rotating along specified trajectories.*

References

[AA95] ABRAMS S., ALLEN P. K.: Swept volumes and their use in viewpoint computation in robot work-cells. In *Proc. IEEE Intl. Sympos. on Assembly and Task Planning* (1995), pp. 188–193.

[AA00] ABRAMS S., ALLEN P. K.: Computing swept volumes. *Journal of Vis. and Comp. Animation 11*, 2 (2000), 69–82.

[AMYO98] ABDEL-MALEK K., YEH H.-J., OTHMAN S.: Swept volumes: void and boundary identification. *Computer-Aided Design 30*, 13 (1998), 1009–1018.

[BDD09a] BARKI H., DENIS F., DUPONT F.: Contributing vertices-based minkowski sum computation of convex polyhedra. *Comput. Aided Des. 41*, 7 (2009), 525–538.

[BDD09b] BARKI H., DENIS F., DUPONT F.: Contributing Vertices-based Minkowski sum of a non-convex polyhedron without fold and a convex polyhedron. In *SMI '09* (2009), Press I. C. S., (Ed.), pp. 73–80.

[BF09] BERNSTEIN G., FUSSELL D.: Fast, exact, linear booleans. *Comput. Graph. Forum 28*, 5 (2009), 1269–1278.

[BGRR96] BASCH J., GUIBAS L. J., RAMKUMAR G. D., RAMSHAW L.: Polyhedral tracings and their convolution, 1996.

[BPK05] BISCHOFF S., PAVIC D., KOBBELT L.: Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4 (2005), 1332–1352.

[BR96] BANERJEE R. P. K., ROSSIGNAC J.: Topologically exact evaluation of polyhedra defined in CSG with loose primitives. *Comput. Graph. Forum 15*, 4 (1996), 205–217.

[BR01] BEKKER H., ROERDINK J. B. T. M.: An efficient algorithm to calculate the minkowski sum of convex 3d polyhedra. In *Int. Conf. on Computational Science (1)* (2001), pp. 619–628.

[BSL99] BLACKMORE D., SAMULYAK R., LEU M. C.: Trimming swept volumes. *CAD 31*, 3 (1999), 215–223.

[CK10] CAMPEN M., KOBBELT L.: Exact and robust (self-)intersections for polygonal meshes. *Comput. Graph. Forum 29*, 2 (2010), 397–406.

[CLM09] CHAZAL F., LIEUTIER A., MONTANA N.: Discrete critical values: a general framework for silhouettes computation. *Comput. Graph. Forum 28*, 5 (2009), 1509–1518.

[CWRR05] CHEN Y., WANG H., ROSEN D. W., ROSSIGNAC J.: *A Point-Based Offsetting Method of Polygonal Meshes*. Tech. rep., 2005.

[dBGH96] DE BERG M., GUIBAS L. J., HALPERIN D.: Vertical decompositions for triangles in 3-space. *Discrete & Computational Geometry 15*, 1 (1996), 35–61.

[Elb97] ELBER G.: Global error bounds and amelioration of sweep surfaces. *Computer-Aided Design 29*, 6 (1997), 441–447.

[FH05] FOGEL E., HALPERIN D.: Exact minkowski sums of convex polyhedra. In *SCG '05: Proc. 21st Annu. Symp. on Computational Geometry* (2005), pp. 382–383.

[FH07] FOGEL E., HALPERIN D.: Exact and efficient construction of minkowski sums of convex polyhedra with applications. *Comput. Aided Des. 39*, 11 (2007), 929–940.

[Gho93] GHOSH P. K.: A unified computational framework for minkowski operations. *Comp. & Graph. 17*, 4 (1993), 357–378.

[GS87] GUIBAS L. J., SEIDEL R.: Computing convolutions by reciprocal search. *Discr. & Comput. Geom. 2* (1987), 175–193.

[GXG08] GUO X., XIE L., GAO Y.: Optimal accurate minkowski sum approximation of polyhedral models. In *ICIC (1)* (2008), pp. 179–188.

[Hac07] HACHENBERGER P.: Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In *Proc. Eur. Symp. Alg.* (2007), vol. 4698, pp. 669–680.

[JSC04] JUNG W., SHIN H., CHOI B. K.: Self-intersection removal in triangular mesh offsetting. *Computer-Aided Design and Applications 1* (2004), 477–484.

[Ju04] JU T.: Robust repair of polygonal models. In *ACM Trans. Graph. 23, 3* (2004), pp. 888–895.

[Kim03] KIM J. J.: Constructing the boundaries of swept volumes for screw motions. *JSME Int. Series C 46*, 3 (2003), 1142–1150.

[KR92] KAUL A., ROSSIGNAC J.: Solid-interpolating deformations: Construction and animation of pips. *Computers & Graphics 16*, 1 (1992), 107–115.

[KVLM03] KIM Y. J., VARADHAN G., LIN M. C., MANOCHA D.: Fast swept volume approximation of complex polyhedral models. In *Proc. Symp. Solid mod. and appl.* (2003), pp. 11–22.

[LBW97] LEU M. C., BLACKMORE D., WANG L.: The sweep-envelope differential equation algorithm and its application to NC machining verification. *CAD 29*, 9 (1997), 629–637.

[Lie08] LIEN J.-M.: A simple method for computing minkowski sum boundary in 3d using collision detection. *8th Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)* (2008).

[Mat75] MATHERON G.: *Random sets and integral geometry*. Wiley New York,, 1975.

[NT99] NOORUDDIN F. S., TURK G.: *Simplification and repair of polygonal models using volumetric techniques*. Technical Report GITGVU -99-37, Georgia Institute of Technology, 1999.

[Par04] PARK S. C.: Triangular mesh intersection. *Vis. Comput. 20*, 7 (2004), 448–456.

[PK08] PAVIĆ D., KOBBELT L.: High-resolution volumetric computation of offset surfaces with feature preservation. *Comput. Graph. Forum 27*, 2 (2008), 165–174.

[PS07] PETERNELL M., STEINER T.: Minkowski sum boundary surfaces of 3d-objects. *Graph. Models 69*, 3-4 (2007), 180–190.

[RK01] ROSSIGNAC J., KIM J. J.: Computing and visualizing pose-interpolating 3d motions. *CAD 33*, 4 (2001), 279–291.

[RKS*07] ROSSIGNAC J., KIM J. J., SONG S. C., SUH K. C., JOUNG C. B.: Boundary of the volume swept by a free-form solid in screw motion. *CAD 39*, 9 (2007), 745–755.

[SH02] SHAUL H., HALPERIN D.: Improved construction of vertical decompositions of three-dimensional arrangements. In *SCG '02: Proc. Symp. on Comp. Geom.* (2002), ACM, pp. 283–292.

[SK01] SCHWANECKE U., KOBBELT L.: Approximate envelope reconstruction for moving solids. *Mathematical Methods for Curves and Surfaces: Oslo 2000* (2001), 455–466.

[SLL94] SCHROEDER W. J., LORENSEN W. E., LINTHICUM S.: Implicit modeling of swept surfaces and volumes. In *Proc. Conf. Visualization* (1994), IEEE Computer Society Press, pp. 40–45.

[VM06] VARADHAN G., MANOCHA D.: Accurate minkowski sum approximation of polyhedral models. *Graphical Models 68*, 4 (2006), 343–355.

[WL90] WELD J. D., LEU M. C.: Geometric representation of swept volumes with application to polyhedral objects. *Int. J. Rob. Res. 9*, 5 (1990), 105–117.

[WSD03] WU Y., SHAH J. J., DAVIDSON J. K.: Improvements to algorithms for computing the minkowski sum of 3-polytopes. *Computer-Aided Design 35*, 13 (2003), 1181–1192.

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3 (2009), 1–10.