Point-Based Minkowski Sum Boundary

Jyh-Ming Lien
jmlien@cs.gmu.edu
George Mason University
Fairfax, VA, 22030, U.S.A.

Abstract

Minkowski sum is a fundamental operation in many geometric applications, including robotics, penetration depth estimation, solid modeling, and virtual prototyping. However, due to its high computational complexity and several nontrivial implementation issues, computing the exact boundary of the Minkowski sum of two arbitrary polyhedra is generally a difficult task. In this work, we propose to represent the boundary of the Minkowski sum approximately using only points. Our results show that this point-based representation can be generated efficiently. An important feature of our method is its straightforward implementation and parallelization. We also demonstrate that the point-based representation of the Minkowski sum boundary can indeed provide similar functionality as mesh-based representations can. We show several applications in motion planning, penetration depth approximation and modeling. An implementation of the proposed method can be obtained from our project webpage at: http://www.cs.gmu.edu/∼jmlien/mksum/

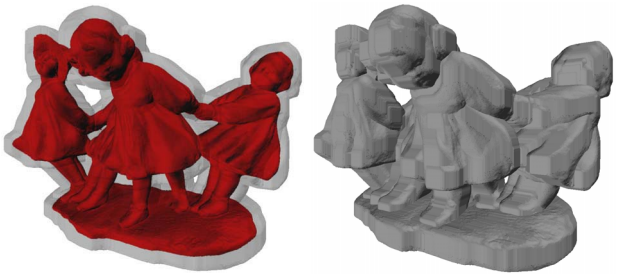


Figure 1. The Minkowski sum boundary (right) of the "dancing children" model (left) and a unit cube. The Minkowski sum boundary is composed of 274,976 points generated in 34.5 seconds using four parallel threads.

1 Introduction

Minkowski sum of two sets P and Q in $\mathbb{R}^d$ is defined as:

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\}. \quad (1)$$

Typically, P and Q represent polygons in $\mathbb{R}^2$ or polyhedra in $\mathbb{R}^3$. Minkowski sum is an important operation due to it's fundamental role in many geometric applications, including image analysis, robotics, penetration depth estimation, solid modeling, and virtual prototyping, to name just a few.

Several methods have been proposed during the last three decades to compute Minkowski sum and its boundary; see surveys in [7,9,27]. In particular, due to the straightforward implementation in images, Minkowski operations comprise a wide spectrum of applications in mathematical morphology [24]. Even though computing Minkowski sums from continuous representations, e.g., polygons, is more difficult than from image-based representations, efficient methods have been proposed, e.g., using *convolutions* [12]. Even in 3-dimensions, several methods [7, 8, 11, 15] are known to compute the Minkowski sum of *convex* polyhedra efficiently.

For general 3-d polyhedra, a typical strategy for computing the Minkowski sum boundary of two polyhedra, denoted as P and Q, is to first apply convex decompositions to P and Q, and then compute the pairwise Minkowski sums between the decompositions of P and Q. The final Minkowski sum boundary is extracted from the union of all the pairwise Minkowski sums. Despite the popularity of this strategy, it has several disadvantages. First, because there can be $O(n + m)$ components produced by convex decomposition, the total number of the pairwise Minkowski sums can be very large, i.e., $O(mn)$, where m and n are the size of P and Q, respectively. For example, to compute the Minkowski sum of the Stanford bunny and the David model, whose convex surface decompositions contain 16549 and 85132 components, resp., we have to compute more than *1.4 billion* pairwise Minkowski sums! In the worst case, the Minkowski sum computation can take $O(m^3n^3)$ time [14]. Second, it is generally difficult to robustly generate the union of the pairwise Minkowski sums. Many degenerate cases need to be considered carefully in implementation. Even though several boolean-operation meth-

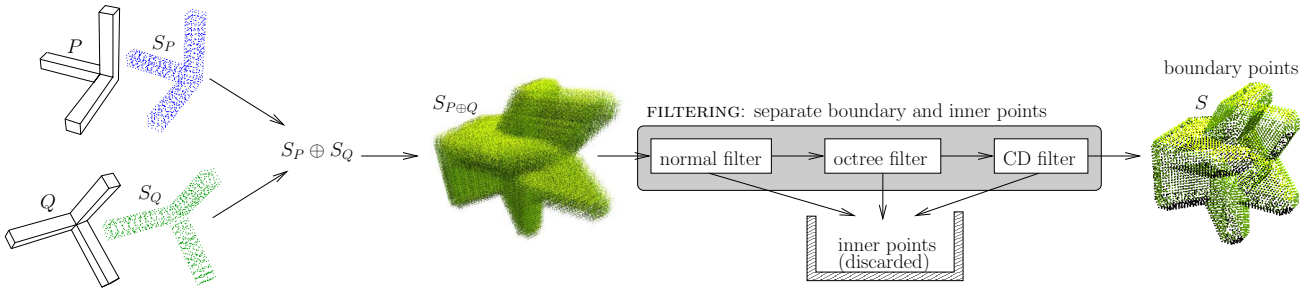


Figure 2. An overview of our method. First, point sets S_P and S_Q are sampled from the input polyhedra P and Q. The the Minkowski sum $S_{P\oplus Q}$ of S_P and S_Q is computed. Finally, the boundary points S is filtered (via normal filter, octree filter and collision detection (CD) filter) from $S_{P\oplus Q}$ to represent the Minkowski sum boundary of P and Q.

ods, e.g., [1, 13], have been proposed, no existing methods can be used to compute Minkowski sum properly due to the errors accumulated after performing a few union operations in a series. The recently proposed approximation-based Minkowski sum [27] is designed to avoid this issue.

It is these difficulties that motivate us to seek an alternative approach. Please note that this work does not attempt to address all these existing issues. Instead, our goal is to provided a very simple method to robustly compute an *approximate* (i.e., point based) and *accurate* (i.e., every point is a valid boundary point) representation of Minkowski sum boundary.

1.1 Our Approach

In this work, we propose an efficient method to compute the Minkowski sum boundary of two polyhedra. Our strategy is to **represent the boundary of the Minkowski sum using only points without connecting them into meshes**. Because of this point-based representation, our method is more efficient and easier to implement than mesh-based representations. In particular, our approach does not require convex decomposition, thus does not need to merge results from the sub-problems. Neither of these steps is trivial.

Our approach is simple. Given two polygons or polyhedra P and Q, our goal is to generate a point set S so that the external boundary $\partial(P \oplus Q)$ of the Minkowski sum $P \oplus Q$ is **well covered** (this term will be defined later in Section 3) by S. Figure 1 shows a result produced by our method.

To generate the point set S, we uniformly sample two point sets from the boundaries of both P and Q and name them S_P and S_Q. Then, each point in S_Q is replaced by S_P. We denote the resulting points of this step as $S_{P\oplus Q}$. Finally, in our last step, we filter out (inner) points that are not on the boundary. Later in this paper, we will introduce three filters, namely collision detection filter (Section 4), normal filter (Section 5), and octree filter (Section 6). In Figure 2, we provide an overview of the proposed method.

It is clear that $S_{P\oplus Q}$ has $\Theta(mn)$ points, where m and n are the size of S_P and S_Q, respectively. Therefore, in the worst case, our approach takes $O(mn\mathrm{T}_{filter})$ time to compute a Minkowski sum boundary, where T_{filter} is the time complexity of filtering a single point, which is dominated by collision detection computation. Fortunately, because the proposed approach does not depend on the solutions obtained from the sub-problems, our method can be easily parallelized to handle large geometric models.

Minkowski sum is an important operation because it can be applied to many geometric problems. Therefore, it is crucial for us to show that our point-based representation can also provide a wide range of applications. In Section 8, we will show that the point-based representation of the Minkowski sum boundary can indeed provide similar functionality as mesh-based representations. We demonstrate the applications in motion planning, penetration depth approximation and solid modeling using the proposed point-based Minkowski sum boundaries.

1.2 Key Contributions

We propose an algorithm to compute Minkowski sum boundary. The resulting representation is point based. Our approach gives up exact and continuous representation but gains several benefits which have not been provided by existing methods. These benefits include:

- efficiency,
- robustness (even for non-manifold models with open surfaces),
- easy implementation (i.e., no convex decomposition and no need to perform union),
- easy parallelization,
- multiresolution representations, and
- similar functionality as mesh-based representations

An implementation of the proposed method can be obtained (for non-commercial use) from our project webpage at: *http://www.cs.gmu.edu/∼jmlien/mksum/*

2 Related Work

Our work is inspired by the increasingly popular work on point set data (PSD) in computer graphics and computer vision, e.g., modeling [21], rendering [2,23], feature extraction [20], collision detection [16], boolean operation [1], mesh offsetting [6], and surface analysis [19]. One of the reasons for its popularity is that the connectivity information is not always easy to obtain and maintain. Similarly, in Minkowski sum computation, while obtaining the point-based representation is easy, obtaining an explicit or continuous representation, e.g., a mesh, can be difficult to compute.

Many methods have been proposed to compute Minkowski sum (see surveys in [7, 9, 27]). Here, we focus on work that computes the Minkowski sum boundaries of polygons and polyhedra.

Ghosh [9] proposed a unified approach to handle 2-d or 3-d convex and non-convex objects by introducing negative shape and slope diagram representation. Slope diagram is closely related to *Gaussian map*, which has been used to compute to implement very efficient Minkowski sum computation of convex objects by Fogel and Halperin [7].

Several other methods have been proposed to handle convex objects. Guibas and Seidel [12] proposed an output sensitive method to compute convolution curves, a superset of the Minkowski sum boundaries. Kaul and Rossignac [15] proposed a linear time method to generate a set of Minkowski sum facets. Output sensitive methods that compute the Minkowski sum of polytopes in d-dimension have also been proposed by Gritzmann and Sturmfels [11] and Fukuda [8].

Because the Minkowski sum of convex polyhedra is easy to compute, most methods that compute the Minkowski sum of non-convex polyhedra first compute the convex decomposition and then compute the union of the Minkowski sums of the convex components [18]. Unfortunately, neither the convex decomposition nor the union of the Minkowski sums is trivial to compute. In this paper, we propose a new method to compute the point-based representation of the Minkowski sum without computing the union and the convex decomposition.

Following the same divide-and-conquer technique, Varadhan and Manocha [27] proposed an approach to generate meshes that approximate Minkowski sum boundary using marching cube technique to extract iso-surface from the signed distance field. They proposed an adaptive subdivision to improve the robustness and efficiency of their method. They demonstrate several applications, including motion planning [26], penetration depth estimation, and morphological operations. Because their approach still depends on convex decomposition, it still suffers from excessive number of convex components in the decomposition.

Peternell et al. [22] proposed a method to compute the Minkowski sum of two solids using points densely sampled from the solids, and compute local quadratic approximations of these points. However, their method only identifies the outer boundary of the Minkowski sum using a regular grid, i.e., no hole boundaries are identified. This can be a serious problem in particular for motion planning and penetration depth computation.

3 Point-Based Minkowski Sum Boundary

In this section, we will describe a method to compute Minkowski sum boundary in point-based representation. Our goal is to produce a set of points to *cover* the boundary of the Minkowski sum of two given polyhedra. More specifically, we will generate a point set S so that S is a d-covering of the Minkowski sum boundary, where d is a user specified value. Intuitively, d controls the sampling density of a boundary. A smaller d will produce a denser "approximation" of the boundary. A more precise definition of d-covering is provided below.

Definition 3.1. *d-covering. We say a set of points S is a d-covering of a surface M if, for every point m of M, there exists a point in S whose distance to m is less than d.*

Our strategy to accomplish this goal is straightforward. Our approach is composed of three main steps. First, we sample two point sets from the input P and Q. Second, we generate the Minkowski sum of the point sets simply using the definition in Equ. 1. Third, we separate the boundary points (both hole and external boundaries) from the internal points. Algorithm 3.1 outlines this strategy. In the following, we will discuss each of these main steps in detail.

Algorithm 3.1: POINT-BASED-MSUM(P, Q, d)

comment: P and Q are polyhedra and d defines sampling density

$S_P \leftarrow$sample(P, d); $S_Q \leftarrow$sample(Q, d)
$S_{P\oplus Q} \leftarrow \emptyset$
for each $p \in S_P$
  do { **for each** $q \in S_Q$
    do $S_{P\oplus Q} \leftarrow \{S_{P\oplus Q}, p + q\}$
$S \leftarrow$ FILTER$(P, Q, S_{P\oplus Q})$ (i)

Sample points. Let P and Q be two polyhedra. We generate two point sets from P and Q, denoted as S_P and S_Q. The point set S representing the Minkowski sum boundary of P and Q is simply

$$(S_P \oplus S_Q) \cap \partial(P \oplus Q) .$$

Because we want the point set S to cover the entire Minkowski sum boundary w.r.t. a user specified interval d, we have to make sure that the points S_P is a d_p-covering of ∂P and the points S_Q is a d_q-covering of ∂Q. It is our task to determine the values of d_p and d_q from the input d.

Fortunately, as shown in Theorem 3.2, we can guarantee that the final point set is at least a d-covering of the Minkowski sum boundary of P and Q by simply letting $d_p = d_q = d$. Moreover, since the boundaries of P and Q are known, we can easily make sure that S_P and S_Q d-cover ∂P and ∂Q, respectively.

Theorem 3.2. *Let S_P and S_Q be two d-covering point sets sampled from two polyhedral surface ∂P and ∂Q and let $S_{P\oplus Q} = S_P \oplus S_Q$ and $S = S_{P\oplus Q} \cap \partial(P \oplus Q)$. Then, S must be a d-covering point set of $\partial(P \oplus Q)$.*

Proof. A facet f on the Minkowski sum boundary can only come from two sources: A facet of P or Q or a pair of edges from P and Q [15]. It is obvious that when the facet f is from a facet of P or Q, $S_{P\oplus Q}$ must have enough points to d-cover the facet f. When the facet f is formed by two edges from P and Q, we should consider the worst case. Since the points from the edges are d-covering, in the worst case, these points will form a grid and each cell in the grid is a $d \times d$ square. In this case, the longest distance from an arbitrary point on the facet f to a grid point is

$$\sqrt{(\frac{d}{2})^2 + (\frac{d}{2})^2} = \frac{d}{\sqrt{2}} < d\,.$$

Therefore, in the worst case, the gird and therefore $S_{P\oplus Q}$ is a d-covering of the facet f. We conclude that $S_{P\oplus Q}$ (thus S) must be a d-covering point set of $\partial(P \oplus Q)$ if S_P and S_Q are d-covering of ∂P and ∂Q. □

Compute Minkowski sum. This step is straightforward. Using S_P and S_Q, we compute $S_{P\oplus Q}$ by simply following the Minkowski sum definition in Eqn. 1. It is obvious that the size of $S_{P\oplus Q}$ is $\Theta(mn)$, where m and n are the size of S_P and S_Q, respectively. Because of this quadratic order of growth, storing the coordinates of the entire point set $S_{P\oplus Q}$ in memory may become unpractical when m and n are both large. Fortunately, this problem can be easily addressed, i.e., we can always compute the point coordinate when needed without storing it.

Extract boundary points. In this final step, we separate (filter) points to two groups: Boundary points and inner points. Boundary points will be returned as our final answer and inner points will be discarded.

We propose three filters in this paper. The first filter, named **normal filter** discussed in Section 5, determines if a pair of sampled points (from P and Q, resp.) is an inner point by examining their *origins* (defined later in Definition 5.1) and orientations. The second filter, named **octree filter** described in Section 6, constructs an octree, which allows us to explore only points near the boundary and avoid definite inner points. These two filters are efficient, but they alone *cannot* filter out all inner points. The third filter, named **CD filter** described in Section 4, uses collision detection to separate boundary points from inner points. This last filter is computational more expensive but it provides an unambiguous decision.

These three filters can be combined to form the FILTER subroutine on line (i) of Algorithm 3.1. Several combinations are studied in our experiments (see Figure 5 in Section 8). Because CD filter is the simplest, we will discuss it first next.

4 Collision Detection (CD) Filter

Minkowski sum boundary is closely related to the concept of "contact space" in Robotics. Every point in the contact space represents a configuration that places the robot in contact with (but without colliding with) the obstacles. Given a translational robot P and obstacles Q, the contact space of P and Q can be represented as $\partial((-P) \oplus Q)$, where $-P = \{-p \mid p \in P\}$. In other words, if a point x is on the boundary of the Minkowski sum of two polyhedra P and Q, then the following condition must be true:

$$(-P^\circ + x) \cap Q^\circ = \emptyset\,,$$

where Q° is the open set of Q and $P+x$ denotes translating P to x. Using this observation, the CD filter simply places $-P$ at a point of $S_{P\oplus Q}$ and test if $-P$ is in collision with Q. If $-P$ is collision free, the point is reported as a point on the Minkowski sum boundary.

5 Normal Filter

In normal filter, we check a pair of points from S_P and S_Q and determine if it will form an inner point. Kaul and Rossignac [15] have shown that a facet of the Minkowski sum boundary can only come from a facet of P and a vertex from Q (or vice versa) or from a new facet formed by two edges of P and Q if the facet, vertex and edges are properly oriented [15]. Our strategy is derived directly from their observation. Since our points are sampled from the polyhedral surface, we first define the *origin* of a sample to ease our discussion.

Definition 5.1. *The origin of a sample x, denoted as $\mathcal{O}(x)$, is a facet, an edge or a vertex of a polyhedron from which x is sampled.*

Let p and q be a pair of points sampled from P and Q, respectively. We decide if $p+q$ is an inner point by checking the orientation of $\mathcal{O}(p)$ and $\mathcal{O}(q)$. There are only five cases we need to consider (the first two cases are illustrated in Figure 3):

1. $\mathcal{O}(p)$ is a vertex and $\mathcal{O}(q)$ is a facet or vice versa.
2. $\mathcal{O}(p)$ and $\mathcal{O}(q)$ are both edges.
3. $\mathcal{O}(p)$ is a vertex and $\mathcal{O}(q)$ is an edge or vice versa.
4. $\mathcal{O}(p)$ and $\mathcal{O}(q)$ are both vertices.
5. Otherwise.

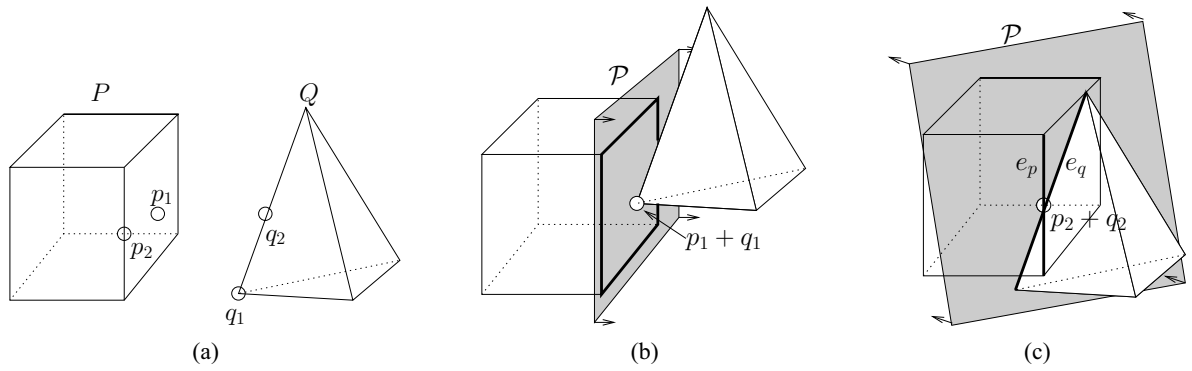


Figure 3. Normal filter. (a) An example of four points sampled from P and Q. Points p_1 and p_2 are sampled from a facet and an edge of P, respectively. Points q_1 and q_2 are sampled from a vertex and an edge of Q, respectively. (b) The point $p_1 + q_1$ is an example of the case 1. (c) The point $p_2 + q_2$ is an example of the case 2.

Case 1. First, we define a supporting plane $\mathcal{P}$ at the point $p+q$ parallel to facet $\mathcal{O}(q)$. Then, we translate P by q so that vertex $\mathcal{O}(p)$ coincides with the point $p+q$. The point $p+q$ must be an inner point when the (open) half space defined by the plane $\mathcal{P}$ intersects any edges incident to vertex $\mathcal{O}(p)$. An example of case 1 is shown in Figure 3(b).

Case 2. Similarly, we define a supporting plane $\mathcal{P}$ at point $p+q$ whose outward normal is the cross product of two vectors parallel to edges $\mathcal{O}(p)$ and $\mathcal{O}(q)$. Then, we translate P by q and Q by p so that edges $\mathcal{O}(p)$ and $\mathcal{O}(q)$ coincide with the plane $\mathcal{P}$. The point $p+q$ must be an inner point when the facets that incident to edges $\mathcal{O}(p)$ and $\mathcal{O}(p)$ are on the different sides of the plane $\mathcal{P}$. An example of this case is shown in Figure 3(c).

Case 3. Case 3 can be divided into two Case 1 and several Case 2. The point $p+q$ must be an inner point when Case 1 reports vertex $\mathcal{O}(p)$ and two incident facets of $\mathcal{O}(q)$ at $p+q$ as inner point and when Case 2 reports *all* edges incident to vertex $\mathcal{O}(p)$ and edge $\mathcal{O}(q)$ at $p+q$ as inner point.

Case 4. Case 4 can be divided into several Case 1 and Case 2. The point $p+q$ must be an inner point when Case 1 reports vertex $\mathcal{O}(p)$ and *all* incident facets of $\mathcal{O}(q)$ as inner point and also reports *all* incident facets of $\mathcal{O}(p)$ and vertex $\mathcal{O}(q)$ as inner points and when Case 2 reports *all* incident edges of $\mathcal{O}(p)$, *all* incident edges of $\mathcal{O}(q)$ at $p+q$ as inner point.

Case 5. All points in this category are considered as inner points.

Lemma 5.2 shows the correctness of the filter. Note that normal filter will not identify all inner points (unless P and Q are both convex), thus it needs to be used with CD filter.

Theorem 5.2. *Normal filter eliminates only inner points.*

Proof. Since boundary points can only exist on a subset of the supporting planes defined by a vertex and a facet or by two edges with properties described above [15], points that are not on these supporting planes must be inner points. □

6 Octree Filter

The goal of this octree filter is to reject points that are far away from the boundary. Our plan is to use a few known boundary points as "seeds" to guide (propagate) the search of unknown boundary points. For the rest of the section, we will first describe how the filter applies to the external boundary (Section 6.1) and then to the hole boundaries (Section 6.2).

6.1 Extract external boundary

The octree filter has two main steps: Obtain initial boundary points (seeds) and explore boundary using seeds.

Initial boundary points (seeds). External boundary can be exacted more easily (than hole boundaries) because we can quickly compute some initial boundary points (seeds) from $S_{P\oplus Q}$. In our implementation we simply use points on the minimum axis-aligned bounding box of $S_{P\oplus Q}$ as our seeds.

Explore boundary. Another reason why exacting external boundary is easier is stated in the following lemma.

Lemma 6.1. *If the Minkowski sum has only one (external) boundary, a point p must be an inner point, if all other points in a ball centered at p with radius d (which is the sampling density in Algorithm 3.1) are all inner points.*

Imagine superimposing a regular grid on $S_{P\oplus Q}$ with cell size d. Initially, each cell is marked as *unknown* cells except those that contain seeds and are marked as *boundary* cells. Now, we can start to explore the boundary by examining the points in the *unknown* cells that are neighboring to *boundary* cells. If all points in an *unknown* cell are reported as

inner points **by CD filter**, then this cell is marked as an *internal* cell. Otherwise the cell is marked as a boundary cell. This process is repeated until no *unknown* cells are next to *boundary* cell.

Adaptive octree. Instead of using a regular grid, we use an adaptive octree. In the adaptive octree, all *boundary* and *internal* cells in the octree have size d, however *unknown* cells can have size larger than d. An *unknown* cell will be subdivided when it is next to a *boundary* cell unless the size of the *unknown* cell is smaller than d. A benefit of using an adaptive octree is to avoid producing a huge number of cells. Exploration of the boundary using the adaptive octree is done in the same manner as using a regular grid. Note that this approach is similar to the surface-tracking algorithms, e.g., [25], in Marching Cubes method and to the inside-outside test in Adams and Dutré's boolean operations [1].

In Lemma 6.2, we show that an octree filter correctly extracts the external boundary.

Theorem 6.2. *Octree filter extracts all points on the external boundary.*

Proof. For simplicity we consider only the approach using a regular grid. The method using an octree can be proved in a similar way.

Because each grid cell has size at most d and the point set $S_{P\oplus Q}$ is a d-covering of the Minkowski boundary, a grid cell that intersects the boundary must contain at least one boundary point. Therefore, it is not possible for us to find an empty or a non-boundary cell on the boundary. This means we can always find all boundary cells (and boundary points) by propagating from one boundary cell. □

6.2 Extract hole boundaries

Hole boundaries are boundaries entirely enclosed in the external boundary. In many applications, such as animation [15], hole boundaries are less important because they are not "visible" from outside. However, for other applications, such as motion planning, hole boundaries usually represent critical pathways and cannot be ignored.

It is more difficult to efficiently extract hole boundary using the method we described above. The reason is that seeds for some hole boundaries are not easy to obtain. If we can find seeds for all hole boundaries, we can explore all boundaries as what we did for the external boundary. In fact, we can classify holes into easy holes and difficult holes. An easy hole has at least one vertex which is formed by a vertex of P and a vertex of Q. Otherwise, a hole is considered as difficult. A difficult hole has vertices formed as the intersections of edges or facets instead of the from the existing vertices. Figure 4 shows an example with one easy hole and three difficult holes.

Initial boundary points (seeds). We can still efficiently identify seeds for many hole boundaries. We identify a small set of boundary points using CD filter with the points that pass the test in the case 4 of the normal filter, i.e., points whose origins are vertices in P and Q. This set of points, in many cases, are small and scattered on the external and the hole boundaries and will be used as seeds for boundary exploration.

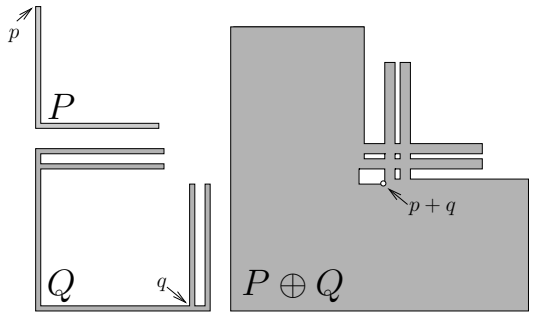


Figure 4. A 2D example shows four hole boundaries. Three small holes are difficult to generate seeds inside. The largest hole can be found more easily because two of its vertices (one circled) are from the vertices of P and Q. (This example is a simplified version of Halperin's Figure 5 [14].)

7 Speedup via Parallelization

More and more dual-core, quad-core and even multi-core processors are commercially available off-the-shelf and make access to parallel computing more easily than ever before. One advantage of our approach is the simplicity of parallelizing the method. In fact, parallelizing Algorithm 3.1 is an example of the so called embarrassingly parallel problem, i.e., we simply need to divide $S_{P\oplus Q}$ into k even-size point sets for k processors and put together the results computed by each processor without worrying about any dependency problems. In our current implementation, we parallelized the proposed method with less than 30 lines (including preparation and synchronization) of C++ code using the POSIX thread libraries. Experimental results are shown in Figure 8 and details of the results will be discussed in the next section.

8 Experimental Results and Applications

In this section, we show experimental results. All the experiments are performed on a PC with two Intel Core 2 CPUs at 2.13 GHz with 4 GB RAM. Our implementation is coded in C++. For detecting collision, we use RAPID [10]. In Section 8.1, we studied the efficiency and robustness of the proposed method using eight examples. In Section 8.2, we demonstrate the applications of the point-based Minkowski sum boundary, including offsetting, sweeping, motion planning and penetration depth approximation.

8.1 Experimental Results

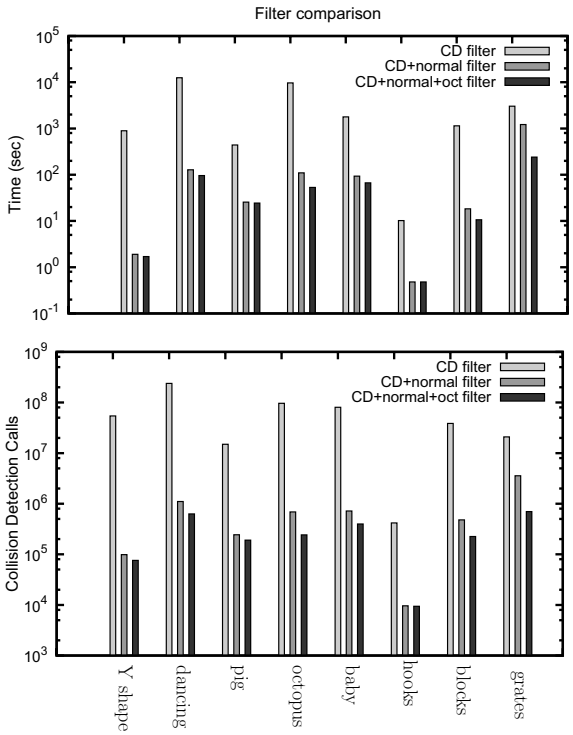
Boundary Point Filters. In this set of experiments, we compare three filters: CD filter, CD+normal (denoted as CD_n) filter, and CD+normal+oct (denoted as CD_n^o) filter. Two sets of experimental results using eight examples are shown in Figure 5. Please notice that the plots in Figure 5 are in logarithmic scale. From the results, we observe that, in all eight examples, computations using CD_n and CD_n^o filters are significantly faster by 1∼4 orders of magnitude than computations using CD filter alone. From Figure 5 we can see that CD_n^o filter takes *at most* 240 seconds for all eight examples while CD filter (except for hooks) requires *at least* 440 seconds. Similar results can also be observed from the collision detection counts. Filters CD_n and CD_n^o make significantly fewer collision detection calls than CD filter does. It is obvious that these significant improvements are due to the normal filter. Even though the octree filter can always improve the efficiency even further, the improvement is not as dramatic.

Robustness. Our goal here is to show that the proposed method is robust under difficult conditions, i.e., our method can generate correct results even for non-manifold models or models with surface openings. In Figure 6, we show the Minkowski sum of a cube and a model made of blocks. The blocks model is constructed by extruding (black) squares from a $\frac{1}{2}$ checkerboard and the cube has the same size of a checkerboard square. If look closer, you should find that the blocks model is non-manifold (all blocks are connected along their edges). As shown in Figure 6 our method correctly generates the Minkowski sum boundary.

Furthermore, we show that our method is sensitive enough to detect a small change that we made to this example. Instead of using the cube described above, we use a slightly (5%) smaller cube. As shown in the bottom two images of Figure 6, our method correctly produces the narrow columns as expected.

Multiresolution. Our method provides an easy way to generate multiresolution representations. By specifying a large d-covering value, we can create a low resolution boundary efficiently. The user can use this low resolution boundary as a quick preview. When the user decides a higher resolution boundary is needed, more sampled can be added. Figure 7 shows an example with four levels of detail.

Multithreading. As mentioned in Section 7, the proposed method can be easily parallelized. This advantage allows us to fully utilize the computation power provided by the multi-core processors. An experimental results obtained from a PC with two dual-core processors is shown in Figure 8. An interesting fact that we observe from Figure 8 is that the gap between the efficiency of filters CD_e and CD_e^o becomes smaller when we increase the number of threads.



	Y shape Figure 2	**dancing** Figure 1	**pig** Figure 9	**octopus** Figure 10
(n_p,n_q)	(7K,7K)	(0.7K,334K)	(13K,1K)	(19K,5K)
$n_\oplus$	67K	275K	90K	80K
	baby Figure 7	**hooks** Figure 8	**blocks** Figure 6	**grates** Figure 11
(n_p,n_q)	(17K,5K)	(0.6K,0.6K)	(41K,0.9K)	(5K,4K)
$n_\oplus$	58K	5K	192K	601K

Figure 5. Comparisons of Minkowski sum computations with CD filter, CD+norm (CD_n) filter and CD+norm+oct (CD_n^o) filter using eight examples. The top two plots show the computation time (using one thread) and collision detection calls of these three filters. The bottom table shows information of each example including the sizes of the sampled points (n_p,n_q) and the size of the Minkowski sum boundary points $n_\oplus$.

8.2 Applications

Modeling. Our method can be used to perform operations such as offsetting, erosion, and sweeping. Figure 1 shows an example of the offsetting operation of the "dancing children" model. Offsetting is done by computing its Minkowski sum with a unit cube or a sphere. Figure 9 shows an example of sweeping operation of a pig model. The sweep volume is generated by computing the Minkowski sum of the pig model and a thin tube representing a trajectory.

Motion Planning. A motion planning problem, which

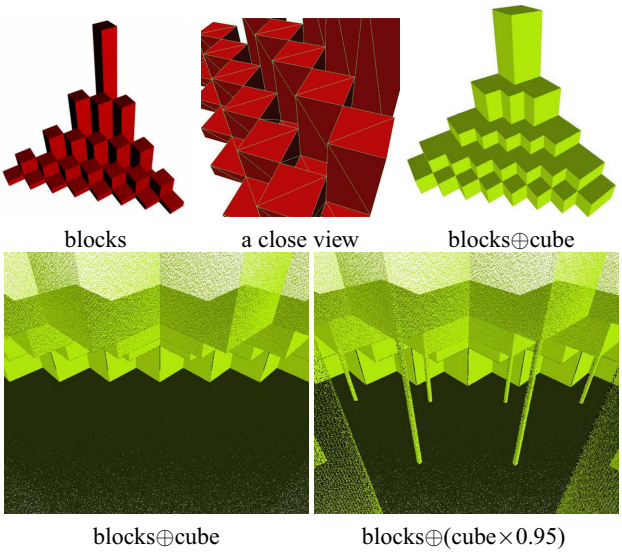


Figure 6. Top: A blocks model built from a $\frac{1}{2}$ checkerboard and its Minkowski sum with a cube with size of a checkerboard square. A close view reveals that the blocks model is non-manifold. Bottom: These two images show the dramatic difference between the Minkowski sums with the cube and with a 5% smaller cube. These images are taken from the same point of view inside of the Minkowski sum boundaries.

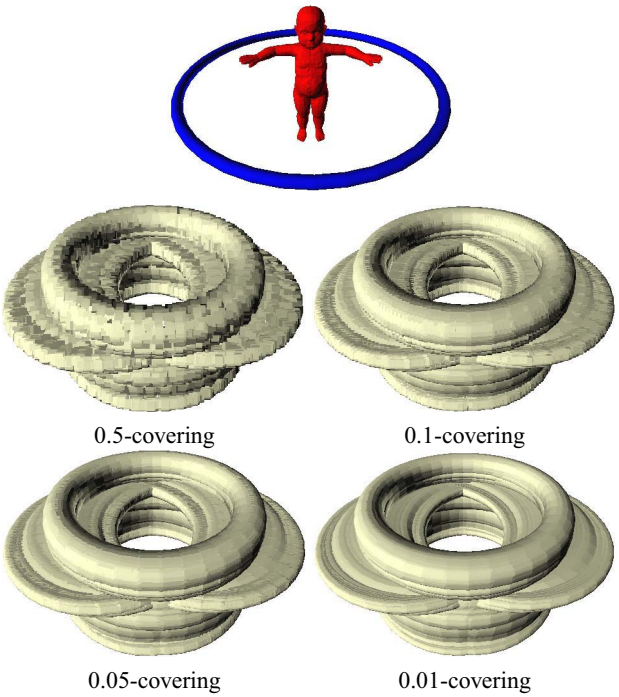


Figure 7. Multiresolution Minkowski sum of a baby model with a torus. Computing 0.5-, 0.1-, 0.05- and 0.01-covering (using four threads) takes 15.2, 26.0, 47.9 and 443.9 seconds, respectively, and generates 23K, 58K, 146K and 2652K points, respectively.

asks us to find a feasible path to bring an object from the start to the goal, can always be reduced to the problem of finding a sequence of consecutive points in the collision-free configuration space (denoted as C-free) [17]. The boundary of the C-free (called *contact space*) is closely related to the Minkowski sum boundary. Let P and Q be a translational robot and obstacle, respectively. The contact space of P and Q is $\partial(-P \oplus Q)$.

Sampling-based motion planners have been shown to solve difficult motion planning problems; see a survey in [4]. These methods approximate the C-free by sampling and connecting random configurations to form a graph (or a tree). However, they also have the difficulty of finding paths that are required to pass through narrow passages. Methods [3, 5] have been proposed to increase the random configurations in narrow passages by carefully sampling around obstacles. However, as far as we know, none of these obstacle-based methods can guarantee to increase *sampling ratio* in narrow passages. On the other hand, our Minkowski sum method can generate points to "cover" the contact space with a desired interval d (see Theorem 3.2) and therefore can guarantee to increase the sampling ratio in narrow passages even when the volume of the narrow passage is near zero. Points produced by our method can be connected into a graph (using simple local planners) as in sampling-based planners.

Penetration Depth Approximation. Penetration depth can be easily approximated using the point-based Minkowski boundary. Given a query configuration of two polyhedra P and Q, the penetration depth is the minimum translational distance of moving P away from colliding with Q.

Using the point-based Minkowski boundary, we can find the penetration depth of P by computing the closest point in S to the position of P, where S is a point set covering $\partial(-P \oplus Q)$. An example of this approach is illustrated in Figure 10. Because S is a d-covering of the true Minkowski sum boundary, we can be sure that $|PD' - PD| < d$ when d is small, where PD and PD' are the true and approximated penetration depths, respectively.

9 Conclusion and Discussion

In this paper, we propose a method that generates point-based Minkowski sum boundaries. We show that generating points on the surface of the Minkowski sum of two models is easier than generating a mesh to represent the Minkowski sum. We proposed three filters, i.e., CD filter, CD+normal filter, and CD+normal+oct filter, to identify boundary points. In the experiments, we observe that the combination of these three filters performs significantly

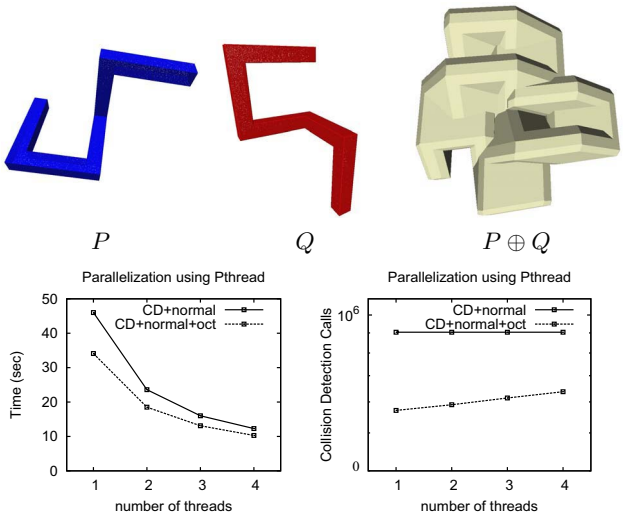


Figure 8. Multithreading. One to four threads are used to compute a 0.01-covering point set of the Minkowski sum boundary of two hook-like models.

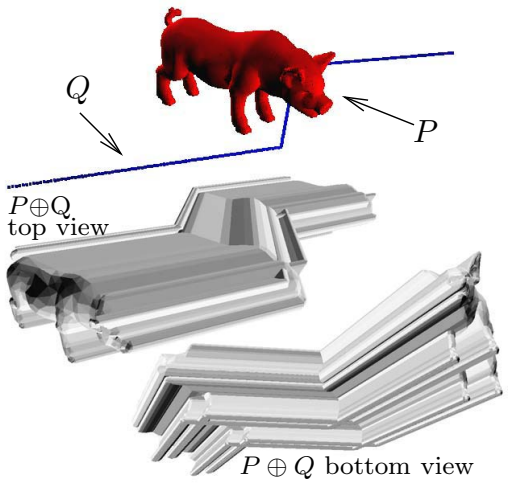


Figure 9. Our proposed method can be used to generate "volumetric sweep". This figure shows an example of creating a sweeping volume by moving a pig model along a line.

faster by *several orders of magnitude* than using only the CD filter. We showed that our method is robust and provides multiresolution and parallelization. We also demonstrate several applications using only points (i.e., without connecting them into meshes) on the Minkowski sum boundary. These applications provide an evidence that the point-based representation can have similar functionality as mesh-based representations.

Limitation and Future work. There are two major drawbacks in the current implementation. First, even though we proved that our method generates a d-covering point set, we may generate much more points than needed. Because points that pass tests in the case 1 or the case 2 in Section 5 may overlap. This overlapping can make points in some areas become $\frac{d}{3}$-covering! It is unclear to us how we can minimize the number points and still provide a d-covering point set. Second, our current implementation does not have any mechanism to create more points to enhance "new" sharp features on the Minkowski sum boundary (i.e., features that do not exist in the input polyhedra), e.g., in Figure 11, many points are needed to capture the small but important features of the grate models. We speculate that both of these two drawbacks are strongly related (re-sampling) issues [1].

Finally, point-based representation may not be used in some situations, such as in CAD, where continuous boundary representations are usually used. Therefore, we are interested in possible approaches and benefits of generating meshes from the points produced from our method.

References

[1] B. Adams and P. Dutre. Interactive boolean operations on surfel-bounded solids. *ACM Trans. Graph.*, 22(3):651–656, 2003.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[4] J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Rob. Res*, 16(6):759–774, 1997.

[5] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1018–1023, May 1999.

[6] Y. Chen, H. Wang, D. W. Rosen, and J. Rossignac. A point-based offsetting method of polygonal meshes, 2006. ASME Journal of Computing and Information Science in Engineering, in review.

[7] E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In *Proc. 8th Wrkshp. Alg. Eng. Exper. (Alenex'06)*, pages 3–15, 2006.

[8] K. Fukuda. From the zonotope construction to the minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004.

[9] P. K. Ghosh. A unified computational framework for minkowski operations. *Computers and Graphics*, 17(4):357–378, 1993.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Com-*

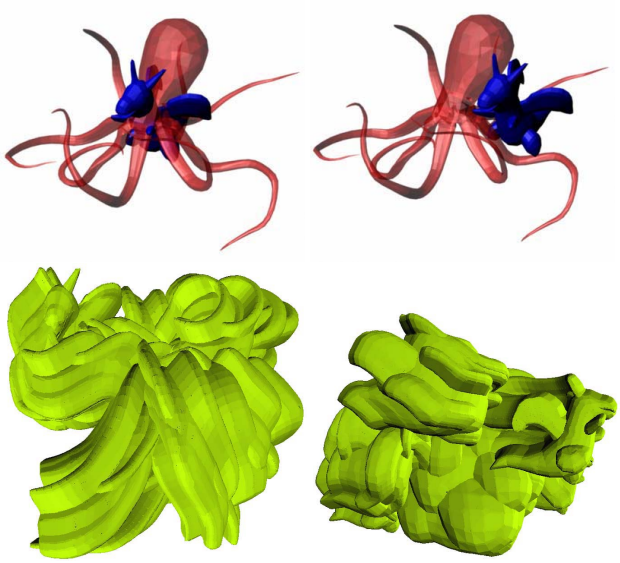


Figure 10. Top: (left) An octopus model is in collision with a dragon model. (right) Models are separated using the closest point on the point-based ∂(octopus$\oplus$-dragon). Bottom: Two different views of the Minkowski sum of the models. The penetration depth is computed as the distance from (0,0,0) to the closest point (-2.4, -3.8, 2.6) in the point set.

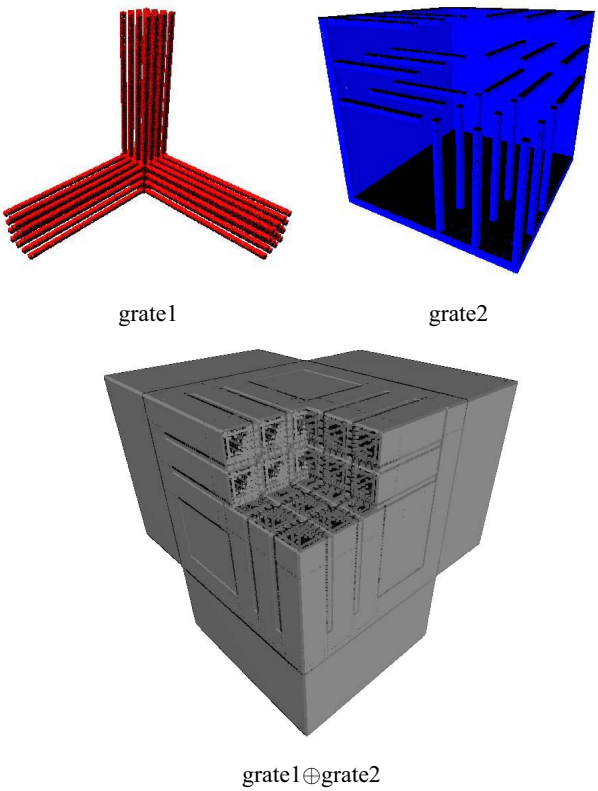


Figure 11. Minkowski sum of two grate-like models. These models imitate the grate models created by Varadhan and Manocha [27].

puter Graphics, 30(Annual Conference Series):171–180, 1996.

[11] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: computational complexity and applications to grobner bases. *SIAM J. Discret. Math.*, 6(2):246–269, 1993.

[12] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[13] P. Hachenberger and L. Kettner. Boolean operations on 3d selective nef complexes: optimized implementation and experiments. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 163–174, New York, NY, USA, 2005. ACM Press.

[14] D. Halperin. Robust geometric computing in motion. *The International Journal of Robotics Research*, 21(3):219–232, 2002.

[15] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In *Proc. Eurographics '91*, pages 493–505, 1991.

[16] J. Klein and G. Zachmann. Point cloud collision detection. In *Computer Graphics forum (EUROGRAPHICS)*, pages 567–576, 2004.

[17] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[18] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[19] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 379–386, New York, NY, USA, 2001. ACM Press.

[20] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 281–289, 2003.

[21] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 641–650, 2003.

[22] M. Peternell, H. Pottmann, and T. Steiner. Minkowski sum boundary surfaces of 3d-objects. Technical report, Vienna Univ. of Technology, August 2005.

[23] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[24] J. Serra, editor. *Image Analysis and Mathematical Morphology, volume 2: Theoretical Advances*. Academic Press, New York, 1988.

[25] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 335–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[26] G. Varadhan and D. Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Robotics Science & Systems*, 2005.

[27] G. Varadhan and D. Manocha. Accurate minkowski sum approximation of polyhedral models. *Graph. Models*, 68(4):343–355, 2006.