# Topology Preserving Approximation of Free Configuration Space

Gokul Varadhan [1] Young J. Kim [2] Shankar Krishnan [3] Dinesh Manocha [1]

[1] *University of North Carolina at Chapel Hill, USA, {varadhan,dm}@cs.unc.edu*

[2] *Ewha Womans University, Korea, kimy@ewha.ac.kr*

[3] *AT&T Labs - Research, Florham Park, NJ, USA, krishnas@research.att.com*

*http://gamma.cs.unc.edu/motion*

***Abstract*— We present a simple algorithm for approximating the free configuration space of robots with low degrees of freedom (DOFs). We represent the free space as an arrangement of contact surfaces. We approximate the free space using an adaptive volumetric grid that is computed by performing simple geometric tests on the contact surfaces. We use an isosurface extraction algorithm to compute a piecewise-linear approximation to the boundary of the free space. We prove that our approximation is topologically equivalent to the exact free space boundary. We also ensure that our approximation is geometrically close to the exact free space boundary by bounding its two-sided Hausdorff error. We have applied our algorithm to compute the free configuration space for the following instances: (1) a 2D polygonal robot with translational and rotational DOFs navigating among polygonal obstacles, and (2) a 3D polyhedral robot translating among polyhedral obstacles. In practice, our algorithm works well on robots with three DOFs.**

## I. Introduction

Configuration space is a fundamental concept in robotics [17], [18]. Consider a robot $\mathcal{R}$ navigating among a stationary rigid obstacle $\mathcal{O}$. The configuration space $\mathcal{C}$ of $\mathcal{R}$ is the set of all possible positions and orientations that $\mathcal{R}$ can assume. The underlying idea of configuration space is to represent the robot as a point in $\mathcal{C}$ and to map the obstacles to $\mathcal{C}$. The obstacle $\mathcal{O}$ maps to a region

$$\mathcal{CO} = \{\mathbf{q} \in \mathcal{C} : \mathcal{R}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\},$$

in $\mathcal{C}$, where $\mathcal{R}(\mathbf{q})$ is the subset of $W$ occupied by $\mathcal{R}$ at the configuration $\mathbf{q}$. $\mathcal{CO}$ is called the *$C$-obstacle region* or the *forbidden region*. The set

$$\mathcal{F} = \mathcal{C} \setminus \mathcal{CO}$$

is called the *free configuration space* or the *free space*. Configuration space reduces the problem of motion planning of a dimensioned robot into the problem of planning the motion of a point within the robot's free space. Configuration space has played a crucial role in helping understand motion planning problems, and led to the development of many motion planning algorithms. The problem of configuration space computation is to compute the free space $\mathcal{F}$. In this paper, we present an algorithm to compute an approximation of $\mathcal{F}$, and give geometric and topological guarantees on its accuracy.

Configuration space computation is a classic problem in algorithmic robotics and computational geometry. This problem arises in several important applications such as motion planning [1], [8], [18]–[20], collision detection and distance computation [3], [5], layout and containment problems in manufacturing [2], spatial reasoning [28], assembly and task planning [23], and tolerance analysis and mechanism design [12].

A general approach for configuration space computation proceeds by enumerating *contact surfaces* for every pair of features from the robot $\mathcal{R}$ and the obstacle $\mathcal{O}$. A contact surface of a geometric feature (vertex, edge, face) of $\mathcal{R}$ and a similar feature (vertex, edge, face) of $\mathcal{O}$ is defined as the set of points in the configuration space that represent configurations of $\mathcal{R}$ at which contact is made between these specific features. The set $\Gamma$ of contact surfaces define an arrangement $\mathcal{A}(\Gamma)$. The free space $\mathcal{F}$ consists of certain cells in this arrangement. Therefore, $\mathcal{F}$ can be computed by computing $\mathcal{A}(\Gamma)$ [1], [10], [14], [19]. The combinatorial complexity of the entire $\mathcal{F}$ can be $O(n^k)$ where $n$ is the number of contact surfaces in $\Gamma$ and $k$ is the dimension of the configuration space [22]. Complexity bounds for $\mathcal{F}$ are also known for certain specific cases of configuration spaces. For instance, in the case where $\mathcal{R}$ is a non-convex polygon with $p$ edges, translating and rotating in a polygonal environment bounded by $m$ edges, the maximum complexity of $\mathcal{F}$ is $\Theta((pm)^3)$. Similarly, if $\mathcal{R}$ is a non-convex polyhedron with $p$ polygons, translating in a polyhedral environment bounded by $m$ polygons, then the maximum complexity of $\mathcal{F}$ is $\Theta((pm)^3)$ [22].

A major bottleneck in using the above approach is arrangement computation. Arrangement computation reduces to computing intersections between pairs of surface primitives and is prone to problems in accuracy and robustness [11]. Two additional factors contribute to the difficulty of arrangement computation. First, the number of surface primitives in the arrangement can be high: the arrangement may have $O(n^2)$ surfaces, where $n$ is the number of features in the two objects. In our applications, the arrangement may consist of several thousands of surface primitives. Second, the surfaces in the arrangement are non-linear in configuration space of robots with rotational degrees of freedom. Computing intersections between non-linear primitives is difficult to implement and

expensive in practice. Therefore, we avoid exact free space computation.

**Main Results**

We present a simple algorithm to approximate $\mathcal{F}$. Our algorithm enumerates a set of contact surfaces whose arrangement defines $\mathcal{F}$. Instead of explicitly computing the arrangement, we approximate $\mathcal{F}$ by generating an adaptive volumetric grid in the configuration space $\mathcal{C}$. Our algorithm ensures that every grid cell satisfies two simple geometric tests: a *complex cell* test and a *star-shaped* test. These two tests are sufficient to capture the topology of $\mathcal{F}$. The resulting grid serves as an implicit representation of $\mathcal{F}$. The main benefit of our representation is that it eliminates the need for arrangement computation. We compute an approximate boundary of $\mathcal{F}$ by performing isosurface extraction on the grid using Marching Cubes. This yields a piecewise linear approximation to $\partial\mathcal{F}$ – the boundary of $\mathcal{F}$. Our approximation is topologically equivalent to $\partial\mathcal{F}$: in particular, it has the same number of connected components and identical genus. We also bound the two-sided Hausdorff distance between $\partial\mathcal{F}$ and our approximation. We have implemented the algorithm and applied it to compute free space approximation for the following instances: (1) a 2D polygonal robot with translational and rotational DOFs navigating among polygonal obstacles, and (2) a 3D polyhedral robot translating among polyhedral obstacles. As compared to prior approaches, our algorithm is relatively simple to implement.

**Organization:** The rest of the paper is organized in the following manner. We give a brief overview of prior work in free space computation in Section 2. Section 3 describes a representation of the free space and gives an overview of our approach. Section 4 presents an approach for computing topology preserving surface approximation. Section 5 presents an application of this approach to the problem of free space computation. Section 6 describes its implementation and highlight its performance.

## II. Previous Work

The problem of free space computation has been well studied. This problem can be reduced to computing the arrangement of contact surfaces. The arrangement computation problem is ubiquitous by nature, and it arises in a number of applications. A survey of different algorithms and complexity bounds for arrangements computations is given in [7], [22].

The maximum complexity of the entire $\mathcal{F}$ can be $O(n^k)$ where $n$ is the number of contact surfaces in $\Gamma$ and $k$ is the dimension of the configuration space [22]. For applications such as motion planning, it is not necessary to compute the entire $\mathcal{F}$; it is sufficient to compute a single connected component of $\mathcal{F}$. Halperin and Sharir [9] showed that the combinatorial complexity of a single cell of an arrangement of $n$ surfaces in three dimensions is $O(n^{2+\epsilon})$, for any $\epsilon > 0$, where the constant of proportionality depends on $\epsilon$ and on the maximum degree of the surfaces. Some of the major issues in the implementation of arrangement computation algorithms are accuracy and robustness problems. It is quite hard to enumerate all degenerate configurations, especially when the primitives (i.e. the contact surfaces) are non-linear primitives.

Few practical algorithms have been proposed for the case of a planar rigid robot with translational and rotational degrees of freedom [1], [19], [20]. These algorithms compute $\mathcal{F}$ by using a discrete number of slices along the orientation parameter (rotational degree of freedom). The boundary of $\partial\mathcal{F}$ is composed of ruled surface patches generated by contacts between a moving vertex $\mathcal{R}$ and an obstacle edge of $\mathcal{O}$ or between a moving edge of $\mathcal{R}$ and an obstacle vertex of $\mathcal{O}$. Avnaim et al. [1] presented an algorithm for constructing $\mathcal{F}$ in $\Theta((pm)^3 \log pm)$ time. Sacks [19], [20] used a similar formulation and developed the first complete motion planning algorithm that is practical for real-world applications. Sacks's algorithm is applicable to polygonal as well as curved primitives. For polygonal primitives, testing for a criticality reduces the problem to solving only a quadratic equation. In the worst case, the algorithm may need to test for $O(m^3n^3)$ criticality conditions.

Many algorithms have been proposed for computing C-obstacle of robots with only translational degrees of freedom. In this case, the C-obstacle is equal to the Minkowski sum of the obstacle and the robot (reflected about its origin). Many algorithms have been proposed for Minkowski sum computation [4], [6], [17], [26]. All of these algorithms reduce the problem to computing either an arrangement or a union of a large set of primitives, which can be difficult in practice. As a result, some algorithms [26] compute only an approximation to the Minkowski sum.

## III. Overview

In this section, we describe a representation of the free space followed by an overview of our approach.

### A. Notation

We use the following notation in the rest of the paper. We use lower case bold letters such as $\boldsymbol{p}, \boldsymbol{q}$ to refer to points in $\mathbb{R}^d$.

The symbols $\mathcal{R}$ and $\mathcal{O}$ denote the robot and the obstacles respectively. $\mathcal{C}$ denotes the configuration space. $\mathcal{F}$ denotes the free space and $\partial\mathcal{F}$ denotes its boundary. To simplify the exposition, we will assume that $\mathcal{C}$ is three-dimensional; however, we note that our algorithm is general and applicable to any configuration space dimension.

Our algorithm uses an adaptive volumetric grid – a spatial subdivision of the configuration space $\mathcal{C}$. The letter $C$ denotes a single cell in the grid. We assume a grid cell is a closed set and consists of a cube-shaped voxel, six faces, twelve edges, and eight vertices. We define a *restriction* of a set $S$ to a grid cell $C$ as $S_C = S \cap C$.

A homeomorphism is a continuous bijective mapping with a continuous inverse. Two objects are homeomorphic or topologically equivalent if there exists a homeomorphism between them.

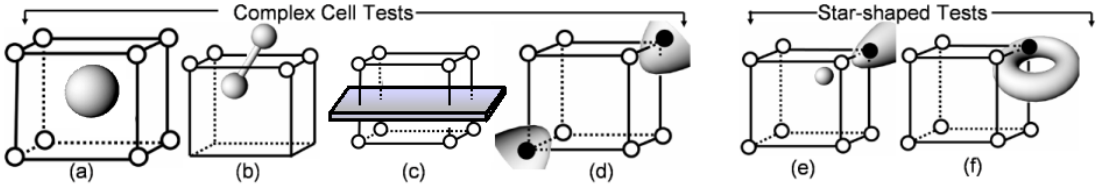



Fig. 1. **Complex cell and Star-shaped Test Cases:** *This figure shows the different cases corresponding to the complex cell and star-shaped test. Figs (a), (b), (c) and (d) show cases of complex voxel, complex face, complex edge, and topological ambiguity. The white and black circles denote positive and negative grid points respectively. Fig. (e) shows the case where the isosurface is not star-shaped w.r.t a voxel. In Fig (f), the restriction of the isosurface to the right face of the cell is not star-shaped.*

Given a set $\mathcal{Q}$, let $d(\boldsymbol{p}, \mathcal{Q})$ denote the distance between a point $\mathbf{p}$ and $\mathcal{Q}$ under some suitable metric (typically Euclidean). The one-sided Hausdorff distance between two sets $\mathcal{P}$ and $\mathcal{Q}$ is defined as follows:

$$h(\mathcal{P}, \mathcal{Q}) \quad = \quad \max\{\min d(\boldsymbol{p}, \mathcal{Q}) \mid \boldsymbol{p} \in \mathcal{P}\}$$

Note that the above definition is not symmetric, i.e., $h(\mathcal{P}, \mathcal{Q})$ is not necessarily equal to $h(\mathcal{Q}, \mathcal{P})$. The two-sided Hausdorff distance is defined as:

$$H(\mathcal{P}, \mathcal{Q}) \quad = \quad \max(h(\mathcal{P}, \mathcal{Q}), h(\mathcal{Q}, \mathcal{P}))$$

### B. Free Space Representation

We assume that the robot $\mathcal{R}$ is a rigid or an articulated object moving among stationary rigid obstacles $\mathcal{O}$. We also assume that the geometry of both $\mathcal{R}$ and $\mathcal{O}$ is accurately known. The free space $\mathcal{F}$ is the set of configurations at which $\mathcal{R}$ does not collide with $\mathcal{O}$. The boundary of $\mathcal{F}$, denoted as $\partial\mathcal{F}$, consists of those configurations of $\mathcal{R}$ at which $\mathcal{R}$ makes contact with $\mathcal{O}$, but does not penetrate into the interior of $\mathcal{O}$. Therefore, $\partial\mathcal{F}$ can be expressed in terms of a collection of *contact surfaces* (C-surfaces), each being the locus of configurations of $\mathcal{R}$ at which a specific feature of $\mathcal{R}$ is in contact with a feature of $\mathcal{O}$. We refer the reader to [14] for a detailed explanation of C-surfaces. We note two important properties of C-surfaces:

**Superset property:** The set $\Gamma$ of C-surfaces define an arrangement in $\mathcal{C}$. $\mathcal{F}$ is a collection of cells in this arrangement; a cell corresponding to a connected component of $\mathcal{F}$. Furthermore, $\Gamma$ is a superset of the boundary $\partial\mathcal{F}$ of free space, i.e., $\partial\mathcal{F} \subseteq \bigcup\{\gamma_i \in \Gamma\}$.

**Orientation property:** We can assign an orientation to each $C$-surface. We explain this with an intuitive argument. Consider a $C$-surface $\gamma$ generated by the contact between a robot feature $f_1$ and an obstacle feature $f_2$. Points on *one side of* $\gamma$ correspond to the case where $f_1$ has penetrated $f_2$. These points belong to C-obstacle. On the other hand, points on the *other side* correspond to no overlap or contact between $f_1$ and $f_2$. We orient $\gamma$ by assigning a normal at $\boldsymbol{p}$ to point "towards C-obstacle". A more precise explanation of the orientation property can be found in [25].

$\partial\mathcal{F}$ can be obtained by computing the arrangement of $\Gamma$. For a 3-dimensional configuration space, the set of 2-dimensional cells in the arrangement provides a partition of the C-surfaces into a set of surface components. Given such a partition, we can combine a subset of the surface components to obtain $\partial\mathcal{F}$. However, this is not feasible in practice due to the difficulty of arrangement computation. Therefore, we avoid exact free space computation.

### C. Our Approach

We compute an approximation to $\partial\mathcal{F}$ using distance field-based techniques. We compute a signed distance field to $\partial\mathcal{F}$. A signed distance field $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a continuous function that at a point $\boldsymbol{p}$ measures the distance between $\boldsymbol{p}$ and $\partial\mathcal{F}$ under a suitable metric (e.g., Euclidean). This value is positive or negative depending on whether the point lies outside or inside $\partial\mathcal{F}$.

A common way of representing the signed distance field is to discretize the continuous distance field into discrete samples – to compute the value of the distance field at the vertices of a volumetric grid. We refer to this step as *sampling* of the distance field. The grid is an approximate representation of the distance field; the accuracy of the approximate representation depends on the *rate of sampling* – the resolution of the grid.

The 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) **Reconstruction**: Use some variant of Marching Cubes algorithm [13], [16] to perform isosurface extraction from the distance field. The extracted isosurface is a piece-wise linear approximation to $\partial\mathcal{F}$.

Two important advantages of the above approach are simplicity and efficiency. Each step is easy to implement. A uniform grid or an adaptive grid (e.g. octree) may be chosen. Isosurface extraction is also reasonably straightforward; Marching Cubes is both simple and fast. Many public domain implementations of Marching Cubes [21] are available. Moreover, this approach does not require arrangement computation.

## IV. Approximate Approach

In our prior work [24], we presented an algorithm for computing topology preserving isosurfaces and used it to perform Boolean operations. We apply this algorithm to the problem of free space computation. We provide a brief description of this algorithm. It is based on the sampling and reconstruction approach presented in Section III-C. 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**

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 (Figs. 1(a) & 1(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 (Fig. 1(c)). It is well known that Marching Cubes produces topologically ambiguous output for certain sign configurations (Fig. 1(d)). These sign configurations are referred to as *ambiguous sign configurations*.

**DEFINITION 1**

1) **Complex cell:** A cell is *complex* if it has a *complex voxel*, *complex face*, *complex edge*, or an *ambiguous sign configuration.*
2) **Complex cell test** ($\mathcal{C}^{\square}$) **:** *A cell $C$ satisfies $\mathcal{C}^{\square}$ if $C$ is not complex.*

Intuitively, $\mathcal{C}^{\square}$ ensures that the surface intersects the grid cell in a simple manner in most cases. If a grid cell does not satisfy $\mathcal{C}^{\square}$, it is subdivided and the algorithm is recursively applied to each of its children cells.

**Star-shaped Test**

A surface $\mathcal{E}$ is *star-shaped* if there exists a point $\boldsymbol{o} \in \mathbb{R}^3$ (called guard) such that for any $\boldsymbol{x} \in \mathcal{E}$ we have $\mathbf{ox} \cap \mathcal{E} = \{\mathbf{x}\}$. Intuitively, the guard can "see" every point on a star-shaped surface.

We now define the star-shaped property for a cell. We say $\mathcal{E}$ is *star-shaped with respect to* (w.r.t) *a voxel* $\vartheta$ if there exists a point $\mathbf{o} \in \mathbb{R}^3$ such that for any $\boldsymbol{x} \in \mathcal{E}_\vartheta = \mathcal{E} \cap \vartheta$ we have $\mathbf{ox} \cap \mathcal{E}_\vartheta = \{\mathbf{x}\}$. Point $\mathbf{o}$ is a guard of $\mathcal{E}_\vartheta$. A similar property is also defined for the faces of the cell. We define $\mathcal{E}$ 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 faces.

**DEFINITION 2**

**Star-shaped test** ($\mathcal{C}^{\bigstar}$) **:** *A cell $C$ satisfies $\mathcal{C}^{\bigstar}$ if $\mathcal{E}$ is star-shaped w.r.t $C$.*

If a cell does not satisfy $\mathcal{C}^{\bigstar}$, it is subdivided and the algorithm is recursively applied to the children cells.

In this manner, by applying $\mathcal{C}^{\square}$ and $\mathcal{C}^{\bigstar}$, the algorithm generates a volumetric grid. It uses Marching Cubes to perform isosurface extraction on the resulting grid. The extracted surface is a piecewise-linear approximation to $\mathcal{E}$.

In [24], we used max-norm distance computation, linear programming, and interval arithmetic to perform the complex cell and star-shaped tests. Performing these computations 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 [24] for additional details.

## V. Free Space Approximation Algorithm

We apply the approximate algorithm described in Sec. IV to the problem of free space computation. We generate an adaptive volumetric grid in $\mathcal{C}$ by performing the complex cell and star-shaped tests on $\partial\mathcal{F}$. The computational techniques presented in [24] to perform these tests assume that the desired surface is defined as a Boolean combination over a set of closed primitives. This assumption does not hold in the case of free space computation: $\partial\mathcal{F}$ is defined as an arrangement of C-surfaces. Furthermore, C-surfaces may have boundaries.

We present a set of techniques for performing complex cell and star-shaped tests for free space computation. The complex cell and star-shaped tests rely on a number of queries: *sign query*, *star-shaped query*, and *cell intersection query*. We first present techniques for answering these queries without computing an explicit representation of $\partial\mathcal{F}$. We then present the adaptive subdivision algorithm that performs the complex cell and star-shaped tests.

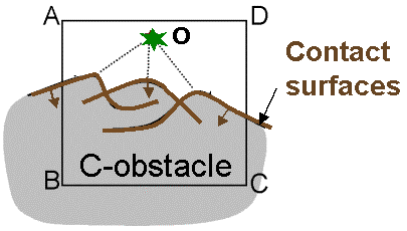



Fig. 2. **Star-shaped Test:** *If in a cell $C$, all the C-surfaces satisfy the contact surface condition (Equation 1), then $\partial\mathcal{F} \cap C$ is star-shaped w.r.t* $\mathbf{o}$*. The arrows indicate the orientation of the C-surfaces.*

### A. Supporting Queries

**Sign Query**

Given a point $\mathbf{q} \in \mathcal{C}$, the sign query determines whether $\mathbf{q}$ belongs to the free space, i.e., if $\mathbf{q} \in \mathcal{F}$. The definition of the free space reduces this query to a collision check between $\mathcal{R}(\mathbf{q})$ and the obstacles: $\mathbf{q} \in \mathcal{F}$ if and only if $\mathcal{R}(\mathbf{q})$ does not intersect any obstacle. Therefore, this query can be implemented using a collision detection routine [15].

**Star-shaped Query**

Consider a grid cell $C$ in the configuration space. The star-shaped query answers the following question: is $\partial\mathcal{F}$ star-shaped w.r.t C? We need to perform two tests on $\partial\mathcal{F}$ – (a) star-shaped w.r.t voxel, and (b) star-shaped w.r.t. each face. We present a conservative technique to perform both tests. We exploit the fact that the C-surfaces form a superset of $\partial\mathcal{F}$ (Superset property, Sec. III-B). This reduces the problem to performing certain computations on the C-surfaces.

Let $S$ be a voxel or a face of cell $C$. Let $\Gamma$ denote the set of all C-surfaces. For each C-surface $\gamma_i \in \Gamma$ that intersects $S$, compute the restriction $\gamma_{i,S} = \gamma_i \cap S$ to $S$. Let $\Gamma_S$ denote the resulting set of surfaces.

We can answer the star-shaped query provided $S$ satisfies the following condition:

**Contact Surface Condition:**

Is there a point $\boldsymbol{o} \in S$ such that
For each $\gamma_{i,S} \in \Gamma_S$ the following holds:
for all $\mathbf{x} \in \gamma_{i,S}$ with normal $\mathbf{n_x}$ we have

$$\mathbf{ox} \cdot \mathbf{n_x} > 0 \tag{1}$$

See Fig. 2. If $S$ satisfies the above condition, then we can answer the star-shaped query. This is formally stated as the following theorem.

**THEOREM 1**

**Star-shaped test***: Suppose $S$ satisfies the contact surface condition (Equation 1). Then*

1) $\mathcal{F}_S \neq \emptyset \iff \mathbf{o} \in \mathcal{F}$.
2) *If $\partial\mathcal{F}_S \neq \emptyset$, then $\partial\mathcal{F}_S$ is star-shaped w.r.t* $\mathbf{o}$.

The proof of this result can be found in [25].

Theorem 1 reduces the star-shaped query on $\partial\mathcal{F}$ to verifying the contact surface condition for the contact surfaces. We can perform this verification using linear programming and interval arithmetic based techniques, as described in [24].

The above test for star-shaped query is conservative: if a cell $C$ passes the test, then $\partial\mathcal{F}$ is star-shaped w.r.t C; however, the converse is not true. It is possible for $\mathcal{F}$ to be star-shaped w.r.t $C$ even if the contact surface condition is not satisfied. The main advantage of the test is that it does not require an explicit representation of $\mathcal{F}$. If $C$ fails the above tests, then we subdivide $C$ and repeat the tests on the subdivided cells. While this may result in some unnecessary subdivision, it preserves the correctness of the algorithm.

**Cell Intersection Query**

The objective of cell intersection query is to test if $\partial\mathcal{F}$ intersects the cell. Specifically, we need to test if $\partial\mathcal{F}$ intersects a voxel, a face, or an edge of a cell. We refer to these three tests collectively as cell intersection queries, and individually as voxel, face, and edge intersection query.

**Voxel/Face Intersection Query:** Let $S$ be a voxel or a face of cell $C$. This query answers whether $\partial\mathcal{F}$ intersects $S$, i.e., if $\partial\mathcal{F}_S \neq \emptyset$. In general, this query is difficult without an explicit representation of $\partial\mathcal{F}$. We answer the intersection query in a special case – when $S$ satisfies the contact surface condition. In this case, we use a test based on the following corollary of Theorem 1.

**COROLLARY 1**

**Cell Intersection query:** *Suppose $S$ satisfies the contact surface condition. Then*

$$\mathbf{o} \in \mathcal{F} \quad \wedge \quad \Gamma_S \neq \emptyset \quad \iff \quad \partial\mathcal{F}_S \neq \emptyset$$

The proof of this result can be found in [25].

We can check if $\mathbf{o} \in \mathcal{F}$ using the sign query. To check if $\Gamma_S \neq \emptyset$, we need to test if any C-surface in $\Gamma$ intersects $S$.

**Algorithm 1** `Adaptive_Subdivision`($C$)
**Input:** Grid cell $C$
**Output:** An adaptive subdivision of $C$.

```
if (C passes the star-shaped test) then
  o = origin of C
  if o ∉ F then
    return C
  end if
  if (C passes the complex cell test) then
    return C
  end if
end if
Subdivide C into children cells C_i
for each child C_i do
  Adaptive_Subdivision(C_i)
end for
```

This can be done using either max-norm distance computation and interval arithmetic based techniques, as described in [24].

**Edge Intersection Query:** Consider an edge $e$ with endpoints $\mathbf{a}$ and $\mathbf{b}$. This query computes the number of points at which $\partial\mathcal{F}$ intersects $e$. If the number of intersection points is greater than 0, then $e$ is intersected by $\partial\mathcal{F}$.

We exploit the superset property of contact surfaces. We compute the intersection of $e$ with all the contact surfaces. Let $I = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_k}\}$ denote the resulting set of intersection points. Since the contact surfaces are a superset of $\partial\mathcal{F}$, some of these intersection points may not belong to $\partial\mathcal{F}$. We perform a test on each $\mathbf{p_i}$ to check if it belongs to $\partial\mathcal{F}$.

Assume that the intersection points in $I$ are sorted along the edge: Point $\mathbf{p_i}$ is closer to $\mathbf{a}$ than $\mathbf{p_{i+1}}$. Define $\mathbf{p_0} = \mathbf{a}$ and $\mathbf{p_{k+1}} = \mathbf{b}$. Compute a set of points $\{\mathbf{q_0}, \ldots, \mathbf{q_k}\}$ where $\mathbf{q_i} = (\mathbf{p_i} + \mathbf{p_{i+1}})/2$. Point $\mathbf{p_i}$ belongs to $\partial\mathcal{F}$ if either $\mathbf{q_{i-1}} \in \mathcal{F}$ or $\mathbf{q_i} \in \mathcal{F}$, which can be tested using the sign query.

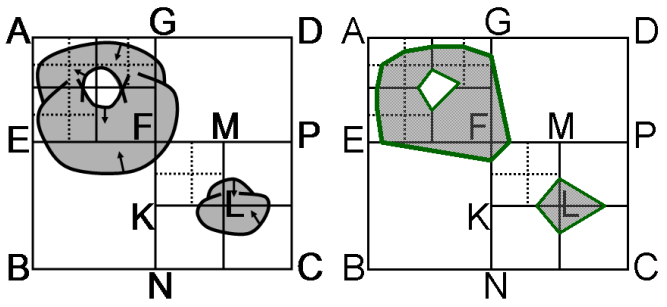
### *B. Adaptive Subdivision Algorithm*

We generate a spatial subdivision of the configuration space $\mathcal{C}$ in the form of an adaptive volumetric grid. The algorithm starts with a single grid cell that bounds all the contact surfaces. The algorithm performs both *complex cell* test and *star-shaped* test on the grid cell. If both the tests pass the grid cell is returned as a leaf node of the adaptive grid. Otherwise the grid cell is subdivided and the algorithm is recursively applied to its children cells. The algorithm terminates when all the grid cells satisfy the complex cell and star-shaped tests.

Below we give details of the complex cell and star-shaped tests. We impose an order in which the two tests must be applied: we require that the star-shaped test be executed before the complex cell test. The reason for imposing this order will be evident below.

**Star-shaped Test**

The star-shaped test performs two tests on $\partial\mathcal{F}$ – (a) star-shaped w.r.t voxel, and (b) star-shaped w.r.t. each face. We use




(a) Adaptive Subdivision (b) Reconstruction

Fig. 3. *Fig. (a) highlights our adaptive subdivision algorithm in 2D. The figure shows an arrangement of oriented contact surfaces and the C-obstacle space (shown in gray). Cell $ABCD$ does not satisfy either the star-shaped or complex cell tests and therefore gets subdivided. Cell $EBNF$ satisfies both the tests and hence does not get subdivided. The adaptive subdivision continues until all the grid cells satisfies the two tests. Fig. (b) shows the result of applying Marching Cubes to the grid generated in Fig. (a). The resulting surface (shown in green) is topologically equivalent to the boundary of the free space.*

the star-shaped query presented in Sec. V-A to perform these tests.

If any of these tests result in the negative, we subdivide the cell and apply the algorithm recursively to the new cells.

**Complex Cell Test**

We perform the complex cell test on a cell $C$ only when $C$ has already satisfied the star-shaped test. This means $C$ satisfies the contact surface condition (Equation 1). Hence we can use Corollary 1 to perform the cell intersection query on $C$. We take advantage of this fact while performing the complex cell test.

To check whether a cell is complex, we perform the following tests:

- **Complex Voxel/Face**: We use the cell intersection query to check whether $\partial\mathcal{F}$ intersects a voxel or face of the cell. If $\partial\mathcal{F}$ intersects the voxel (face), then we determine if the voxel (face) is complex by checking for a sign change at the cell vertices. The signs at the cell vertices are computed using the sign query. If $\partial\mathcal{F}$ does not intersect the voxel (face), then the voxel (face) is not considered complex.
- **Complex Edge**: We use the edge intersection query to test if an edge is complex. An edge is complex if the $\partial\mathcal{F}$ intersects the edge in more than one point.
- **Ambiguity**: We use the signs at the grid vertices to resolve cases corresponding to face and voxel ambiguity.

If any of these tests results in the affirmative, the cell is complex, and we subdivide it and apply the algorithm recursively to the new cells.

Alg. 1 shows the pseudo-code of our adaptive subdivision algorithm. Fig. 3 illustrates the algorithm in 2D.

### C. Geometric and Topological Guarantees

The adaptive subdivision algorithm generates an adaptive volumetric grid such that every grid cell satisfies the complex cell and star-shaped tests. In [24], we have shown that this is a sufficient condition for topology preserving isosurface extraction: applying Marching Cubes to such a grid produces a piecewise-linear approximation $\mathcal{A}$ that is topologically equivalent to $\partial\mathcal{F}$. In particular, $\mathcal{A}$ has the same number of connected components and genus as $\partial\mathcal{F}$. Furthermore, we have also described a simple extension to the adaptive subdivision algorithm that bounds the Hausdorff distance $H(\mathcal{A}, \partial\mathcal{F})$, also referred to as the *Hausdorff error*. The main idea is to check if a cell satisfies the following test:

**DEFINITION 3**

**Hausdorff test ($\mathcal{C}^\epsilon$) :** *Given an $\epsilon > 0$, a cell $C$ satisfies $\mathcal{C}^\epsilon$ if*

$$H(\mathcal{A}_C, \gamma_{i,C}) < \epsilon \quad \forall \quad \gamma_i \in \Gamma \text{ such that } \gamma_{i,C} \neq \emptyset$$

*where $\gamma_{i,C} = \gamma_i \cap C$ is the restriction of $\gamma_i$ to $C$.*

If every grid cell bounding $\partial\mathcal{F}$ satisfies $\mathcal{C}^\epsilon$, then we can show that $H(\mathcal{A}, \partial\mathcal{F})$ is bounded by $\epsilon$. This property is used in the subdivision algorithm to obtain a bounded Hausdorff error. A detailed explanation of this technique can be found in [25].

The following guarantees follow from the results in [24].

**THEOREM 2**

*If every cell in the volumetric grid satisfies $\mathcal{C}^\square$, $\mathcal{C}^\bigstar$, and $\mathcal{C}^\epsilon$ then*

1) **Geometric Guarantee:** *Given any $\epsilon > 0$, our algorithm outputs a free space approximation $\mathcal{A}$ such that $H(\mathcal{A}, \partial\mathcal{F}) < \epsilon$.*
2) **Topological Guarantee:** *Our free space approximation $\mathcal{A}$ is topologically equivalent to $\partial\mathcal{F}$.*

Together, the geometric and topological guarantees ensure an accurate free space approximation.

## VI. Implementation and Results

In this section, we describe the implementation of our algorithm and demonstrate its performance on configuration space generation examples. We used C++ programming language with the GNU g++ compiler under Linux operating system. Table I highlights the performance of our algorithm on these models. All timings were obtained on a 2 GHz Pentium IV PC with a GeForce 4 graphics card and 1 GB RAM.

| Model | Complexity | | Performance | | | |
|---|---|---|---|---|---|---|
| | Num of Edges | | # Surf | Grid Gen | Isosurface | Free Space Size |
| | Obstacle | Robot | size | (s) | (s) | size |
| Assembly | 224 | 224 | 256 | 6 | 1.8 | 22,928 |
| Maze | 30 | 24 | 1,550 | 12 | 1.68 | 21,402 |
| Gears | 36 | 72 | 3,929 | 212 | 3.2 | 66,389 |

TABLE I

*Performance: This table highlights the performance of our algorithm on different models. The model complexity is provided in terms of the number of edges of the polygonal objects. The table shows the number of contact surfaces, the time for grid generation, the time for isosurface extraction, and the size of the free space boundary.*

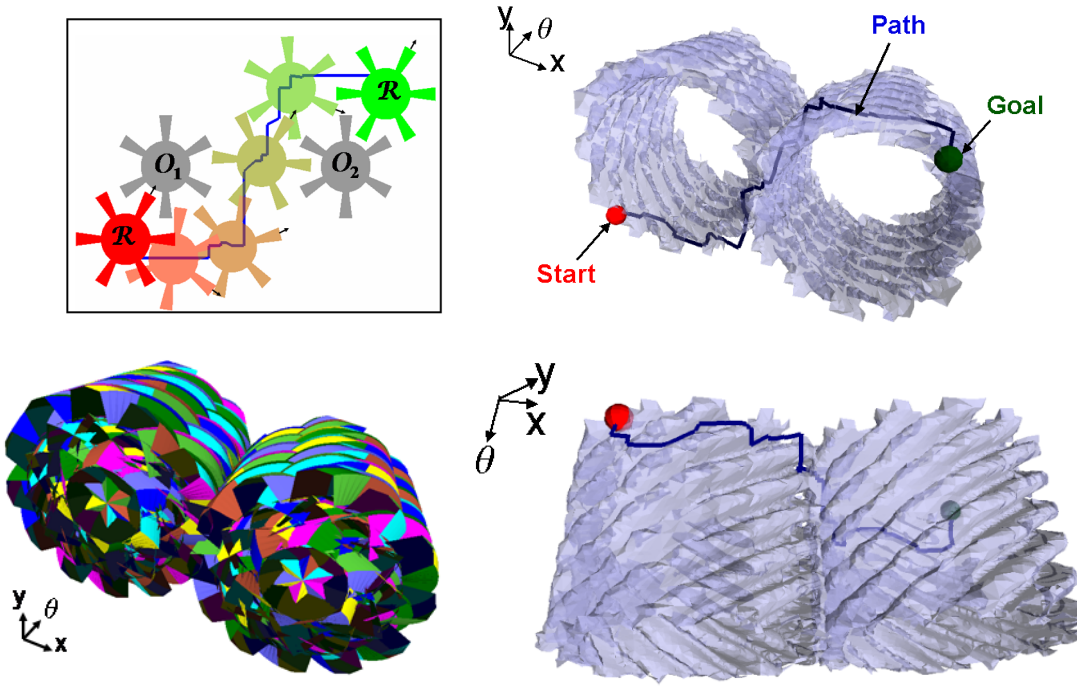



Fig. 4. **Configuration space of a planar robot capable of translation and rotation** *This figure highlights application of our free space approximation algorithm. The top left image shows a gear-shaped robot $\mathcal{R}$ navigating amongst two gear-shaped obstacles ($\mathcal{O}_1$ & $\mathcal{O}_2$) shown in gray. The start and goal configurations of the robot are shown in red and green respectively. Several intermediate configurations are also shown. The bottom left image is a color-coded image of the C-surfaces. The two images on the right show two views of our free space approximation (drawn translucently). The images also show a collision-free path that was computed using the star-shaped roadmap algorithm [27].*

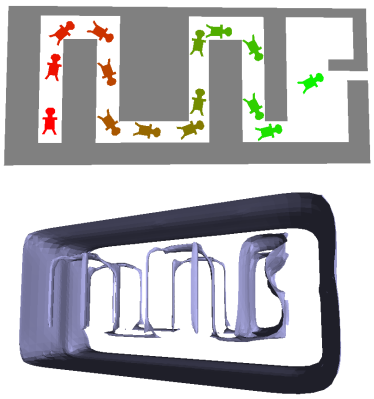


Fig. 5. *The top image shows a non-convex planar robot navigating through a maze. The robot is capable of both translation and rotation. The bottom image shows our approximation to the robot's free space.*

We demonstrate the application of our algorithm to two examples. Fig. 5 shows a robot navigating within a maze model. The free configuration space is defined in terms of $1,550$ contact surfaces. The figure shows a view of our free space approximation.

Fig. 4 shows another example of a gear-shaped robot navigating among two gear-shaped obstacles. We enumerated a set of $3,929$ contact surfaces. We computed a collision-free path for this example using the *star-shaped roadmap* algorithm [27], which is a deterministic sampling algorithm for complete motion planning. The images show the path superimposed onto the free space approximation.

Fig. 6 shows an application of our algorithm to 3D translational assembly planning. The figure shows our free space approximation.

Table I lists the time in computing the boundary of the free configuration space. The table also shows the complexity of the grid size and the boundary of free configuration space.

### A. Limitations

Our algorithm assumes that the free space does not have any tangential contacts. A tangential contact occurs when two $C$-surfaces touch each other at a point thus forming a narrow passage of width zero in the free space. This may occur in cases where the robot must touch an obstacle in order to pass through a narrow passage to reach the goal configuration. Dealing with such degenerate cases is a difficult problem.

A bottleneck in our approach is the large number of C-surfaces. Typically, our method enumerates $O(n^2)$ C-surfaces where $n$ is the number of features in the robot and the obstacles. Many of these C-surfaces lie within C-obstacle and do not contribute to the actual boundary of the free space, thus adding an unnecessary overhead to the algorithm. We can alleviate this problem by using better *culling* techniques to eliminate such C-surfaces [29].

Our algorithm is not specific to a fixed configuration space dimension. In theory, it is applicable to robots with arbitrary DOF. However, the theoretical complexity and the implementation complexity grow considerably with DOF.

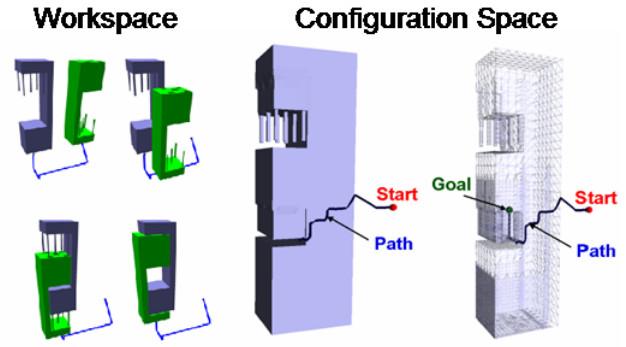



Fig. 6. *This example demonstrates the application of configuration space approximation algorithm to assembly planning. There are two parts each with pegs and holes. The goal is to assemble the two parts so that the pegs of one part fit into the holes of the other. We applied our free space approximation algorithm to this example. The images on the right show two views of the free configuration space approximation (one solid and one wireframe). The images also show a collision-free path computed using the star-shaped roadmap algorithm [27].*

## VII. Conclusions and Future Work

We have presented a practical algorithm for approximating the free configuration space of robots with translational and rotational degrees of freedom. Unlike previous methods, our algorithm avoids computing an arrangement of the C-surfaces. Instead, we use a grid-based method to compute a geometrically close and topologically correct approximation. The algorithm is simple to implement in practice.

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 higher degrees of freedom (e.g. 6-DOF) configuration space computation. It is possible to extend our current algorithm to approximate the configuration space of curved planar objects. The contact surface generation step of our algorithm will need to be extended: this could be done similarly to the approach presented in [19]. Finally, we would like to improve the adaptive subdivision algorithm by making our approach less conservative, thus improving the overall performance of our algorithm.

## Acknowledgment

This project was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel. Young J. Kim was supported in part by the grant R08-2004-000-10406-0 of KRF, the STAR program of MOST, the Ewha SMBA consortium and the ITRC program.

## References

[1] 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.

[2] Karen Daniels and Victor Milenkovic. Multiple translational containment: Approximate and exact algorithms. pages 205–214, 1995.

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

[4] 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.

[5] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects. *Internat. J. Robot. Autom.*, 4(2):193–203, 1988.

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

[7] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

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

[9] D. Halperin and M. Sharir. Almost tight upper bounds for the single cell and zone problems in three dimensions. 14:385–410, 1995.

[10] D. Halperin and M. Sharir. Arrangements and their applications in robotics: Recent developments. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 495–511. A. K. Peters, Wellesley, MA, 1995.

[11] C. Hoffmann. Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1:143–156, 2001.

[12] Leo Joskowitz and Elisha Sacks. HIPAIR: Interactive mechanism analysis and design using configuration spaces. In *ACM Symposium on Computational Geometry*, pages V5–V6, 1995.

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

[14] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[15] M. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003.

[16] 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.

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

[18] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[19] E. Sacks. Practical sliced configuration space for curved planar pairs. *International Journal of Robotics Research*, 18(1), 1999.

[20] E. Sacks. Deterministic path planning for planar assemblies. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2001.

[21] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice-Hall Inc, New Jersey, NJ, 1997.

[22] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL, 1997.

[23] U. Thomas, M. Barrenscheen, and F. Wahl. Efficient assembly sequence planning using stereographical projections of c-space obstacles. *Proc. of the 5th IEEE International Symposium on Assembly and Task Planning*, 2003.

[24] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Eurographics Symposium on Geometry Processing*, 2004.

[25] Gokul Varadhan. *Accurate Sampling Based Methods for Surface Extraction and Motion Planning*. 2005.

[26] Gokul Varadhan and Dinesh Manocha. Accurate minkowski sum approximation of polyhedral models. In *Pacific Conference on Computer Graphics and Applications*, pages 392–401, 2004.

[27] Gokul Varadhan and Dinesh Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[28] P. G. Xavier and R. A. LaFarge. A configuration space toolkit for automated spatial reasoning: Technical results and ldrd project final report. Technical Report SAND97-0366, Sandia Natioanl Laboratories, 1997.

[29] Liangjun Zhang, Young J. Kim, Gokul Varadhan, and Dinesh Manocha. Fast c-obstacle query computation for motion planning. In *IEEE International Conference on Robotics and Automation*, 2006.