Fast C-obstacle Query Computation for Motion Planning

Liangjun Zhang [1] Young J. Kim [2] Gokul Varadhan [1] Dinesh Manocha [1]

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

[2] *Dept. of Computer Science and Engineering, Ewha Womans University, Korea, kimy@ewha.ac.kr*

http://gamma.cs.unc.edu/cobstacle

***Abstract*— The configuration space of a robot is partitioned into free space and C-obstacle space. Most of the prior work in collision detection and motion planning algorithms is targeted towards checking whether a configuration or a 1D path lies in the free space. In this paper, we address the problem of checking whether a C-space primitive or a spatial cell lies completely inside C-obstacle space, without explicitly computing the boundary of C-obstacle. We refer to the problem as the *C-obstacle query*. We present a fast and conservative algorithm to perform this C-obstacle query. Our algorithm uses the notion of generalized penetration depth that takes into account both translational and rotational motion. We compute the generalized penetration depth for polyhedral objects and compare it with the extent of the motion that the polyhedral robot can undergo.**

Our approach is general and useful for designing practical algorithms for complete motion planning of rigid robots. We have integrated our query computation algorithm with star-shaped roadmaps [1] – a deterministic sampling approach for complete motion planning. We have applied our modified planning algorithm to planar robots undergoing translational and rotational motion in complex 2D environments. Our algorithm is able to perform the C-obstacle query in milliseconds and improves the performance of the complete motion planning algorithm.

I. Introduction

The concept of robot's *configuration space* is widely used for motion planning of rigid and articulated robots [2], [3]. The robot is represented as a configuration in the parameter space that encodes the robot's degree-of-freedom (DOF). The obstacles in the workspace map to the forbidden regions (or *C-obstacle space*) in the configuration space. The complement of C-obstacle space is called *free space*. The free space is the set of collision-free configurations, i.e. the robot does not collide with the obstacles. The goal of motion planning is to check whether there exists a path from the initial configuration to the final configuration that lies completely within the free space.

The complexity of computing the boundary of C-obstacle space is exponential in the number of $DOFs$. Furthermore, exact C-obstacle space computation is prone to floating point errors and degeneracies. As a result, it is hard to compute the boundary of C-obstacle space exactly, even for low-DOF robots. Instead of computing this boundary explicitly, many planning algorithms perform collision and proximity queries to check whether a configuration lies in the free space or not. This reduces to checking whether a specific placement of the robot in the workspace collides with any obstacle or not. The probabilistic roadmap planners (PRM) [4] also check whether a continuous 1D path segment connecting two free configurations lies in the free space.

In this paper, we address the problem of checking whether a configuration space primitive or a cell lies completely in C-obstacle space (i.e. *C-obstacle query*). Formally speaking, this query is defined as checking whether the following predicate P is true:

$$P(\mathcal{A}, \mathcal{B}, Q): \ \forall \mathbf{q} \in Q, \ \mathcal{A}(\mathbf{q}) \cap \mathcal{B} \neq \emptyset. \tag{1}$$

Here, $\mathcal{A}$ is a robot, $\mathcal{B}$ represents obstacles and Q is a C-space primitive or a cell; $\mathcal{A}(\mathbf{q})$ represents the placement of $\mathcal{A}$ at the configuration $\mathbf{q}$. Q may be a line segment, a cell or a contact surface that is generated from the boundary features of the robot and the obstacles.

The C-obstacle query is useful for cell decomposition based algorithms for motion planning [2]. These algorithms subdivide the configuration space into cells and need to check whether a cell is fully contained either in the free space or in C-obstacle space. The C-obstacle query also arises in sampling based approaches for motion planning, especially complete motion planning. These include the star-shaped roadmap algorithm [1], which is a deterministic sampling algorithm and subdivides the configuration space into a collection of cells in a hierarchical fashion. Given that the time and space complexity of these methods grows quickly with the level of subdivision, it is important to identify cells that lie in C-obstacle so that no further subdivision is performed on those cells.

Most prior work has focused on checking whether such a continuous primitive lies inside the free space, such as collision-free checking for a path segment [5], [6]. However, these techniques are not directly applicable to the C-obstacle query. Previous methods to perform C-obstacle query require enumerating contact surfaces and tend to be inefficient in practice [2]. In the workspace, C-obstacle query is equivalent to deciding whether the robot and the obstacles overlap at all the configurations that belong to the primitive or the cell. Conceptually, this is opposite to the problem of *Continuous Collision Detection* (CCD) for collision-free path checking [5], [6]. However, C-obstacle query computation is more difficult. Standard techniques based on bounding volume hierarchies cannot be directly used to check whether a cell or a primitive completely lies in the C-obstacle space. Moreover, the C-obstacle query is often used for multi-dimensional primitives,

such as cells or contact surfaces. On the other hand, prior work on CCD has been restricted to 1-dimensional paths.

A. Main Results

We present a fast C-obstacle query algorithm for rigid robots. Our algorithm efficiently checks whether a continuous C-space primitive, such as a cell or a contact surface, lies inside the C-obstacle space. Our algorithm uses the notion of *generalized penetration depth*, which takes into account object's translational as well as rotational motion to compute the extent of penetration. In order to perform C-obstacle cell query, we first compute a lower bound on the generalized penetration depth PD^g between a robot $\mathcal{A}$ and an obstacle $\mathcal{B}$. The lower bound on PD^g is calculated by decomposing both $\mathcal{A}$ and $\mathcal{B}$ into convex polytopes and computing all possible pairwise, convex PD^g between these polytopes and taking their maximum value. We extend Schwarzer *et al.*'s method [6] for a 1-dimensional path segment to a multi-dimensional set of paths to compute an upper bound on the motion trajectory of a moving robot. We compare the lower bound of PD^g with the upper bound on the motion trajectory to perform the C-obstacle query. Our approach is conservative and provides a sufficient condition for the query. In practice, our algorithm is fast and performs the query in a few milliseconds for 2D rigid robots. We have used our algorithm to accelerate the performance of star-shaped roadmap algorithm for complete motion planning.

B. Organization

The rest of the paper is organized as follows. In Section II, we briefly survey related work in this area. We introduce the notation and give an overview of our approach in Section III. In Section IV, we present our algorithm for computing a lower bound on PD^g and extend our algorithm to perform queries on contact surfaces in Section V. We highlight the performance of these algorithms and its application to star-shaped roadmaps in Section V.

II. Related Work

In this section, we briefly review the previous work on C-obstacle query, continuous collision detection (CCD) for local planning, penetration depth, and motion bound calculation.

A. C-obstacle Query

A C-obstacle query algorithm based on contact surface enumeration and convex decomposition has been described in [2]. A configuration cell lies entirely inside C-obstacle if it is contained by C-obstacle region formed by a pair of convex pieces from the robot and the obstacles respectively. One drawback of this method is that it enumerates all the contact surfaces for every convex pair to test the containment. Moreover, it is rather difficult to extend this approach to higher DOFs robots.

B. Continuous Collision Detection and Local Planning

The goal of continuous collision detection (CCD) is to determine whether all configurations along a continuous path are collision-free or not. Different types of approaches have been proposed for this purpose: algebraic solving approach [7], [8], swept volume-based approach [9], adaptive bisection approach [5], [6] and kinetic data structures approach [10], [11]. A typical application of CCD is local motion planning such as probabilistic roadmap approach (PRM) [4], where one needs to quickly determine whether a given path segment is collision-free or not [6].

C. Penetration Depth

Penetration Depth (PD) is a distance measure to describe the extent of inter-penetration between two overlapping objects. One can define PD differently depending on whether considering only translational motion or considering both translational and rotational motion in the measure. A classical definition of translational PD, PD^t, is defined as a minimum translational distance to make two objects disjoint. This definition can be formulated in terms of the *Minkowski sum* of two objects [12]–[16]. The generalized PD, PD^g takes into account both the translational and rotational motion [17].

D. Motion Bound Calculation

Schwarzer *et al.* [6] propose a method to bound a motion trajectory by calculating the maximal length of curve segments traced by all points on a moving robot with constant linear and angular velocities. This technique has been extended to bound the motion of an articulated robot, and the upper bound on the motion trajectory is obtained by taking the weighted sum of differences between all configuration parameters along the motion trajectory. Redon *et al.* [18] bound the motion trajectory of linear swept spheres (LSS) by using interval arithmetic and this bound has been used for dynamic collision checking between a moving avatar and the virtual environment.

III. C-obstacle Query Algorithm

In this section we introduce our notation and give an overview of our C-obstacle query algorithm.

A. Notations

We use the following notation throughout the rest of the paper. We define a rigid robot $\mathcal{A}$ moving among stationary rigid obstacles $\mathcal{B}_1, ..., \mathcal{B}_n$. For a rigid robot in 2D with translational and rotational $DOFs$, its C-space resides in $R^2 \times SO(2)$. A configuration $\mathbf{q}$ in this space represented by three independent configuration parameters: x, y representing its translational components, and ϕ representing its rotational angle. A line segment in C-space connecting configurations $\mathbf{q}_a$ and $\mathbf{q}_b$ is represented as $\pi_{\mathbf{q}_a,\mathbf{q}_b}$. A cell C in this C-space is defined as a Cartesian product of the following form (Figure 1):

$$C = [x_1, x_2] \times [y_1, y_2] \times [\phi_1, \phi_2].$$

We denote $\mathcal{A}(\mathbf{q})$ as a placement of a robot $\mathcal{A}$ at a configuration $\mathbf{q}$. Let $l(\mathbf{t})$ be an arbitrary curve in C-space. When $\mathcal{A}$ moves

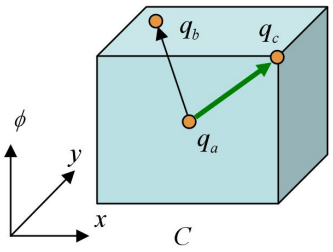

Fig. 1. *The upper bound of the motion of the robot when it is restricted to the cell C: C is a cell in $R^2 \times SO(2)$ with configuration parameters x, y, and ϕ. Initially, the robot is placed at $\mathbf{q}_a$, the center of the cell. The bounding motion of the robot, when it moves along the diagonal segment, i.e. from $\mathbf{q}_a$ to $\mathbf{q}_c$, is greater than or equal to the bounding motion moving along line segment from $\mathbf{q}_a$ to q_b, where q_b is an arbitrary point on the boundary of the cell.*

along l, any point $\mathbf{p}$ on $\mathcal{A}$ traces a distinct curve in workspace. Let $\mathbf{p}(\mathbf{q})$ as the position of the point $\mathbf{p}$ at the configuration $\mathbf{q}$. The length μ of the curve traced by the point $\mathbf{p}$ is:

$$\mu(\mathbf{p}, l) = \int ||\dot{\mathbf{p}}(l(t))||\, d(l(t)). \tag{2}$$

B. Motion Bound Calculation

The formulation of C-obstacle query requires an extent of the motion that the robot $\mathcal{A}$ can undergo in workspace, when it is restricted to a C-space primitive Q. If the underlying primitive is a 1-dimensional curve (e.g., l), Schwarzer *et al.* [6] define the bounding motion λ as:

$$\lambda(\mathcal{A}, l) = UpperBound(\mu(\mathbf{p}, l) \mid \mathbf{p} \in \mathcal{A}).$$

In particular, when the rigid robot $\mathcal{A}$ moves along a line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ in C-space, Schwarzer *et al.* compute a bounding motion λ as the weighted sum of the difference between $\mathbf{q}_a$ and $\mathbf{q}_b$ for each component x, y and ϕ:

$$\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = |\mathbf{q}_{b.x} - \mathbf{q}_{a.x}| + |\mathbf{q}_{b.y} - \mathbf{q}_{a.y}| + R_\phi \times |\mathbf{q}_{b.\phi} - \mathbf{q}_{a.\phi}|. \tag{3}$$

The weight R_ϕ is defined as the maximum Euclidean distance between every point $\mathbf{p}$ on $\mathcal{A}$ and the rotation center. This equation can be easily extended to a rigid robot in 3D with higher translational and rotational $DOFs$.

We define the bounding motion λ of the robot when it is restricted within a cell C instead of a curve:

$$\lambda(\mathcal{A}, C) = \max\{\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) \mid \mathbf{q}_b \in \partial C\}, \tag{4}$$

where $\mathbf{q}_a$ is the center of C, and $\mathbf{q}_b$ is any point on ∂C, the boundary of C (Figure 1).

Among all line segments $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, the diagonal line segments have the maximum difference on each configuration component. According to Eq. (3), the maximum of the bounding motion $\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b})$ is achieved by any diagonal line segment of the cell. Therefore, the bounding motion for the cell C is equivalent to the bounding motion over any diagonal line segment $\pi_{\mathbf{q}_a,\mathbf{q}_c}$:

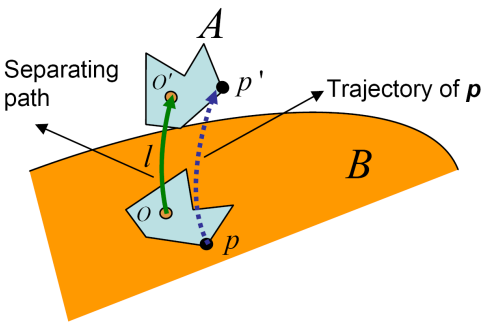

Fig. 2. *Notations of separating path and trajectory length for generalized penetration depth PD^g: The robot $\mathcal{A}$ with a reference point o initially intersects with the obstacle $\mathcal{B}$. The curve l is called a separating path, since when $\mathcal{A}$ moves along this path, it finally separates from $\mathcal{B}$. The length of the trajectory traced by any point $\mathbf{p}$ on $\mathcal{A}$ is the arc length of its moving trajectory pp'. Eq. (6) uses these two notations to define the generalized penetration depth PD^g.*

$$\lambda(\mathcal{A}, C) = \lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_c}), \tag{5}$$

where $\mathbf{q}_a$ is the center of the cell and $\mathbf{q}_c$ is any corner of the cell.

C. Generalized Penetration Depth

The formulation of our C-obstacle query also requires an extent of inter-penetration between the robot and the obstacle. The translational PD, PD^t, is often defined as a minimum translational distance to separate two overlapping objects.

$$PD^t(\mathcal{A}, \mathcal{B}) = \min(\{\| \mathbf{d} \| \mid interior(\mathcal{A} + \mathbf{d}) \cap \mathcal{B} = \emptyset\}).$$

However, this notion is not directly applicable to our method, because a rigid robot can have both translational and rotational $DOFs$.

We adopt the generalized penetration depth PD^g by [17], which takes both translational and rotational motion into account. The generalized PD can be defined using the notions of separating path and trajectory length. As Fig. 2 illustrates, a separating path l in C-space is such a curve that when a robot moves along l, the robot can be completely separated from the obstacle. The length of the trajectory of any point $\mathbf{p}$ on the robot $\mathcal{A}$ moving along l is defined by Eq. (2).

Given a set L of all possible candidates of separating paths, PD^g between a robot $\mathcal{A}$ and an obstacle $\mathcal{B}$ is defined as:

$$PD^g(\mathcal{A}, \mathcal{B}) = \min(\{\max(\{\mu(\mathbf{p}, l) | \mathbf{p} \in \mathcal{A}\}) | l \in L\}). \tag{6}$$

A useful property related to PD^g is as follows:

THEOREM 1 *For two convex polytopes $\mathcal{A}$ and $\mathcal{B}$, we have*

$$PD^g(\mathcal{A}, \mathcal{B}) = PD^t(\mathcal{A}, \mathcal{B}).$$

The proof of this result can be found in [17]. Furthermore, like PD^t, PD^g satisfies the property: $PD^g(\mathcal{A}, \mathcal{B}) = 0$ if and only if $\mathcal{A}$ and $\mathcal{B}$ are disjoint.

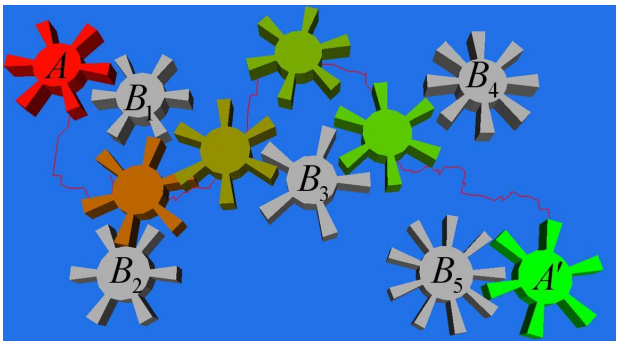

Fig. 3. *Gear example: This figure illustrates an application of our C-obstacle query algorithm to speedup a complete motion planner - the star-shaped roadmap algorithm. In this example, the object Gear needs to move from initial configuration A to goal configuration $\mathcal{A}'$ by translating and rotating within the shaded rectangular 2D region. There are five gear-like obstacles $B_i (i = 1, ..., 5)$ in this example, and there are narrow passages between B_1 and B_2, and B_4 and B_5, and no path existing in the passage between B_3 and B_5. We show the robot's intermediate configurations for the found path. Using our C-obstacle query, we can achieve about 2.5 times speed up for the star-shaped roadmap algorithm for this example.*

D. C-obstacle Cell Query Criterion

We now state a sufficient condition for C-obstacle cell query, i.e., to check whether $\mathcal{A}$ and $\mathcal{B}$ overlap at every configuration $\mathbf{q}$ in a cell C (Fig. 1).

Lemma 1: For a cell C with the center $\mathbf{q}_a$, the predicate $P(\mathcal{A}, \mathcal{B}, C)$ is true if:

$$PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B}) > \lambda(\mathcal{A}, C). \tag{7}$$

Proof: We want to prove that Eq. (7) implies that there is no free configuration on any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, where $\mathbf{q}_b$ is any configuration on the boundary of the cell C. According to the definition of PD^g, the maximum trajectory length for every point on the robot $\mathcal{A}$ moving along a possible separating path should be greater than or equal to $PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B})$. Moreover, according to Eq. (4), the trajectory length of the robot when it moves along $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ is less than or equal to $\lambda(\mathcal{A}, C)$. Since $PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B}) > \lambda(\mathcal{A}, C)$, the minimum motion required to separate the robot $\mathcal{A}$ from the obstacle $\mathcal{B}$ is larger than the maximum motion the robot $\mathcal{A}$ could make. Therefore, there is no free configuration along any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$.

Because there is no free configuration in every line segment between $\mathbf{q}_a$ to $\mathbf{q}_b$, this concludes that every configuration in the cell C lies inside C-obstacle space, and the predicate $P(\mathcal{A}, \mathcal{B}, C)$ holds. ■

We use $Lemma$ 1 to conservatively decide whether a given cell C lies inside C-obstacle space. The C-obstacle query algorithm includes two parts: computing a lower bound on PD^g for the robot $\mathcal{A}(\mathbf{q}_a)$ and the obstacle $\mathcal{B}$, which is presented in the next section, and computing an upper bound on motion: $\lambda(\mathcal{A}, C)$, which can be easily computed by Eqs. (5) and (3).

IV. Lower Bound on Generalized Penetration Depth

In this section, we present our algorithm for efficiently computing a lower bound on PD^g. The algorithm is based on the fact that PD^g is equal to PD^t for convex polytopes [17]. As a result, we can obtain a lower bound on PD^g by (1) decomposing non-convex models into convex pieces and (2) taking a maximum value of PD^gs between all pairwise combinations of convex pieces. More precisely, the algorithm can be described as follows:

1) As a preprocess, decompose a non-convex robot $\mathcal{A}$ and obstacle $\mathcal{B}$ into convex pieces; i.e., $\mathcal{A} = \cup\mathcal{A}_i$, where $i = 1, ..., M$ and $\mathcal{B} = \cup\mathcal{B}_j$, where $j = 1, ..., N$.
2) During run-time query, place $\mathcal{A}$ at configuration $\mathbf{q}$ to obtain $\mathcal{A}(\mathbf{q})$.
3) For each pair of $(\mathcal{A}_i(\mathbf{q}), \mathcal{B}_j)$,
   a) Perform collision detection to check for overlaps.
   b) If the pair overlaps, $PD^g_k = PD^t((\mathcal{A}_i(\mathbf{q}), \mathcal{B}_j)$, where $k = (i-1)N + j$; otherwise $PD^g_k = 0$.
4) Lower bound on $PD^g = \max(PD^g_k)$ for all k.

A. Translational Penetration Depth Computation

In our method, the lower bound on generalized PD^g computation is decoupled into a set of PD^t queries among convex pieces. The PD^t between two convex polytopes can be computed [15], [19], [20]. These methods compute PD^t by calculating a minimum distance from the origin to the surface of Minkowski sum between two convex polytopes.

Given that our C-obstacle query requires the lower bound on PD^g, this imposes that the PD^t computation used in our method should be exact or be a lower bound too. In particular, the method in [19] guarantees such a lower bound and [15] provides a even tighter lower bound based on an iterative method.

B. Convex Decomposition

In our method, the convex decomposition is performed as a preprocess. Our query algorithm also works when the decomposed convex pieces may overlap with each other. Moreover, our method also allows the union of decomposed convex pieces to cover only a proper subset of the initial model, i.e. $\cup\mathcal{A}_i \subset \mathcal{A}$. We use these properties to reduce the number of convex pieces.

C. Bounding Volume Hierarchy Acceleration

Our lower bound on generalized PD^g computation can be accelerated by standard bounding volume hierarchy-based collision detection technique. For two disjoint convex pieces, their PD^t is trivially set to 0. In practice, there are many disjoint pairwise combinations of convex pieces for two non-convex overlapping polytopes. Therefore, to accelerate our query algorithm, we detect such disjoint pairs and prune them away. We employ standard bounding volume hierarchy-based collision detection technique such as axis-aligned bounding box (AABB) or oriented bounding box (OBB) [21], to conservatively check whether the convex pieces are disjoint.

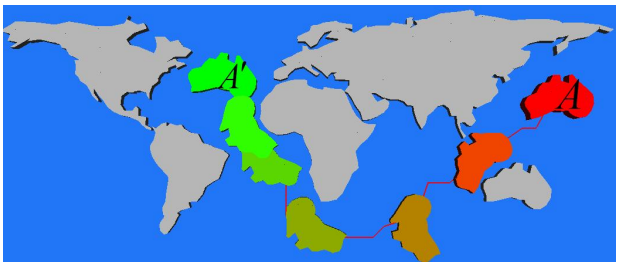

Fig. 4. *World Map Example: This figure shows an application of our C-obstacle query to speed up a complete motion planner - the star-shaped roadmap method. In this example, the object needs to move from initial configuration A to goal configuration A' by translating and rotating within the shaded rectangular 2D region. Our C-obstacle query can achieve about 2.0 times speedup for this example.*

V. Applications

In this section, we highlight several applications of our C-obstacle query algorithm. First, we apply our C-obstacle cell query to the star-shaped roadmap approach [1] to improve its performance. Next, we apply the query algorithm to a different C-space primitive – a contact surface. We highlight the performance of our query algorithm on both these applications.

A. C-obstacle Query for Star-Shaped Roadmaps

We have applied our C-obstacle query algorithm to accelerate the star-shaped roadmap algorithm [1], [22]. This algorithm computes a roadmap that captures the connectivity of free space and is able to perform complete motion planning. It constructs the roadmap by performing an adaptive subdivision in C-space. The adaptive subdivision method generates a volumetric grid in C-space such that every grid cell C satisfies the following property: the portion $\mathcal{F}_C$ of free space contained within C is star-shaped, i.e., there exists a point $\mathbf{o} \in \mathcal{F}_C$ such that $\mathbf{o}$ can "see" every point in $\mathcal{F}_C$. This star-shaped property of the grid cells is used to extract a roadmap of the free space that captures its connectivity.

One of the major performance bottlenecks in the original star-shaped roadmap arises from processing many cells that lie in the C-obstacle space and do not contribute to the final roadmap. For example, in the $Gear$ example (shown in Fig. 3), during C-space subdivision, about 70% cells that completely lie inside C-obstacle and handling these cells takes about 40% of the total time.

In order to accelerate the star-shaped roadmap construction, we apply the C-obstacle query algorithm to conservatively identify the cells that are inside C-obstacle and cull them away.

Another benefit of the C-obstacle query is to accelerate the checking of non-existence of any collision-free path. The star-shaped roadmap method determines that no path exists between the start and the goal configurations if the corresponding parts of the roadmap are disconnected, i.e., separated by C-obstacle. This computation is accelerated by using the C-obstacle query.

B. Contact Surface Culling

Contact surface or C-surface is defined to be the locus of configurations of a robot at which a specific feature of the robot is in contact with a feature of an obstacle. The boundary of free space can be extracted from an arrangement of contact surfaces.

Removing contact surfaces that do not contribute to the boundary of free space (i.e., contact surfaces lying inside C-obstacle) is an important step in terms of accelerating the process of constructing the boundary of free space [23]. Since the time complexity of these computations is a polynomial function of the number of contact surfaces in d dimension, effectively identifying and culling contact surfaces can significantly improve the performance of constructing the boundary of free space. In order to determine whether a contact surface lies inside the C-obstacle space, we first compute an axis aligned cell that bounds the given contact surface. If this cell lies inside C-obstacle, the associated C-surface lies completely inside C-obstacle, and can be culled away. This computation can be performed by using our C-obstacle cell query algorithm.

C. Experimental Results and Analysis

We have implemented the C-obstacle query algorithm for 2D robots with two translational and one rotational $DOFs$. For the PD^t computation, we use the Mink2D package [1]. We employ the OBB intersection test to accelerate the intersection test between two convex polygons.

Our query algorithms have been integrated into the star-shaped roadmap method. We test the enhanced planner on complex 2D benchmarks with translational and rotational $DOFs$. Figures 3, 4 and 5 illustrate the collision-free paths computed by our planner for the $Gear$, $World\ Map$ and $Piano$ models. In order to demonstrate the effectiveness our C-obstacle cell query, we define the *Cell Culling Ratio* as:

$$Cell\ Culling\ Ratio = \frac{number\ of\ culled\ C\text{-}obstacle\ cells}{number\ of\ C\text{-}obstacle\ cells}. \quad (8)$$

Table I illustrates that our C-obstacle query algorithm can achieve from 65% up to 80% Cell Culling Ratio in our benchmarks. Table I also shows the average time for each C-obstacle query. For the $Gear$ example, it takes about $0.12ms$ for a single query. The executed number of C-obstacle query is 111,313, and the total execution time for C-obstacle query is 13.30s (Table II).

Table II shows the performance speedup for the star-shaped roadmap method. We observe 2-3 times speedup in our benchmarks. The effectiveness of our contact surface culling is shown in Table I.

VI. Conclusion and Future Work

We have presented a fast C-obstacle query algorithm for rigid robots. The algorithm can check whether a cell or a contact surface lies inside the C-obstacle space. We have presented a novel technique to perform the query based on penetration depth computation and bounds on the motion trajectory. Our C-obstacle query algorithm is general and can be used with

[1] http://www.cs.tau.ac.il/~efif/collision_detection/

	Gear	Piano	World Map
Cell Culling Ratio	75.21%	67.21%	65.52%
Time Per Cell Culling	0.12ms	0.04ms	0.06ms
Surface Culling Ratio*	11.11%	20.20%	22.15%
Time for all C-surface Queries	3.28s	0.40s	11.27s

TABLE I

EFFECTIVENESS OF C-OBSTACLE CELL AND SURFACE QUERY: FOR 2D EXAMPLES, OUR QUERY CAN IDENTIFY ABOUT 65% TO 80% C-OBSTACLE CELLS. THE AVERAGE QUERY TIME VARIES FROM 0.04MS TO 0.12MS. *THE SURFACE CULLING RATIO IS DEFINED AS THE RATIO BETWEEN CULLED CONTACT SURFACES OVER ALL INPUT CONTACT SURFACES.

	Gear	Piano	World Map
Time of Original Method(s)	261.4	47.0	160.5
Time of Accelerated Method(s)	110.4	15.9	78.7
Speedup	2.4	2.9	2.0
Time for C-obstacle Cell Query(s)	13.3	0.8	1.8

TABLE II

PERFORMANCE: THE PERFORMANCE IMPROVEMENT IN THE STARSHAPED ROADMAP METHOD BY USING C-OBSTACLE CELL QUERY. FOR THE COMPLEX 2D EXAMPLE - *Gear*, OUR ALGORITHM CAN IMPROVE ITS PERFORMANCE BY 2.4 TIMES.

cell decomposition based planners. Moreover, our algorithm is easy to implement and efficient in practice. We have applied our C-obstacle query to accelerate the performance of the star-shaped roadmap algorithm for complete motion planning. Our experimental results show that by integrating our C-obstacle cell and contact surface query algorithms with this method, we can improve its performance.

There are several directions to pursue for future work. We are interested in further improving the effectiveness of our C-obstacle query algorithm. We would like to extend the algorithm to handle articulated robots. Finally, we would like to combine the C-obstacle query with probabilistic roadmap methods to design a hybrid planner that can handle high DOF robots.

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] G. Varadhan and D. Manocha, "Star-shaped roadmaps - a determinstic sampling approach for complete motion planning," in *Proc. of Robotics: Science and Systems*, 2005.

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

[3] H. C. et al., *Principles of Robot Motion*. The MIT Press, 2005.

[4] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, pp. 12(4):566–580, 1996.

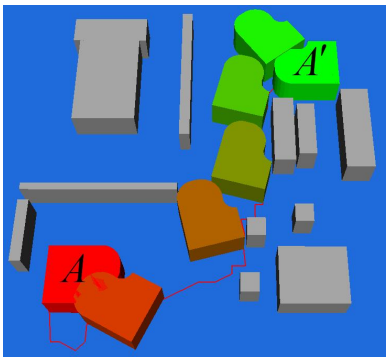

Fig. 5. *Piano example: our C-obstacle query algorithm can speedup the star-shaped roadmap method about 2.9 times.*

[5] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004.

[6] F. Schwarzer, M. Saha, and J. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Tr. on Robotics*, vol. 21, no. 3, pp. 338–353, June 2005.

[7] J. F. Canny, "Collision detection for moving polyhedra," *IEEE Trans. PAMI*, vol. 8, pp. 200–209, 1986.

[8] B. Kim and J. Rossignac, "Collision prediction for polyhedra under screw motions," in *ACM Conference on Solid Modeling and Applications*, June 2003.

[9] K. Abdel-Malekl, D. Blackmore, and K. Joy, "Swept volumes: Foundations, perspectives, and applications," *International Journal of Shape Modeling*, 2002.

[10] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, "Deformable free space tiling for kinetic collision detection," 2000, to appear.

[11] D. Kirkpatrick, J. Snoeyink, and B. Speckmann, "Kinetic collision detection for simple polygons," in *ACM Symposium on Computational Geometry*, 2000, pp. 322–330.

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

[13] P. Agarwal, L. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir, "Penetration depth of two convex polytopes in 3d," *Nordic J. of Computing*, vol. 7, no. 3, pp. 227–240, 2000.

[14] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst*, vol. SMC-15, pp. 224–233, 1985.

[15] G. van den Bergen, "Proximity queries and penetration depth computation on 3d game objects," *Game Developers Conference*, 2001.

[16] P. Agarwal, L. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir, "Penetration depth of two convex polytopes in 3d," *Nordic J. Computing*, vol. 7, pp. 227–240, 2000.

[17] L. Zhang, Y. Kim, G. Varadhan, and D. Manocha, "Generalized penetration depth computation," University of North Carolina at Chapel Hill, Tech. Rep., 2005.

[18] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Interactive and continuous collision detection for avatars in virtual environments," in *Proceedings of IEEE VR Conference*, 2004.

[19] S. Cameron, "Enhancing GJK: Computing minimum and penetration distance between convex polyhedra," *IEEE International Conference on Robotics and Automation*, pp. 3112–3117, 1997.

[20] Y. Kim, M. Lin, and D. Manocha, "Deep: Dual-space expansion for estimating penetration depth between convex polytopes," in *Proc. IEEE International Conference on Robotics and Automation*, May 2002.

[21] M. Lin and D. Manocha, "Collision and proximity queries," in *Handbook of Discrete and Computational Geometry*, 2003.

[22] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha, "A simple algorithm for complete motion planning of translating polyhedral robots," in *Workshop on the Algorithmic Foundations of Robotics*, 2004.

[23] G. Varadhan, Y. J. Kim, S. Krishnan, and D. Manocha, "Topology preserving approximation of free configuration space," in *Proc. IEEE International Conference on Robotics and Automation*, 2006.