Accurate Minkowski Sum Approximation of Polyhedral Models

Gokul Varadhan Dinesh Manocha

University of North Carolina at Chapel Hill
{varadhan,dm}@cs.unc.edu
http://gamma.cs.unc.edu/recons

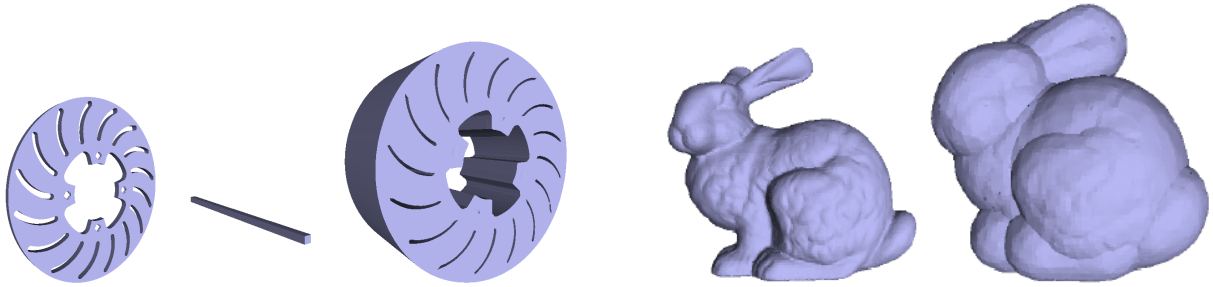


Figure 1. *The images on the left show the Minkowski sum of Brake Hub and Rod models. The Minkowski sum is obtained by sweeping one object around the other. The images on the right show the offset of the Bunny model. Offset of an object is obtained by sweeping a sphere around the object. It is a special case of the Minkowski sum with a sphere.*

Abstract

We present an algorithm to approximate the 3D Minkowski sum of polyhedral objects. Our algorithm decomposes the polyhedral objects into convex pieces, generates pairwise convex Minkowski sums and computes their union. We approximate the union by generating a voxel grid, computing signed distance on the grid points and performing isosurface extraction from the distance field.

The accuracy of the algorithm is mainly governed by the resolution of the underlying volumetric grid. Insufficient resolution can result in unwanted handles or disconnected components in the approximation. We use an adaptive subdivision algorithm that overcomes these problems by generating a volumetric grid at an appropriate resolution. We guarantee that our approximation has the same topology as the exact Minkowski sum. We also provide a two-sided Hausdorff distance bound on the approximation. Our algorithm is relatively simple to implement and works well on complex models. We have used it for exact 3D translation motion planning, offset computation, mathematical morphological operations and bounded-error penetration depth estimation.

1 Introduction

The problem of Minkowski sum computation arises in many applications including solid modeling, digital geometry processing, robotics, dynamic simulation and computer animation. The Minkowski sum of two sets P and Q is the set of points $\{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in P,\ \boldsymbol{q} \in Q\}$. Minkowski sum has a number of applications. They are useful as a tool to compute collision-free paths in robot motion planning [20], computer-aided design and manufacturing [18], satellite layout [4], penetration depth computation and dynamic simulation [16]. They have also been used for morphing [15], offset computation [21] and mathematical morphological operations [23].

Our goal is to compute the boundary of the 3D Minkowski sum of two polyhedral models. The Minkowski sum of two convex polytopes (with n features) can have $O(n^2)$ combinatorial complexity and is relatively simple to compute. On the other hand, the Minkowski sum of non-convex polyhedra can have complexity as high as $O(n^6)$ [13]. One of the commonly used approach to compute Minkowski sums decomposes the two non-convex polyhedra into convex pieces, computes their pairwise Minkowski sums and finally the union of the pairwise Minkowski sums. The main bottleneck in implementing such an algorithm is computing the union of pairwise Minkowski sums. Given m polyhedral primitives, their union can have combinatorial complexity $O(m^3)$ [2] and m can be high in the context of Minkowski sum computation (e.g. a few thousand). Furthermore, robust computation of the boundary of the union and handling all degeneracies remains a major issue [13, 1]. As a result, no practical algorithms are known for robust

computation of exact Minkowski sum of complex polyhedral models.

Main Results: We present a novel algorithm to approximate the Minkowski sum of polyhedral models. Instead of computing the exact union, we use distance field-based techniques to approximate the union of the pairwise Minkowski sums. Our algorithm generates an adaptive volumetric grid, computes a distance field, and performs isosurface extraction from it to obtain an approximation to the Minkowski sum. The accuracy of the algorithm is mainly governed by the rate of sampling, i.e., the resolution of the underlying volumetric grid. Insufficient resolution can result in unwanted handles or disconnected components in the approximation. Due to lack of resolution, the approximation may not capture many of the features, e.g. small holes, present in the exact Minkowski sum. We use an adaptive subdivision algorithm that generates a volumetric grid at a sufficient resolution such that a faithful approximation can be obtained by performing isosurface extraction on the resulting grid. We ensure a good quality of approximation by guaranteeing the correct topology as well as bounding the two-sided Hausdorff distance between the approximation and the exact Minkowski sum.

In order to speed up the computation, we employ two types of culling techniques during adaptive subdivision. Our algorithm performs *cell culling* to eliminate the grid cells that do not contain a part of the Minkowski sum boundary. Our algorithm also takes advantage of *primitive culling* and performs efficient distance and inside/outside queries by only considering a small subset of primitives, while preserving the correctness of these queries. In practice, these culling techniques improve the performance of the algorithm by more than *two orders* of magnitude.

We have used our Minkowski sum approximation algorithm for a number of applications. These include:

- Exact robot motion planning of robots with translation degrees of freedom.
- Offsets and mathematical morphological operations.
- Penetration depth estimation between overlapping polyhedra with tight error bounds.

Our algorithm is simple to implement and we have tested its performance on a number of benchmarks. The underlying polyhedral models consist of several hundreds of triangles. The computation of Minkowski sum takes few minutes on a 2 GHz Pentium IV processor.

Some of the novel results of our approach include:

- Approximate algorithm for computing Minkowski sum of polyhedral models.
- Culling techniques to improve the performance of adaptive subdivision and sampling scheme.
- Guaranteed topology and 2-sided Hausdorff distance bounds on the approximation.
- Application to motion planning, offset, and penetration depth computation.

To the best of our knowledge, this is the first algorithm that can compute a topologically accurate approximation of Minkowski sum of complex polyhedral models.

Organization

The rest of our paper is organized as follows. In Section 2, we review the earlier work on Minkowski sum computation. Section 3 gives an overview of our approach. In Section 4, we present our approximate algorithm to compute the boundary of Minkowski sum. Sections 5 discusses its application to motion planning, offsets and morphological operations, and penetration depth computation. We highlight its performance on various benchmarks in Section 6. Sections 7 discusses some limitations of our approach.

2 Previous Work

In this section, we give a brief survey of the related work. Many algorithms have been proposed for Minkowski sum computation in computational geometry and solid modeling [20, 12, 9, 11]. A survey can be found in [13].

Guibas and Seidel [12] proposed an output-sensitive algorithm for Minkowski sum of convex polytopes. They defined an operation, called convolution, on 2D planar tracings. Basch et al. [3] extended the convolution computation to 3D and defined it on polyhedral tracings. Convolution is a superset of the Minkowski sum and the exact Minkowski sum is extracted using arrangement computation.

Seong et al. [22] proposed an algorithm to compute Minkowski sum of surfaces generated by slope-monotone closed curves. Flato and Halperin [10] presented algorithms for robust construction of planar Minkowski sums and used it for computing 2D configuration space obstacles.

Lozano-Perez [20] used Minkowski sum to construct configuration space obstacles and used it for path planning. Kaul and Rossignac [15] proposed use of Minkowski sums for morphing and computer animation. They used weighted Minkowski sum to construct a parameterized interpolating polyhedron (PIP) that smoothly interpolates between two polyhedra. When the two polyhedra are nonconvex, PIP's faces form a superset of the Minkowski sum boundary.

Ghosh [11] presented a unified algorithm for computing 2D and 3D Minkowski sum of both convex and non-convex polyhedra based on a *slope diagram representation*. They reduce the problem of Minkowski sum into computing the slope diagrams of the two objects, merging their slope diagrams and extracting a boundary from the merged slope diagram.

Many of the above algorithms compute only a superset of surfaces that contribute to the Minkowski sum boundary [15, 3]; they do not explicitly compute the boundary. Evans et al. [9] present an approach for computing an explicit boundary of Minkowski sum of polyhedral models. Their algorithm is based on decomposing the two polyhedra into a collection of *affine* cells, computing pairwise Minkowski

sums between pairs of *transversal* affine cells, and computing their exact union. Affine cells correspond to features of the polyhedron such as a face, vertex, or an edge, etc. They have presented results on Minkowski sum of simple polyhedral models with a low polygon count. Although our overall approach is similar in some respects, there are crucial differences. We decompose the two polyhedra into convex pieces instead of affine cells. This produces fewer pieces compared to the number of affine cells. Moreover, instead of computing an exact union, we compute an approximate union. We believe this makes our algorithm applicable to more complex models. To the best of our knowledge, none of the previous algorithms can robustly compute Minkowski sum of complex polyhedral models.

3 Minkowski Sum Computation

In this section, we present some background on Minkowski sum computation and give a brief overview of our approximate approach.

3.1 Notation

We use lower case bold letters such as $\boldsymbol{p}, \boldsymbol{q}$ to refer to points in $\mathcal{R}^3$. We denote the complement of a set S as $\overline{S}$. All our primitives are closed solids. We use upper case letters such as P, Q, P_1, P_2, to refer to them. ∂P denotes the boundary of a primitive P. The letter C denotes a grid cell used for sampling. The exact Minkowski sum and our approximation are denoted as $\mathcal{M}$ and $\mathcal{A}$ respectively.

3.2 Overview

The Minkowski sum, $P \oplus Q$, is defined as a set of pairwise sums of points from P and Q. In other words, $P \oplus Q = \{\boldsymbol{p} + \boldsymbol{q} |\ \boldsymbol{p} \in P, \boldsymbol{q} \in Q\}$.

It is relatively easier to compute Minkowski sums of convex polytopes as compared to general polyhedral models. Minkowski sum of two convex polyhedra can have $O(n^2)$ complexity. However, for non-convex polyhedra in 3D, the Minkowski sum can have $O(n^6)$ worst-case complexity [7].

One common approach for computing Minkowski sum of general polyhedra is based on *convex decomposition* [20]. It uses the following property of Minkowski sum. If $P = P_1 \cup P_2$, then $P \oplus Q = (P_1 \oplus Q) \ \cup \ (P_2 \oplus Q)$. The resulting algorithm combines this property with convex decomposition for general polyhedral models:

1. Compute a convex decomposition for each polyhedron
2. Compute the pairwise convex Minkowski sums between all possible pairs of convex pieces in each polyhedron.
3. Compute the union of pairwise Minkowski sums.

After the second step, there can be $O(n^2)$ pairwise Minkowski sums. The pairwise convex Minkowski sums are convex. Their union can have $O(n^6)$ complexity [2].

Our algorithm for Minkowski sum computation is based on the above framework. We now discuss each of the above steps in detail.

3.3 Convex Decomposition

The problem of computing an optimal convex decomposition of a non-convex polyhedron is known to be NP-hard. Chazelle proposed one of the earliest convex decomposition algorithms [5], which can generate $O(r^2)$ convex parts and uses $O(nr^3)$ time where n and r the number of polygons and notches in the original polyhedron. However, no robust implementation of this algorithm is known. Most practical algorithms for convex decomposition perform surface decomposition or tetrahedral volumetric decomposition [6, 8]. Typically, these methods can generate $O(n)$ convex parts and each of them has a few faces.

We used a modification of the convex decomposition scheme available in a public collision detection library, SWIFT++ [8]. This method is an implementation of the algorithm presented in [6]. It performs surface decomposition and generates a set of convex patches c_i's of the object boundary ∂P. Furthermore, we compute a convex hull of each surface patch, c_i, and denote the resulting polytope by C_i. The C_i's constitute a convex decomposition of object P. C_i's consists of two types of faces: *real faces* that belong to the original polyhedron and *virtual faces* that are artifacts of the convex hull computation. In general, the union of C_i's need not cover the entire volume of P. C_i's may create some undesirable voids in the interior of P that are bounded by the virtual faces. We disregard these voids by explicitly checking for virtual faces in our distance and inside/outside queries.

Given two polyhedra P and Q each with n triangles, the convex decomposition method typically divides each polyhedron into $O(n)$ convex parts. In practice, each convex part usually has very few polygons ($4 - 8$ on an average). Computing pairwise Minkowski sums between all pairs of convex pieces results in $O(n^2)$ pairwise Minkowski sums. Although this quadratic complexity may seem high, it should be viewed in context of the high complexity of Minkowski sum ($O(n^6)$). Even though we may need to compute the union of a large number of primitives (pairwise Minkowski sums), the primitives themselves are relatively simple and typically have low combinatorial complexity. Our approximate algorithm is well suited to this problem.

3.4 Pairwise Minkowski Sum Computation

We compute the pairwise Minkowski sums between all possible pairs of convex pieces, C_i^P and C_j^Q, belonging to P and Q, respectively. Let us denote the resulting Minkowski sum as M_{ij}. We use a *convex hull algorithm* to compute M_{ij}. Its complexity is $O(n^2)$ where n is the number of polygons in C_i^P and C_j^Q. While more efficient (in terms of time complexity) algorithms are known, e.g. [12], the constant factors in the time complexity can be high and it is non-trivial to implement them robustly. Moreover, C_i^P, C_j^Q and M_{ij} usually have a constant combinatorial complexity. Hence we use the simpler convex hull algorithm described

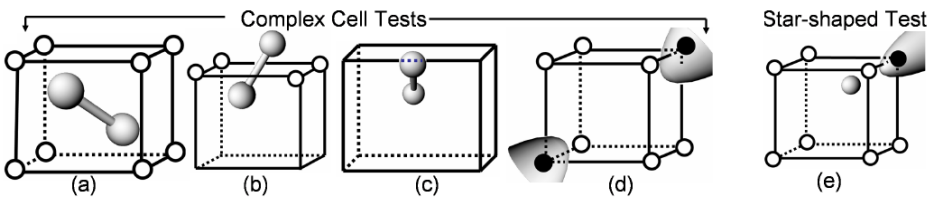


Figure 2. *This figure shows the different cases corresponding to the complex cell and star-shaped test. Figs (a), (b), (c) and (d) show cases of complex voxel, complex face, complex edge, and topological ambiguity. The white and black circles denote positive and negative grid points respectively. Fig. (e) shows the case where the surface is not star-shaped w.r.t a cell.*

below.

Convex Hull Approach: It is based on the following property:

$$P \oplus Q = \mathbf{CH}(\{\mathbf{v}_i + \mathbf{v}_j | \mathbf{v}_i \in V_P, \mathbf{v}_j \in V_Q\}) \quad (1)$$

Here, **CH** denotes the convex hull operator, and V_P, V_Q represent the sets of vertices, respectively in polyhedra P and Q. Based on this fact, we compute the Minkowski sum as follows:

1. Compute the vector sum between all possible pairs of vertices from each polytope.
2. Compute their convex hull.

3.5 Union Computation

Our goal is to compute $\mathcal{M}$, the boundary of the final solid that corresponds to the result of the Minkowski sum. $\mathcal{M}$ is given by the union of the pairwise Minkowski sums: $\mathcal{M} = \cup_{i,j} M_{ij}$. However, computing an exact union of the pairwise Minkowski sums is not practical. This is due to the large number of pairwise Minkowski sums. In our benchmarks, $\mathcal{M}$ is defined by union of tens of thousands of primitives (pairwise Minkowski sums). Exact boundary evaluation of this size is slow and prone to robustness problems. Abrams and Allen [1] discuss these problems in context of computing union of polyhedral models for swept volume computation. The commercial CAD systems have not been designed to perform Boolean operations on a high number of primitives (e.g. thousands of polyhedra). As a result, they are either not robust or too slow in terms of computing the union of a high number of polyhedral models.

Instead of computing $\mathcal{M}$ exactly, we approximate it using distance field-based techniques. For each closed primitive P, we compute a signed distance field D. A signed distance field $D(\boldsymbol{p})$ is a continuous function that at a point $\boldsymbol{p}$ measures the distance between $\boldsymbol{p}$ and the surface of P. This value is positive or negative depending on whether the point lies outside or inside the closed primitive. Distance fields are attractive because they map geometric operations such as union and intersection into min/max operations on the distance fields of the primitives. Our overall approach proceeds in the following steps:

1. **Sampling**: Generate an adaptive voxel grid and compute the signed distance field at its grid points.
2. **Operation**: For each geometric operation (e.g., union), perform a min/max operation on the signed distance field of the primitives.
3. **Reconstruction**: Use some variant of Marching Cubes algorithm [19, 14] to perform isosurface extraction from the distance field. The extracted isosurface is our approximation to the boundary of the Minkowski sum.

4 Minkowski Sum Approximation

In this section, we present our approximation algorithm and show its application to Minkowski sum computation.

4.1 Approximate Algorithm

Varadhan et al. [26] have presented an algorithm for computing topology preserving isosurfaces and have used it for performing Boolean operations. We apply their algorithm to the problem of Minkowski sum computation. We provide a brief description of their algorithm. It is based on the sampling and reconstruction approach presented in Section 3.5. Given a Boolean expression defined over a set of primitives, it generates an adaptive volumetric grid. Let $\mathcal{E}$ denote the boundary of the final solid defined by the Boolean expression. The algorithm starts with a single grid cell that encloses $\mathcal{E}$. It performs two tests, *complex cell test* and *star-shaped test*, to decide whether to subdivide a grid cell.

Complex Cell Test: A cell is *complex* if it has a *complex voxel*, *face*, *edge*, or an *ambiguous sign configuration*. We define a voxel (face) of a grid cell to be *complex* if it intersects $\mathcal{E}$ and the grid vertices belonging to the voxel (face) do not exhibit a sign change (see Figs. 2(a) & 2(b)). The sign of a vertex is positive if it lies within $\mathcal{E}$, negative otherwise. An edge of the grid cell is said to be *complex* if $\mathcal{E}$ intersects the edge more than once. It is well known that Marching Cubes produces topologically ambiguous output for certain sign configurations (see Fig. 2(d)). We classify grid cells with such sign configurations as complex.

Intuitively, the complex cell criterion ensures that the surface intersects the grid cell in a simple manner in most cases. If a grid cell is complex, it is subdivided and the algorithm is recursively applied to each of its children.

Star-shaped Test This test ensures that the surface $\mathcal{E}$ restricted to a cell is *star-shaped* within that cell. Let S be a nonempty subset of $\mathcal{R}^n$. The set Kernel(S) consists of all $\boldsymbol{s} \in S$ such that for any $\boldsymbol{x} \in S$, we have $\boldsymbol{s} + \lambda(\boldsymbol{x} - \boldsymbol{s}) \in$

$S, \forall \lambda \in [0,1]$. S is *star-shaped* if Kernel$(S) \neq \emptyset$. Intuitively, a star-shaped primitive has a representative point (called the origin) such that all the points in the primitive are visible from the origin. If $\mathcal{E}$ is not star-shaped w.r.t the cell (see 2(e)), the cell is subdivided and the algorithm is recursively applied to the children cells.

In this manner, by applying the above two tests, the algorithm generates a volumetric grid. It uses Marching Cubes to perform isosurface extraction on the resulting grid. The extracted surface is an approximation to $\mathcal{E}$.

Varadhan et. al. [26] use max-norm distance computation and linear programming to perform the complex cell and star-shape tests respectively. Max-norm distance computation is used to determine whether $\mathcal{E}$ intersects a voxel/face/edge of the cell or not. Linear programming is used to check whether $\mathcal{E}$ restricted to a cell is star-shaped or not. Performing these tests does not require an explicit representation of $\mathcal{E}$. They can be performed even when $\mathcal{E}$ is defined as a Boolean combination of a number of primitives. We refer the reader to [26] for a detailed explanation.

4.2 Application to Minkowski Sum Computation

We use the approximate algorithm described above to the problem of Minkowski sum computation. In particular, we approximate the union of the pairwise Minkowski sums M_{ij}'s. A naive application of the approximate algorithm can result in poor performance. This is because in the context of Minkowski sum computation, we are dealing with a very large number of primitives (M_{ij}'s). We present a number of culling techniques to improve the efficiency of the algorithm. Together, they improve the overall performance significantly.

4.2.1 Cell Culling

During Boolean operations, only a subset of the boundaries of the primitives contribute to the *final surface*, the boundary of the solid defined by the Boolean operation. Let $C_{M_{ij}}$ be the set of cells intersecting the boundary of a primitive M_{ij}. Therefore $\cup C_{M_{ij}}$ is the set of cells intersecting the boundary of some primitive. Only a subset of cells $C_{\mathcal{M}} \subset \cup C_{M_{ij}}$ contain the final surface. Typically $|C_{\mathcal{M}}| << |\cup C_{M_{ij}}|$. We use max-norm distance to determine whether a cell intersects the final surface or not. Our algorithm disregards those cells that lie inside a primitive and are guaranteed not to intersect the final surface. This process is cell culling. It considerably improves the performance of our algorithm.

We use a *voxel intersection* test to determine whether the final surface intersects a cell (cube-shaped voxel) or not. Our test is based on the following fact: the surface intersects a voxel if and only if the unsigned max-norm distance between the center of the voxel and the surface is less than half the voxel length. This test can also be generalized to axis-aligned cells by suitably defining a weighted max-norm.

Suppose a cell C lies within a primitive M_{ij}. Let d_{ij} denote the signed max-norm distance to M_{ij} at the center of C. Let l denote the length of cell C. We have $d_{ij} < -l/2 < 0$. This is because C is contained inside M_{ij}. We obtain a lower bound δ on the distance to $\mathcal{M}$ by computing $\min_{ij} d_{ij}$. We have $\delta < d_{ij} < -l/2$ or $|\delta| > l/2$. As a result, our voxel intersection test guarantees that the final surface does not intersect C. In this manner, our adaptive subdivision algorithm disregards cell C and does not consider it for further subdivision.

4.2.2 Primitive Culling

To apply the complex cell and star-shaped tests, our algorithm needs to perform two types of queries. These include distance and sign (inside/outside status) computation. These queries are performed several times for each grid cell and therefore impact the overall performance of the algorithm. These queries are *global* in scope in that the answer to the query depends on all the primitives (M_{ij}'s). For example, to check whether a point lies outside the union, we need to check if it lies outside every primitive. Given the large number of primitives, this can slow down the overall algorithm considerably.

Our objective is to perform *local* queries such that the answer to the query depends only on a small subset of primitives. In particular, when performing a query within a cell, we would like to inspect only those primitives that intersect the cell. Of course, we have to do this in a manner that preserves the correctness of the query.

Suppose we want to perform the inside/outside query to determine whether a point $\boldsymbol{p}$ lies inside the Minkowski sum, i.e., if $\boldsymbol{p} \in \cup M_{ij}$. We take advantage of the fact that this query does not have to be performed within cells that are eliminated due to cell culling. Suppose $\boldsymbol{p}$ is contained within a cell C. If $C \subset M_{ij}$, then C would have been eliminated due to cell culling (see Sec. 4.2.1). Therefore it suffices to consider the case when $C \not\subset M_{ij}$ for all primitives M_{ij}. For such a cell, it is sufficient to consider the set M_C consisting of primitives M_{ij} that intersect C. The following theorem guarantees the correctness of the query. A similar result holds for the distance query.

THEOREM 1

Given a point $\boldsymbol{p}$ contained within a grid cell C such that $C \not\subset M_{ij}$ for all primitives M_{ij}, we have

$$\boldsymbol{p} \in \cup\{M_{ij}\} \Longleftrightarrow \boldsymbol{p} \in \cup\{M_{ij} \in M_C\}$$

Proof: Consider any primitive $M_{kl} \notin M_C$. In other words, the boundary of M_{kl} does not intersect cell C. Two cases arise: either M_{kl} lies completely outside C or it encloses C. In the first case, $\boldsymbol{p} \notin M_{kl}$ and so we have

$$\boldsymbol{p} \in \cup M_{ij} \Longleftrightarrow \boldsymbol{p} \in \cup\{M_{ij} \mid i \neq k \text{ or } j \neq l\}$$

i.e., M_{kl} does not make a difference to the answer. In the second case, we have $C \subset M_{kl}$ which is a contradiction. This concludes the proof.

The above result is important for Minkowski sum computation because we are dealing with a very large number of primitives. For example, the Minkowski sum benchmarks

shown in Fig. 5 consist of tens of thousands of primitives. Using the above theorem, each query only considers a small subset — on an average around 100 to 200 — of primitives. This drastically improves the overall performance of the algorithm.

We can extend the above result to perform additional culling. Given a primitive P intersecting cell C, we can decompose P into two primitives $P_1 = P \cap C$ and $P_2 = P \cap \overline{C}$ where $\overline{C}$ denotes the complement of C. Given a point $\boldsymbol{p}$ contained within C, we have $\boldsymbol{p} \in P \iff \boldsymbol{p} \in P_1$. In other words the inside/outside status of $\boldsymbol{p}$ depends only on the subset of the primitive that is contained within the cell.

We use this property to achieve *triangle culling*. We are dealing with triangulated primitives. An inside/outside query for a triangulated primitive takes time proportional to the number of triangles in the primitive. Let T_{ij} be the set of triangles in M_{ij} and $T_{ij}^C \subset T_{ij}$ be the subset of triangles that lie within C. $T^C = \cup_{ij} T_{ij}^C$ is the set of all the triangles contained within C. To determine the inside/outside status of $\boldsymbol{p}$, we only need to consider T^C. In this manner, our algorithm disregards triangles outside C.

4.3 Geometric and Topological Guarantees

The following guarantees follow from the results in [26].

THEOREM 2

If all the cells in the volumetric grid satisfy the complex cell and star-shape tests, then

1. **Geometric Guarantee:** *Given any $\epsilon > 0$, our algorithm outputs a Minkowski sum approximation $\mathcal{A}$ such that Two-sided Hausdorff Distance$(\mathcal{A}, \mathcal{M}) < \epsilon$.*
2. **Topological Guarantee:** *Our Minkowski sum approximation $\mathcal{A}$ has the same topology as the exact surface $\mathcal{M}$.*

Together, the geometric and topological guarantees ensure a good quality of the Minkowski sum approximation.

5 Applications

We describe three applications of our approximate algorithm. These are motion planning, morphological operations, and penetration depth computation.

5.1 Motion Planning

Motion planning is an important problem in algorithmic robotics. The basic problem is to find a collision-free path for a robot among rigid objects. We consider the case of a 3D polyhedral robot undergoing translation motion among 3D polyhedral obstacles. This problem is often formulated using a configuration space approach. The free configuration space is the set of all possible positions in which the robot avoids contact with the obstacles. It can be expressed as the complement of the Minkowski sum of the robot and the obstacles [20].

For the purpose of path planning, it suffices to have a representation that captures the connectivity of the free configuration space. We have used our Minkowski sum approximation algorithm to compute such a representation and used it for designing an exact path planner. Our planner is guaranteed to find a path, if one exists, even through narrow passages. Details can be found in [25]. Fig. 3 shows application of our algorithm to assembly planning. It consists of two parts each with pegs and holes. The goal is to assemble the two parts so that the pegs of one part fit into the holes of the other. This problem can be reduced to a motion planning problem by treating one of the parts as a robot and the other as the obstacle.

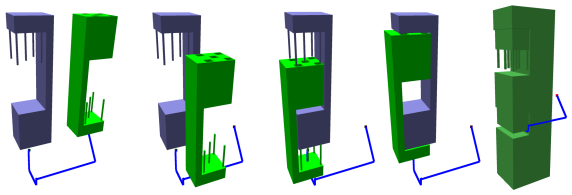


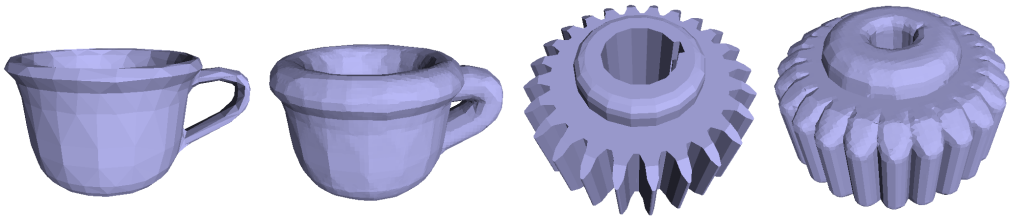
Figure 3. Assembly Planning: *This benchmark shows application of our algorithm to assembly planning. The four images on the left shows a path that the robot can take so that the two parts could be assembled. The rightmost image shows the Minkowski sum and the path of the robot in configuration space. This is a challenging example because the goal configuration is lodged within a narrow passage in the configuration space. Our algorithm took 12 secs to find a valid path (shown in blue).*

5.2 Offsets and Mathematical Morphological Operations

We apply our approximation algorithm to perform mathematical morphological operations. Mathematical morphology has been used in image analysis for a long time. A systematic treatment is given in [23]. The primary morphological operations, from which many others are constructed, are dilation and erosion. Dilation of an object P by an object Q is same as the Minkowski sum $P \oplus Q$. Q is usually referred to as the structuring element. Erosion of an object P by structuring element Q selects the locus of points swept by the origin of Q where P entirely contains the translated Q. Erosion can be expressed in terms of the Minkowski sum operation as: $\overline{\overline{P} \oplus Q'}$ where Q' denotes a copy of Q reflected about the origin. Our Minkowski sum approximation algorithm can be used to perform morphological operations on polyhedral models.

An interesting case of morphological operations is where the structuring element Q is a sphere. In this special case, dilation reduces to the *offset* operation [21]. The offset of a solid is obtained by adding to the solid all the points that lie within a distance r. Mathematically it is defined as Offset$(P) = \{\boldsymbol{p} \mid \exists\, \boldsymbol{q} \in P,\ ||\boldsymbol{p} - \boldsymbol{q}|| \leq r\}$. Offset is a special case of Minkowski sum – it can be expressed as the Minkowski sum with a sphere. The exact computation of the offset is difficult because it requires union computation of a large number of higher order surfaces.

The offset of a triangulated object P consists of three types of regions:



Cup $(1,000$ tris) Cup $\oplus$ Sphere Gear $(2,382$ tris) Gear $\oplus$ Sphere

Figure 4. *Offsets: The figures show two models, Cup and Gear, with* $1,000$ *and* $2,382$ *triangles respectively. Our approximation algorithm computed their offsets by computing their Minkowski sum with a sphere. It took* 33 *and* 84 *secs to compute the offsets for the two models. The approximate boundary consisted of* $14,895$ *and* $22,742$ *triangles.*

- A spherical region around a vertex v_i of P. This region is part of a sphere S_i of radius r centered at v_i.
- A cylindrical region around an edge e_j of P. This region is part of a cylinder C_j of radius r and whose axis is same as e_j.
- A planar region due to a triangle t_k of P obtained by displacing t_k along its outward normal by a distance r. This results in a triangular prism P_k.

Let $O = \cup_i S_i \bigcup \cup_j C_j \bigcup \cup_k P_k$. The dilation $P \oplus S$ is same as offset of P and is given by $P \bigcup O$. The erosion $P \ominus S$ is given by $P \setminus O$. In case of erosion, the triangular prisms P_k are obtained by displacing the triangle along the inward normal.

In this manner, the problem of performing mathematical morphological operations (with sphere as the structuring element) reduces to performing Boolean operations on polyhedra, triangular prisms, spheres, and cylinders. We use our approximation algorithm described in Sec. 4 to perform these Boolean operations. Note that this approach does not require convex decomposition of the object. For an object with n triangles, we perform a Boolean on $3 * n + 1$ primitives. Figure 1 shows the offset of the *Bunny* model. Fig. 4 shows offsets of two models.

5.3 Penetration Depth Computation

We use our Minkowski sum approximation algorithm to estimate the penetration depth between two polyhedral models. We guarantee that our estimate of penetration depth is arbitrarily close to the actual value. The penetration depth of two intersecting polyhedra P and Q, $PD(P, Q)$, is the minimum translational distance that one of the polyhedra must undergo to render them disjoint. It is well known that one can reduce the problem of computing the PD between P and Q to a minimum distance query on the surface of their Minkowski sum , $P \oplus -Q$.

Based on this approach, [16] presented an approximate algorithm to estimate the PD using graphics hardware. This approach is very efficient and can compute penetration depth of complex models quickly. One limitation of this approach is that due to the limited precision of the rasterization hardware, the estimated PD can be very different from the actual PD and there are no tight error bounds on the estimate.

We can use our Minkowski sum approximation algorithm to obtain a penetration depth estimate that is arbitrarily close to the actual value. Given any $\epsilon > 0$, we compute an approximation $\mathcal{A}_\epsilon$, such that Two-sided Hausdorff Distance$(\mathcal{A}_\epsilon, \mathcal{M}) < \epsilon$ (Theorem 2). Our penetration depth estimate δ is given by $\delta = D(O_{Q-P}, \mathcal{A}_\epsilon)$. It is easy to prove that our estimate δ is close to the actual PD. In particular, we can show that $\delta - \epsilon < PD(P, Q) < \delta + \epsilon$. Thus $\alpha = \delta - \epsilon$ and $\beta = \delta + \epsilon$ provide bounds on the PD. By decreasing ϵ, we can obtain arbitrarily tight bounds on the actual PD.

In addition to the penetration depth estimate, the above bounds α and β can be used to obtain a potential set of penetrating features. Due to space limitations, we skip the details. The potential set of penetrating features is obtained by considering a set of features belonging to the pairwise Minkowski sums that lie within an annulus of radii α and β centered at the origin.

6 Implementation and Performance

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

6.1 Implementation

We implemented our algorithms on a 2 GHz Pentium IV PC with 1 GB main memory. We used the Extended Marching Cubes (EMC) algorithm [17] to perform the isosurface extraction. It requires computing directed distance at the grid points. Our algorithm is simple to implement. It only requires performing distance and inside/outside queries. Directed distance [17] and max-norm distances [24] to convex primitives can be computed efficiently.

6.2 Performance

We tested our algorithm on a number of complex models. The model complexity (Table 1) varied from several hundred to few thousand triangles. Figures 1 and 4 show the offset of three models: *Bunny*, *Cup* and *Gear*. Figure 1 shows the Minkowski sum of *Brake Hub* and *Rod* models. The final Minkowski sum has a number of narrow holes that

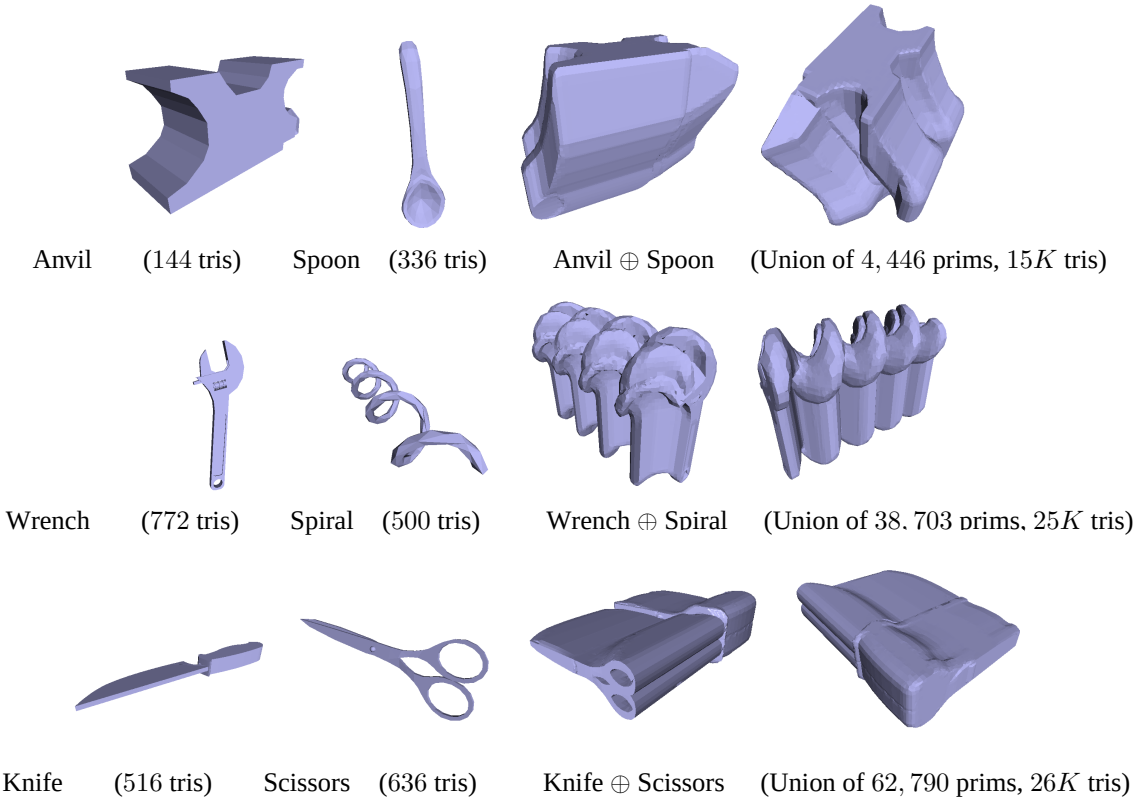


Figure 5. *Benchmarks: This figure shows three different benchmarks. The left two columns show the two primitives whose Minkowski sum is being computed. The triangle counts for the two primitives are shown in brackets. Two views of the approximation computed by our algorithm are shown in the right. For the three models, the Minkowski sum reduced to computing the union of* $4,446$, $38,703$ *and* $62,790$ *primitives respectively. Our algorithm took* 63, 316 *and* 778 *secs respectively to generate an approximation. The approximate boundary consists of* $15K$, $25K$ *and* $26K$ *triangles respectively (see Table 1).*

contribute to a high genus. Our algorithm produces an approximation with the correct topology. Fig. 5 shows the Minkowski sum of a number of CAD models. Fig. 6 shows a complex benchmark consisting of two *Grates*. This is a very challenging scenario as the resulting Minkowski sum has very high complexity. It has numerous thin and needle-like features. Our algorithm was able to reconstruct all the complex features. Fig. 3 shows an application to motion planning. Table 1 shows the model complexity and performance of our algorithm on these benchmarks. Sampling is the most time consuming step in the algorithm. Fig. 7 highlights the performance of our algorithm on different benchmarks, showing the level of subdivision.

The culling techniques improve the performance significantly. We applied our algorithm without any culling techniques to the Anvil and Spoon benchmark (Figure 1). It took more than 7 hours to generate an approximation, as compared to 63 secs using culling techniques.

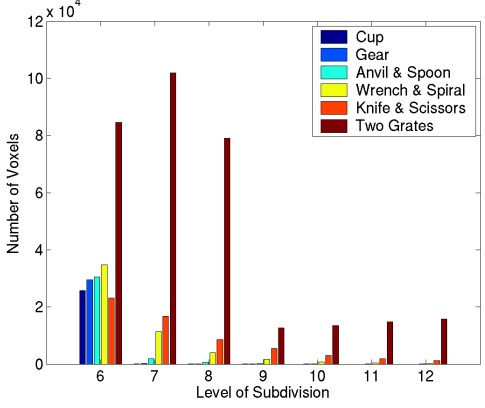


Figure 7. *The histogram shows the number of voxels in our adaptive voxel grid for different benchmarks. It highlights the number of voxels at each level of subdivision.*

7 Limitations

The complex cell and star-shaped criteria are conservative. As a result, the sampling algorithm may result in conservative subdivision. Our algorithm may not be able to handle all degenerate configurations in the input model. These include cases when the model has artifacts such as self intersections. Our algorithm can only generate manifold boundaries and is not applicable to the cases where the exact boundary is non-manifold. Our sampling algorithm cannot

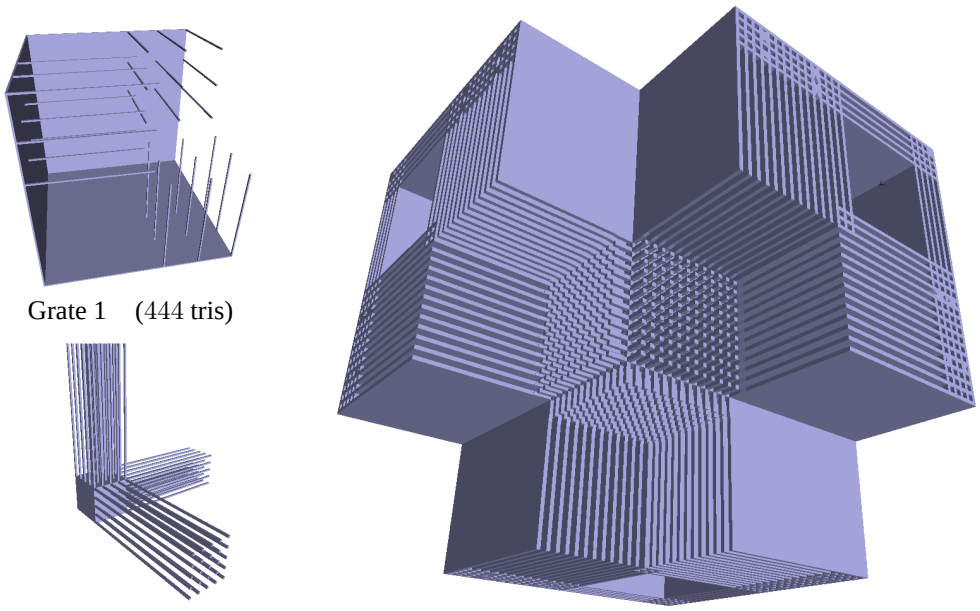


Figure 6. *The left figure show two grates with* 444 *and* $1,134$ *triangles respectively. We decomposed them into* 163 *and* 409 *convex pieces respectively and computed the pairwise Minkowski sums between the convex pieces. The final Minkowski sum is given by the union of* $66,667$ *pairwise Minkowski sums. Our approximation algorithm computed an approximation (shown in the right) in* $3,162$ *secs (*52 *minutes). It was able to reconstruct the complex features present on the boundary.*

handle cases where two primitives (the pairwise Minkowski sums) are touching tangentially. One way of resolving this problem is by choosing an alternative way of subdividing the grid cells (instead of octree subdivision). We are exploring this alternative in our ongoing work [25].

The main bottleneck in our approach is the convex decomposition method. Typically, it produces $O(n)$ convex pieces. Given two polyhedra each with n triangles, we usually obtain $O(n^2)$ pairwise convex Minkowski sums whose union needs to be computed. Since this set of pairwise convex Minkowski sums is an input to our approximation algorithm, its large size impacts the performance of the overall algorithm. Although our algorithm is able to approximate their union much faster and robustly compared to exact union algorithms, it still needs to pay the penalty for the large input size. It takes few minutes to compute Minkowski sums of models composed of hundreds of triangles. Using a better convex decomposition method can alleviate this problem.

8 Conclusion and Future Work

We have presented an algorithm to approximate the 3D Minkowski sum of polyhedral objects. Our algorithm guarantees that the approximation has the correct topology and provides two-sided Hausdorff distance bounds on the approximation. We employ cell and primitive culling techniques to improve the performance of our algorithm. We have applied our algorithm to offset computation, morphological operations, and penetration depth computation of complex polyhedral models. We have also used it for exact motion planning with translational degrees of freedom.

As part of future work, we would like to improve our sampling algorithm to make it less conservative and thereby improve its performance. We would like to use better convex decomposition algorithms. It is well-known that the Minkowski sum of two star-shaped polyhedra is a star-shaped polyhedra. We could exploit this property and design our overall approach based on star-shaped decomposition instead of convex decomposition. The main advantage of this approach is that the star-shaped decomposition of a polyhedron would typically result in fewer primitives. Furthermore, we would like to develop similar algorithms for arrangement and envelope computation.

9 Acknowledgments

This research is supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards ACI 9876914 and ACR-0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, DARPA Contract N61339-04-C-0043 and Intel. We thank Danny Halperin for many fruitful discussions on the complexity and practical implementations of Minkowski sum.

References

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

	Primitive 1			Primitive 2		Num Convex	Performance (sec)			Output
	Num Tris	Num Pieces		Num Tris	Num Pieces	Prims	Convex Mink	Sampling	Recons	Num Tris
Cup	1000	338	Sphere	-	1	338	1.2	32	0.08	14,895
Gear	2,382	744	Sphere	-	1	744	3.6	81	0.09	22,742
Brake Hub	4,736	1777	Rod	24	1	1,777	4.3	135	0.04	45,753
Anvil	144	57	Spoon	336	78	4,446	3.9	59	0.02	15,638
Wrench	772	291	Spiral	500	133	38,703	27	289	0.06	25,280
Knife	516	273	Scissors	636	230	62,790	36	742	0.06	26,038
Grates 1	444	163	Grates 2	1134	409	66,667	40	3120	1.5	358,030

Table 1. *Benchmarks: This table shows the performance of our algorithm on different models. The columns on the left show the statistics of the two primitives whose Minkowski sum is computed. They show the number of triangles in each primitive and the number of convex pieces generated by convex decomposition. The column, Num Convex Prims, shows the number of convex Minkowski sums generated. The right three columns show the time taken to generate the convex Minkowski sums, sampling (grid generation) and isosurface reconstruction.*

2000.

[2] B. Aronov, M. Sharir, and B. Tagansky. The union of convex polyhedra in three dimensions. *SIAM J. Comput.*, 26:1670–1688, 1997.

[3] J. Basch, L. Guibas, G. Ramkumar, and L. Ramshaw. Polyhedral tracings and their convolutions. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 1996.

[4] J.-D. Boissonnat, E. de Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Symposium on Computational Geometry*, pages 67–76, 1997.

[5] B. Chazelle. Convex decompositions of polyhedra. In *ACM Symposium on Theory of Computing*, pages 70–79, 1981.

[6] B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications*, 7:327–342, 1997.

[7] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[8] S. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3):500–510, 2001.

[9] R. C. Evans, M. A. O'Connor, and J. R. Rossignac. Construction of minkowski sums and derivatives morphological combinations of arbitrary polyhedra in cad/cam systems. *US Patent 5159512*, 1992.

[10] E. Flato and D. Halperin. Robust and efficient construction of planar minkowski sums. In *Abstracts 16th European Workshop Comput. Geom.*, pages 85–88, 2000. Eilat.

[11] P. Ghosh. A unified computational framework for minkowski operations. In *Computers and Graphics, 17(4), pp.357-378*, 1993.

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

[13] D. Halperin. Robust geometric computing in motion. *International Journal of Robotics Research, 21(3)*, 2002.

[14] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 21(3), 2002.

[15] A. Kaul and J. Rossignac. Solid-interpolating deformations: Construction and animation of PIPs. In W. Purgathofer, editor, *Eurographics '91*, pages 493–505. North-Holland, Sept. 1991.

[16] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics*, 2002.

[17] L. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature-sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH*, pages 57–66, 2001.

[18] I.-K. Lee, M.-S. Kim, and G. Elber. Polynomial/rational approximation of Minkowski sum boundary curves. *Graphical Models and Image Processing*, 60(2):136–165, 1998.

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

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

[21] J. Rossignac and A. Requicha. Offsetting operations in solid modeling. *Comput. Aided Geom. Design*, 3:129–148, 1986.

[22] J.-K. Seong, M.-S. Kim, and K. Sugihara. The minkowski sum of two simple surfaces generated by slope-monotone closed curves. *Geometric Modeling and Processing: Theory and Applications*, 2002.

[23] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, UK, 1982.

[24] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, and D. Manocha. Efficient max-norm distance computation and reliable voxelization. *Eurographics Symposium on Geometry Processing*, 2003.

[25] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. A simple algorithm for complete motion planning of translating polyhedral robots. Technical report, Workshop on Algorithmic Foundations of Robotics, 2004.

[26] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. Topology preserving isosurface extraction using adaptive subdivision. Technical report, Eurographics Symposium on Geometry Processing, 2004.