Ray-Shooting Algorithms for Robotics

Yu Zheng and Katsu Yamane

Abstract Ray shooting is a well-studied problem in computer graphics. It also occurs in robotics as a collision detection problem in 3-D object space or a contact force optimization problem in 6-D wrench space. However, the ray-shooting algorithms derived in computer graphics are limited to 3-D polyhedra and not suited for general convex sets in high-dimensional space. This paper discusses several general ray-shooting algorithms and their applications to these problems in robotics.

1 Introduction

In computational geometry and computer graphics, the ray-shooting problem deals with computing the first intersection point on the surface of given objects by a query ray and has been well studied for over four decades [1, 8, 13, 14]. In these efforts, most of the practical algorithms are limited to 2-D or 3-D polytopes.

It has been discovered that several fundamental problems in robotics are equivalent to a ray-shooting problem. Ong and Gilbert [10] proposed a unified distance measure, called the growth distance, for both separated and penetrated objects and applied it to collision-free path planning [11]. The growth distance computation can be reduced to a ray-shooting problem. Liu et al. [6, 2, 7] cast several problems in grasping, such as grasping force optimization and force-closure grasp test and synthesis, into a ray-shooting problem. However, these ray-shooting problems deal with the intersection of a ray with a set, which is specified by nonlinear parametric functions combined with complex operations on sets, such as the Minkowski sum, rather than a 3-D polytope with given vertices or facets. Some of them are also in higher-dimensional space than 3-D. Therefore, the ray-shooting techniques in computational geometry and computer graphics are not suited here. The previous solutions to these problems relied on general-purpose optimization techniques

Yu Zheng, Katsu Yamane
Disney Research Pittsburgh, USA, e-mail: {yu.zheng, kyamane}@disneyresearch.com

[10, 6] and suffered the low computational efficiency. Recently, two procedures were proposed to more quickly solve such a ray-shooting problem [16, 18].

In this paper, we extensively discuss the algorithms for a general ray-shooting problem and their applications in robotics. The original contributions include:

- A better stopping criterion for the algorithm [16] to enhance its accuracy.
- Generalization of the algorithm [18] to any case with guaranteed convergence.
- Discussion on a new algorithm and hybrid uses of these algorithms to gain higher computational efficiency.
- Application of these ray-shooting algorithms to growth distance computation, in addition to contact force optimization in the previous work [16, 18].

The rest of this paper is organized as follows. Sect. 2 defines the ray-shooting problem and summarizes algorithms to be used in developing ray-shooting algorithms in Sect. 3. Sects. 4 and 5 show their applications with numerical examples. Conclusions and future work are included in Sect. 6. In the following discussion, we will use many convex geometry concepts, for which we refer readers to [4].

2 Problem Statement and Preliminaries

In this section, we give a mathematical definition of the ray-shooting problem and summarize two existing algorithms for minimum distances [3, 15].

2.1 Definition of the Ray-Shooting Problem

Let A be a compact convex set with nonempty interior in $\mathbb{R}^n$, $\mathbf{r}$ a nonzero vector in $\mathbb{R}^n$, and $R(\mathbf{r})$ the ray emanating from the origin $\mathbf{0}$ of $\mathbb{R}^n$ in the direction $\mathbf{r}$, i.e.,

$$R(\mathbf{r}) \triangleq \{\lambda \mathbf{r} \in \mathbb{R}^n \mid \lambda \geq 0\}. \tag{1}$$

The ray-shooting problem first needs to determine

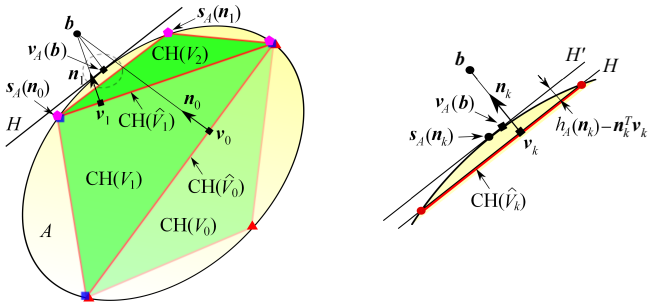
1. whether A and $R(\mathbf{r})$ intersect.

If A and $R(\mathbf{r})$ intersect, then it is also aimed at computing

2. the farthest intersection point $\mathbf{z}_A(\mathbf{r})$ of A with $R(\mathbf{r})$ from the origin $\mathbf{0}$;
3. a set of affinely independent points in A, denoted by $Z_A(\mathbf{r})$, such that $\mathbf{z}_A(\mathbf{r})$ can be written as a convex combination of $Z_A(\mathbf{r})$;
4. the normal $\mathbf{n}$ of the hyperplane that passes through $\mathbf{z}_A(\mathbf{r})$ and supports A.

In the algorithms discussed later to solve the ray-shooting problem, the support function h_A and the support mapping $\mathbf{s}_A$ of A are often used, which are defined by

$$h_A(\mathbf{u}) \triangleq \max_{\mathbf{a} \in A} \mathbf{u}^T \mathbf{a}, \qquad \mathbf{s}_A(\mathbf{u}) \triangleq \arg\max_{\mathbf{a} \in A} \mathbf{u}^T \mathbf{a}. \tag{2}$$



(a) (b)

Fig. 1 Illustration of the GJK algorithm in 2-D space. (a) Iteration that leads $\mathbf{v}_k$ to $\mathbf{v}_A(\mathbf{b})$. (b) Stopping criterion $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k < \varepsilon_{\text{GJK}}$. H and H' are hyperplanes with normal $\mathbf{n}_k$ passing $\mathbf{v}_k$ and $\mathbf{s}_A(\mathbf{n}_k)$, respectively, and bounds $\mathbf{v}_A(\mathbf{b})$ in the gap between them with the width of $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k$.

2.2 Summary of the GJK Algorithm [3]

Let $\mathbf{b}$ be an arbitrary point in $\mathbb{R}^n$. The purpose of this algorithm is to compute the distance $d_A(\mathbf{b})$ between $\mathbf{b}$ and A and the closest point $\mathbf{v}_A(\mathbf{b})$ in A to $\mathbf{b}$, defined by

$$d_A(\mathbf{b}) \triangleq \min_{\mathbf{a}\in A} \|\mathbf{a}-\mathbf{b}\|, \qquad \mathbf{v}_A(\mathbf{b}) \triangleq \arg\min_{\mathbf{a}\in A} \|\mathbf{a}-\mathbf{b}\|. \tag{3}$$

Since A is compact and convex, $\mathbf{v}_A(\mathbf{b})$ exists and is unique. If $\mathbf{b} \notin A$, then $d_A(\mathbf{b}) > 0$ and the hyperplane H with normal $\mathbf{b} - \mathbf{v}_A(\mathbf{b})$ passing through $\mathbf{v}_A(\mathbf{b})$ supports A at $\mathbf{v}_A(\mathbf{b})$, as depicted in Fig. 1a; otherwise $d_A(\mathbf{b}) = 0$ and $\mathbf{v}_A(\mathbf{b}) = \mathbf{b}$.

Both $d_A(\mathbf{b})$ and $\mathbf{v}_A(\mathbf{b})$ can be computed by the GJK algorithm, as illustrated in Fig. 1. It starts with any affinely independent set V_0 in A and iterates by $V_{k+1} = \hat{V}_k \cup \{\mathbf{s}_A(\mathbf{n}_k)\}$, where $\hat{V}_k$ is a minimal subset of V_k to represent $\mathbf{v}_k$ (the closest point in $\text{CH}(V_k)$ to $\mathbf{b}$) as its convex combination and $\mathbf{n}_k$ is the unit vector from $\mathbf{v}_k$ to $\mathbf{b}$. If $h_A(\mathbf{n}_k) > \mathbf{n}_k^T \mathbf{v}_k$, then $d_{\text{CH}(V_{k+1})}(\mathbf{b}) < d_{\text{CH}(V_k)}(\mathbf{b})$. Hence, $d_{\text{CH}(V_k)}(\mathbf{b})$ is strictly decreasing with the iteration and converges to $d_A(\mathbf{b})$ while $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k < \varepsilon_{\text{GJK}}$, where ε_{GJK} is the termination tolerance. Then, $\mathbf{v}_k$ converges to $\mathbf{v}_A(\mathbf{b})$ and $\hat{V}_k$ gives an affinely independent set in A, denoted by $V_A(\mathbf{b})$, to represent $\mathbf{v}_A(\mathbf{b})$ as its convex combination. If $\mathbf{b} \in A$, then $\mathbf{v}_A(\mathbf{b}) = \mathbf{b}$ and $\mathbf{b}$ is a convex combination of $V_A(\mathbf{b})$.

2.3 Summary of the ZC Algorithm [15]

Let $\text{CO}(A)$ denote the convex cone of A [4], which consists of all nonnegative combinations of A. This algorithm computes the distance $d_{\text{CO}(A)}(\mathbf{b})$ and the closest point $\mathbf{v}_{\text{CO}(A)}(\mathbf{b})$ between $\mathbf{b}$ and $\text{CO}(A)$, defined similarly to (3) with A replaced by $\text{CO}(A)$, as illustrated in Fig. 2. It starts with a linearly independent set V_0 in A and iterates by $V_{k+1} = \hat{V}_k \cup \{s_A(\mathbf{n}_k)\}$, where $\hat{V}_k$ is a minimal subset of V_k to represent $\mathbf{v}_k$ (the

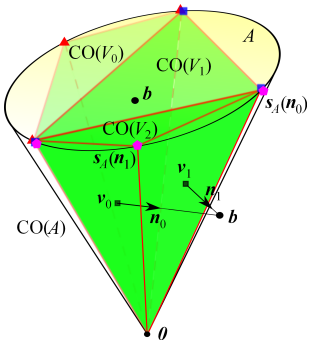


Fig. 2 Illustration of the ZC algorithm in 3-D space. If $\mathbf{b} \in \mathrm{CO}(A)$, then a linearly independent subset of A, namely V_1 here, is obtained to contain $\mathbf{b}$ in its convex cone.

closest point in $\mathrm{CO}(V_k)$ to $\mathbf{b}$) as its positive combination and $\mathbf{n}_k$ is the unit vector from $\mathbf{v}_k$ to $\mathbf{b}$. This iteration leads $d_{\mathrm{CO}(V_k)}(\mathbf{b})$ to $d_{\mathrm{CO}(A)}(\mathbf{b})$ and $\mathbf{v}_k$ to $\mathbf{v}_{\mathrm{CO}(A)}(\mathbf{b})$ while $h_A(\mathbf{n}_k) < \varepsilon_{\mathrm{ZC}}$, where $\varepsilon_{\mathrm{ZC}}$ is the termination tolerance. The final $\hat{V}_k$ provides a linearly independent set in A, denoted by $V_{\mathrm{CO}(A)}(\mathbf{b})$, to represent $\mathbf{v}_{\mathrm{CO}(A)}(\mathbf{b})$ as its positive combination. If $\mathbf{b} \in \mathrm{CO}(A)$, then $d_{\mathrm{CO}(A)}(\mathbf{b}) = 0$ and $\mathbf{v}_{\mathrm{CO}(A)}(\mathbf{b}) = \mathbf{b}$, and $\mathbf{b}$ is a positive combination of $V_{\mathrm{CO}(A)}(\mathbf{b})$.

3 Ray-Shooting Algorithms

In this section, we discuss procedures to solve the ray-shooting problem and their hybrid uses for higher efficiency. In general, each procedure generates a sequence of points on $R(\mathbf{r})$ approaching $\mathbf{z}_A(\mathbf{r})$. Hence, the convergence of these procedures can be easily proved by using the monotone-convergence principle [12].

3.1 A GJK-Based Procedure

A procedure based on the GJK algorithm to solve the ray-shooting problem was proposed in [14, 16, 17]. It computes $\mathbf{z}_A(\mathbf{r})$ by iterating a point $\mathbf{b}_k$ on $R(\mathbf{r})$ outside of A, as depicted in Fig. 3a. Initially, $\mathbf{b}_0$ can be taken to be $h_A(\mathbf{r})\mathbf{r}/\mathbf{r}^T\mathbf{r}$, which is the intersection point of the supporting hyperplane $H_0 = H(\mathbf{r}, \mathbf{s}_A(\mathbf{r}))$ of A with $R(\mathbf{r})$ and not in the interior of A. Then, we update the point $\mathbf{b}_k$ by

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \frac{d_A(\mathbf{b}_k)}{\mathbf{r}^T\mathbf{n}_k}\mathbf{r} \tag{4}$$

where $d_A(\mathbf{b}_k)$ and $\mathbf{v}_A(\mathbf{b}_k)$ are the minimum distance and the closest point from A to $\mathbf{b}_k$, respectively, which are computed by the GJK algorithm, and $\mathbf{n}_k$ is the unit vector

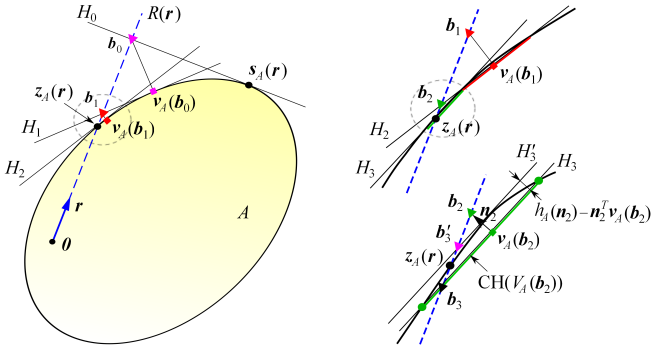


Fig. 3 Illustration of the GJK-based ray-shooting algorithm in 2-D space. (a) Iteration. $\mathbf{b}_0$ is the intersection point of H_0 with $R(\mathbf{r})$. H_1 is the hyperplane with normal $\mathbf{b}_0 - \mathbf{v}_A(\mathbf{b}_0)$ passing through $\mathbf{v}_A(\mathbf{b}_0)$ and intersects $R(\mathbf{r})$ at $\mathbf{b}_1$, which is closer to $\mathbf{z}_A(\mathbf{r})$ than $\mathbf{b}_0$. (b) Stopping criterion. Upper: Although $d_A(\mathbf{b}_1)$ is small, this does not guarantee that $\mathbf{b}_1$ is close enough to and can be taken to be $\mathbf{z}_A(\mathbf{r})$. Lower: A better stopping criterion is that $R(\mathbf{r})$ intersects $\mathrm{CH}(V_A(\mathbf{b}_2))$ (the green line segment). Then, their intersection point $\mathbf{b}_3$ can be adopted as $\mathbf{z}_A(\mathbf{r})$ and $V_A(\mathbf{b}_2)$ as $Z_A(\mathbf{r})$.

from $\mathbf{v}_A(\mathbf{b}_k)$ to $\mathbf{b}_k$. The iteration (4) can be terminated with the conclusion that $R(\mathbf{r})$ does not intersect A if $\mathbf{r}^T\mathbf{b}_k < \max\{-h_A(-\mathbf{r}),0\}$, or $\mathbf{r}^T\mathbf{n}_k = 0$ while $d_A(\mathbf{b}_k) \neq 0$, or $\mathbf{r}^T\mathbf{n}_k < 0$ [17]. Or else, $d_A(\mathbf{b}_k)$ will decrease to zero and $\mathbf{b}_k$ will approach $\mathbf{z}_A(\mathbf{r})$.

Instead of the stopping criterion $d_A(\mathbf{b}_k) < \varepsilon$ used in [14, 16, 17], we verify whether $R(\mathbf{r})$ intersects $\mathrm{CH}(V_A(\mathbf{b}_k))$ or $\mathbf{r}$ is a nonnegative combination of $V_A(\mathbf{b}_k)$, as explained in Fig. 3b. If so, we can compute their intersection point, which is equal to $\mathbf{b}_{k+1}$ from (4) and contained in A, since $\mathrm{CH}(V_A(\mathbf{b}_k))$ is contained in the hyperplane $H_{k+1} = H(\mathbf{n}_k, \mathbf{v}_A(\mathbf{b}_k))$. The hyperplane $H'_{k+1} = H(\mathbf{n}_k, \mathbf{s}_A(\mathbf{n}_k))$ supports A and intersects $R(\mathbf{r})$ at the point $\mathbf{b}'_{k+1} = h_A(\mathbf{n}_k)\mathbf{r}/\mathbf{r}^T\mathbf{n}_k$, which is outside of A. It is evident that $\mathbf{z}_A(\mathbf{r})$ is between $\mathbf{b}'_{k+1}$ and $\mathbf{b}_{k+1}$. The GJK algorithm stops iterating when $h_A(\mathbf{n}_k) - \mathbf{n}_k^T\mathbf{v}_A(\mathbf{b}_k) < \varepsilon_{\mathrm{GJK}}$, which implies $\mathbf{n}_k^T(\mathbf{b}'_{k+1} - \mathbf{b}_{k+1}) < \varepsilon_{\mathrm{GJK}}$. From this we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_{k+1}\| \leq \|\mathbf{b}'_{k+1} - \mathbf{b}_{k+1}\| < \varepsilon_{\mathrm{GJK}}\|\mathbf{r}\|/\mathbf{r}^T\mathbf{n}_k$. Therefore, we take $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_{k+1}$ and $Z_A(\mathbf{r}) = V_A(\mathbf{b}_k)$, as depicted in the lower figure of Fig. 3b.

3.2 An Internal Expanding (IE) Procedure

A preliminary version of this procedure was proposed in [18]. However, its convergence was unclear in some rare singular situations. Here we solve this issue and present a generalized version with guaranteed convergence.

Assume that we have a facet F_0 in A such that $R(\mathbf{r})$ passes through its interior. Let $\mathfrak{F}_k$ be a collection of facets in A and initially $\mathfrak{F}_0 = \{F_0\}$. In the following iteration, $\mathfrak{F}_k$ may comprise more than one facet, but $R(\mathbf{r})$ intersects every facet in $\mathfrak{F}_k$ at the same point, which is on the face $\mathrm{CH}(S_k)$ shared by those facets and denoted by $\mathbf{b}_k$,

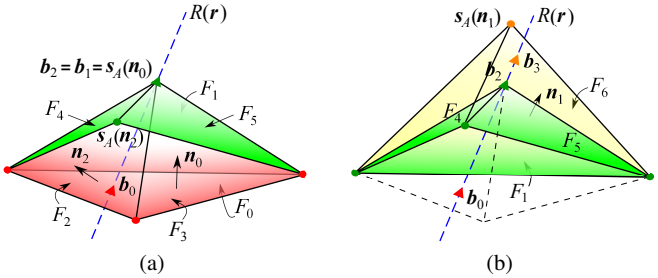
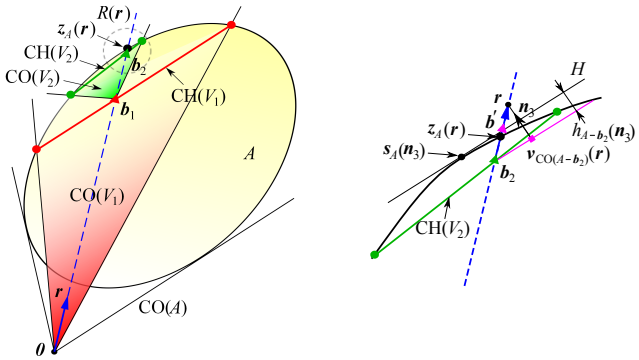


Fig. 4 Illustration of the IE ray-shooting algorithm in 3-D space. (a) $\mathfrak{F}_0 = \{F_0\}$ and $R(\mathbf{r})$ passes through an interior point $\mathbf{b}_0$ of F_0. Then, $\mathfrak{F}_0^+ = \mathfrak{F}_0$. Assume that $R(\mathbf{r})$ happens to pass $\mathbf{s}_A(\mathbf{n}_0)$. Then, $\mathfrak{F}_1 = \{F_1, F_2, F_3\}$. Suppose that F_2 is the facet in $\mathfrak{F}_1$ closest to the origin and $\mathbf{s}_A(\mathbf{n}_2)$ lies on the different side of F_2 or F_3 from the origin. Then, $\mathfrak{F}_1^+ = \{F_2, F_3\}$ and F_4 and F_5 are generated to replace F_2 and F_3. As a result, $\mathfrak{F}_2 = \{F_1, F_4, F_5\}$. (b) Suppose that F_1 is the facet in $\mathfrak{F}_2$ closest to the origin and $\mathbf{s}_A(\mathbf{n}_1)$ lies on the different side of any facet in $\mathfrak{F}_2$ from the origin. Then, $R(\mathbf{r})$ intersects a new face, i.e., F_6, at a point $\mathbf{b}_3$ closer to $\mathbf{z}_A(\mathbf{r})$ than $\mathbf{b}_2$. As $\mathbf{b}_3$ is in the interior of F_6, $\mathfrak{F}_3 = \{F_6\}$.

where S_k is the set of vertices of the face. Let $\mathbf{n}_i$ be the unit normal to each facet in $\mathfrak{F}_k$ such that $\mathbf{n}_i^T \mathbf{r} > 0$. Then $\mathbf{n}_i^T \mathbf{b}_k$ is the distance from the origin to facet i in $\mathfrak{F}_k$. We find facet i^* in $\mathfrak{F}_k$ whose distance from the origin is the minimum and compute $h_A(\mathbf{n}_{i^*})$ and $\mathbf{s}_A(\mathbf{n}_{i^*})$. If $h_A(\mathbf{n}_{i^*}) - \mathbf{n}_{i^*}^T \mathbf{b}_k > \varepsilon$, we extract a sub-collection $\mathfrak{F}_k^+$ from $\mathfrak{F}_k$ such that $\mathbf{n}_i^T \mathbf{s}_A(\mathbf{n}_{i^*}) > \mathbf{n}_i^T \mathbf{b}_k$ for any facet in $\mathfrak{F}_k^+$, which implies that $\mathbf{s}_A(\mathbf{n}_{i^*})$ lies on the different side of the facet from the origin. Let $\mathfrak{R}$ be the collection of facets of all facets in $\mathfrak{F}_k^+$, which are ridges in $\mathbb{R}^n$, and $\mathfrak{R}_i$ the collection of facets of a facet in $\mathfrak{F}_k^+$ that contain $\mathrm{CH}(S_k)$. Then, we will meet two situations, as depicted in Fig. 4a:

1. $\mathfrak{F}_k^+$ is equal to $\mathfrak{F}_k$: In this case, $\mathbf{s}_A(\mathbf{n}_{i^*})$ lies on the different side of every facet in $\mathfrak{F}_k$ from the origin. Then, $R(\mathbf{r})$ intersects one of the facets formed by each ridge in $\mathfrak{R} \setminus \cup_i \mathfrak{R}_i$ with the point $\mathbf{s}_A(\mathbf{n}_{i^*})$ and the intersection point, which is assigned to $\mathbf{b}_{k+1}$, is farther from the origin than $\mathbf{b}_k$. Hence, we reconstruct the facet collection $\mathfrak{F}_{k+1}$ with the facets intersected by $R(\mathbf{r})$ and the ridge collections $\mathfrak{R}_i$ for each facet in $\mathfrak{F}_{k+1}$ accordingly. Note that $R(\mathbf{r})$ may pass through a common face of some of the newly formed facets. In that case, $\mathfrak{F}_{k+1}$ consists of all such facets and S_{k+1} consists of vertices of the face.
2. $\mathfrak{F}_k^+$ is a proper sub-collection of $\mathfrak{F}_k$: In this case, we construct $\mathfrak{F}_{k+1}$ as follows. First, $\mathfrak{F}_{k+1}$ contains $\mathfrak{F}_k \setminus \mathfrak{F}_k^+$. Besides, for each ridge in $\mathfrak{R}_i$ that is not contained in other $\mathfrak{R}_i$ or shared by other facets in $\mathfrak{F}_k^+$, we add the facet formed by the ridge with the point $\mathbf{s}_A(\mathbf{n}_{i^*})$ to $\mathfrak{F}_{k+1}$. After doing this for every ridge collection $\mathfrak{R}_i$, we obtain a new facet collection $\mathfrak{F}_{k+1}$. However, $\mathbf{b}_{k+1}$ and S_{k+1} remain the same as $\mathbf{b}_k$ and S_k, respectively.

By the iteration in case 2, though $\mathbf{b}_k$ and S_k do not change, the minimum distance from the origin to the facets in $\mathfrak{F}_k$ is increased. On the other hand, the minimum distance is bounded above by the distance from the origin to the common face shared by the facets in $\mathfrak{F}_k$. Hence, by repeating this iteration, case 1 will become true, and $\mathbf{b}_k$ will be updated with a new point closer to $\mathbf{z}_A(\mathbf{r})$ (see Fig. 4b). As $\mathbf{b}_k$ approaches $\mathbf{z}_A(\mathbf{r})$, the stopping condition $h_A(\mathbf{n}_{i^*}) - \mathbf{n}_{i^*}^T \mathbf{b}_k < \varepsilon$ will reach, where i^* indicates the



(a) (b)

Fig. 5 Illustration of the ZC-based algorithm for ray-shooting in 2-D space. (a) It computes a subset V_k of A in every iteration such that $\mathbf{r}$ is a positive combination of $V_k - \mathbf{b}_{k-1}$. Then $\mathbf{b}_k$ is the intersection point of $\mathrm{CH}(V_k)$ with $R(\mathbf{r})$ and approaches $\mathbf{z}_A(\mathbf{r})$ as the iteration proceeds. (b) The iteration can be stopped by $h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\mathrm{ZC}}$, where $\mathbf{n}_{k+1}$ is the unit vector from $\mathbf{v}_{\mathrm{CO}(A-\mathbf{b}_k)}(\mathbf{r})$ to $\mathbf{r}$. Then the point $\mathbf{b}_k$ and the hyperplane H with normal $\mathbf{n}_{k+1}$ passing through $\mathbf{s}_A(\mathbf{n}_{k+1})$ bounds $\mathbf{z}_A(\mathbf{r})$ between them, which implies that $\mathbf{b}_k$ is close enough to $\mathbf{z}_A(\mathbf{r})$.

facet in $\mathfrak{F}_k$ closest to the origin. Then, we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_k\| < \varepsilon\|\mathbf{r}\|/\mathbf{n}_{i^*}^T\mathbf{r}$, and the hyperplane $H(\mathbf{n}_{i^*}, \mathbf{b}_k)$ supports A at $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_k$. This procedure can handle the situation that $R(\mathbf{r})$ passes through a face of dimension lower than $n-1$ with ensured convergence and generalizes the algorithm proposed in [18].

3.3 A ZC-Based Procedure

We propose a novel procedure based on the ZC algorithm, which iterates a point in A towards $\mathbf{z}_A(\mathbf{r})$. We first let $\mathbf{b}_0 = \mathbf{0}$ and compute $d_{\mathrm{CO}(A)}(\mathbf{r})$ using the ZC algorithm. Then, $R(\mathbf{r})$ intersects A if and only if $\mathbf{r} \in \mathrm{CO}(A)$ or equivalently $d_{\mathrm{CO}(A)}(\mathbf{r}) = 0$.

If $R(\mathbf{r})$ intersects A, the ZC algorithm gives $d_{\mathrm{CO}(A-\mathbf{b}_k)}(\mathbf{r}) = 0$ and a set of linearly independent points in $A - \mathbf{b}_k$, denoted by $\mathbf{a}_1 - \mathbf{b}_k, \mathbf{a}_2 - \mathbf{b}_k, \ldots, \mathbf{a}_L - \mathbf{b}_k$, such that

$$\mathbf{r} = \sum_{l=1}^{L} c_l(\mathbf{a}_l - \mathbf{b}_k) \tag{5}$$

where $\mathbf{a}_l \in A$ and $c_l > 0$ for all l. Let $V_k = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_L\}$, $\sigma = \sum_{l=1}^{L} c_l$, and

$$\mathbf{b}_{k+1} = \mathbf{b}_k + \frac{1}{\sigma}\mathbf{r} = \sum_{l=1}^{L} \frac{c_l}{\sigma}\mathbf{a}_l. \tag{6}$$

From (6) it follows that $\mathbf{b}_{k+1}$ is a point on $R(\mathbf{r})$ and it is also a convex combination of V_k, which implies that $\mathbf{b}_{k+1} \in A \cap R(\mathbf{r})$, as depicted in Fig. 5a. Since $\sigma > 0$, $\mathbf{b}_{k+1}$

is strictly farther from the origin and strictly closer to $\mathbf{z}_A(\mathbf{r})$ than $\mathbf{b}_k$. By repeating this iteration, $\mathbf{b}_{k+1}$ will converge to $\mathbf{z}_A(\mathbf{r})$ and V_k will become $Z_A(\mathbf{r})$.

As $\mathbf{b}_k$ approaches $\mathbf{z}_A(\mathbf{r})$ and the boundary of A, the ZC algorithm in a certain iteration will terminate with $h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\text{ZC}}$ and $d_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r}) > 0$, where $\mathbf{n}_{k+1}$ is the unit vector from $\mathbf{v}_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r})$ to $\mathbf{r}$, as depicted in Fig. 5b. The hyperplane $H(\mathbf{n}_{k+1}, \mathbf{s}_A(\mathbf{n}_{k+1}))$ supports A and intersects $R(\mathbf{r})$ at $\mathbf{b}' = h_A(\mathbf{n}_{k+1})\mathbf{r}/\mathbf{r}^T\mathbf{n}_{k+1}$. Then, $\mathbf{b}'$ and $\mathbf{b}_k$ bound $\mathbf{z}_A(\mathbf{r})$ between them and $\mathbf{n}_{k+1}^T(\mathbf{b}' - \mathbf{b}_k) = h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\text{ZC}}$. Therefore, we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_k\| \leq \|\mathbf{b}' - \mathbf{b}_k\| < \varepsilon_{\text{ZC}}\|\mathbf{r}\|/\mathbf{r}^T\mathbf{n}_{k+1}$. Furthermore, $H(\mathbf{n}_{k+1}, \mathbf{b}_k)$ can be regarded as a hyperplane of support to A at $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_k$.

3.4 Hybrid Uses

3.4.1 A Bi-GJK Procedure

From Sect. 3.3, we know that whether $R(\mathbf{r})$ intersects A can be easily determined by the ZC algorithm and, if so, a point in $A \cap R(\mathbf{r})$ can be obtained, as depicted by $\mathbf{b}_1$ in Fig. 5a, which we denote by $\hat{\mathbf{b}}$ here. On the other hand, $\check{\mathbf{b}} = h_A(\mathbf{r})\mathbf{r}/\mathbf{r}^T\mathbf{r}$ is a point on $R(\mathbf{r})$ outside the interior of A, as depicted by $\mathbf{b}_0$ in Fig. 3a. Apparently, $\mathbf{z}_A(\mathbf{r})$ lies on the line segment between $\hat{\mathbf{b}}$ and $\check{\mathbf{b}}$, which allows us to use the bisection method to compute $\mathbf{z}_A(\mathbf{r})$. We call the GJK algorithm to compute the minimum distance $d_A(\mathbf{b})$ between the midpoint $\mathbf{b} = (\check{\mathbf{b}} + \hat{\mathbf{b}})/2$ and A. If $d_A(\mathbf{b}) > 0$, i.e., $\mathbf{b} \notin A$, then we can update $\check{\mathbf{b}}$ with the point from (4) by setting $\mathbf{b}_k = \mathbf{b}$ and $\mathbf{n}_k = (\mathbf{b} - \mathbf{v}_A(\mathbf{b}))/\|\mathbf{b} - \mathbf{v}_A(\mathbf{b})\|$, which is a point on $R(\mathbf{r})$ closer to $\mathbf{z}_A(\mathbf{r})$ than $\check{\mathbf{b}}$ but still outside of A. Otherwise, we can replace $\hat{\mathbf{b}}$ by $\mathbf{b}$. This procedure can stop as the GJK-based procedure in Sect. 3.1, as shown in Fig. 3b, or once $\|\check{\mathbf{b}} - \hat{\mathbf{b}}\| < \varepsilon$.

3.4.2 A ZC-IE Procedure

The IE procedure can be much faster than the other two procedures if $R(\mathbf{r})$ always passes through the interior of a new facet in every iteration. In that case, the only computation in an iteration is to determine the facet intersected by $R(\mathbf{r})$ among n new facets and record its vertices. By contrast, the GJK-based or ZC-based procedure needs to run the GJK or ZC algorithm in every iteration, whose computation cost is much higher. However, there is no guarantee that $R(\mathbf{r})$ can always pass through the interior of a facet. In case that $R(\mathbf{r})$ happens to pass through only a low-dimensional face shared by several facets, we have to compute and maintain a facet collection and a ridge collection for each facet as discussed in Sect. 3.2, which increases the computation cost of every iteration. To keep a relatively lower computation cost, instead we can call the ZC-based iteration as discussed in Sect. 3.3. If V_k resulting from the ZC algorithm consists of n points, which implies that $\text{CH}(V_k)$ is a facet in A and $R(\mathbf{r})$ passes through its interior, then we can switch back to the IE procedure; otherwise, we continue the ZC-based iteration.

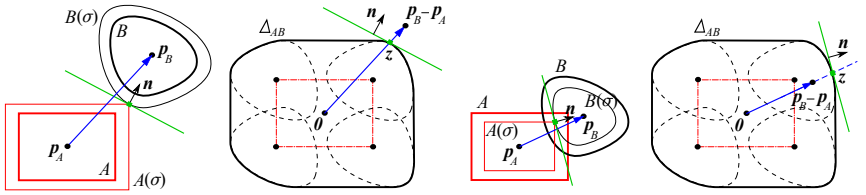


Fig. 6 Illustration of reducing the growth distance between two (a) separated or (b) penetrated convex sets A and B to a ray-shooting problem in 2-D space.

4 Application to Collision Detection

Determining the status (i.e., separation, contact, or penetration) between two objects is a fundamental problem in robotics and other fields, such as computer animation and haptics. To do this, a natural way is to compute the minimum Euclidean distance between them [3, 5], but the distance computation for penetrated objects is difficult due to the existence of many local minima. To overcome this trouble and have a unified distance measure for both separated and penetrated objects, the growth distance was proposed [10]. So far, however, there is no efficient algorithm to compute it, which significantly impeded its application. In this section, we show that the computation of growth distance can be reduced to a ray-shooting problem and present numerical results obtain using the aforementioned algorithms.

4.1 Computation of the Growth Distance

Assume that A is a compact convex set with nonempty interior in $\mathbb{R}^n$, which represents an object and can be specified by triangles meshes or parametric functions, as long as its support function and mapping can be computed. The object A in the global coordinate frame can be expressed as

$$A = \mathbf{p}_A + \mathbf{R}_A(A_0) \tag{7}$$

where $\mathbf{p}_A \in \mathbb{R}^n$ is an interior point of A indicating its position with respect to the global coordinate frame and $\mathbf{R}_A \in SO(n)$ denotes the orientation of A, and A_0 is the description of the object in the local coordinate frame attached at $\mathbf{p}_A$.

As depicted in Fig. 6, the growth model of A is defined by

$$A(\sigma) \triangleq \mathbf{p}_A + \sigma\mathbf{R}_A(A_0) \tag{8}$$

where $\sigma \geq 0$. Let B be another object and $B(\sigma)$ its growth model defined in the same way as $A(\sigma)$. The growth function of the object pair (A,B) is defined by [10]

$$g(A,B) \triangleq \sigma^* = \min_{A(\sigma)\cap B(\sigma)\neq\emptyset,\ \sigma\geq 0} \sigma. \tag{9}$$

The growth function (9) computes the minimum scale factor σ^* such that $A(\sigma^*)$ and $B(\sigma^*)$ are not strictly separated from each other, which implies that $A(\sigma^*)$ just contacts $B(\sigma^*)$, as shown in Fig. 6. Then, we can deduce that A and B separate if $\sigma^* > 1$, A and B contact if $\sigma^* = 1$, and A and B penetrate if $\sigma^* < 1$. Since $A(\sigma) \cap B(\sigma) \neq \emptyset$ is equivalent to $\mathbf{0} \in A(\sigma) - B(\sigma)$, from (8) we can rewrite (9) as

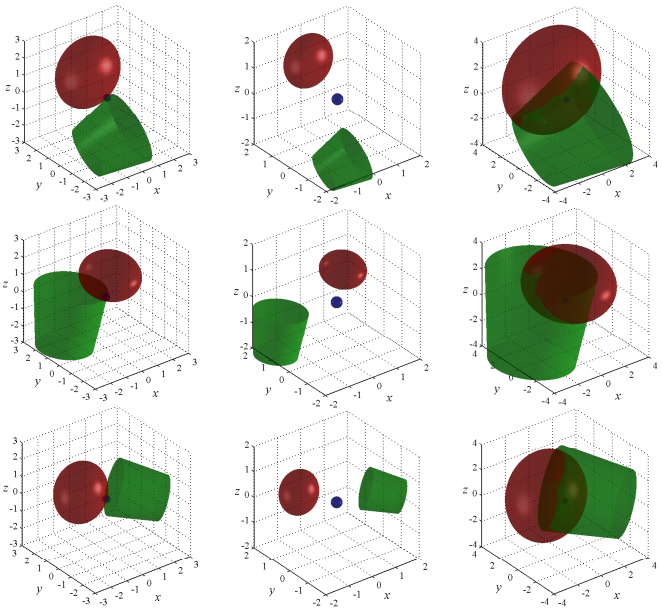
$$g(A,B) = \min_{\frac{1}{\sigma}(\mathbf{p}_B-\mathbf{p}_A)\in\Delta_{AB},\ \sigma\geq 0} \sigma \tag{10}$$

where $\Delta_{AB} = \mathbf{R}_A(A_0) - \mathbf{R}_B(B_0)$. Equation (10) implies that the calculation of $g(A,B)$ is a ray-shooting problem between Δ_{AB} and $R(\mathbf{p}_B - \mathbf{p}_A)$, as depicted in Fig. 6. It can be deduced that $g(A,B) = \|\mathbf{p}_B - \mathbf{p}_A\| / \|\mathbf{z}_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)\|$. Furthermore, from the set $Z_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)$ and the normal $\mathbf{n}$ of the supporting hyperplane of Δ_{AB} at $\mathbf{z}_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)$, we can derive the contact point between $A(\sigma^*)$ and $B(\sigma^*)$ and the normal at the contact. Due to the page limit, we omit this derivation here.

4.2 Numerical Examples

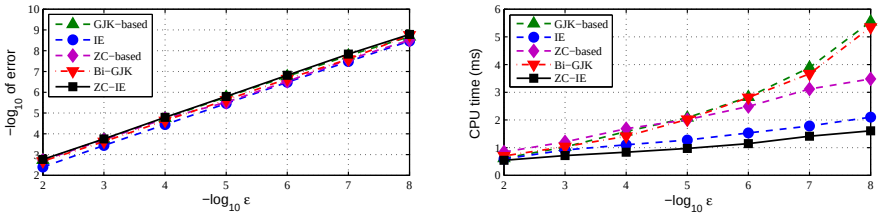
We implement the aforementioned ray-shooting algorithms in MATLAB on a laptop with an Intel Core i7 2.67GHz CPU and 3GB RAM and apply them to computing $g(A,B)$ between an ellipsoid and a truncated cone, as depicted in Fig. 7. The surface of an ellipsoid or truncated cone is specified by parametric functions, but their sizes and relative positions and orientations are randomly generated. First, we set the two objects to be in contact with each other (Fig. 7a) and shrink (Fig. 7b) or enlarge (Fig. 7c) them by 2 times such that they become separated or penetrated. Then, the true values of $g(A,B)$ are 1, 2, and 0.5 in the three cases (contact, separate, and penetrate), respectively. We generate 3000 such pairs of ellipsoids and truncated cones for each case and report the average absolute error between the computed $g(A,B)$ and the true values and the average CPU running time of each algorithm, as exhibited in Tables 1 and 2, respectively. As a comparison with the existing approach, we also write $g(A,B)$ as a convex programming problem as in [10] and solve it using the active-set algorithm provided by the Optimization Toolbox of MATLAB with default settings. Fig. 8a shows the average error in $g(A,B)$ computed by the ray-shooting algorithms versus their termination tolerance for the three cases. Finally, we randomly generate 100 ellipsoids and truncated cones and compute $g(A,B)$ between any two of them. Then, the average running time of each algorithm is listed in the last row of Table 2. Fig. 8b plots the running time versus the termination tolerance. Table 2 also displays the number of iterations of each algorithm in the above tests.

From the numerical results, we can see that the ray-shooting algorithms are as accurate as the active-set algorithm in MATLAB and their running times are one order of magnitude shorter. It is no surprise that the result accuracy can be improved



(a) (b) (c)

Fig. 7 Growth distance computation between an ellipsoid and a truncated cone. (a) An ellipsoid contacting the side surface (upper), one edge (middle), or one base (lower) of a truncated cone. Then $g(A,B) = 1$. (b),(c) An ellipsoid and a truncated cone obtained by shrinking or enlarging those in (a) by 2 times about their centroids. Then $g(A,B) = 2$ or $g(A,B) = 0.5$.



(a) (b)

Fig. 8 Absolute error and CPU time versus the termination tolerance ε ($\varepsilon_{\mathrm{GJK}} = \varepsilon_{\mathrm{ZC}} = \varepsilon$). (a) Absolute error in the computed $g(A,B)$. (b) CPU running time of each ray-shooting algorithm.

by simply reducing the termination tolerance. It should be noted that the computation costs for one iteration of these algorithms are different. Although the number of iterations of the IE or ZC-IE procedure is bigger than or close to those of the other algorithms, they are still faster because the computation in each of their iteration is much simpler. Moreover, the running time of the IE or ZC-IE procedure increases much slower as the termination tolerance decreases.

Table 1 Absolute error of each algorithm from the ground truth $g(A,B)$. ($\varepsilon_{\text{GJK}} = \varepsilon_{\text{ZC}} = \varepsilon = 10^{-5}$)

	GJK-based	IE	ZC-based	Bi-GJK	ZC-IE	Active-Set
contact	9.62×10^{-7}	2.91×10^{-6}	1.24×10^{-6}	1.68×10^{-7}	9.10×10^{-7}	5.14×10^{-7}
separate	3.92×10^{-6}	5.92×10^{-6}	5.10×10^{-6}	4.60×10^{-6}	3.52×10^{-6}	5.42×10^{-6}
penetrate	2.42×10^{-7}	1.51×10^{-6}	4.82×10^{-7}	7.04×10^{-7}	2.18×10^{-7}	4.85×10^{-7}

Table 2 Average CPU running time (in milliseconds) and number of iterations (rounded up to the next highest integer in parentheses) of each algorithm to compute $g(A,B)$. ($\varepsilon_{\text{GJK}} = \varepsilon_{\text{ZC}} = \varepsilon = 10^{-5}$)

	GJK-based	IE	ZC-based	Bi-GJK	ZC-IE	Active-Set
contact	2.05 (2)	1.48 (15)	2.51 (20)	1.95 (5)	1.16 (17)	12.77 (9)
separate	1.85 (2)	1.43 (15)	2.36 (19)	1.60 (5)	1.08 (16)	13.23 (10)
penetrate	2.23 (2)	1.44 (15)	2.61 (20)	2.14 (5)	1.17 (18)	13.23 (10)
random	2.08 (2)	1.28 (17)	2.01 (21)	2.02 (6)	0.97 (18)	20.05 (16)

5 Application to Contact Force Optimization

5.1 Problem Description

Consider a robot system making m contacts with the environment, such as a multi-fingered robot hand grasping an object or a legged robot standing on the ground. To maintain the whole system in equilibrium, the resultant wrench $\mathbf{w}_{\text{res}}$ from all contact forces must counterbalance the external wrench $\mathbf{w}_{\text{ext}}$ (sum of the other wrenches) [9], which can be formulated with respect to the global coordinate frame as

$$\mathbf{w}_{\text{res}} = \sum_{i=1}^{m} \mathbf{G}_i \mathbf{f}_i = -\mathbf{w}_{\text{ext}} \tag{11}$$

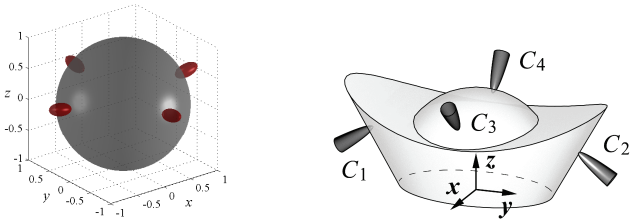
where $\mathbf{G}_i = \begin{bmatrix} \mathbf{n}_i & \mathbf{o}_i & \mathbf{t}_i & \mathbf{0} \\ \mathbf{p}_i\times\mathbf{n}_i & \mathbf{p}_i\times\mathbf{o}_i & \mathbf{p}_i\times\mathbf{t}_i & \mathbf{n}_i \end{bmatrix} \in \mathbb{R}^{6\times4}$, $\mathbf{p}_i$ is the position of contact i, and $\mathbf{n}_i$, $\mathbf{o}_i$ and $\mathbf{t}_i$ are the unit normal and tangent vectors at contact i in the global coordinate frame and satisfy $\mathbf{n}_i = \mathbf{o}_i \times \mathbf{t}_i$. The contact force $\mathbf{f}_i$ has four components, i.e., three pure force components f_{i1}, f_{i2}, f_{i3} along $\mathbf{n}_i$, $\mathbf{o}_i$, $\mathbf{t}_i$, respectively, and a spin moment f_{i4} about $\mathbf{n}_i$. To maintain a stable contact, $\mathbf{f}_i$ must stay within the friction cone [9]

$$F_i \triangleq \left\{ \mathbf{f}_i \in \mathbb{R}^4 \mid f_{i1} \geq 0,\ \sqrt{\frac{f_{i2}^2 + f_{i3}^2}{\mu_i^2} + \frac{f_{i4}^2}{\mu_{si}^2}} \leq f_{i1} \right\} \tag{12}$$

where μ_i and μ_{si} are the tangential and torsional friction coefficients, respectively.

A major problem in the research of multi-contact robotic systems is to determine whether there exist feasible contact forces $\mathbf{f}_i \in F_i$, $i = 1,2,\ldots,m$ to resist a given external wrench as (11) and compute the minimum contact forces if so. The overall contact force magnitude is often measured by

$$\sigma_{L_1} \triangleq \sum_{i=1}^{m} f_{i1} \quad \text{or} \quad \sigma_{L_\infty} \triangleq \max_{i=1,2,\ldots,m} f_{i1}. \tag{13}$$



(a) (b)

Fig. 9 Examples for contact force optimization. (a) Four contacts grasping a sphere. (b) Four contacts grasping an ingot (also used in [16, 18]).

As described in [6, 16, 18], this problem can be reduced to a ray-shooting problem between the ray $R(-\mathbf{w}_{\text{ext}})$ and the set

$$W_{L_1} \triangleq \text{CH}\left(\bigcup_{i=1}^{m} W_i\right) \quad \text{or} \quad W_{L_\infty} \triangleq \text{CH}\left(\bigoplus_{i=1}^{m} W_i\right) \tag{14}$$

where $W_i = \mathbf{G}_i(U_i)$ and $U_i \triangleq \left\{\mathbf{f}_i \in \mathbb{R}^4 \mid f_{i1} = 1,\ (f_{i2}^2 + f_{i3}^2)/\mu_i^2 + f_{i4}^2/\mu_{si}^2 = 1\right\}$. It can be derived that the minimum values of σ_{L_1} and σ_{L_∞} equal $\|\mathbf{w}_{\text{ext}}\|/\|\mathbf{z}_{W_{L_1}}(-\mathbf{w}_{\text{ext}})\|$ and $\|\mathbf{w}_{\text{ext}}\|/\|\mathbf{z}_{W_{L_\infty}}(-\mathbf{w}_{\text{ext}})\|$, respectively. Also, the corresponding contact forces can be computed through the computation of $Z_{W_{L_1}}(-\mathbf{w}_{\text{ext}})$ or $Z_{W_{L_\infty}}(-\mathbf{w}_{\text{ext}})$. The detailed derivation can be found in [6, 16, 18] and is omitted here due to the page limit.

As an extension to the previous work [6, 16, 18], we would like to point out here that the ray-shooting algorithms also work in the case where the overall contact force magnitude is measured and needs to be minimized as

$$\bar{\sigma}_{L_1} \triangleq \sum_{i=1}^{m} \|\mathbf{f}_i\| \quad \text{or} \quad \bar{\sigma}_{L_\infty} \triangleq \max_{i=1,2,\ldots,m} \|\mathbf{f}_i\| \tag{15}$$

where $\|\mathbf{f}_i\| \triangleq \sqrt{f_{i1}^2 + f_{i2}^2 + f_{i3}^2 + (\mu_i^2/\mu_{si}^2) f_{i4}^2}$ is the magnitude of $\mathbf{f}_i$ rather than its normal component f_{i1} used in (13). To do this, we only need to redefine $U_i \triangleq \left\{\mathbf{f}_i \in F_i \mid f_{i1}^2 + f_{i2}^2 + f_{i3}^2 + (\mu_i^2/\mu_{si}^2) f_{i4}^2 = 1\right\}$. The derivation and the test of the ray-shooting algorithms to compute minimum contact forces in terms of a different force magnitude measure will be included in a complete version of this paper.

5.2 Numerical Examples

We first verify the accuracy of the ray-shooting algorithms applied to the contact force optimization with the example shown in Fig. 9a, where four contacts are located symmetrically on a sphere and have the same elevation angle α. Then, it can be derived that the minimum normal contact force for holding the sphere is

Table 3 Relative error of each algorithm to minimize σ_{L_1}.

λ	GJK-based	IE	ZC-based	Bi-GJK	ZC-IE	Active-Set
10^{-1}	6.86×10^{-16}	4.93×10^{-6}	0	0	0	3.63×10^{-9}
10^{-2}	1.35×10^{-14}	1.98×10^{-6}	0	1.83×10^{-16}	0	1.65×10^{-6}
10^{-3}	1.38×10^{-13}	5.38×10^{-6}	6.88×10^{-14}	6.91×10^{-14}	6.88×10^{-14}	2.88×10^{-8}
10^{-4}	0	2.84×10^{-6}	6.89×10^{-13}	6.89×10^{-13}	6.89×10^{-13}	3.92×10^{-3}
10^{-5}	6.89×10^{-12}	infeasible	infeasible	infeasible	infeasible	1.09×10^{-1}
10^{-6}	infeasible	infeasible	infeasible	infeasible	infeasible	infeasible

$G/4(\mu\cos\alpha-\sin\alpha)$ for each contact and no feasible contact forces exist to do so if $\alpha\geq\tan^{-1}\mu$, where G is the gravity of the sphere and μ is the tangential friction coefficient. Thus, the minimum values of σ_{L_1} and σ_{L_∞} are $G/(\mu\cos\alpha-\sin\alpha)$ and $G/4(\mu\cos\alpha-\sin\alpha)$, respectively. Here, we take $G=10$ N, $\mu=0.2$, and $\alpha=(1-\lambda)\tan^{-1}\mu$ for $\lambda=10^{-1},10^{-2},\ldots,10^{-6}$, respectively. The termination tolerance is 10^{-5} for each ray-shooting algorithm. For comparison, we also formulate the contact force optimization as a convex programming problem and solve it using the active-set algorithm provided by MATLAB, for which the maximum number of function evaluations and the maximum number of iterations are both set to be sufficiently big such that its result can be comparably accurate. Tables 3 and 4 display the relative error in the result of each algorithm compared with the true minimum value of σ_{L_1} or σ_{L_∞}. It is shown that, in this particular case, the results of the ray-shooting algorithms except the IE procedure are highly accurate. Actually, they compute the minimum contact forces by evaluating the support mapping of W_{L_1} or W_{L_∞} only once, which can be done analytically as discussed in [15], and the error in their results is just the round-off error. As the elevation angle α of each contact approaches $\tan^{-1}\mu$, some algorithms report that the problem becomes infeasible. This is because the termination tolerance ε is too big for those cases. To relieve this numerical issue and obtain a more accurate result, we have to choose a smaller ε.

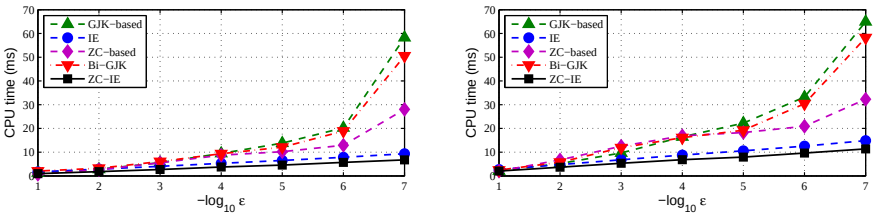
We also verify the efficiency of these algorithms with the grasping example used in [16, 18], as shown in Fig. 9. We compute contact forces with minimum σ_{L_1} or σ_{L_∞} with respect to 10^3 random $\mathbf{w}_{\text{ext}}$ and report the average CPU running time and number of iterations of each algorithm, as exhibited in Table 5. It can be seen that the ray-shooting algorithms are several times faster than the active-set algorithm in MATLAB. In particular, the IE and ZC-IE procedures are more efficient than the other ray-shooting algorithms due to the much simpler computation in their iterations, though they need more iterations. Also, their running times rise slower along with the decrease of the termination tolerance, as revealed in Fig. 10. Furthermore, the ZC-IE procedure is even faster than the IE procedure because it saves the computation for maintaining the facet and ridge collections, which is only required on the rare occasion that the ray passes through a face of low dimension. In our implementation of algorithms, the IE and ZC-IE procedures have the same initialization. Then, the difference in their numbers of iterations shown in Table 5 implies that the occasion happens and causes the two procedures to follow different iteration steps.

Table 4 Relative error of each algorithm to minimize σ_{L_∞}.

λ	GJK-based	IE	ZC-based	Bi-GJK	ZC-IE	Active-Set
10^{-1}	0	6.53×10^{-6}	0	0	0	7.50×10^{-9}
10^{-2}	0	5.49×10^{-6}	0	6.96×10^{-15}	0	1.18×10^{-8}
10^{-3}	1.38×10^{-13}	5.19×10^{-6}	1.38×10^{-13}	1.38×10^{-13}	1.38×10^{-13}	5.63×10^{-6}
10^{-4}	0	5.39×10^{-6}	0	0	0	9.65×10^{-5}
10^{-5}	1.38×10^{-11}	infeasible	infeasible	infeasible	infeasible	3.00×10^{-4}
10^{-6}	1.50×10^{-16}	infeasible	infeasible	infeasible	infeasible	1.70×10^{-3}

Table 5 Average CPU running time (in milliseconds) and number of iterations (rounded up to the next highest integer in parentheses) of each algorithm for contact force optimization.

	GJK-based	IE	ZC-based	Bi-GJK	ZC-IE	Active-Set
minimize σ_{L_1}	13.80 (4)	6.50 (33)	10.22 (26)	12.11 (8)	4.57 (30)	38.94 (24)
minimize σ_{L_∞}	22.23 (4)	10.55 (54)	18.27 (47)	19.25 (10)	7.93 (53)	47.36 (31)



(a) (b)

Fig. 10 CPU running time versus the termination tolerance ε ($\varepsilon_{\mathrm{GJK}} = \varepsilon_{\mathrm{ZC}} = \varepsilon$) of each ray-shooting algorithm to minimize (a) σ_{L_1} or (b) σ_{L_∞}.

6 Conclusions and Future Work

In this paper, we discussed general ray-shooting algorithms and their applications in robotics. We first described three individual procedures and their hybrid uses to compute the ray-shooting problem. They do not use any existing optimization techniques and their implementation is very straightforward. Then, their performance qualities, including the computational accuracy and efficiency, have been verified and compared with each other in the application to growth distance computation and contact force optimization. Numerical examples show that the ray-shooting algorithms provide better ways to solve these fundamental problems in robotics than some existing methods. Particularly, the ZC-IE procedure, which is newly proposed in this paper, is notably faster than the other ray-shooting algorithms and able to compute results at the same level of accuracy.

Because of the page limit, we could not go into the implementation details of these algorithms and leave it for future publications. As future work, we also would like to apply the algorithms to a dynamic environment, such as collision detection for moving objects or contact force optimization with respect to a time-varying external wrench. Then, by taking advantage of time coherence, they may achieve constant time complexity. We will conduct more experiments on the algorithms,

especially in dynamic environments, and report their results in a complete version of this paper. By doing this, we expect to give a comprehensive evaluation of their performance and figure out how to choose an algorithm for a specific problem. In addition, we are exploring other applications of these ray-shooting algorithms.

References

1. P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 517–526, 1992.
2. D. Ding, Y.-H. Liu, Y. Wang, and S. G. Wang. Automatic selection of fixturing surfaces and fixturing points for polyhedral workpieces. *IEEE Transactions on Robotics and Automation*, 17(6):833–841, 2001.
3. E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 4(2):193–203, 1988.
4. S. R. Lay. *Convex Sets and their Applications*. John Wiley & Sons, New York, NY, USA, 1982.
5. M. Lin and J. Canny. A fast algorithm for incremental distance calculation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1008–1014, Sacramento, CA, 1991.
6. Y.-H. Liu. Qualitative test and force optimization of 3-D frictional form-closure grasps using linear programming. *IEEE Transactions on Robotics and Automation*, 15(1):163–173, 1999.
7. Y.-H. Liu, M.-L. Lam, and D. Ding. A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain. *IEEE Transactions on Robotics*, 20(5):805–816, 2004.
8. J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Computational Geometry*, 10(1):215–232, 1993.
9. R. M. Murray, Z. X. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, USA, 1994.
10. C. J. Ong and E. G. Gilbert. Growth distance: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation*, 12(6):888–903, 1996.
11. C. J. Ong and E. G. Gilbert. Robot path planning with penetration growth distance. *Jounral of Robotic Systems*, 15(2):57–74, 1998.
12. W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, NY, USA, third edition, 1976.
13. L. Szirmay-Kalos, V. Havran, B. Balázs, and L. Szécsi. On the efficiency of ray-shooting acceleration schemes. In *Proceedings of the Spring conference on Computer Graphics*, pages 97–106, Budmerice, Slovakia, 2002.
14. G. van den Bergen. Ray casting against general convex objects with application to continuous collision detection. http://www.dtecta.com, 2004.
15. Y. Zheng and C.-M. Chew. Distance between a point and a convex cone in n-dimensional space: computation and applications. *IEEE Transactions on Robotics*, 25(6):1397–1412, 2009.
16. Y. Zheng and C.-M. Chew. A numerical solution to the ray-shooting problem and its applications in robotic grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2080–2085, Kobe, Japan, May 2009.
17. Y. Zheng, C.-M. Chew, and A. H. Adiwahono. A GJK-based approach to contact force feasibility and distribution of multi-contact robots. *Robotics and Autonomous Systems*, 59(3-4):194–207, 2011.
18. Y. Zheng, M. C. Lin, and D. Manocha. A fast n-dimensional ray-shooting algorithm for grasping force optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1300–1305, Anchorage, Alaska, May 2010.