Computer Aided Geometric Design 25 (2008) 652–666

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Covering Minkowski sum boundary using points with applications

Jyh-Ming Lien *

MSN 4A5, 4400 University Drive, Fairfax, VA 22030, USA

ARTICLE INFO

Article history:
Available online 17 July 2008

PACS:
02.40.Dr

Keywords:
Point-based representation
Minkowski sum approximation
Parallelization
Geometric modeling
Motion planning
Penetration depth estimation

ABSTRACT

Minkowski sum is a fundamental operation in many geometric applications, including robotic motion planning, penetration depth estimation, solid modeling, and virtual prototyping. However, due to its high computational complexity and several non-trivial 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 demonstrate that the point-based representation of the Minkowski sum boundary can indeed provide similar functionality as the mesh-based representations can. We show several applications in motion planning, penetration depth approximation and geometric modeling. An implementation of the proposed method can be obtained from our project webpage.

1. Introduction

Minkowski sum is an important operation due to it's fundamental role in many geometric applications, including image analysis, robotic motion planning, penetration depth estimation, solid modeling, and virtual prototyping, to name just a few. In Fig. 1, we show the Minkowski sum of the David model and a sphere. The Minkowski sum of two sets P and Q in $\mathbb{R}^d$ is defined as:

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

Typically, P and Q represent polygons in $\mathbb{R}^2$ or polyhedra in $\mathbb{R}^3$.

During the last three decades, several methods have been proposed to compute the Minkowski sum and its boundary; see surveys in Ghosh (1993), Varadhan and Manocha (2006), Fogel and Halperin (2006). In particular, due to the straightforward implementations for images, Minkowski operations comprise a wide spectrum of applications in mathematical morphology (Serra, 1988). Even though computing the Minkowski sums for the continuous representations, e.g., polygons, is more difficult than for the image-based representations, many efficient methods have been proposed, e.g., using *convolutions* (Guibas and Seidel, 1987). Even in 3-dimensions, several methods (Kaul and Rossignac, 1991; Fogel and Halperin, 2006; Gritzmann and Sturmfels, 1993; Fukuda, 2004) 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 (Hachenberger, 2007). 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

* Tel.: +1 703 993 9546.
E-mail address: jmlien@cs.gmu.edu.

0167-8396/$ – see front matter
doi:10.1016/j.cagd.2008.06.006

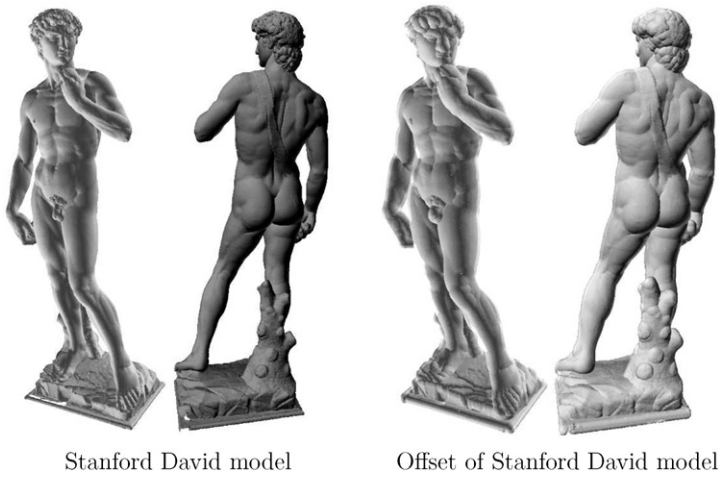


Fig. 1. The Minkowski sum boundary (right) of the "David" model (left) and a unit sphere. The Minkowski sum boundary is composed of 773 021 points generated in 224 seconds using four threads. The David model is composed of 493 904 facets.

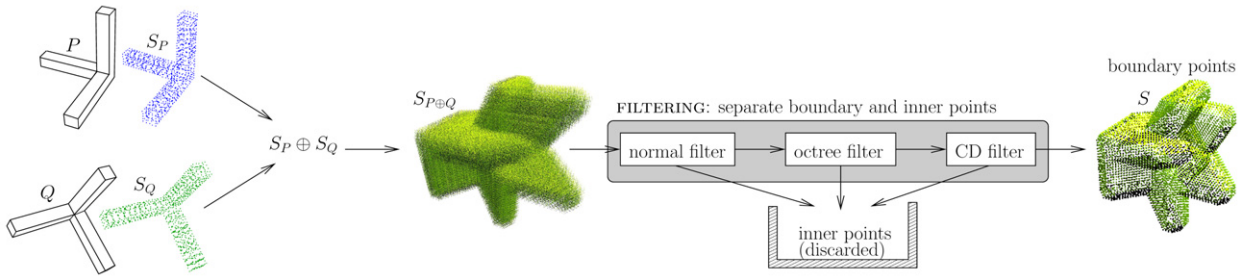


Fig. 2. An overview of our method. First, two point sets S_P and S_Q are sampled from the input polyhedra P and Q. Next, the Minkowski sum $S_{P\oplus Q}$ of S_P and S_Q is computed. Finally, the boundary points S are filtered (via normal filter, octree filter and collision detection (CD) filter) from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

can be $O(n+m)$ components produced by convex (surface) 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 16 549 and 85 132 components, resp., we have to compute more than *1.4 billion* pairwise Minkowski sums. 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. The recently proposed approximation-based approach (Varadhan and Manocha, 2006) is designed to avoid the union step.

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 is easier to implement than the methods that attempt to generate 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.

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 Minkowski sum 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 Fig. 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 (when both P and Q are convex), our approach takes $O(mnT_{\text{filter}})$ time to compute a Minkowski sum boundary, where T_{filter} is the time complexity of filtering a single point, which is dominated by the collision detection computation. Fortunately, as it will become more clear when we show our results in Section 9, when the input models are non-convex, many inner points will be filtered out before reaching the collision detection stage. Moreover, 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 9.2, 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 (Section 9.1),
- robustness (can even work for non-manifold models with open surfaces) (Section 9.1),
- easy implementation (i.e., no convex decomposition and no need to perform union),
- easy parallelization (Section 7),
- multiresolution representations (Fig. 8), and
- similar functionality as mesh-based representations (Section 9.2).

The preliminary version of this work is presented in the proceedings of the Pacific Graphics 2007 (Lien, 2007). An implementation of the proposed method can be obtained from our project webpage.

2. Related work

Our work is inspired by the increasingly popular work on point set data in computer graphics and computer vision, e.g., modeling (Pauly et al., 2003b), rendering (Rusinkiewicz and Levoy, 2000; Alexa et al., 2003), feature extraction (Pauly et al., 2003a), collision detection (Klein and Zachmann, 2004), mesh offsetting (Chen et al., 2006), and surface analysis (Pauly and Gross, 2001). 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 Ghosh, 1993; Varadhan and Manocha, 2006; Fogel and Halperin, 2006). Here, we focus on work that computes the polygonal and polyhedral Minkowski sum boundaries.

Ghosh (Ghosh, 1993) 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 efficiently compute the Minkowski sum of convex objects by Fogel and Halperin (2006).

Several other methods have been proposed to handle convex objects. Guibas and Seidel (1987) proposed an output sensitive method to compute convolution curves, a super-set of the Minkowski sum boundaries. Kaul and Rossignac (1991) 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 (1993) and Fukuda (2004).

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 (Lozano-Pérez, 1983). Unfortunately, neither the convex decomposition nor the union of the Minkowski sums is trivial. 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 (2006) 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 (Varadhan and Manocha, 2005), penetration depth estimation, and morphological operations. Because their approach still depends on convex decomposition, this approximate method still suffer from excessive number of convex components in the decomposition.

Peternell et al. (2005) 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 when we study problems in motion planning and penetration depth computation.

3. Point-based Minkowski sum boundary

In this section, we will describe a method to compute the Minkowski sum boundary in the 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 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 Eq. (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.

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 value d, we have to make sure that the points S_P form a d_p-covering of ∂P and the points S_Q form 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 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 cover ∂P and ∂Q, respectively.

Theorem 2. *Let S_P and S_Q be two d-covering point sets sampled from two polyhedral surfaces ∂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 form a d-covering 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 (Kaul and Rossignac, 1991). 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 worst case, these points will be vertices of 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

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

Therefore, in worst case, the grid 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 the 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 Eq. (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 coordinates when needed without storing it.

Algorithm 3.1 (*Point-based*-Msum P, Q, d)**.**
comment: P and Q are input polyhedra and d defines the sampling density

$S_P \leftarrow \text{sample}(P, d)$

$S_Q \leftarrow \text{sample}(Q, d)$

$S_{P\oplus Q} \leftarrow \emptyset$

for each $p \in S_P$

do { **for each** $q \in S_Q$
  do $S_{P\oplus Q} \leftarrow S_{P\oplus Q} \cup (p+q)$

$S \leftarrow \text{FILTER}(P, Q, S_{P\oplus Q})$ (i)

Extract the boundary points. In this final step, we separate (filter) points to two groups: boundary points and inner points. The boundary points will be returned as our final answer and the 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 3) 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 the definite inner points. These two filters are efficient, but they alone *cannot* filter out all the inner points. The third filter, named **CD filter** described in Section 4, uses collision detection to separate the boundary points from the 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 of these filters are further studied in our experiments (Fig. 5 in Section 9). Because CD filter is the simplest, we will discuss it first next.

4. Collision Detection (CD) filter

In Robotics, the contact space, in which every point represents a configuration that places the robot in contact with (but without colliding with) the obstacles, can be computed using Minkowski sum. 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 (1991) 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 (Kaul and Rossignac, 1991). 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 3. 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 Fig. 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.

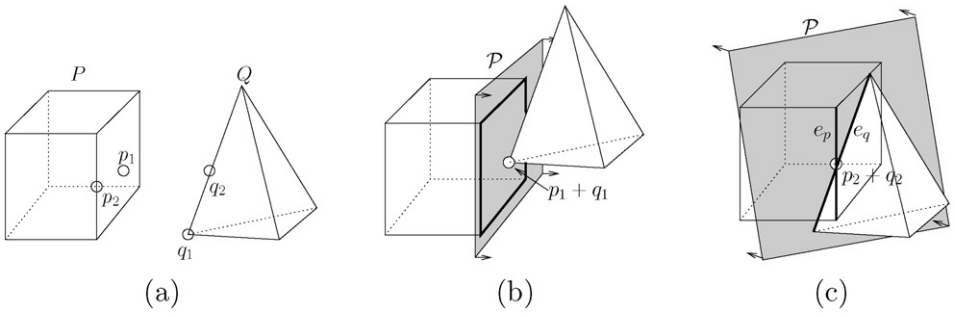


Fig. 3. Normal filter. (a) An example of four points sampled from P and Q. Points p_1 and p_2 are sampled from a facet and an edge of P, respectively. Points q_1 and q_2 are sampled from a vertex and an edge of Q, respectively. (b) The point $p_1 + q_1$ is an example of the case 1. (c) The point $p_2 + q_2$ 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 Fig. 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 Fig. 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 4. *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 (Kaul and Rossignac, 1991), points that are not on these supporting planes must be inner points. □

Lemma 4 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 the CD filter.

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 extracted 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 5. *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 the 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 a *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., Shekhar et al. (1996), in Marching Cubes method.

In Lemma 6, we show that the octree filter correctly extracts the external boundary.

Lemma 6. *The 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.

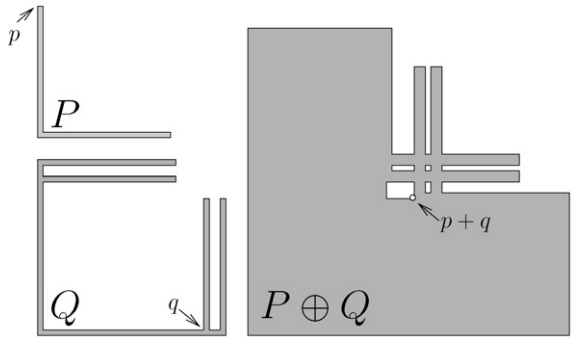


Fig. 4. A 2D example shows four hole boundaries. Three small holes are difficult to generate seeds inside. The largest hole can be found more easily because two of its vertices (one circled) are from the vertices of P and Q. (This example is a simplified version of Halperin's Fig. 5 (Halperin, 2002).)

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 the boundaries entirely enclosed in the external boundary. In many applications, such as animation (Kaul and Rossignac, 1991), 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. Fig. 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.

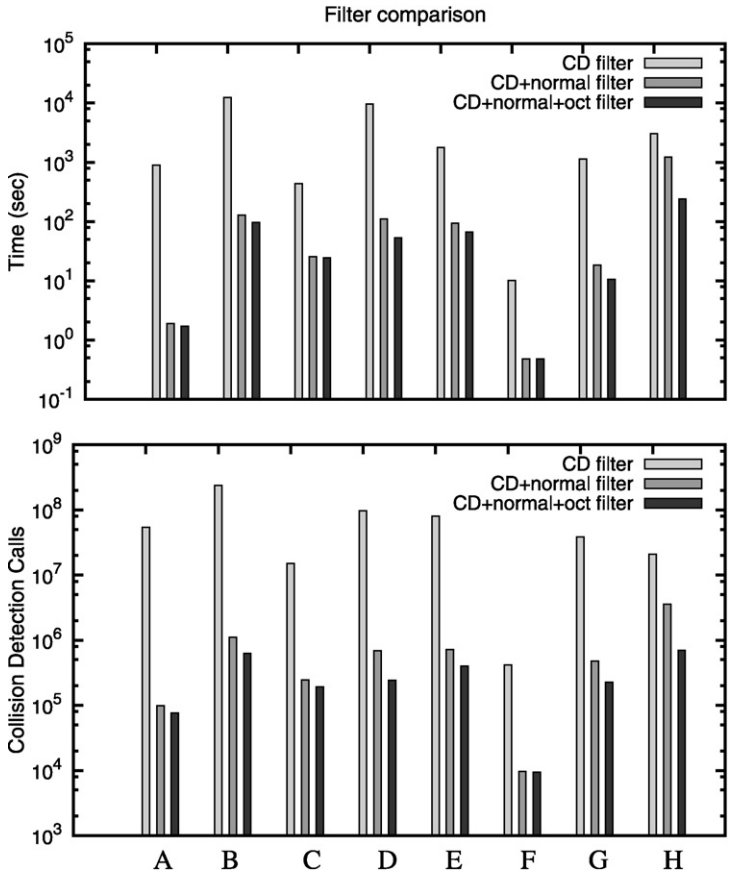
7. Speedup via parallelization

More and more dual-core, quad-core and even multi-core processors are commercially available and make access to parallel computing more easily than ever. 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 library. Our experimental results in Fig. 9 show a near linear speedup. Details of the results will be discussed in Section 9.

8. Detecting collisions

For detecting collisions, we use a modified version of RAPID (Gottschalk et al., 1996). The modification is mainly for the purpose of parallelization.

Another issue that we have to deal with when working with RAPID is that RAPID cannot distinguish if two polyhedra are in a contact configuration or are in fact in collision. To work around this problem, we perturb each point that we sampled with an infinitesimally small vector pointing in the outward direction of the facet where the point is sampled from. After the perturbation, the point will most likely become collision free if it is indeed on the Minkowski sum boundary. On the other hand, if the point is an inner point, the point will most likely remain inside the Minkowski sum after the perturbation. The exceptions to the above cases rarely occur, i.e., when the boundary degenerates to an isolated vertex, an edge or a sliver (a very small volume). Since our method focuses on providing an approximation to the Minkowski sum boundary, we do not consider these rare cases in this work.



	A (Fig. 2) **Y ⊕ Y**	B (Fig. 10) **dancing ⊕ cube**	C (Fig. 11) **pig ⊕ line**	D (Fig. 13) **octopus ⊕ dragon**
n_p	(7 K, 7 K)	(0.7 K, 334 K)	(13 K, 1 K)	(19 K, 5 K)
$n_\oplus$	67 K	275 K	90 K	80 K
	E (Fig. 8) **baby ⊕ torus**	F (Fig. 9) **hook ⊕ hook**	G (Fig. 6) **blocks ⊕ cube**	H (Fig. 15) **grate ⊕ grate**
n_p	(17 K,5 K)	(0.6 K,0.6 K)	(41 K,0.9 K)	(5 K,4 K)
$n_\oplus$	58 K	5 K	192 K	601 K

Fig. 5. Comparisons of Minkowski sum computations with CD filter, CD + norm (CD_n) filter and CD + norm + oct (CD_{no}) filter using eight examples. The top two plots show the computation time (using one thread) and the collision detection calls of these three filters. The bottom table shows information of each example including the size of the sampled points n_p and the size of the Minkowski sum boundary points $n_\oplus$. For all the computations in this experiment, we use $d = 0.1$.

9. 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++. In Section 9.1, we study the efficiency and robustness of the proposed method using eight examples. In Section 9.2, we demonstrate the applications of the point-based Minkowski sum boundary, including offsetting, sweeping, motion planning and penetration depth approximation.

9.1. 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_{no}) filter. Two sets of experimental results using eight examples are shown in Fig. 5. Please notice that the plots in Fig. 5 are in logarithmic scale. From the results, we observe that, in all eight examples, the computations using CD_n and CD_{no} filters are significantly faster (by $1 \sim 4$ orders of magnitude) than the computations using CD filter alone. From Fig. 5 we can see that CD_{no} 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.

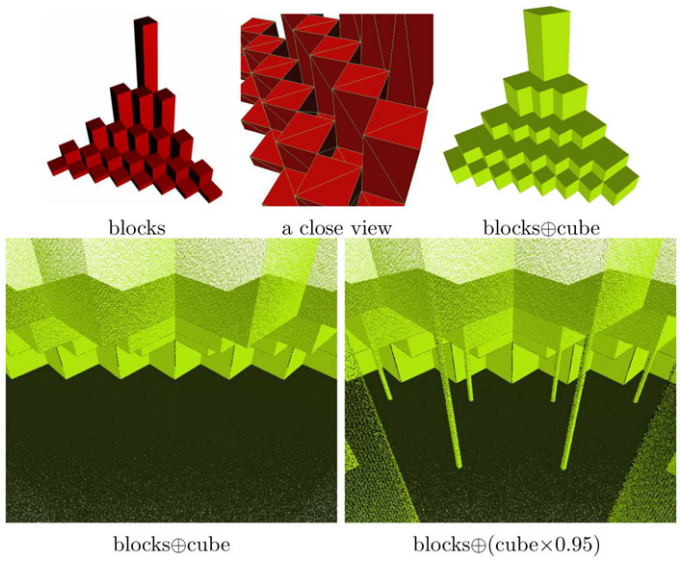


Fig. 6. Top: A blocks model built from a $\frac{1}{2}$ checkerboard and its Minkowski sum with a cube with size of a checkerboard square. A close view reveals that the blocks model is non-manifold. Bottom: These two images show the dramatic difference between the Minkowski sums with the cube and with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

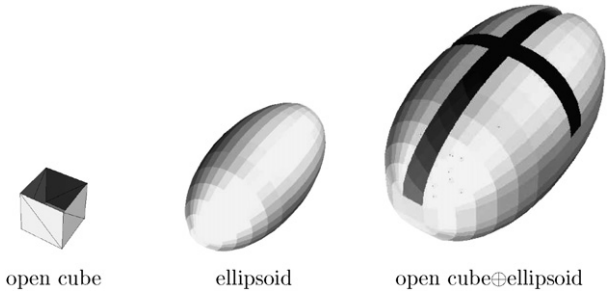


Fig. 7. The Minkowski sum of meshes with boundary. The cube's top is removed to create a boundary. The resulting Minkowski sum of the open cube and an ellipsoid is generated with an opening at the top.

Filters CD_n and CD_{no} 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 Fig. 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 touch other blocks along their vertical edges). As shown in Fig. 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 Fig. 6, our method correctly produces the narrow columns as expected.

In Fig. 7, we show the Minkowski sum of an ellipsoid and a cube with its top facet removed. In this case, as one may expect, their Minkowski sum boundary is no longer closed. Even though the Minkowski sum of open meshes is ill-defined (i.e., adding a 3-d volume and a 2-manifold with boundary in 3-d), the proposed method successfully generates a natural solution. This feature can be useful to some geometric operations, such as offsetting, when the input models are not watertight.

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. Fig. 8 shows an example with four levels of detail.

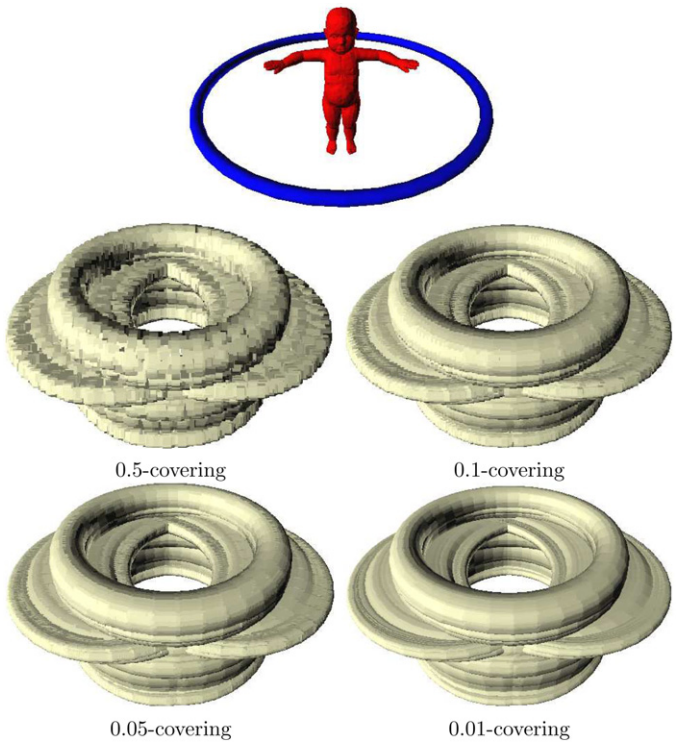


Fig. 8. Multiresolution Minkowski sum of a baby model with a torus. Computing 0.5-, 0.1-, 0.05- and 0.01-covering (using four threads) take 15.2, 26.0, 47.9 and 443.9 seconds, respectively, and generate 23 K, 58 K, 146 K and 2652 K points, respectively.

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 Fig. 9. An interesting fact that we observe from Fig. 9 is that the gap between the efficiency of filters CD_n and CD_{no} becomes smaller when we increase the number of threads.

9.2. Applications

Modeling. Our method can be used to perform operations such as offsetting, erosion, and sweeping. Figs. 1 and 10 show examples of the offsetting operation of the David and the "dancing children" model. Offsetting is done by computing its Minkowski sum with a unit cube or a sphere. Fig. 11 shows an example of sweeping operation of a pig model. The swept 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 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 $\mathcal{C}_{\text{free}}$) (Latombe, 1991). The boundary of the $\mathcal{C}_{\text{free}}$ (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 (Barraquand et al., 1997). These methods approximate the $\mathcal{C}_{\text{free}}$ 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 (Amato et al., 1998; Boor et al., 1999) 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 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.

More specifically, the key idea of our planner is that we can in fact decompose the point-based Minkowski sum boundary into a set of (approximately) star-shaped components. A point set is star shaped if and only if there exists at least one point in the set which can *see* all the points of the set (Lien, 2007). Fig. 12(a) shows an approximate start shape of the point x. Due to this interesting property, multiple overlapping star-shaped components covering $\mathcal{C}_{\text{free}}$ can easily form a network to represent the topology of $\mathcal{C}_{\text{free}}$. Such a network is usually called a roadmap (Latombe, 1991) and the motion planning problem can be solved by connecting the start and the goal configurations through the roadmap.

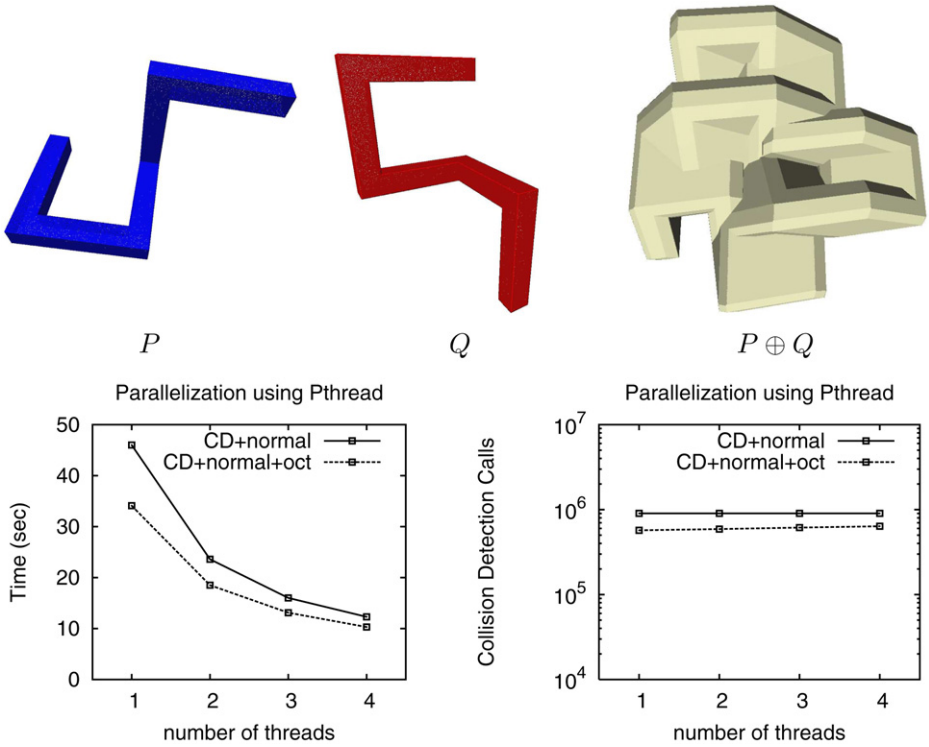


Fig. 9. Multithreading. One to four threads are used to compute a 0.01-covering point set of the Minkowski sum boundary of two hook-like models.

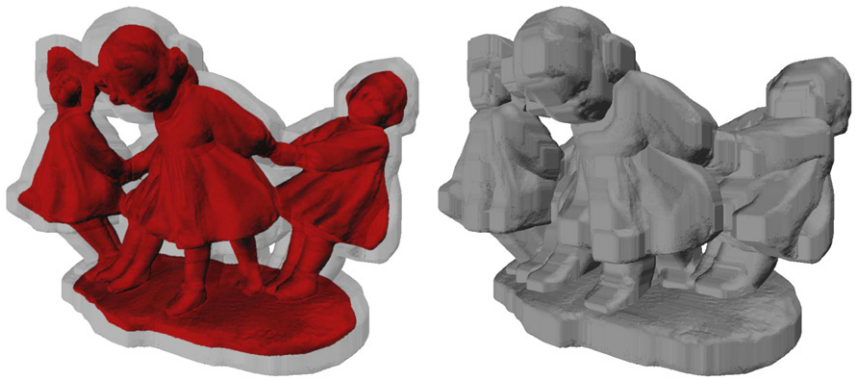


Fig. 10. The Minkowski sum boundary (right) of the "dancing children" model (left) and a unit cube. The Minkowski sum boundary is composed of 274 976 points generated in 34.5 seconds using four parallel threads.

The motion planning problem shown in Fig. 12(b) has a robot tightly fit into the hole of the obstacle. Our goal is to remove the robot from the holes. To generate the points covering the contact space, we sample 48 points from the robot and 74 points from the obstacle. The Minkowski sum has 1327 points on its boundary and the proposed method solves the problem in 0.58 seconds.

The main benefit of this motion planner is a deterministic method that can be potentially applied to solve problems involving high dimensional configuration spaces. In addition, our recent work also shows that when a problem can be solved by reusing some critical configurations (e.g., configurations in narrow passages), point-based Minkowski sum boundary can even further improve the efficiency of the existing motion planning methods (Lien, 2008).

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 $\partial(-P \oplus Q)$. Because S is a d-covering of the true Minkowski sum

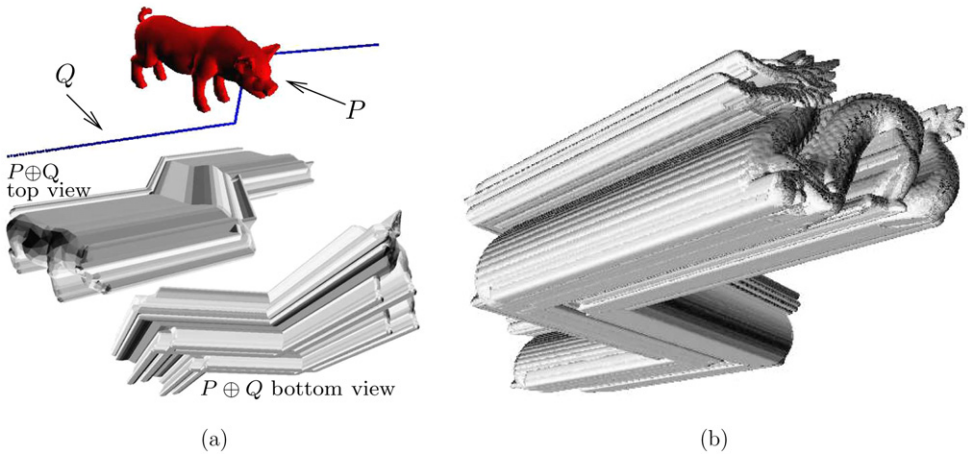


Fig. 11. Our proposed method can be used to generate "swept volume." Figure (a) shows an example of creating a swept volume by moving a pig model along a line. Figure (b) shows a swept volume of a more complex dragon model (822 504 facets) with the same path.

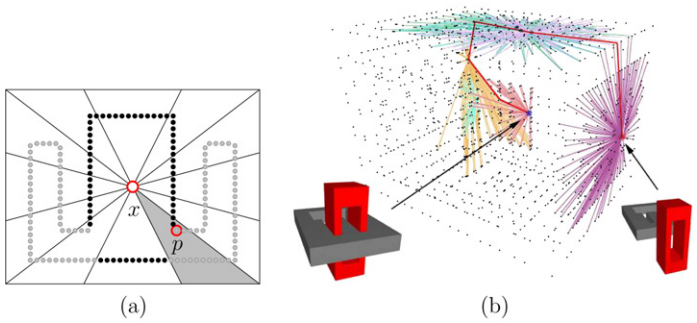


Fig. 12. (a) We identify a set of visible points (shown as the dark dots) of the point x by defining 'thick' viewing lines. The point p is invisible from x because p is blocked by other visible points of x. (b) The start and the goal configurations are shown along with the points that cover the contact space. A path that connects the start and the goal is shown in the thick line. The path is exacted from a set of 6 overlapping start-shape components.

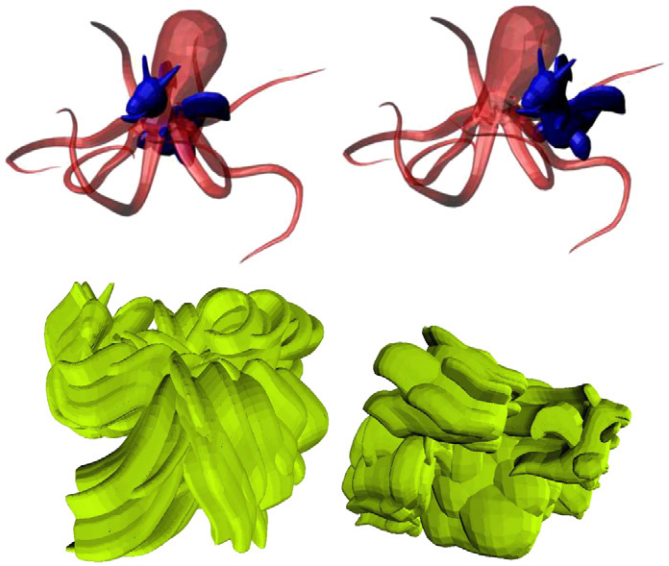


Fig. 13. Top: (left) An octopus model is in collision with a dragon model. (right) Models are separated using the closest point on the point-based ∂(octopus⊕-dragon). Bottom: Two different views of the Minkowski sum of the models. The penetration depth is computed as the distance from (0,0,0) to the closest point (−2.4, −3.8, 2.6) in the point set.

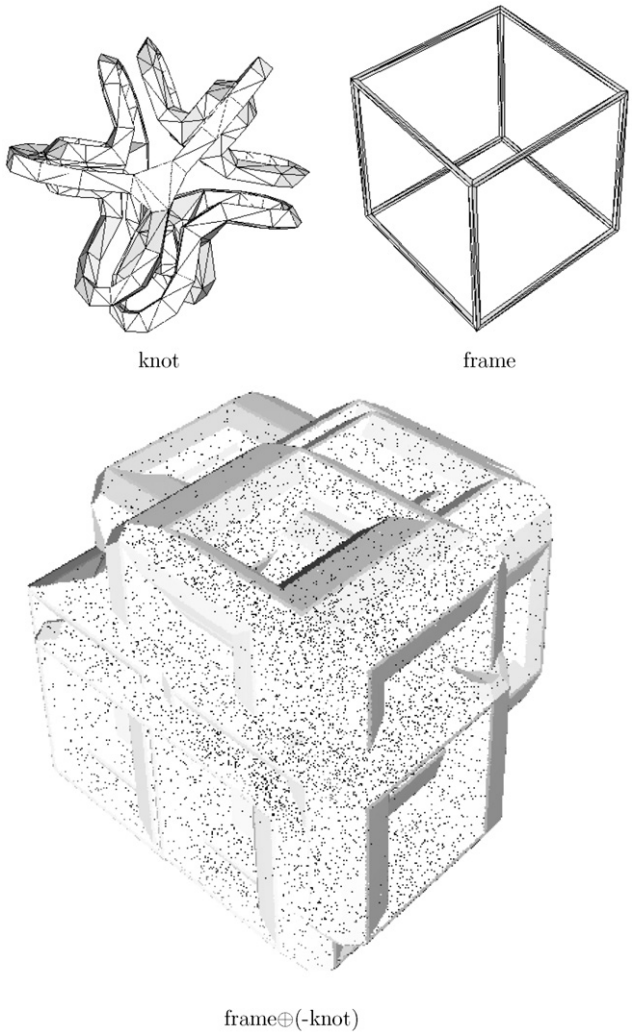


Fig. 14. A (slightly) harder PD problem. The Minkowski of two high genus models: the frame and the (-knot). Their Minkowski sum contains many hole boundaries.

boundary, we can be sure that $|PD' - PD| < d$ when d is small, where PD and PD' are the true and the approximate penetration depths, respectively.

An example of this approach is illustrated in Fig. 13. The Minkowski sum of an octopus (8276 facets) and a dragon (2328 facets) contains 226 773 points when $d = 0.05$. We then use the ANN library (Arya et al., 1998) to compute the closest point from a query point. The distance between the closest point and a query point is the approximate penetration depth. On average, each penetration depth query takes only 0.1 milliseconds (after the k-d tree in the ANN is initialized).

Another (slightly harder) example is shown in Fig. 14. In this example, both P (knot) and Q (frame) have several handles. When placing the knot in the center of the frame, it is not as intuitive as the previous example what the penetration depth of the knot will be. The main reason for this is because their Minkowski sum has many hole boundaries. Our method solves this problem by generating 769 969 points using $d = 0.1$. By moving the knot from $(0, 0, 0)$ to $(-3.2, 1.1, -0.04)$, the knot and the will frame become collision free.

In addition, when we decrease the value of d to 0.05, the number of points increases to 3 108 892 and the closest point becomes $(-3.21, 1.09, -0.008)$, which is in fact very close to the previous estimation when $d = 0.1$. On average, for both $d = 0.1$ or $d = 0.05$, each penetration depth query between the knot and the frame takes about 0.1 milliseconds using the proposed method and the ANN.

10. 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 that represents the Minkowski

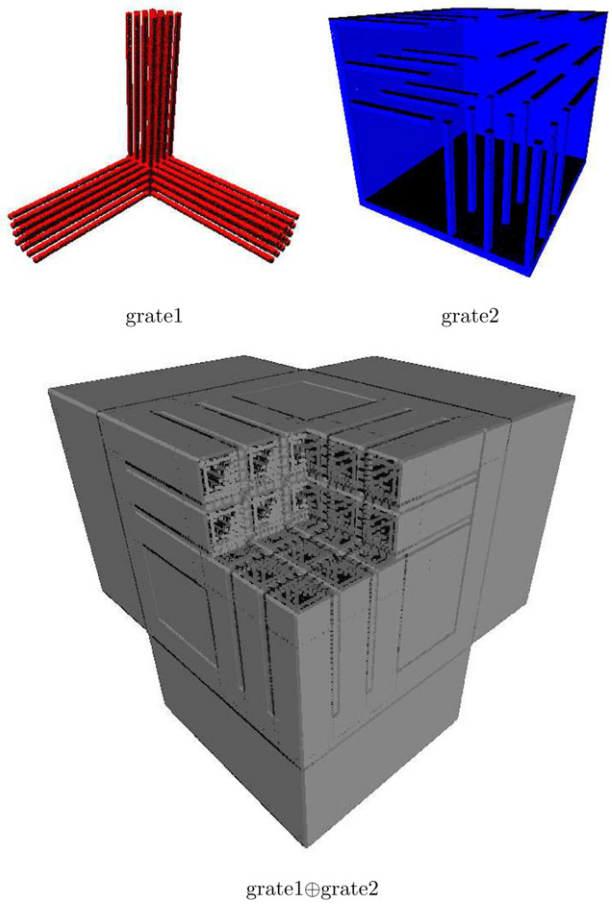


Fig. 15. Minkowski sum of two grate-like models. These models imitate the grate models created by Varadhan and Manocha (Varadhan and Manocha, 2006).

sum. We proposed three filters, i.e., CD filter, octree filter, and normal filter, to identify boundary points. In the experiments, we observe that the combination of these three filters performs significantly 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 a mesh) on the Minkowski sum boundary. These applications provide an evidence that the point-based representation can have similar functionality as the 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). For example, in Fig. 15, 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 (sampling) issues.

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 by our method.

Acknowledgement

The "dancing children" model is provided courtesy of IMATI-GE by the AIMSHAPE Shape Repository. The "David" and the "Dragon" models are obtained from the Stanford 3D Scanning Repository. The author also thanks anonymous reviewers for their valuable comments.

References

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

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

Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A., 1998. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM 45, 891–923.

Barraquand, J., Kavraki, L.E., Latombe, J.-C., Li, T.-Y., Motwani, R., Raghavan, P., 1997. A random sampling scheme for path planning. Int. J. Robot. Res. 16 (6), 759–774.

Boor, V., Overmars, M.H., van der Stappen, A.F., 1999. The Gaussian sampling strategy for probabilistic roadmap planners. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), vol. 2, May 1999, pp. 1018–1023.

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

Fogel, E., Halperin D., 2006. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In: Proc. 8th Workshp. Alg. Eng. Exper., Alenex '06, pp. 3–15.

Fukuda, K., 2004. From the zonotope construction to the Minkowski addition of convex polytopes. J. Symbolic Comput. 38 (4), 1261–1272.

Ghosh, P.K., 1993. A unified computational framework for Minkowski operations. Computers and Graphics 17 (4), 357–378.

Gottschalk, S., Lin, M.C., Manocha, D., 1996. OBBTree: A hierarchical structure for rapid interference detection. Computer Graphics, 30 (Annual Conference Series), pp. 171–180.

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

Guibas, L.J., Seidel, R., 1987. Computing convolutions by reciprocal search. Discrete Comput. Geom. 2, 175–193.

Hachenberger, P., 2007. Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In: Proc. 15th Annual European Symposium on Algorithms (ESA), pp. 669–680.

Halperin, D., 2002. Robust geometric computing in motion. Int. J. Robot. Res. 21 (3), 219–232.

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

Klein, J., Zachmann, G., 2004. Point cloud collision detection. In: Computer Graphics Forum (EUROGRAPHICS), pp. 567–576.

Latombe, J.-C., 1991. Robot Motion Planning. Kluwer Academic Publishers, Boston, MA.

Lien, J.-M., 2007. Approximate star-shaped decomposition of point set data. In: Proceedings of the IEEE/Eurographics Symposium on Point Based Graphics (PBG).

Lien, J.-M., 2007. Point-based Minkowski sum boundary. In: PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, Washington, DC, USA, IEEE Computer Society, pp. 261–270.

Lien, J.-M., 2008. Hybrid motion planning using Minkowski sums. In: Proc. Robotics: Sci. Syst. IV, Zurich, Switzerland, http://www.roboticsproceedings.org/rss04/p13.html.

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

Pauly, M., Gross, M., 2001. Spectral processing of point-sampled geometry. In: SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. ACM Press, New York, NY, pp. 379–386.

Pauly, M., Keiser, R., Gross, M., 2003a. Multi-scale feature extraction on point-sampled surfaces. In: Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 281–289.

Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M., 2003b. Shape modeling with point-sampled geometry. In: SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, pp. 641–650.

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

Rusinkiewicz, S., Levoy, M., 2000. Qsplat: a multiresolution point rendering system for large meshes. In: SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co, New York, NY, pp. 343–352.

Serra, J. (Ed.), 1988. Image Analysis and Mathematical Morphology, vol. 2: Theoretical Advances. Academic Press, New York.

Shekhar, R., Fayyad, E., Yagel, R., Cornhill, J.F., 1996. Octree-based decimation of marching cubes surfaces. In: Proceedings of the 7th Conference on Visualization '96. VIS '96, Los Alamitos, CA, USA. IEEE Computer Society Press, pp. 335–342.

Varadhan, G., Manocha, D., 2005. Star-shaped roadmaps – a deterministic sampling approach for complete motion planning. In: Proc. Robotics: Sci. Syst. (RSS).

Varadhan, G., Manocha, D., 2006. Accurate Minkowski sum approximation of polyhedral models. Graph. Models 68 (4), 343–355.