A Simple Algorithm for Complete Motion Planning of Translating Polyhedral Robots

Gokul Varadhan[1], Shankar Krishnan[2], T.V.N. Sriram[3], and Dinesh Manocha[4]

[1] University of North Carolina at Chapel Hill varadhan@cs.unc.edu
[2] AT&T Labs - Research krishnas@research.att.com
[3] University of North Carolina at Chapel Hill tvn@cs.unc.edu
[4] University of North Carolina at Chapel Hill dm@cs.unc.edu
http://gamma.cs.unc.edu/motion

Summary. We present an algorithm for complete path planning for translating polyhedral robots in 3D. Instead of exactly computing an explicit representation of the free space, we compute a roadmap that captures its connectivity. This representation encodes the complete connectivity of free space and allows us to perform exact path planning. We construct the roadmap by computing deterministic samples in free space that lie on an adaptive volumetric grid. Our algorithm is simple to implement and uses two tests: a *complex cell test* and a *star-shaped test*. These tests can be efficiently performed on polyhedral objects using max-norm distance computation and linear programming. The complexity of our algorithm varies as a function of the size of narrow passages in the configuration space. We demonstrate the performance of our algorithm on environments with very small narrow passages or no collision-free paths.

1 Introduction

Path planning is an important problem in algorithmic robotics. The basic problem is to find a collision-free path for a robot among rigid objects and it has been well-studied for over three decades. Some of the earlier interest was in developing algorithms for *complete* path planning. An algorithm is complete, if it is guaranteed to find a solution when one exists and to return failure otherwise. It is well known that any complete planner will run in exponential time in the number of degrees-of-freedom (dofs) of the robot [13]. Most practical algorithms for complete path planning are restricted to 2D polygonal objects or 3D convex polytopes or special objects e.g. ladders, discs or spheres.

Given the complexity of a complete path planner, most of the effort in the last two decades has been on development of approximate approaches including those based on cell decomposition and potential field [13]. These approaches can be *resolution complete* if the resolution parameters are selected properly, but not exact or complete. Other algorithms are based on probabilistic roadmaps [11], which have been successfully applied to many high-dof robots. However, they may not terminate in a deterministic manner when there is no collision-free path.

In this paper, we restrict ourselves to complete path planning for a 3D polyhedral robot undergoing translation motion among 3D polyhedral obstacles. The configuration space of the robot can be computed based on Minkowski sum of the robot and the obstacles. 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 $O(n^6)$. A 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 polyhedra, their union can have combinatorial complexity $O(m^3)$ and m can be high in the context of Minkowksi sum computation. Furthermore, robust computation of the boundary of the union and handling all degeneracies remains a major open issue. As a result, no good algorithms are known for robust computation of exact Minkowski sum of 3D polyhedral models and complete path planning.

Main Contributions: We present a novel algorithm for complete path planning for translating polyhedral robots in 3D. We perform convex decomposition and compute pairwise Minkowski sums of resulting convex polytopes. Instead of exactly computing the union of these polytopes, we compute a connectivity roadmap that captures the connectivity of the free space. We generate the connectivity roadmap by taking deterministic samples on an adaptive volumetric grid. We employ two main tests during sample generation: *complex cell test* and *star-shaped test*. These tests can be efficiently performed for polyhedral objects using max-norm computation and linear programming. The complexity of our algorithm varies as a function of the size of narrow passages in the configuration space.

Our algorithm is simple to implement in practice. We highlight its performance on two environments with few hundred polygons with either very small narrow passages or no collision-free paths. In these configurations, our algorithm takes a few seconds to either compute a collision-free path or guarantees that no path exists.

Organization: The rest of the paper is organized in the following manner. We give a background on motion planning in Section 2 and briefly survey related work. Section 3 gives an overview of our approach. We describe connectivity roadmaps in Section 4 and present an algorithm to compute the roadmap in Section 5. We highlights its performance on some environments in Section 6. In Section 7, we give an analysis of our approach and discuss few of its limitations. We conclude in Section 8.

2 Background and Prior Work

In this section, we define the general motion planning problem. Let $\mathcal{R}$ be a robot consisting of a collection of rigid subparts moving in a Euclidean space W, called *workspace*, represented as $\mathbb{R}^d$, with $d = 2$ or 3. Let $\mathcal{O}_1, \ldots, \mathcal{O}_q$

be fixed rigid objects, hereafter referred as *obstacles* embedded in W. Assume that the geometry of $\mathcal{R}, \mathcal{O}_1, \ldots, \mathcal{O}_q$ is accurately known, and that there are no kinematic constraints to limit the motion of $\mathcal{R}$. The position and orientation of the subparts define the *placement* of $\mathcal{R}$ (also referred to as a *configuration* of $\mathcal{R}$). The set of all placements of $\mathcal{R}$ defines a configuration space $\mathcal{C}$. The motion planning problem is defined as: given an initial and goal placement of $\mathcal{R}$, generate a *path* ρ specifying a continuous sequence of placements of $\mathcal{R}$ avoiding contact with the $\mathcal{O}_i$'s, starting at initial placement and terminating at the goal placement. Report failure if no such path exist. Every obstacle $\mathcal{O}_i, i = 1, \ldots, q$, in W maps to the region

$$\mathcal{C}O_i = \{\mathcal{Z} \in \mathcal{C} : \mathcal{R}(\mathcal{Z}) \cap \mathcal{O}_i \neq \phi\},$$

in $\mathcal{C}$, where $\mathcal{R}(\mathcal{Z})$ is the subset of W occupied by $\mathcal{R}$ at the placement $\mathcal{Z}$. The union of all $\mathcal{C}O_i$, $\bigcup_{i=1}^{q} \mathcal{C}O_i$ is called *C-obstacle region* or *forbidden region.* The *free configuration space* is defined as the set

$$\mathcal{F} = \mathcal{C} \setminus \bigcup_{i=1}^{q} \mathcal{C}O_i.$$

A *free path* between two free configurations $\mathcal{Z}_{init}$ and $\mathcal{Z}_{goal}$ is a continuous map $\rho : [0,1] \rightarrow \mathcal{F}$, $\rho(0) = \mathcal{Z}_{init}$ and $\rho(1) = \mathcal{Z}_{goal}$.

2.1 Previous Work

Motion planning has been extensively studied in the literature for more than three decades. In this section, we limit our discussion to algorithms for exact motion planning or to polyhedral objects undergoing translation motion in 3D. At a broad level they can be classified into the following techniques: *roadmaps*, *cell decomposition*, and specialized algorithms for 2D and 3D objects. A comprehensive survey of motion planning results is presented in [13].

Roadmaps

The idea underlying this approach is to convert the path planning problem in k-dimensional configuration space to path planning in network of 1-dimensional curves maintaining the connectivity in the robot's free space. The various types of roadmaps proposed to achieve this task are *visibility graph*, *Voronoi diagram or retraction approach*, and *silhouettes.*

Visibility graph method [13] reduces the problem of motion planning to a graph search. Using this approach, Lozano-Perez and Wesley [15] proposed an $O(n^3)$ algorithm which was improved to $O(n^2 \log n)$ by Lee [14] and to $O(n^2)$ by Guibas and Hershberger [9]. An output sensitive algorithm of $O(k+n \log n)$, where k is output size, was proposed by Ghosh and Mount [7]. In practice, this technique is mostly used for motion planning in two dimensional configuration spaces.

The retraction approach uses the concept of retraction in topology by defining a continuous mapping of the robot's free space $\mathcal{F}$ onto one-dimensional network of curves lying in $\mathcal{F}$. When $\mathcal{C} = R^2$ and the robot and obstacles are polygonal, the Voronoi diagram is a roadmap obtained by retraction [21]. This approach provides the additional property that the obtained paths maximize the clearance between the robot and the obstacles.

Silhouette method was proposed by Canny [4]. Unlike approaches surveyed earlier, this method does not make any assumptions about the configuration space and is a complete path planning algorithm that runs in single exponential time in the configuration space dimension.

Cell Decomposition

These methods are most extensively applied for motion planning problem [13]. The crux of this approach is to partition the robot's free space into a collection of non-overlapping cells and to construct a connectivity graph representing the cell adjacency. One of the first exact cell decomposition methods for solving the general motion planning problem was by Schwartz and Sharir [17].

Motion Planning Algorithms for 2D and 3D Robots

Different exact algorithms have been proposed for complete motion planning of 2D and 3D robots. Many of them are based on computing the Minkowski difference and the exact representation of C-obstacle region [15]. Kedem and Sharir [12] presented an efficient algorithm for a polygonal robot among polygonal obstacles with 3-dof configuration space. A similar algorithm for polygonal objects was also developed by Avnaim and Boissonant [3]. Sacks [16] presented a practical configuration space computation algorithm for pairs of curved planar parts. Halperin [10] presented efficient and robust algorithms to compute the Minkowski sum of 2D polygonal objects and used them for exact motion of planning of 2D objects undergoing translation motion. Aronov and Sharir [1] present a randomized algorithm to plan the motion of a convex polyhedron translating in 3-space amidst convex polyhedral obstacles. Vleugels and Overmars [21] presented a spatial subdivision algorithm to approximate the Voronoi diagram of an environment with convex primitives and used it for motion planning using retraction.

3 Overview

In this section, we formulate the problem of computing a collision free path for a polyhedral object undergoing translation motion in 3D using Minkowski sums, and provide an overview of our approach.

3.1 Motion Planning of Translating Robot

In this paper, our focus is on complete motion planning of translating polyhedral robots in the presence of polyhedral obstacles. It is well known that for a translating robot $\mathcal{R}$ and an obstacle $\mathcal{O}$, the C-obstacle $\mathcal{C}O = \mathcal{O} \oplus (-\mathcal{R})$, where $\oplus$ is the Minkowski sum and $-\mathcal{R}$ is $\mathcal{R}$ reflected about the origin. This formulation shrinks the robot to a point object, and the obstacles $\mathcal{O}_i$s are transformed to the respective Minkowski sums $\mathcal{C}O_i$.

3.2 Minkowski Sum

The Minkowski sum of two subsets P and Q of an affine space is defined as $P \oplus Q = \{\mathbf{x} | \ \mathbf{x} = \mathbf{p} + \mathbf{q}, \mathbf{p} \in P, \mathbf{q} \in Q\}$. It is relatively easier to compute Minkowski sums of convex polytopes as compared to general polyhedral models. For convex polytopes in 3D, the Minkowski sum can be computed in $O(n \log n + k)$ time, where k is the total number of features of the Minkowski sums [8]. In the worst case, $k = O(n^2)$. However, for non-convex polyhedra in 3D, the Minkowski sum can have $O(n^6)$ worst-case complexity [5].

One common approach for computing Minkowski sum of general polyhedra is based on *convex decomposition.* 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].

3.3 Our Approach

Our algorithm for computing the Minkowski sum is based on the decomposition property. We have a set of convex primitives consisting of the pairwise convex Minkowski sums whose union is the Minkowski sum. Although the above approach provides a simple algorithmic framework, the union computation is non-trivial. This is because the combinatorial complexity can be very large. Exact computation of the boundary of the union is prone to robustness problems and degeneracies. It is very difficult to compute the exact union in 3D. As a result, it is difficult to compute an exact representation for the free space $\mathcal{F}$.

Our goal is to obtain a representation for $\mathcal{F}$ that captures its connectivity and allows us to perform exact path planning. The *connectivity roadmap* is one such representation. The roadmap consists of a graph that encodes the complete connectivity of $\mathcal{F}$. We construct this graph by computing deterministic samples in $\mathcal{F}$ that lie on an adaptive volumetric grid. The ability to perform exact path planning using this approach relies critically on the sampling of the volumetric grid. We provide a simple sampling criterion for adaptive grid generation based on linear programming and max-norm distance computation. We construct a weighted graph using the vertices and edges of the grid that lie in $\mathcal{F}$. The weight of each edge is the Euclidean distance between the corresponding vertices. Then the path planning problem reduces to computing shortest paths in this graph.

4 Connectivity Roadmaps

In this section, we define a connectivity roadmap that captures the connectivity of the free space $\mathcal{F}$. We begin by introducing some notation.

Given a set S, two points $\mathbf{p}, \mathbf{q} \in S$ are connected if there exists a path between $\mathbf{p}$ and $\mathbf{q}$ that lies in S. We use the shorthand notation $\mathbf{p} \overset{S}{\longleftrightarrow} \mathbf{q}$ to mean that $\mathbf{p}$ and $\mathbf{q}$ are connected in S. The connectivity relation is symmetric. Given an undirected graph $G = (V, E)$ and two vertices $\mathbf{v}, \mathbf{w} \in V$, $\mathbf{v} \overset{G}{\longleftrightarrow} \mathbf{w}$ means that $\mathbf{v}$ and $\mathbf{w}$ are connected in G, i.e., there exists a path between $\mathbf{v}$

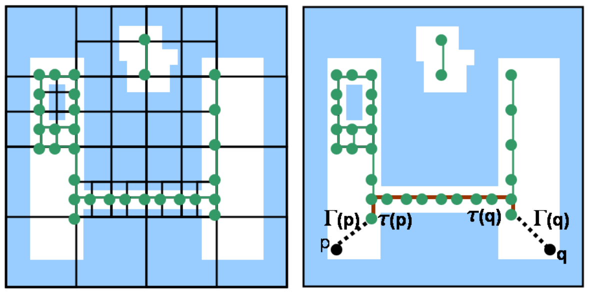


Fig. 1. Connectivity Roadmap: *We construct a connectivity roadmap by generating an adaptive voxel grid. The roadmap consists of a connectivity graph which is obtained by considering the set of grid points and grid edges that lie in free space (shown in green in the left figure). Connectivity roadmap also consists of a transfer function τ that maps points in free space to a vertex in the connectivity graph. As shown in the right figure, the source $\mathbf{p}$ and destination $\mathbf{q}$ get mapped to vertices $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ respectively. The problem of path planning between $\mathbf{p}$ and $\mathbf{q}$ reduces to simple graph search between $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ in the connectivity graph.*

and $\mathbf{w}$ consisting of a sequence of edges in E. As before, $\mathcal{F}$ refers to the free space. $\partial\mathcal{F}$ denotes the boundary of $\mathcal{F}$. The sign of a point is positive if it lies in $\mathcal{F}$, negative otherwise.

A connectivity roadmap consists of a connectivity graph G.

Definition 1. *The connectivity graph $G = (V, E, w)$ is a weighted undirected graph defined by a set V of points in free space, a set E of line segments in free space that connect pairs of points in V, and a weight function $w : E \rightarrow \mathbb{R}$. V and E correspond to the set of vertices and edges of G, respectively.*

Definition 2. *A transfer function $\tau : \mathcal{F} \rightarrow V$ is a mapping such that*

$$\forall \mathbf{p} \in \mathcal{F},\ \mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{p})$$

The connectivity graph satisfies the following two properties:

- **Property 1.** It encapsulates the connectivity of the free space. Every point $\mathbf{p} \in \mathcal{F}$ is connected to atleast one vertex in the connectivity graph. This implies that there exists a *transfer function* τ that maps a point in free space to a vertex in the connectivity graph. There also exists a *transfer path function* Γ such that $\Gamma(\mathbf{p})$ returns the path between $\mathbf{p}$ and $\tau(\mathbf{p})$ in $\mathcal{F}$.
- **Property 2.** It captures the connectivity of free space. Two points $\mathbf{p}, \mathbf{q} \in \mathcal{F}$ are connected in $\mathcal{F}$ if and only if the two vertices $\tau(\mathbf{p}), \tau(\mathbf{q}) \in V$ are connected in G, i.e.,

$$\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q} \iff \tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q})$$

Definition 3. *The connectivity roadmap is defined as the tuple (G, τ, Γ)*

Fig. 1 shows an example of a connectivity roadmap. By combining the above two properties, we see that the connectivity roadmap satisfies the following property:

$$\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q} \iff \mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{p}) \quad \tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q}) \quad \tau(\mathbf{q}) \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q}$$

This property enables us to perform complete path planning. As long as the source and destination are connected, we can find a path between them using the connectivity roadmap. Suppose we wish to find a path between two points $\mathbf{p}, \mathbf{q} \in \mathcal{F}$. Assume they are connected. We compute a mapping on the connectivity graph using the transfer function τ. The transfer path function Γ gives us the path between $\mathbf{p}$ and $\tau(\mathbf{p})$, and $\mathbf{q}$ and $\tau(\mathbf{q})$. Moreover, there exists a path between $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ in the connectivity graph. This path can be found easily by performing a simple graph search. Call this path γ. Thus we obtain the path $\Gamma(\mathbf{p}) :: \gamma :: \Gamma(\mathbf{q})$ between $\mathbf{p}$ and $\mathbf{q}$, where :: denotes a path concatenation. In this manner, the connectivity roadmap reduces the problem of path planning to computing a graph shortest path between $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$.

The above property also provides us with a test for non-existence of a path. If no path exists between $\mathbf{p}$ and $\mathbf{q}$ in $\mathcal{F}$, we can detect this by testing if $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ are disconnected in G (by Property 2).

5 Connectivity Roadmap Construction

In this section, we describe our algorithm to compute the connectivity roadmap.

5.1 Sampling

We construct a roadmap by performing a sampling of the free space. Unlike previous approaches such as probabilistic roadmaps (PRMs) that generate samples randomly [11], we construct a roadmap in a deterministic fashion. Our goal is to sample the free space sufficiently to capture its connectivity. If we do not sample the free space adequately, we may not detect valid paths that pass through the narrow passages in the configuration space.

In our prior work [20], we proposed a sampling algorithm to generate an octree grid for the purpose of topology preserving surface extraction. We use this sampling algorithm to capture the connectivity of free space. We provide a brief description of the octree generation algorithm. We refer the reader to [20] for a detailed description. The algorithm starts with a single grid cell that is large enough to capture relevant features of $\mathcal{F}$. 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 $\partial\mathcal{F}$ 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 $\mathcal{F}$, negative otherwise. An edge of the grid cell is said to be *complex* if $\partial\mathcal{F}$ intersects the edge more than once (see Fig. 2(c)).

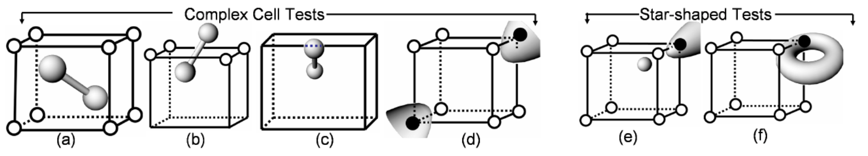


Fig. 2. *This figure shows the different cases corresponding to the complex cell and star-shaped test. Figs (a), (b), (c) and (d) show cases of complex voxel, complex face, complex edge, and ambiguous sign configuration. The white and black circles denote positive and negative grid points respectively.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

There are two types of sign ambiguities — *face ambiguity* and *voxel ambiguity* [22]. When the signs at the vertices of a single face alternate during counterclockwise (or clockwise) traversal, the resulting configuration is a face ambiguity. A voxel ambiguity results when any pair of diagonally opposite vertices have one sign while the other vertices have a different sign (see Fig. 2(d)). Either of these cases is defined as an ambiguous sign configuration. 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. We use max-norm distance computation to perform the complex cell test [20]. Max-norm distance is used to determine whether $\partial\mathcal{F}$ intersects a voxel/face/edge of the cell. It can be computed efficiently for polyhedral primitives. If a grid cell is complex, it is subdivided and the algorithm is recursively applied to each of its children.

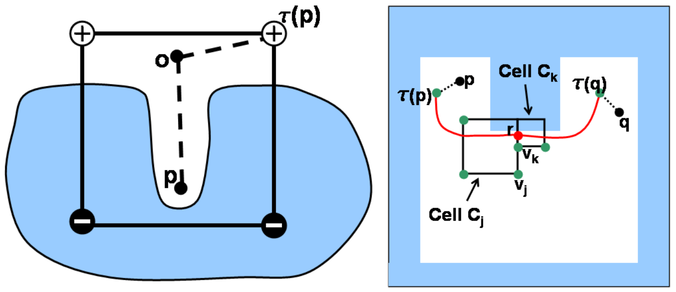
Star-shaped Test

Let S be a nonempty subset of $\mathbb{R}^n$. The set Kernel(S) consists of all $\mathbf{s} \in S$ such that for any $\mathbf{x} \in S$, we have $\mathbf{s} + \lambda(\mathbf{x} - \mathbf{s}) \in S, \forall\lambda \in [0, 1]$. S is *star-shaped* if Kernel(S) $\neq \emptyset$. We refer to a point belonging to Kernel(S) as an origin of S. A star-shaped primitive has at least one representative point (origin) such that all the points in the primitive are visible from it.

$\mathcal{F}$ is defined to be *star-shaped with respect to (w.r.t) a voxel* v if $\mathcal{F}_v = \mathcal{F} \cap v$ is star-shaped. Similarly, $\mathcal{F}$ is defined to be *star-shaped w.r.t a face* f if the two-dimensional set, $\mathcal{F}_f = \mathcal{F} \cap f$, is star-shaped. $\mathcal{F}$ is said to be *star-shaped w.r.t a cell* if it is star-shaped w.r.t the cell's voxel, and each of its six faces. If $\mathcal{F}$ is not star-shaped w.r.t the cell (see Figs. 2(e) & 2(f)), then the cell is subdivided and the algorithm is recursively applied to the children cells.

We use linear programming to perform the star-shaped test [20]. By definition, a polyhedron is star-shaped if it has a non-empty kernel. If $\mathbf{p}$ is a point belonging to the kernel, then each face of the polyhedron with centroid $\mathbf{c}$ and outward normal $\mathbf{n}$ defines the linear constraint $\mathbf{n} \cdot (\mathbf{c} - \mathbf{p}) > 0$ on $\mathbf{p}$. As a result, the kernel is non-empty if the set of constraints admits a feasible solution for $\mathbf{p}$.

The star-shaped and complex cell tests can be performed very efficiently. Moreover, performing these tests does not require an explicit representation of $\mathcal{F}$. It works even when the surface is represented as a Boolean combination



(a) Transfer Function (b) Connectivity

Fig. 3. *The left figure shows how to define the transfer function τ. We wish to define a path from $\mathbf{p}$ to one of the positive vertices of the grid cell that lies completely in $\mathcal{F}$. Due to the star-shaped property, there exists a straight line path from $\mathbf{p}$ to the origin $\mathbf{o}$ and from $\mathbf{o}$ to a grid vertex. This defines the transfer function τ and the transfer path function Γ for point $\mathbf{p}$. The right figure highlights the connectivity graph. If there exists a path $\mathcal{P}$ between $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ in free space $\mathcal{F}$, then there exists a path between them in the connectivity graph G. This is because the set of connectivity graph vertices belonging to the cells along path $\mathcal{P}$ is connected in G*

of other primitives. This fits with our representation of $\mathcal{F}$ as the complement of the union of individual Minkowski sums.

5.2 Roadmap Computation

The recursive subdivision algorithm discussed above generates an adaptive voxel grid. In this section, we show how to use the grid to construct a connectivity roadmap. In particular, we use the grid to define a connectivity graph $G = (V, E, w)$ and a transfer function τ.

Connectivity Graph

We extract a graph from the adaptive grid as follows. Let V be the set of grid vertices that lie in $\mathcal{F}$ and E be the set of grid edges that lie in $\mathcal{F}$. The weight function $w : E \rightarrow \mathbb{R}$ is defined as the Euclidean distance between the edge vertices. The connectivity graph is defined as the undirected graph $G = (V, E, w)$.

Transfer Function

We define a transfer function by defining a mapping from any arbitrary point in free space to a grid vertex. We note that a naive function that maps the point by snapping to the closest grid vertex need not guarantee a collision free path.

Consider a point $\mathbf{p} \in \mathcal{F}$ belonging to a grid cell C. We consider two cases. **Case 1**: Suppose C is not intersected by $\partial\mathcal{F}$. Since C contains point $\mathbf{p} \in \mathcal{F}$, we have $C \subseteq \mathcal{F}$. In particular, all the grid vertices of C lie in $\mathcal{F}$. We pick any one of the grid vertices of C, say $\mathbf{v}$. By definition of the connectivity graph $G = (V, E, w)$, we have $\mathbf{v} \in V$. We let $\tau(\mathbf{p}) = \mathbf{v}$. Since C is convex, the straight line segment between $\mathbf{p}$ and $\mathbf{v}$, $\mathbf{pv}$, is contained within C, and therefore lies within $\mathcal{F}$. Therefore, the transfer function satisfies the property $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{p})$. Further, the transfer path function $\Gamma(\mathbf{p}) = \mathbf{pv}$.

Case 2: Consider the case where C is intersected by $\partial\mathcal{F}$. Due to the star-shaped property, $\mathcal{F}_C = \mathcal{F} \cap C$ is star-shaped. Let $\mathbf{o}$ be the origin of $\mathcal{F}_C$. Because the cell is not complex, there exists at least one grid vertex in $\mathcal{F}_C$. Let this vertex be $\mathbf{v}$. We let $\tau(\mathbf{p}) = \mathbf{v}$. Due to the star-shaped property, both $\mathbf{v}$ and $\mathbf{p}$ are "visible" from $\mathbf{o}$. Since the line segments $\mathbf{po}$ and $\mathbf{ov}$ lie in $\mathcal{F}$, we have $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{p})$ (see Fig. 3(a)). This also gives us the transfer path function, $\Gamma(\mathbf{p}) = \mathbf{po} :: \mathbf{ov}$.

5.3 Connectivity Guarantee

In this section, we show that the connectivity roadmap defined above captures the connectivity of $\mathcal{F}$. This is formally expressed in Theorem 1. We begin by presenting a lemma. Given a cell C, the connectivity graph restricted to C is given by $G_C = (V_C, E_C)$ where V_C and E_C denote the set of grid points and grid edges respectively of cell C that lie in $\mathcal{F}$. We have dropped the weight function from the graph notation for convenience. It is assumed to be the Euclidean distance function.

Lemma 1. *Given a cell C, the graph $G_C = (V_C, E_C)$ is connected.*

Proof: Consider any two grid vertices $\mathbf{v}, \mathbf{w} \in V_C$. We prove that $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{w}$. It suffices to prove that there exists a sequence of grid cell edges connecting $\mathbf{v}$ and $\mathbf{w}$ that do not intersect $\mathcal{F}$. We consider three cases:

- **Case 1**: The grid points $\mathbf{v}$ and $\mathbf{w}$ are the endpoints of an edge of the cell. Since both $\mathbf{v}$ and $\mathbf{w}$ have the same (positive) sign and the edge is not complex, this edge cannot intersect $\partial\mathcal{F}$. Therefore the edge belongs to E_C and we have $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{w}$.
- **Case 2**: The grid vertices $\mathbf{v}$ and $\mathbf{w}$ lie diagonally opposite on a cell face. The case where the other two grid vertices on the face are negative corresponds to a case of face ambiguity. Therefore, at least one of the other two grid vertices (say $\mathbf{u}$) on the face has a positive sign. $\mathbf{v}$ and $\mathbf{u}$ are two positive grid vertices on the endpoints of an edge of the cell. Therefore by Case 1, we have $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{u}$. Similarly, we have $\mathbf{u} \overset{G_C}{\longleftrightarrow} \mathbf{w}$. This implies $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{w}$.
- **Case 3**: The grid points $\mathbf{v}$ and $\mathbf{w}$ are diagonally opposite vertices of the cell. The case where all the other grid vertices are negative corresponds to a case of voxel ambiguity. Therefore, at least one other grid vertex $\mathbf{u}$ has a positive sign. Depending on $\mathbf{u}$'s position, the vertices $\mathbf{v}$ and $\mathbf{u}$ either reduce to Case 1 or 2 ($\mathbf{u}$ and $\mathbf{w}$ will belong to the other case). Therefore, $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{u}$ and $\mathbf{u} \overset{G_C}{\longleftrightarrow} \mathbf{w}$ which implies $\mathbf{v} \overset{G_C}{\longleftrightarrow} \mathbf{w}$.

□

Since G_C is a subgraph induced by G, any two vertices $\mathbf{v}$, $\mathbf{w} \in V_C$ satisfy $\mathbf{v} \overset{G}{\longleftrightarrow} \mathbf{w}$.

Theorem 1.

$$\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q} \iff \tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q})$$

Proof: We first prove that if $\tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q})$, then we have $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q}$. By construction of the transfer function, we have $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{p})$ and $\mathbf{q} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{q})$. Moreover, because graph G consists of vertices and edges in $\mathcal{F}$, we have

$$\tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q}) \implies \tau(\mathbf{p}) \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{q})$$

Therefore, we have $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q}$.

We now prove the converse. Let $\mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q}$. We have

$$\tau(\mathbf{p}) \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{p}, \quad \mathbf{p} \overset{\mathcal{F}}{\longleftrightarrow} \mathbf{q}, \quad \mathbf{q} \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{q})$$

Therefore, we have $\tau(\mathbf{p}) \overset{\mathcal{F}}{\longleftrightarrow} \tau(\mathbf{q})$. If $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ belong to the same grid cell, then Lemma 1 ensures that $\tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q})$.

Consider the case where $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ belong to different cells, $C_{\mathbf{p}}$ and $C_{\mathbf{q}}$ respectively. There exists a path between $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ in $\mathcal{F}$. Let $C_i, i = 0, \ldots n$, be the set of cells that are intersected by this path such that $C_0 = C_{\mathbf{p}}$ and $C_n = C_{\mathbf{q}}$. Suppose the path passes from a cell C_j into an adjacent cell C_k. Let the corresponding connectivity graphs restricted to C_j and C_k be $G_{C_j} = (V_j, E_j)$ and $G_{C_k} = (V_k, E_k)$ respectively. According to Lemma 1, both V_j and V_k are connected in graph G. The path passes from cell C_j to C_k through a face of the cell. Let f_j be the face of C_j that is incident on C_k and f_k be the face of C_k that is incident on C_j. Since grid cells C_j and C_k can be at different resolutions, f_j and f_k need not be identical. The path penetrates faces f_j and f_k at a common point $\mathbf{r}$ that lies in $\mathcal{F}$ (see Fig. 3(b)). Since both faces f_j and f_k are not complex and have at least one point in $\mathcal{F}$, they contain positive vertices $\mathbf{v_j} \in V_j$ and $\mathbf{v_k} \in V_k$ respectively that lie in $\mathcal{F}$. We will show below that $\mathbf{v_j} \overset{G}{\longleftrightarrow} \mathbf{v_k}$. As a result, $V_j \cup V_k$ is connected in graph G. This is true of all the cells along the path. Therefore, $\tau(\mathbf{p})$ and $\tau(\mathbf{q})$ are connected in graph G and we have $\tau(\mathbf{p}) \overset{G}{\longleftrightarrow} \tau(\mathbf{q})$.

We now return to the proof of the result $\mathbf{v_j} \overset{G}{\longleftrightarrow} \mathbf{v_k}$. The octree subdivision ensures that one of the faces f_j and f_k is a subset of the other. Let f denote the larger of the two faces. Vertices $\mathbf{v_j}$ and $\mathbf{v_k}$ lie on f. Using the facts that f is not complex and that $\mathcal{F}$ is star-shaped w.r.t f, it can be shown that the set $\mathcal{F}_f = \mathcal{F} \cap f$ is connected [20]. This implies that vertices $\mathbf{v_j}$ and $\mathbf{v_k}$ are connected in $\mathcal{F}_f$, i.e., $\mathbf{v_j} \overset{\mathcal{F}_f}{\longleftrightarrow} \mathbf{v_k}$. We wish to prove that $\mathbf{v_j} \overset{G}{\longleftrightarrow} \mathbf{v_k}$.

This is a two-dimensional version of the above theorem. Since our constraints of complex cell and star-shapedness extend to all the faces and edges of the cell, we can apply our argument recursively. The base case of our recursion is the one-dimensional case where we need to show that two points on an edge connected in $\mathcal{F}$ are also connected through G. This is readily shown by observing that the edges of the grid cells are not complex.

□

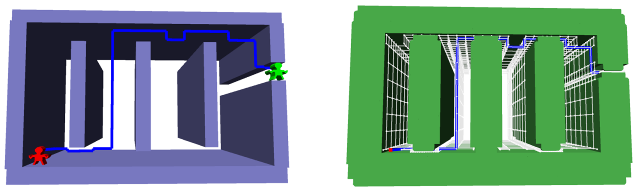


Fig. 4. Maze problem: *The left image shows a robot navigating within a maze model from a source (shown in red) to a destination (shown in green). The right image show the configuration space obstacle along with the connectivity graph. A path exists and passes through a number of narrow passages. Our algorithm generated a connectivity roadmap with* $18K$ *vertices in* 6 *secs and was able to find a path (shown in blue) in* 0.07 *secs.*

6 Implementation and Results

In this section, we describe the implementation of our algorithm and demonstrate its performance on path planning examples with narrow passages that test our algorithm. We used C++ programming language with the GNU g++ compiler under Linux operating system. Table 1 highlights the performance of our algorithm on these models. All timings are on a 2 GHz Pentium IV PC with a GeForce 4 graphics card and 1 GB RAM.

We compute a convex decomposition of the two polyhedra and compute pairwise Minkowski sums between the convex pieces. We used a modification of the convex decomposition scheme available in a public collision detection library, SWIFT++ [6]. We used a *convex hull* algorithm to compute the pairwise Minkowski sums. This algorithm adds the vectors of each vertex of one polyhedron with that of every vertex of the other polyhedron to get a point cloud. It computes a convex hull of the point cloud to obtain the pairwise Minkowski sum. Its time complexity is $O(f^2)$ where f is the number of features in the two convex polyhedra. In practice, this step doesn't take too much time (see Table 1). We used Dijkstra's single source shortest path algorithm to perform the graph search on the connectivity graph. We used the routine provided by a public domain library, Boost Graph Library [18]. Its running time is $O(|V| + |E|) \log |V|$ where $|V|$ and $|E|$ are the number of vertices and edges in the graph.

Table 1 provides a breakup of the total time. It shows that most of the time is spent on roadmap construction and a very small fraction of the total time is spent on graph search and pairwise Minkowski sum computation. Given an environment with static obstacles and a robot, we need to construct the connectivity roadmap just once. We can perform planning between a new pair of initial and final positions without having to recompute the roadmap.

Fig. 4 shows a robot navigating within a maze model. A path exists and passes through a number of narrow passages. Our algorithm successfully found a path through the narrow passages. It generated a connectivity roadmap with $18K$ vertices in 6 secs and was able to find a path in 0.07 secs. We also considered a scenario where the maze was modified such that there was no collision-free path. Our algorithm took 7 secs to generate a connectivity roadmap with $25K$ vertices and found out in 0.07 secs that no path exists.

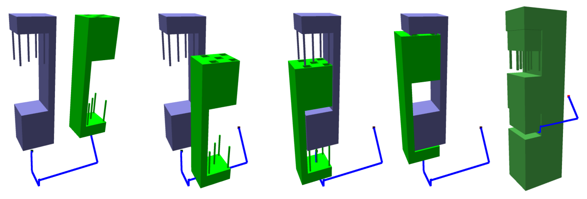


Fig. 5. Assembly problem: *This benchmark shows application of our algorithm to assembly planning. The four images on the left shows a path that the robot can take so that the two parts could be assembled. Our algorithm took* 12 *secs to generate a connectivity roadmap with* $28K$ *vertices and was able to find a path (shown in blue) in* 0.11 *secs. The rightmost image shows the configuration space obstacle 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.*

Fig. 5 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. Our algorithm took 12 secs to generate a connectivity roadmap with $28K$ vertices and was able to find a path (shown in blue) in 0.11 secs.

Model	Combinatorial Complexity				Performance			Roadmap
	Num Tris		Num Convex		Convex Mink	Roadmap	Graph	
	Obstacle	Robot	Obstacle	Robot	(s)	(s)	(s)	size
Maze (Path exists)	96	136	8	10	0.08	6	0.07	18,453
Maze (No Path)	100	136	8	10	0.08	7	0.07	25,920
Assembly	224	224	16	16	0.1	12	0.11	28,815

Table 1. *This table highlights the performance of our algorithm on different models. The model complexity is indicated in terms of the triangle count and number of convex pieces for the obstacles and robot. The table shows the time taken to compute pairwise convex Minkowski sum, construct the connectivity roadmap and to perform the graph search. The rightmost column shows the size of the connectivity roadmap, which is equal to the number of vertices in the connectivity graph.*

Fig. 6 shows the performance of our roadmap construction algorithm on the maze model (Fig 4) as a function of the robot size. As we vary the robot size, the size and the number of narrow passages in the configuration space change, resulting in varying performance.

7 Discussion

7.1 Performance

The overall performance is dominated by the roadmap construction step of our algorithm. Its time complexity is output-sensitive — it is $O(N)$ where N is the size of the roadmap. The roadmap size is primarily dependent on the narrow passges in the free space. Our algorithm ensures that it captures the connectivity of free space by performing additional sampling in the vicinity of narrow passages. The amount of sampling is related to the size and number of narrow passages (see Fig. 6). For example, suppose two configuration space

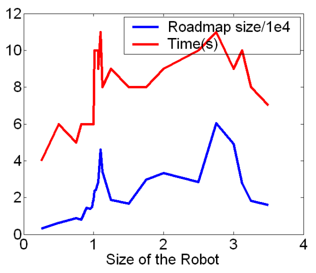


Fig. 6. *This plot shows the performance of our roadmap construction algorithm on the maze model as a function of the robot size. We have shown two measures of performance: roadmap construction time and roadmap size. The x-axis shows the scaling factor applied to the robot shown in Fig. 4.*

obstacles are at a distance ϵ apart such they form a narrow passage of width ϵ. In order to obtain samples within the narrow passage, the grid cells in the vicinity may need to be subdivided until they become smaller than ϵ.

Moreover, the volumetric grid generated by our sampling algorithm is coordinate system dependent. A change of coordinate system can result in a volumetric grid with different levels of subdivision. Consequently, the size of the roadmap depends on the coordinate system.

7.2 Translational & Rotational Motion Planning

Our basic approach is also applicable to a 2D robot undergoing rotation and translation motion in a plane. We can reformulate the problem of computing the connectivity of its configuration space by reducing rotation to a swept volume computation and the translational part remains a Minkowski sum computation. Let the robot $\mathcal{R}$ and obstacles $\mathcal{O}$ be in a $2D$ workspace. We can think of this $2D$ workspace to be embedded in a $3D$ workspace where the z-axis represents the rotation (θ-axis). We map both the robot and obstacles to an alternate space, the swept volume space, as follows: we extrude the robot along the θ-axis while rotating it about θ-axis simultaneously. In other words, we are computing the swept volume of the robot under screw motion. We extrude the obstacles $\mathcal{O}$ along the θ-axis, which is the swept volume of the obstacles under translation. As a result, we obtain representations $\mathcal{R}_{SV}$ and $\mathcal{O}_{SV}$ for the robot and obstacles, respectively, in the swept volume space. Then the configuration space obstacle corresponds to performing a *sliced Minkowski sum* operation on $\mathcal{R}_{SV}$ and $\mathcal{O}_{SV}$. This is related to the concept of computing critical slices in the configuration space [12, 16]. By using the above formulation, we can use our path planning algorithm to compute a collision-free path.

7.3 Tangential Contact

The sampling algorithm presented earlier uses an octree to perform the adaptive subdivision. This presents a problem when the configuration space obstacles are in tangential contact. In such a case, if we use axis-aligned hyperplanes to subdivide the cell, the subdivision process may not terminate. Dealing with such degenerate configurations is a difficult problem. One possible approach to handle the termination issue is to augment our original subdivision strategy by including supporting planes that arise from the individual Minkowski primitives themselves. This is akin to binary space partitioning (BSP) technique. Cells generated by this approach are no longer cubical, but general convex polyhedra. It is easy to prove that such a technique will always terminate [19].

7.4 Limitations

Our sampling algorithm uses two criteria: complex cell test and star-shaped test to guide the subdivision. These criteria are conservative. Consequently, our algorithm may perform unnecessary subdivision. This can reduce the performance of our algorithm and result in a larger roadmap than necessary. The convex decomposition method can result in a large number of convex pieces. Given two polyhedra each with n convex pieces, we obtain n^2 pairwise convex Minkowski sums. Since this set of pairwise convex Minkowski sums is an input to our algorithm, its large size affects the performance of the overall algorithm.

8 Conclusion and Future Work

We have presented an algorithm for complete path planning for translating polyhedral robots in 3D. Our algorithm is based on constructing a connectivity roadmap that captures the connectivity of the free space. It is guaranteed to find a collision-free path if one exists. Otherwise it detects non-existence of any collision-free path.

Our algorithm is simple to implement in practice. It uses two tests: a complex cell test and a star-shaped test. These tests can be efficiently performed for polyhedral objects using max-norm distance computation and linear programming. The complexity of our algorithm varies as a function of the size of the narrow passages in configuration space. We highlight the performance of our algorithm on two environments with very small narrow passages or no collision-free paths.

There are many avenues for future work. For some applications, a robot is allowed to be in contact with the obstacles. We would like to extend our algorithm to accommodate this. We are interested in application of our algorithm to the problem of motion planning for a robot with translational and rotational degrees of freedom. Finally, we would like to improve the roadmap generation algorithm by making our approach less conservative and improve the overall performance.

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 sums in 3D.

References

1. B. Aronov and M. Sharir. On translational motion planning in 3-space. In *ACM Symposium on Computational Geometry*, pages 21–30, 1994.

2. B. Aronov, M. Sharir, and Boaz Tagansky. The union of convex polyhedra in three dimensions. *SIAM Journal on Computing*, 26:1670–1688, 1997.
3. Francis Avnaim and J.-D. Boissonnat. Practical exact motion planning of a class of robots with three degrees of freedom. In *Proc. of Canadian Conference on Computational Geometry*, page 19, 1989.
4. J.F. Canny. *The Complexity of Robot Motion Planning.* ACM Doctoral Dissertation Award. MIT Press, 1988.
5. D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.
6. 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.
7. S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 11–19, 1987.
8. L. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom*, 2:175–193, 1987.
9. L. J. Guibas and J. Hershberger. Computing the visibility graph of n line segments in $O(n^2)$ time. *Bull. EATCS*, 26:13–20, 1985.
10. D. Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002.
11. L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580, 1996.
12. K. Kedem and M. Sharir. An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. *Discrete Comput. Geom.*, 5:43–75, 1990.
13. J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.
14. D. T. Lee. Proximity and reachability in the plane. Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL, 1978.
15. T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.
16. E. Sacks. Practical sliced configuration space for curved planar pairs. *International Journal of Robotics Research*, 18(1), 1999.
17. J. T. Schwartz and M. Sharir. On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351, 1983.
18. J. G. Siek, L. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual.* Addison-Wesley, Boston, 2002.
19. TVN Sriram, Gokul Varadhan, and Dinesh Manocha. Handling degeneracies during adaptive subdivisions. In *UNC Technical Report TR04-14*, 2004.
20. G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. Technical report, Department of Computer Science, University of North Carolina, 2004.
21. J. Vleugels and M. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222, 1997.
22. Jane Wilhelms and Allen Van Gelder. Topological considerations in isosurface generation extended abstract. *Computer Graphics*, 24(5):79–86, 1990.