Robotica (2014) volume 32, pp. 867–887. © Cambridge University Press 2013
doi:10.1017/S0263574713001057

New computational method for three-fingered force-closure test

Nattee Niparnan, Thanathorn Phoka, Yuttana Suttasupa and Attawith Sudsang*

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, 10330, Thailand

(Accepted October 29, 2013. First published online: December 5, 2013)

SUMMARY
This paper proposes an efficient implementation of a force-closure test for frictional three-finger grasps. The implementation is based on a condition that transforms force-closure testing into the problem of convex hull intersection in projective space. The proposed implementation further reduces the problem into the problem of computing whether a line segment intersects a convex hull of at most four points. Implementation results are presented along with a thorough performance analysis and comparison with several existing methods. The results are also verified with arbitrary precision floating point computation. This provides comparison of qualitative error resulting from floating point roundoff. The result shows that the proposed implementation outperforms other methods in terms of speed and precision.

KEYWORDS: Grasping; Force closure; Projective geometry.

1. Introduction

Grasping has been an active research topic in robotics since the pioneering works of Salisbury and Roth.[1,2] A reader is referred to Bicchi and Kumar[3] and Bicchi[4] for recommended surveys of the topics. There is one central question being shared among all grasping problems: Whether an object can be *securely grasped*? Several definitions of a secure grasp have been proposed, but a particular grasping property that has captured the maximum attention is undoubtedly *force closure*, which indicates a grasp that can resist any external disturbance. The best known force-closure condition states that a grasp achieves force closure when the convex hull of vectors describing exertable force–torque, called *wrench*, contains the origin. This condition is the root of many force-closure tests. Tests that adopt this approach are usually general, being able to handle any number of contacts and dimensions. Ferrari and Canny[5] uses this concept to provide a quality metric for force-closure grasp. Other examples include a qualitative test of Liu,[6] which is used in many subsequent works of Ding *el al.*[33] and Liu *et al.*[34] The Q Distance of Zhu and Wang and the gauge function are also included in this category.[7,8]

Algorithms for calculating a distance between two convex objects can be used for this problem. Examples include the GJK algorithm,[9] a very fast collision checking algorithm which was recommended for force-closure testing,[10] and other algorithms that compute distance between a point and a convex object.[11,12] As opposed to a general test, there exist specific tests that are applicable under some criteria. For example, Nguyen proposes a simple test of force closure for two fingers in 2D.[13] The linear programming system in the work of Ponce and Faverjon[14] is also a test for a planar three-finger grasp. This approach is usually more efficient than a general test since it is specially designed by exploiting *a priori* knowledge of a particular case. A good example on how significant this can help is demonstrated in the work of Li *et al.*,[15] where a fast planar three-finger force-closure test is presented. Although there are several force-closure testing methods, unfortunately there exists no performance comparison between these methods under the same input and setting.

* Corresponding author. E-mail: attawith@gmail.com

A main contribution of this paper is to present a comparative study of available force-closure tests in dealing with three-fingered grasps under the Coulomb friction. Grasping with three contact points is the simplest form of grasp that can achieve force closure on 3D objects. Its practical advantage not only attracts a great deal of research attention but also a wide range of applications. This is evidenced by increasing popularity of three-fingered hand such as the Barrett hand, which recently has become a practical alternative to the less flexible specialized gripper solution. One important basic problem in this area is the determination whether a given three-fingered grasp achieves force closure. Although several methods in the literature can handle this problem, there exists no comparison to describe how well they perform relative to one another. In robotic grasping, a computational aspect is usually given a weaker emphasis than the mechanical counterpart. New methods are often proposed without any comparison with the existing ones. As a result, we are often left to choose the method to be used without adequate information. In our study, many methods that can verify three-fingered force closure are implemented as described in their original papers, and their performances are evaluated under the same input and setting in terms of speed and accuracy. It is proven in Li *et al.*[15] that 3D three-fingered grasps under the Coulomb friction can be reduced to 2D computation. Hence, most of the results in this paper consider 2D cases, while 3D are discussed briefly.

Another contribution of this paper is a novel implementation of the method by Mason and Brost,[16] for which computational implementation has never been reported. This is by no means trivial since intricate geometric nature of the method is needed to be translated into an efficient algorithm and implementation. More precisely, our presented algorithm relies on representing wrenches associated with the input grasp using a 2D representation, referred to in Brost and Mason as the *force-dual space*. With this representation, we transform force-closure test into the problem of computing whether a line segment intersects a convex polygon of at most four vertices. Interestingly, it turns out that this decade old method by Mason and Brost under our novel implementation outperforms many methods proposed recently in the literature. This is quite a surprise since this method was originally proposed primarily for intuition and for working problem by hand (using graphical drawings). Our comparative study considers speed and accuracy of different methods. We pay great attention to the accuracy of the results by comparing them with reference results computed using exact arithmetic. This comparison and the accompanied analysis reveal drastic difference in the performance of different methods and several interesting issues, including incompleteness and errors, regarding existing force-closure tests. To the best of our knowledge, this presentation provides the first performance comparison on available force-closure tests that has been conducted at this level of thoroughness. We hope that this comparative study would help stimulate development of more efficient methods and facilitate the readers in choosing a suitable force-closure test or implementing their own.

The paper is organized as follows. Section 2 briefly describes necessary background on grasping and force closure. Our proposed implementation is presented in Section 3. Section 4 describes existing tests in brief, and Section 4.2 presents the comparison between these tests and our method. Sections 5 and 6 give a brief explanation on how our method is used in the 3D case and its example, respectively. Finally, Section 7 concludes the paper.

2. Background

In this section, we review some background necessary for understanding the proposed algorithm. In particular, we briefly review the concept of force closure and the main idea of the graphical analysis of frictional contact forces presented in Brost and Mason[16] (see also chapter 5 in Mason[17]). The discussion only concerns planar grasps.

A grasp achieves force closure when it can counterbalance any external disturbance to the object being grasped. The external disturbance and the effect of a grasping device are represented as the force $\boldsymbol{f}$ and the torque $\boldsymbol{\tau}$. It is conventional to combine the force $\boldsymbol{f} = (f_x, f_y)$ and the torque $\boldsymbol{\tau}$ into an entity called the *wrench*, $\boldsymbol{w} = (f_x, f_y, \boldsymbol{\tau})$. This work assumes the hard contact model, which means that a contact is unable to exert a pure torque. This implies that the effect of a contact point at $\boldsymbol{p}$ that exerts the force $\boldsymbol{f}$ can be represented by the wrench $\boldsymbol{w} = (\boldsymbol{f}, \boldsymbol{p} \times \boldsymbol{f}) \in \mathbb{R}^3$ (recall that for $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$, their cross product $\boldsymbol{x} \times \boldsymbol{y}$ is the scalar $x_1 y_2 - y_1 x_2$). This work also assumes frictional contact with the Coulomb friction. Let $\boldsymbol{n}$ be the inward pointing normal vector of the object at $\boldsymbol{p}$, and let μ be the coefficient of friction. The Coulomb friction indicates that, in order to ensure non-slipping contact, the exerted force must lie inside a cone defined by two vectors $\boldsymbol{f}_l$ and

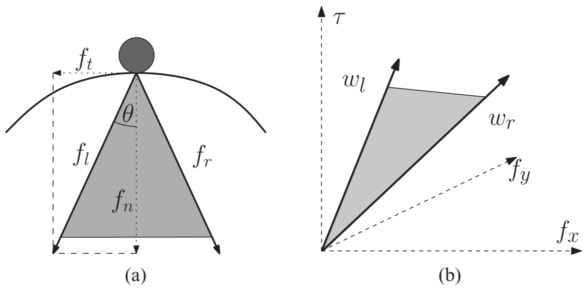

Fig. 1. (a) Set of non-slip force under the Coulomb friction. The vectors f_t and f_n are the tangential and the normal forces, respectively. Let μ be the friction coefficient, $f_t/f_n = \mu$. In other words, the angle between the normal force and the boundary force is $\theta = \arctan(\mu)$. (b) Wrenches associated with a contact point form a fan in the wrench space.

$\boldsymbol{f}_r$, that have an angle $\theta = \tan^{-1}\mu$ with the normal vector $\boldsymbol{n}$ (see Fig. 1). In other words, a force that is a positive combination of $\boldsymbol{f}_l$ and $\boldsymbol{f}_r$ can be exerted by the contact. The wrenches associated with $\boldsymbol{f}_l$ and $\boldsymbol{f}_r$ are $\boldsymbol{w}_l = (\boldsymbol{f}_l, \boldsymbol{p} \times \boldsymbol{f}_l)$ and $\boldsymbol{w}_r = (\boldsymbol{f}_r, \boldsymbol{p} \times \boldsymbol{f}_r)$, respectively. Similarly, a wrench that is a positive combination of $\boldsymbol{w}_l$ and $\boldsymbol{w}_r$ is associated with some corresponding force of the contact point. Therefore, it suffices to describe a contact by $\boldsymbol{w}_l$ and $\boldsymbol{w}_r$ only. These wrenches are called *primitive wrenches*.

In force-closure analysis, a contact wrench is allowed to take arbitrarily large magnitude, leaving only its direction to affect the determination of force closure. As a result, it is obvious that force closure is achieved when all the contact wrenches positively span the entire wrench space (i.e., any disturbing wrench can be resisted by the fingers' wrenches). This is equivalent to the following well-known force-closure condition.

Proposition 2.1. Force closure is achieved when the origin is contained in the interior of the convex hull of all the primitive wrenches.

Proposition 2.1 is the root of several methods for force-closure test. The reader is referred to Mishra *et al.*[18] for the detail of the proof. In this paper, the proposition is transformed into a computational implementation by using a representation from the graphical analysis of frictional contact forces in Brost and Mason.[16] The representation and related ideas that can be derived directly from the paper will be described in the remainder of this section.

Since only the wrench direction is counted toward force-closure determination, it suffices to represent a wrench in the wrench space (f_x, f_y, τ) as a point on the unit sphere centered at the origin. As an alternative, we can instead represent a wrench using its intersection (called *dual point*) with one of the following structures: (1) plane $\tau = 1$, (2) plane $\tau = -1$, or (3) the unit circle on the plane $\tau = 0$ with its center at the origin. Note that the dual point on the plane $\tau = 1$ of the wrench (f_x, f_y, τ) with positive τ is simply $(f_x/\tau, f_y/\tau)$, requiring two divisions. The benefit of this representation is that it allows a convex cone of wrenches to be conveniently represented. As described in Brost and Mason,[16] it is obvious from the geometry that each convex cone of wrenches with positive (resp. negative) torques can be represented as the convex hull of the corresponding dual points on the plane $\tau = 1$ (resp. $\tau = -1$). See Fig. 2(a) for examples.

Let us consider all primitive wrenches of a grasp and assume that their dual points lie on the planes $\tau = 1$ or $\tau = -1$ only (this assumption can be easily fulfilled by ensuring that the origin of the primary space does not lie on the line that supports any friction cone's boundary). On each of these two planes, let us construct the convex hull of all the dual points that lie on the plane. If all the dual points lie only on one plane, it is obviously impossible for the grasp to achieve force closure. To determine force closure, Proposition 2.1 can be consulted. It can be easily shown from geometry that the proposition is satisfied when there exists a line through the origin of the wrench space that intersects each of the two convex hulls in its relative interior[1] (see Fig. 2(a)). Instead of

[1] A relative interior of a set is the interior relative to the affine hull of the set. Intuitively speaking, a relative interior consists of all points not on the relative edge of the set, e.g., a relative interior of a line segment is the segment minus its endpoints, regardless of the dimension where the line is situated.

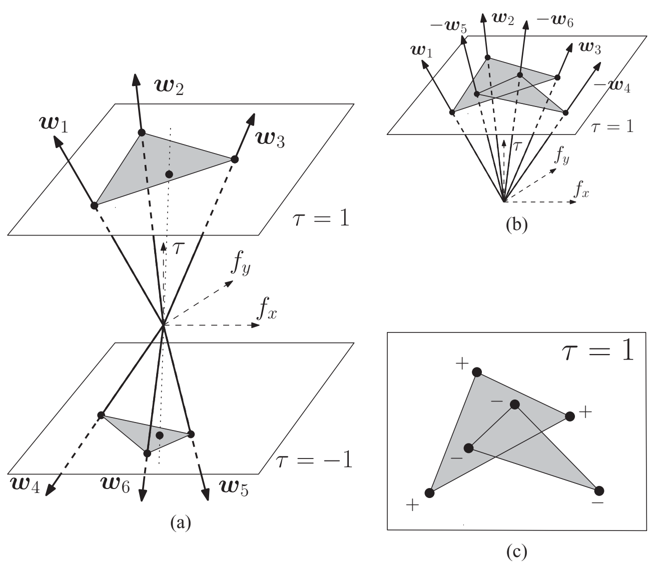

Fig. 2. Wrenches as represented on the dual space. We consider an example case of six wrenches. We would like to determine if these wrenches achieve force closure. (a) $\boldsymbol{w}_1$, $\boldsymbol{w}_2$, $\boldsymbol{w}_3$ having positive torques are represented as points on $\tau = 1$, and $\boldsymbol{w}_4$, $\boldsymbol{w}_5$, $\boldsymbol{w}_6$ having negative torques are represented as points on $\tau = -1$. The shaded regions are convex hulls representing the convex cone of the wrenches on their respective planes. The dotted line represents a line through the origin that intersects the relative interiors of both convex hulls. This indicates force closure. (b) Wrenches with negative torques are represented by the intersections of their negative with $\tau = 1$ plane. (c) Simplified illustration showing only the dual points. Symbols $+$ and $-$ indicate positive and negative dual points, respectively. The intersection between the relative interiors of two convex hulls indicates force closure.

directly checking for the existence of such line, the following equivalent method can be used. First, label all dual points in the plane $\tau = 1$ as positive dual points. Second, for each primitive wrench $\boldsymbol{w}$ with a negative torque, draw the dual point of its inverse $-\boldsymbol{w}$ on the plane $\tau = 1$ and label it as a negative dual point. Third, construct the convex hull of all positive dual points and the convex hull of all negative dual points. It is clear that Proposition 2.1 is satisfied when the two convex hulls intersect and some intersection point is contained in the relative interior of both convex hulls. Note that the mapping from a primitive wrench with positive (resp. negative) torque to a positive (resp. negative) dual point in the plane $\tau = 1$ will also be used in the algorithm of the next section.

Despite sharing the same dual representation, the procedure described above is different from the method presented in Brost and Mason.[16] It is described in Brost and Mason that to confirm force closure, one needs to find in the plane $\tau = 1$ a pair of intersecting line segments, one joining positive dual points and the other joining negative dual points, or a triangle formed by three dual points of the same sign that contains another dual point of the opposite sign. Our procedure avoids the enumeration of triangles which results in a more efficient implementation. The advantage is more obvious when the number of fingers increases, since our procedure takes $O(N \lg N)$ for N fingers, while the original method presented in Brost and Mason obviously takes $O(N^3)$. As mentioned in the Introduction, the original method in Brost and Mason is developed primarily for intuition and working problems by hand. Extra effort is needed to transform the underlying idea of the method into an efficient and robust computational implementation.

3. The Implementation

In this section, we present a detailed algorithm for three-fingered frictional force-closure test. The algorithm demonstrates how the concept described in the previous section can be efficiently implemented.

Given a grasp defined by three contact points together with the corresponding contact normal vectors and the coefficient of friction, the following algorithm determines whether a grasp achieves force closure:

1. Choose one of the contact points as the origin and compute the two primitive wrenches of this contact point according to the chosen origin. Denote these two wrenches by $\boldsymbol{u}$ and $\boldsymbol{v}$.
2. According to the origin chosen in step 1, compute the four primitive wrenches of the remaining two contact points. If it is not true that some of these primitive wrenches have positive torques and some have negative torques, report that the grasp does not achieve force closure and halt.
3. Compute the positive and negative dual points on the plane $\tau = 1$ of the primitive wrenches obtained from step 2. The dual point of a wrench with zero torque, which is a point at infinity in the direction of the wrench, is regarded as a positive dual point. In practice, a point at infinity is presented by a vector indicating its direction and an additional flag indicating that it is a point at infinity.
4. Depending on the number of positive and negative dual points from step 3, apply one of the following steps:
   - Two positive and two negative dual points: Let the dual points of $\boldsymbol{u}$ and $\boldsymbol{v}$ be positive dual points.
   - One positive and three negative dual points, or one negative and three positive dual points: Let the dual point of $\boldsymbol{u}$ be a positive dual point and the dual point of $-\boldsymbol{v}$ be a negative dual point.
5. It is guaranteed by step 4 that there must be four dual points of the same sign on the plane $\tau = 1$. Compute the convex hull of these four dual points and detect if the convex hull intersects the line segment formed by the remaining two dual points. If the intersection contains a point in the relative interior of the convex hull and the line segment, report that the grasp achieves force closure.

This algorithm is essentially the convex hull-based method of Section 2 that is optimized for three-fingered grasps. The optimization stems from the choice of the origin made in step 1. As the two primitive wrenches of the chosen contact point possess zero torque, their dual points which are points at infinity are obtained without any computation. Saving four out of 12 floating-point divisions in dual point computation practically yields considerable speedup. In addition, the freedom to map the primitive wrenches $\boldsymbol{u}$ and $\boldsymbol{v}$ to positive or negative dual points is exploited in step 4 to ensure that four dual points are of one sign (either positive or negative) and the remaining two dual points are of the other sign. This grouping allows us to apply our brute-force convex hull computation for four input points which is more efficient in this case than generic convex hull algorithms. The optimization described above, of course, comes at an increased complexity, i.e., dual points at infinity have to be explicitly dealt with. Fortunately, our convex hull computation and the intersection detection (step 5) are based on the cross product operation which can be easily extended to handle points at infinity. In the remainder of this section, this extension will be described along with our four-point convex hull algorithm and the intersection detection.

It should be noted that a degenerate case when some wrenches are co-punctual is possible. When wrenches are co-punctual their associated dual points are identical. At least four wrenches are needed to satisfy the condition in Proposition 2.1. Hence, only the case of four and five dual points needs to be considered. For both cases, we can map the primitive wrenches $\boldsymbol{u}$ and $\boldsymbol{v}$ such that three dual points are positive (or negative). We then compute the convex hull of three points instead, which is much easier and straightforward. There might exist only one dual point of the opposite sign. Thus, the intersection is only to check whether this dual point lies strictly inside the convex hull. Again, this is straightforward.

3.1. The cross product

Our convex hull algorithm for four input points and the detection of intersection between a line segment and a polygon (to be described shortly) rely critically on a test to determine for three points, say $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, whether $\boldsymbol{a}$ lies on the directed line $\boldsymbol{bc}$, or on which side of it. The test can be performed by inspecting the sign of the cross product $\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}$. A positive, negative, or zero outcome indicates whether $\boldsymbol{a}$ lies, respectively, on the left, on the right, or exactly on the directed line $\boldsymbol{bc}$. For convenience, let $\operatorname{sgn}(x)$ be -1, 0, or $+1$ when x is respectively negative, zero, or positive. To describe how to compute $\operatorname{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}})$ when $\boldsymbol{a}$, $\boldsymbol{b}$, or $\boldsymbol{c}$ could be points at infinity, some properties of points at infinity are needed.

In Projective Geometry, a point at infinity is an intersection of parallel lines.[19] Informally speaking, the farthest point of a ray is a point at infinity. Unlike an ordinary point which is defined by its position, a point at infinity $\boldsymbol{p}$ is defined by its direction vector, denoted by $\boldsymbol{w}(\boldsymbol{p})$; rays pointing in the same direction meet at the same point at infinity (defined by the direction of the rays). We can therefore infer that a vector from an ordinary point to a point at infinity is in the same direction as the vector that defines the point at infinity. All points at infinity are defined to lie on the same line called the line at infinity. When traversing the line at infinity in the counterclockwise direction, all ordinary points lie on the left. With the properties described above, the following rules are added for computing $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}})$:

1. When only $\boldsymbol{a}$ is a point at infinity: Since $\overrightarrow{\boldsymbol{ba}}$ is parallel to $\boldsymbol{w}(\boldsymbol{a})$, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = \mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \boldsymbol{w}(\boldsymbol{a}))$.
2. When $\boldsymbol{c}$ is at infinity but $\boldsymbol{b}$ is not: Since $\overrightarrow{\boldsymbol{bc}}$ is parallel to $\boldsymbol{w}(\boldsymbol{c})$, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = \mathrm{sgn}(\boldsymbol{w}(\boldsymbol{c}) \times \overrightarrow{\boldsymbol{ba}})$. If $\boldsymbol{a}$ is also at infinity, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = \mathrm{sgn}(\boldsymbol{w}(\boldsymbol{c}) \times \boldsymbol{w}(\boldsymbol{a}))$.
3. When $\boldsymbol{b}$ is at infinity but $\boldsymbol{c}$ is not: Since $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = -\mathrm{sgn}(\overrightarrow{\boldsymbol{cb}} \times \overrightarrow{\boldsymbol{ca}})$ and $\overrightarrow{\boldsymbol{cb}}$ is parallel to $\boldsymbol{w}(\boldsymbol{b})$, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = -\mathrm{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \overrightarrow{\boldsymbol{ca}})$. If $\boldsymbol{a}$ is also at infinity, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = -\mathrm{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{a}))$.
4. When $\boldsymbol{b}$ and $\boldsymbol{c}$ are points at infinity: $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = \mathrm{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{c}))$. If $\boldsymbol{a}$ is also at infinity, $\mathrm{sgn}(\overrightarrow{\boldsymbol{bc}} \times \overrightarrow{\boldsymbol{ba}}) = 0$.

3.2. *Computing the convex hull of four points*

In fact, Graham's scan algorithm can be used here by replacing the standard cross product with the extended version described in Section 3.1.[20] However, for the case of four input points, the following brute-force method demands less overhead, and is therefore more efficient.

Let the four input points be arbitrarily denoted by $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{q}_1$, and $\boldsymbol{q}_2$, and let us define segments $p = \overline{\boldsymbol{p}_1\boldsymbol{p}_2}$ and $q = \overline{\boldsymbol{q}_1\boldsymbol{q}_2}$. The underlying idea of our convex hull algorithm comes from observing how p and q lie with respect to each other. Note that the output convex hull is represented here as a sequence of points arranged counterclockwise. Such sequence can be shifted freely, e.g., sequences $(\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4)$, $(\boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_1)$, and $(\boldsymbol{c}_4, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3)$ represent the same convex hull.

To characterize how segments p and q lie with respect to each other, let us define $t_i^p = \mathrm{sgn}(\overrightarrow{\boldsymbol{p}_1\boldsymbol{p}_2} \times \overrightarrow{\boldsymbol{p}_1\boldsymbol{q}_i})$, $i = 1, 2$, and $t_i^q = \mathrm{sgn}(\overrightarrow{\boldsymbol{q}_1\boldsymbol{q}_2} \times \overrightarrow{\boldsymbol{q}_1\boldsymbol{p}_i})$, $i = 1, 2$. The sign of a cross product is computed as described in Section 3.1 so that input points at infinity can be handled. From the definition, t_i^p (resp. t_i^q) takes on the value of 0, $+1$, or -1 when $\boldsymbol{q}_i$ (resp. $\boldsymbol{p}_i$) is exactly on, on the left or on the right of segment p (resp. segment q). With this setting, arrangement of segments p and q can be classified into one of the following three cases.

3.2.1. When $\boldsymbol{t_1^p = t_2^p}$ *and* $\boldsymbol{t_1^q = t_2^q}$. This case is illustrated in Fig. 3(a). Obviously, the resulting convex hull is a concatenation of the endpoints of segments p and q, i.e. $(\boldsymbol{p}_k, \boldsymbol{p}_l, \boldsymbol{q}_m, \boldsymbol{q}_n)$ where $k \neq l$, $m \neq n$, and $k, l, m, n \in \{1, 2\}$. Assignment of k, l, m, and n has to be made such that $(\boldsymbol{p}_k, \boldsymbol{p}_l, \boldsymbol{q}_m, \boldsymbol{q}_n)$ is in the counterclockwise order. It is easy to verify that this requirement can be satisfied by the following rule: When $t_1^q = t_2^q = +1$, we set $k = 1$ and $l = 2$, otherwise we set $k = 2$ and $l = 1$, and when $t_1^p = t_2^p = +1$, we set $m = 1$ and $n = 2$, otherwise we set $m = 2$ and $n = 1$.

3.2.2. When $\boldsymbol{t_1^p \neq t_2^p}$ *and* $\boldsymbol{t_1^q \neq t_2^q}$. An example of this case is shown in Fig. 3(b). Since the condition indicates that segments p and q cross each other, the resulting convex hull is an interwoven sequence of the endpoints from the two segments, i.e. $(\boldsymbol{p}_1, \boldsymbol{q}_m, \boldsymbol{p}_2, \boldsymbol{q}_n)$, where $m \neq n$ and $m, n \in \{1, 2\}$. Assignment of m and n needs to cause the sequence to follow the counterclockwise order. This requires m and n to be chosen such that $t_m^p = -1$ or $t_n^p = +1$ (either t_m^p or t_n^p is allowed to be zero in case of three collinear points; see Fig. 3(c) for example).

3.2.3. When $\boldsymbol{t_1^p = t_2^p}$ *and* $\boldsymbol{t_1^q \neq t_2^q}$, *or when* $\boldsymbol{t_1^p \neq t_2^p}$ *and* $\boldsymbol{t_1^q = t_2^q}$. As illustrated in Fig. 3(d), the resulting convex hull consists of only three points; one input point is discarded. Let us describe only the case in which $t_1^p = t_2^p$ and $t_1^q \neq t_2^q$ (the other case is treated likewise). In this case, either $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$ (not both) has to be discarded because it lies inside the convex hull. If $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are on the left side of segment p ($t_1^p = t_2^p = +1$), the convex hull can be written in the form $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_m)$, $m \in \{1, 2\}$. When no three points are collinear, it is easy to verify that the value of m must be chosen such that $t_m^q = -1$. However, when some three points are collinear, it is possible that $t_1^q \neq -1$ and $t_2^q \neq -1$. When this occurs, choose m such that $t_m^q = 0$ in order to correctly eliminate the redundant point. For the case that $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are on the right side of segment p, we will have the resulting convex hull $(\boldsymbol{p}_2, \boldsymbol{p}_1, \boldsymbol{q}_m)$ where the value of m must be chosen such that $t_m^q = +1$ if possible, otherwise choose m such that $t_m^q = 0$.

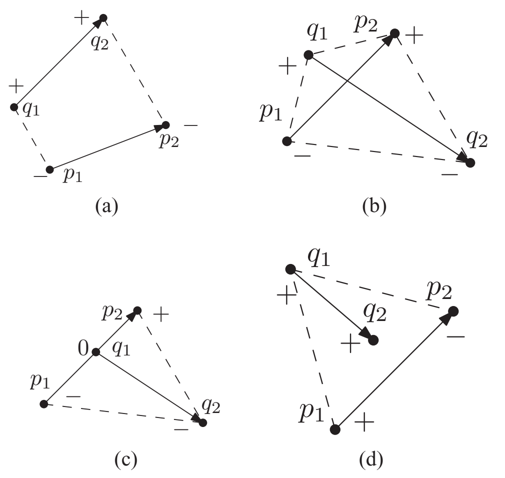

Fig. 3. Arrangements of two segments forming a convex hull. The convex hull is represented by dashed lines. Signs + and − indicate the value of t_i^p and t_i^q. The plus sign (+) indicates that the point is on the left of the other segment, while the minus sign means that the point is on the right of the other segment.

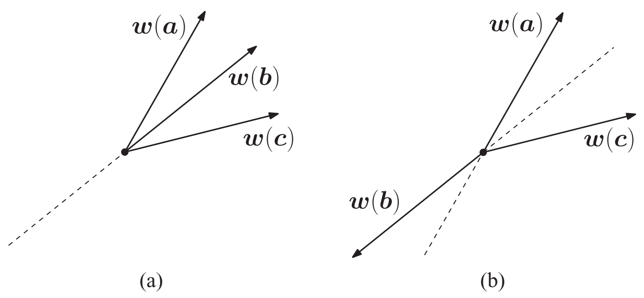
$\boldsymbol{w}(\boldsymbol{a})$
$\boldsymbol{w}(\boldsymbol{b})$
$\boldsymbol{w}(\boldsymbol{c})$
(a)
(b)

Fig. 4. Arrangement of three points at infinity. (a) $\boldsymbol{w}(\boldsymbol{b})$ is between $\boldsymbol{w}(\boldsymbol{a})$ and $\boldsymbol{w}(\boldsymbol{c})$. Vectors $\boldsymbol{w}(\boldsymbol{a})$ and $\boldsymbol{w}(\boldsymbol{c})$ lie on the opposite side with respect to $\boldsymbol{w}(\boldsymbol{b})$. (b) Another case where $\boldsymbol{w}(\boldsymbol{a})$ and $\boldsymbol{w}(\boldsymbol{c})$ lie on the opposite side with respect to $\boldsymbol{w}(\boldsymbol{b})$ but $\boldsymbol{w}(\boldsymbol{b})$ does not lie between $\boldsymbol{w}(\boldsymbol{a})$ and $\boldsymbol{w}(\boldsymbol{c})$. The three vectors positively span the plane. It should be noted that $\boldsymbol{w}(\boldsymbol{b})$ and $\boldsymbol{w}(\boldsymbol{c})$ also lie on the opposite side with respect to $\boldsymbol{w}(\boldsymbol{a})$.

A special case worth attention is when three of the input points, say $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ are at infinity. These are by definition collinear. As shown in Fig. 4(a), to safely discard $\boldsymbol{b}$, it is necessary that $\boldsymbol{a}$ and $\boldsymbol{c}$ are on different sides of $\boldsymbol{b}$, i.e., $\text{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{c})) = -\text{sgn}(\boldsymbol{w}(\boldsymbol{b}) \times \boldsymbol{w}(\boldsymbol{a}))$. This condition is only necessary since $\boldsymbol{b}$ may not be discarded when $\boldsymbol{w}(\boldsymbol{a})$, $\boldsymbol{w}(\boldsymbol{b})$, and $\boldsymbol{w}(\boldsymbol{c})$ positively span the plane. In this case, $\boldsymbol{b}$ and $\boldsymbol{c}$ are also on different sides of $\boldsymbol{a}$ (see Fig. 4(b)) and the convex hull of these three points covers the entire plane. Any segment must intersect this convex hull, no detection is required.

The computation of the convex hull can be summarized as Algorithm 1. The algorithm takes as input t_i^p and t_i^q, the signs of the cross products as described above. It returns true to indicate that the convex hull spans the entire space and no convex hull is computed. It returns false with ch as the computed convex hull described by a list of vertices.

3.3. Intersection detection

A convex hull is represented by a sequence of extreme points $\boldsymbol{p}_1, ..., \boldsymbol{p}_k$. A pair of adjacent points (with the last point adjacent to the first) define a facet of the convex hull. Let $\text{ri}(\cdot)$ denote the relative interior. Given a convex hull $\mathcal{C}$ whose facets are $c_1, c_2, \ldots, c_k$, a segment s can intersect the interior of the convex hull in only three cases: (1) $\text{ri}(s)$ does not intersect any c_i but lies in $\text{ri}(\mathcal{C})$, (2) $\text{ri}(s)$ intersects $\text{ri}(c_i)$ for some i such that s and c_i do not lie on the same line, and (3) $\text{ri}(s)$ intersects the common boundary point of c_i and c_{i+1} when the other boundary points of c_i and c_{i+1} lie on different sides of the line supporting s. Fig. 5 demonstrates the examples of each case.

Case 1 can be detected by enumerating all facets of the convex hull and test whether a point in $\text{ri}(s)$, say the middle point of s, lies on the same side of all facets. For case 2, we enumerate every facet and check its intersection with s. The relative interior of two segments intersects and does not

Algorithm 1 Convex hull of four points

```
                                                            ▷ Handle collinear cases
if $t_i^p = 0$ then                                         ▷ $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_i$ are collinear
    let $\boldsymbol{q}_j$ be the other end point of $q$.
    if $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_i$ positively span the plane then
        return True
    end if
    let $ch$ be the extreme points amongst $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_i$
    if $t_j^p = 0$ then                                     ▷ all points are collinear
        if $ch, \boldsymbol{q}_j$ positively span the plane then
            return True
        end if
        let $ch$ be the extreme points amongst $ch, \boldsymbol{q}_j$
        return False
    end if
end if
if $t_i^q = 0$ then                                         ▷ $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{p}_i$ are collinear
    process similarly to the case of $t_i^p == 0$
end if
                        ▷ No point is collinear, $t_i^p$ and $t_i^q$ are either $-1$ or $+1$
if $t_1^p = t_2^p$ then
    if $t_1^q = t_2^q$ then
        see Section 3.2.1
    else
        see Section 3.2.3
    end if
else
    if $t_1^q = t_2^q$ then
        see Section 3.2.3
    else
        see Section 3.2.2
    end if
end if
```

lie on the same line only when different endpoints of one segment lie on different sides of the other segment and *vice versa*. Case 3 occurs when an endpoint of facets of $\mathcal{C}$ lies on ri(s) and some part of ri(s) lies in ri($\mathcal{C}$). Let $\boldsymbol{p}_i$ be the vertex of $\mathcal{C}$ that lies on ri(s). We have to assert that $\boldsymbol{p}_{i-1}$ and $\boldsymbol{p}_{i+1}$ lie on different sides of s. Figs. 5(d) and (e) show examples of case 3. In Fig. 5(f), case 3 does not occur, although one vertex of $\mathcal{C}$ lies in ri(s). Note that all the cross products to determine which side $\boldsymbol{p}_{i-1}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}$ lie with respect to the line containing s are already computed in the testing of case 2.

To accommodate the situation in which extreme points of the convex hull $\mathcal{C}$ or endpoints of the segment s might be points at infinity, we exploit the modified cross product of Section 3.1 with the following special treatment. For the case 1 above, when s is a ray (one ordinary end point, and one end point at infinity), instead of the middle point, we take any point on the ray that is not the ordinary endpoint.

Algorithm 2 describes the intersection detection. The algorithm requires input $\boldsymbol{p}_i$, the vertices of the convex hull in counterclockwise order, and $\boldsymbol{s}_1$,$\boldsymbol{s}_2$, the end points of segment s.

3.4. Extension to any number of fingers

The method described above is specially developed to benefit three finger cases. However, the general idea based on convex hull intersection introduced in Section 2 is applicable to any number of fingers. In particular, the two convex hulls can be constructed using algorithms such as Graham's Scan algorithm and detecting whether the two convex hulls can intersect by computing their Minkowski difference and checking whether the origin is contained inside the difference. For N fingers, computing the two convex hulls takes $O(N \lg N)$ and computing their Minkowski difference and testing origin

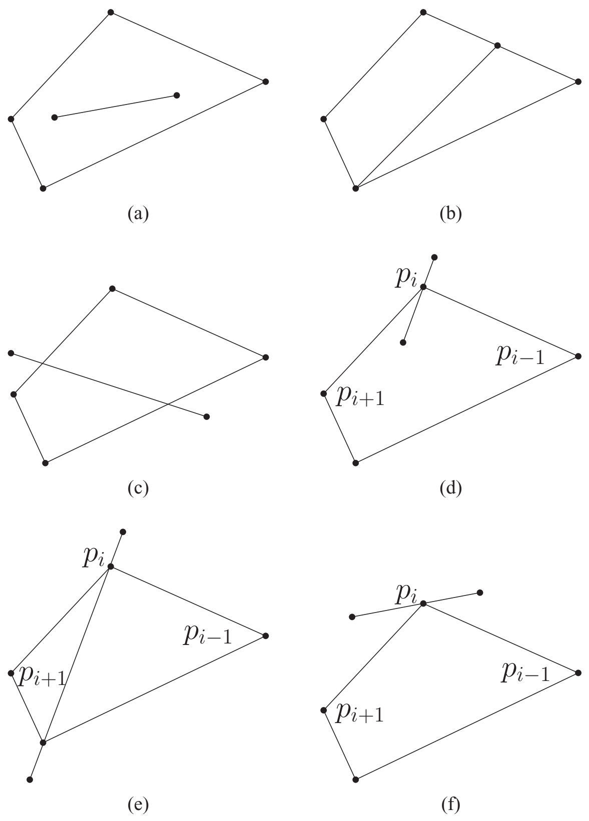

Fig. 5. Examples of convex hulls and segments. (a) and (b) satisfy the first condition, (c) satisfies the second. The last three figures represent the case when the segment intersects the common boundary of the convex hull. (d) and (e) satisfy the last condition, while (f) has no intersection.

containment takes $O(N)$, provided that the primitive wrenches are sorted by their angles when constructing the convex hulls. The standard Minkowski difference algorithm can be easily extended to handle points at infinity using the idea given in Section 3.1. An alternative is to make sure that no dual points are at infinity. This can be done by choosing the origin that does not lie on the boundary of any friction cone. A practical approach is to generate a new origin at random until the condition is satisfied. This generate-and-test scheme is more efficient than directly computing an appropriate origin since it is rare that a randomly chosen point lies exactly on the boundary of a friction cone. This test can even be omitted altogether in practice in preference of speed over completeness.

4. Comparison

Although several force-closure tests were presented in the past decade, we can rarely find a comparative study regarding the performance of these tests. New methods were often proposed without comparing any existing ones. Results are usually presented under different computational settings using different input sets. Obviously, it is difficult, if not impossible, for the reader to fairly compare and evaluate the strength and weakness across available methods. A solution is to establish a common framework for performance benchmark that allows new methods to be compared simply by publishing their results that are produced in compliance with the framework. We take the first step toward this direction. In particular, we implement selected methods in C++ programming language and test them under the same running environment using the same input. Validity of the results are assessed using reference obtained from computation with exact arithmetic. Interestingly, the comparison identifies many discrepancies never stated in the respective original works. This helps us analyze the source of errors exhibited by each method. Of course, performance depends strongly on the quality of implementation. The implementation of the selected methods is done with great

Algorithm 2 Intersection detection

```
  $\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{p}_1$
  $\boldsymbol{m} \leftarrow$ any point in ri($s$)
  if $\text{sgn}(\overrightarrow{\boldsymbol{p}_1\boldsymbol{p}_2} \times \overrightarrow{\boldsymbol{p}_1\boldsymbol{m}}) = \text{sgn}(\overrightarrow{\boldsymbol{p}_2\boldsymbol{p}_3} \times \overrightarrow{\boldsymbol{p}_2\boldsymbol{m}}) = \ldots = \text{sgn}(\overrightarrow{\boldsymbol{p}_k\boldsymbol{p}_1} \times \overrightarrow{\boldsymbol{p}_k\boldsymbol{m}})$ then
      return True                                                    ▷ Case 1
  end if
  for $i = 1 \rightarrow k + 1$ do
      $t_j^{pi} \leftarrow \text{sgn}(\overrightarrow{\boldsymbol{p}_i\boldsymbol{p}_{i+1}} \times \overrightarrow{\boldsymbol{p}_i\boldsymbol{s}_j})$
      $t_i^s \leftarrow \text{sgn}(\overrightarrow{\boldsymbol{s}_1\boldsymbol{s}_2} \times \overrightarrow{\boldsymbol{s}_1\boldsymbol{p}_i})$
  end for
  for $i = 1 \rightarrow k$ do
      if $t_1^{pi} \times t_2^{pi} < 0$ then                 ▷ $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are on different side
          if $t_i^s \times t_{i+1}^s < 0$ then
              return True                                            ▷ Case 2
          end if
          if $t_i^s == 0$ and $t_{i-1}^s \times t_{i+1}^s < 0$ then
              return True                                            ▷ Case 3
          end if
      end if
  end for
  return False
```

consideration on optimization. All the source codes and data used in our study are available upon request to the reader of the paper.

4.1. *Selected methods*

Plethora of force-closure assertion methods exist in the literature. Some methods pursue generality so that they can be applicable for any number of contacts and/or dimensions of the work space. Examples include the ray shooting method by Liu,[6] the Q Distance method in Zhu and Wang,[7] and a quantitative test by Zhu and Ding,[8] who also suggest in ref. [10] the use of GJK, a well-known collision detection algorithm in computer graphics, as a general test of force closure. Han *et al.* proposes a method that can operate on a wide range of finger type using a convex optimization technique.[21] The method of Brost and Mason[16] is also applicable with any number of contacts, although it cannot be used in a 3D grasp, except for the case of three-finger, where it is shown in Li *et al.*[15] that the method can be easily adopted. Cornella and Suarez[22] propose a condition for planar force-closure grasps, which is applicable with any number of fingers, to identify independent contact region of a polygonal object. Related condition is also presented for a discretized object in Cornella and Suarez.[23]

Many researchers derive methods that are specialized to a particular number of contacts and/or dimensions of the workspace. These methods usually exploit *a priori* knowledge of the specialized situation to achieve better performance. Example includes the disposition algorithm of Li *et al.*,[15] which is very fast in assertion of force closure of three fingers. Tung and Kak[24] propose a geometrical approach to compute two-finger force-closure grasp of a polygonal object. Ponce *et al.*[14,25,26] derive several conditions for two- to four-finger grasps and present algorithms for synthesizing grasps. Grasp synthesis is also considered in many other works such as Pollard and Wolf..[27] Cheong *et al.*[28,29] uses a representation similar to force-dual representation to compute all 2D force-closure grasp.

Among the aforementioned methods, many are presented as explicit tests, while several are implicitly integrated in their grasp synthesis methods. We select seven recent methods that distinguish themselves as stand-alone tests of force closure. These methods and our novel implementation will be compared in this section. The first method is the method of Brost and Mason,[16] in which we provide an efficient implementation based on convex hull intersection. The next one is the disposition method of Li *et al.*,[15] which is included as a representative of specialized methods for three-finger grasps. The other five methods are general methods which can accommodate any number of fingers in any dimension. One of them is GJK in van den Bergen.[30] The other two methods, i.e., the ray shooting method of Liu[6] and the quantitative test of Zhu and Ding,[8] are based on linear programming. Another ray shooting-based method by Zheng and Chew[11] is also included. Finally, the standard convex hull routine, QHull in Barber *et al.*[31] is chosen. In our comparison, only QHull is deployed with arbitrary precision arithmetic so that its results can be used as reference.

4.1.1. Straightforward implementation of Brost and Mason. The work of Brost and Mason[16] provides the fundamental of a dual representation of force. The work suggests that force closure can be asserted in the force-dual representation. As originally described in Brost and Mason, the test is simply to check whether one point of a given sign lies strictly inside the triangle defined by three points of the other sign, or whether two segments of different signs intersect. It was presented originally as an instruction for drawing and inspection. No explicit computational implementation detail was given. Nevertheless, we derive a method that we believe to be a straightforward implementation. To do so, the instruction is translated into an implementation using computational geometry algorithms, e.g., checking for point containment is implemented as an intersection test with convex hull.

More precisely, the straightforward implementation (as noted toward the end of Section 2) is as follows: We compute two convex hulls, one for positive dual points and the other for negative dual points, and test whether their interior intersects. We use Graham's scan algorithm for convex hull construction and apply the intersection detection described in Section 3.3.

It should be noted that this method differs from our novel implementation (Section 3). We include the straightforward implementation of Brost and Mason[16] to emphasize that our implementation actually outperforms the straightforward implementation.

4.1.2. Disposition method of Liet al. [15] This method distinguishes itself from the others because it specifically takes into account the nature of three-finger grasps. It can quickly identify a certain class of non-force-closure grasps. In short, this method removes unnecessary portion of the force cones and show that, with the trimmed force cones, to achieve force closure, it is necessary and sufficient that there exists an intersection point of the boundary of two force cones that lies inside the third cone. The disposition of the cone requires only a few algebraic calculations (see Fig. 5 in Li *et al.*[15]).

4.1.3. GJK algorithm. Invented by Gilbert *et al.*,[9] GJK algorithm is a well-known collision detection algorithm. It determines the closest distance between two convex polyhedra. A variation of the GJK algorithm also supports a binary query; instead of reporting a distance between two objects, it decides whether the two objects collide with each other. Taking the Minkowski difference of two convex polyhedra as an input, the algorithm determines whether the origin lies inside the convex hull of the Minkowski difference. As suggested in Zhu *et al.*,[10] this query is suitable for a qualitative test of force closure. In this work, we use the GJK separating-axis algorithm presented in van den Bergen..[30] The separating-axis algorithm is an improved version of the GJK algorithm and is designed for binary query. The algorithm can quickly determine whether the origin lies inside the convex hull of wrenches. However, it should be noted that the separating-axis algorithm cannot distinguish whether the origin lies on the boundary or in the interior of the convex hull. The original GJK algorithm in Gilbert *et al.*[9] is an iterative algorithm that runs in linear time with respect to the number of wrenches.[32] The GJK separating-axis algorithm improves on this by reducing the running time used in each iteration.

4.1.4. Ray shooting algorithm of Liu. Liu[6] proposes a simple test of force closure by drawing a ray from an arbitrary point P within the convex hull of primitive wrenches to the origin and calculate the intersection X between the ray and the boundary of the convex hull. Force closure is achieved if and only if the distance between the point P and the origin is smaller than the distance between the point P and the intersection point X. This condition transforms the problem into the ray shooting problem which is extensively studied in the field of Computational Geometry and is solvable by linear programming. This test is used in many subsequent works of Liu.[33,34]

4.1.5. Numerical ray shooting algorithm of zheng and Chew. Like the work of Liu,[6] this force-closure test is also based on ray shooting computation. The difference is that instead of using linear programming to solve the ray shooting problem, Zheng and Chew[11] introduced an efficient numerical solution based on an iterative method for computing the distance between a point and a convex set. Like most numerical methods, a proper termination tolerance ϵ is required.

4.1.6. Q Distance of Zhu and Wang. Zhu and Wang[7] propose the concept of Q distance for analysis and synthesis of force-closure grasps. The Q distance is a grasp quality measure, calculated for the convex hull of primitive wrenches, denoted by A. Given a *ruler* convex polyhedron Q, which contains the origin, the Q distance determines a positive scaling factor of Q (if exists) such that the scaled Q is entirely contained in A. The Q distance consists of two subfunctions, Q^+ distance and Q^- distance. The Q^+ distance determines whether there exists a positive scaling factor that causes a nonempty intersection between A and the scaled Q. This is required for the calculation of Q^-, which actually determines the Q distance. It can be formulated as a set of linear programming problems. In our experiments, the ruler polyhedron Q is set to be a regular tetrahedron to achieve maximum computation speed.

4.1.7. Quick hull with arbitrary precision arithmetic. A straightforward approach to force-closure testing is to directly compute the convex hull of primitive wrenches. The convex hull can be described by the intersection of its bounding half spaces. To test whether the origin lies strictly inside the convex hull, we test whether the origin lies in the interior of every bounding half space (described by the corresponding bounding facet). We use the Quick Hull algorithm in Barber *et al.*[31] to compute the convex hull and its bounding facets. The algorithm is implemented using the CGAL library.[35] The internal computation of the library relies on an arbitrary precision number support from GNU Multiple Precision Arithmetic Library.[36] Employing the arbitrary precision arithmetic avoids the problem of roundoff errors encountered when using ordinary floating point operations. Of course, using exact arithmetic requires tremendous computing power. The method is therefore selected only as a reference method for validating the results.

4.2. Numerical examples and results on three fingers

We compare the running time of our implementation with the others. In total, eight methods are compared. One is our implementation given in Section 3 and the others are those from Section 4.1. Nine test objects, shown in Fig. 6, are used. For each object, 200 contact points are randomly sampled from its boundary to form a discrete point set. Specifically, we define $p(t)$ as the parameterized path that represents the boundary. Each point is sampled by drawing t from a uniform distribution $U(0, 1)$. For a polygonal object, care is taken not to sample a point at the vertices.

Using our method and that from Section 4.1, every combination of three contact points from each set is tested whether it achieves force closure. The friction coefficient of 1120/6351 is used, which yields a friction cone with half cone angle at approximately 10.001°. Note that the friction coefficient used in every experiment is a rational number so that the sine and cosine values of the corresponding half friction cone angle take on rational numbers. This is to ensure that the reference method is free from floating point operations and can generate true results. All methods are implemented in C++, running on an Intel Core i7 3.4 GHz machine with 8-GB RAM. Of all the methods, only the Quick Hull employs arbitrary precision arithmetic in order to ensure the validity of the result. The other methods use primitive data type (64-bit floating point).

Table I presents the running time of each method to test all $C_{200,3} = 1{,}313{,}400$ grasps. Each row represents the result from each test object. The first column shows the actual running time of our algorithm, labeled as "NS." The remaining columns show the percentage between the actual running time of a particular method and our method. The columns labeled BM, LLC, GJK, Liu, ZC, and ZW represent the methods in Sections 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5, and 4.1.6, respectively. The Quick Hull method, which is implemented with arbitrary precision arithmetic, is not included since the calculation obviously requires tremendous amount of time (more than 1000 folds of our method). The result of the Quick Hull method is included in the experiment only for the purpose of result validation, which will be discussed afterward.

The result shows that our method is the fastest one, taking less than two-third of the second fastest, the BM method. On average, each force-closure test by our NS method takes less than 0.2 microseconds. The results show that there is a large gap between the first three methods (NS, BM, and LLC) and the remaining methods. The major difference is that NS, BM, and LLC operate in 2D space, while the remaining methods operate in 3D space. We analyze these faster three methods in terms of arithmetic operations they perform. In the case of three fingers, it is unnecessary to analyze the performance in terms of asymptotic complexity since the input size is fixed. We instead count the number of major operations performed by each method. The major operations considered here are divisions and cross product operations.

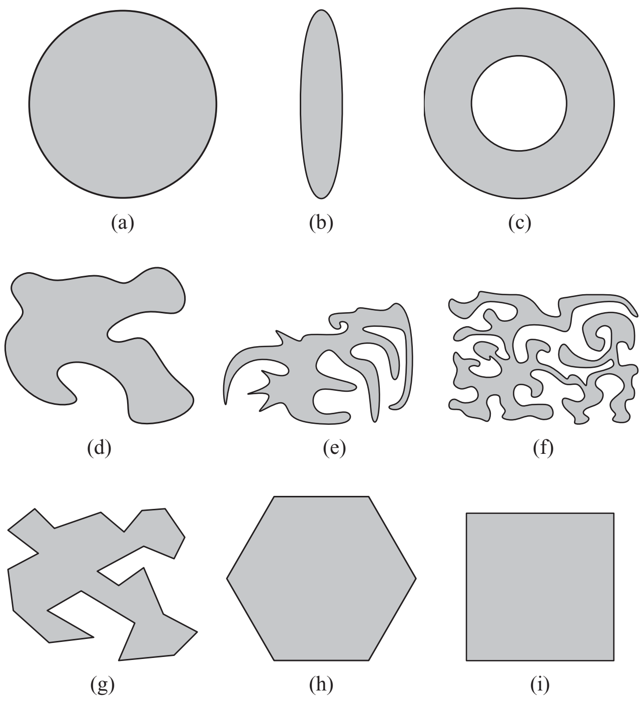

Fig. 6. Test objects.

Table I. Actual running time. The friction coefficient is 1120/6351. The running time of the comparing methods is given as percentages with respect to that of NS.

		Running time (%)					
Test objs.	Time of NS (s)	BM	LLC	GJK	Liu	ZC	ZW
(a)	0.243	179	280	942	2921	3363	6445
(b)	0.252	161	182	759	2982	2924	4009
(c)	0.254	169	220	879	2768	3260	5750
(d)	0.271	139	192	763	2703	2792	3688
(e)	0.273	130	203	732	2672	2750	3583
(f)	0.278	132	195	712	2750	2673	3511
(g)	0.267	144	196	770	2722	2849	3813
(h)	0.253	170	233	921	2887	3201	5433
(i)	0.259	143	219	841	2858	2902	4994
Avg.	0.261	152	213	813	2807	2968	4581

We first compare our method against the BM method. Our method takes eight divisions to convert wrenches into the force dual representation, comparing with 12 divisions required by the BM method. From our preliminary experiment, division takes more than half of the total running time of the BM method. Another speedup of our method over BM comes from the grouping that always yields four points to construct a convex hull. Convex hull of four points can be constructed by our brute force algorithm using four cross products. This is obviously more efficient than the BM method, where convex hull is constructed using Graham's scan, which requires sorting. A four-point convex hull also reduces the number of cross products required to detect the intersection. At worst, it takes $4 + 16$ cross products, where four of them is to check whether a middle point lies inside the convex hull, while the remaining 16 comes from the intersection checking of a segment against the boundary of the convex hull. For the BM method, it is possible that the dual representation results in two groups of three points each. Hence, we have to identify intersection of 3^2 segments, which can be done in $9 \times 4 = 36$ cross products. Moreover, we have to test whether a point lies in the convex hull of each

Table II. Number of errors. The friction coefficient is 1120/6351. The column labeled "QHULL" gives the total number of force-closure grasps computed using arbitrary precision arithmetic system.

		Number of errors						
Test objs.	QHULL	NS	BM	LLC	GJK	Liu	ZC	ZW
(a)	538,395	-	-	-	-	-	-	-
(b)	137,598	-	-	-	2386	-	-	-
(c)	463,537	-	-	-	56	-	-	-
(d)	146,353	-	-	8	1676	-	-	-
(e)	140,013	-	-	16	10,673	-	-	-
(f)	122,081	-	-	21	6021	10	-	-
(g)	163,627	-	-	-	777	-	-	-
(h)	369,807	-	-	-	4	10,762	-	-
(i)	315,531	-	-	-	-	62,266	-	-
Total	2,396,942	-	-	45	21,593	73,038	-	-

group, which takes four cross products. Hence, BM takes at most $4 + 4 + 36 = 44$ cross products. It is clear that our method uses fewer divisions and cross products than BM.

The BM method outperforms the LLC method by a noticeable fraction. LLC calculates the position of intersection between two lines. Determining the intersection point between two lines is an expensive operation comparable with six cross products and two divisions. There are at most six pairs of lines to be taken into account, which amounts to 12 divisions and 36 cross products in the worst case. Moreover, LLC also requires some additional calculation. In the worst case, it requires seven cross products to trim one boundary of friction cone and there are at most six boundaries to be trimmed. This amounts to 42 cross products. In conclusion, LLC uses at most $36 + 42 = 78$ cross products. This supports the empirical result that LLC takes more running time than BM.

Table I represents only the running time, taking no account of accuracy. Every selected method is essentially numerical in nature. Thus, unless an arbitrary precision arithmetic system is used in the implementation, the result is inevitably affected by roundoff error of the floating point representation. In Table II, we compare the validity of the result with the reference method, the Quick Hull algorithm using the arbitrary precision arithmetic system. For each method, we count the number of errors of the solution. The first column shows the number of force closure-grasps reported by the reference method, labeled "QHULL." Each of the remaining columns shows the number of errors of the corresponding method. It is to be noted that this is a limited comparison considering only the validity of the result, a complete numerical sensitivity is not addressed, and it is worth further study.

Table II shows that only our method, the BM method, the ZC method, and the Q Distance method report solutions that totally agree with the reference method. On the other hand, results from LLC, GJK, and Liu dissent from the reference. We examine the cause of these errors. Note that the ZC method also suffers from some errors when the threshold ϵ is set greater than 7×10^{-3}.

First, we consider the disposition method of Li *et al.*[15] Unlike the other cases, most of the conflicting results of the disposition method stem not from the floating point inaccuracy but from the input that is not covered by the case analysis in the proof of the algorithm. In brief, this method considers every permutation of the three contact points, using one contact point (at a time) as the origin and examining the sign of torques produced by the remaining two contact points. If one of the two contact points can produce only positive torque (resp. negative torque), part of the force cone of the other contact point that produces positive torque (resp. negative torque) is disposed because that portion of the cone obviously cannot help forming a force-closure grasp. It is stated in Proposition 3 of Li *et al.* that after all permutations of friction cones have undergone such disposition, existence of the intersection of the disposed double-side friction cones is a necessary and sufficient condition of force closure.

The problem arises when no disposition can take place, i.e., when each cone can produce both positive and negative torques no matter which contact point is chosen to be the origin. The proof of Proposition 3 in Li *et al.*[15] states in its second case analysis that this event can occur only when each pair of contact points lies on the same side of the double-side friction cone of the remaining contact point. The proof proceeds to derive that a two-finger force-closure grasp can always be found when

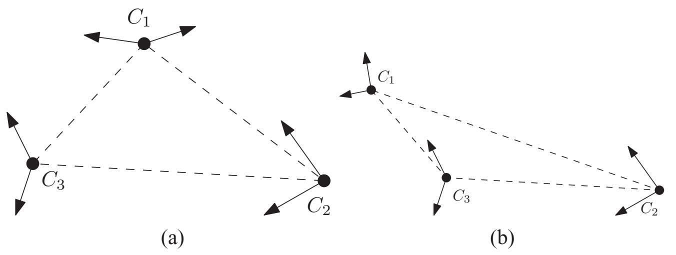

Fig. 7. (a) The case covered by the disposition method. Both contact points C_i and C_j lie on the same side of the double-side friction cone of C_k for any permutation of i, j, k. (b) The case not covered by the disposition method. C_1 lies in the positive friction cone of C_3, while C_2 lies in the negative friction cone. It is clear that all forces point leftward and it does not achieve force closure.

this event is encountered. The case analysis is unfortunately not complete: when the aforementioned event occurs, it is *not* necessary that each pair of contact points lies on the same side of the double-side friction cone of the remaining contact point. An example is shown in Fig. 7(b), where each cone can produce both positive and negative torques, but C_1 and C_2 lie on different sides of the double-side friction cone of C_3. In this example, it is obvious that no force-closure grasp can be formed (all forces in one half plane) as opposed to the report in error by the algorithm that a two-finger force-closure grasp is found. For our test objects, many relatively simple shapes do not exhibit this fallacy. This case is more likely to occur when there are three contact points that lie almost collinear and their normal directions point relatively toward the same direction. Such behavior appears noticeably in the two complex objects (e)–(f), which could be identified from the fault of the result.

Next, we consider the errors of GJK. These errors are due to the numerical nature of the algorithm. The GJK algorithm is sensitive to the shape of the convex hull of wrenches. It should be noted that even though wrench representation merges the notion of force and torque together, the unit of each component in a wrench is different. In theory, arbitrary change in the scale of each unit has no effect to the force-closure property of the grasp. For example, changing the distance measurement of an object results in change of position scaling of contact points and subsequently changes the torque portion of the wrench while force portion remains intact. Scaling up the object results in a larger torque, while scaling down the object reduces the torque. However, the direction of the force cone does not change. When the object is enlarged, the shape of the convex hull becomes flatter and thinner. This has a great impact on GJK algorithms.

All of our test objects have the diameter in the range of several hundred units, while the normal direction of a contact point is a unit vector. This exhibits the aforementioned problem of scaling. To verify this conjecture, we scale down each object by a hundred times, effectively rendering the convex hull to become more rounded. The result is that the number of errors decreased significantly.

Finally, we consider the ray shooting method of Liu.[6] From all test data, it can be seen that the results from the ray shooting method mostly agree with the reference method, except for the hexagon and the rectangle cases (rows (h) and (i) in the table). This is because several special cases of the convex hull of wrenches can be found in these objects. Generally, the convex hull is a full rank polyhedron in 3D space. However, it is possible that the convex hull can be degenerated. A common degenerate case occurs when a convex hull is not at full rank and does not contain the origin. This problem is also identified and an enhance version that is capable of handling the degenerate case is presented in ref. [37].

This case can occur very easily when all contact points lie in the same linear side of an object (see Fig. 8(a)). In such a case, all contact points have the same normal direction, but different positions. These contact points yield two groups of wrenches, each has the same force, but different torque (see Fig. 8(b)). The convex hull of the wrenches obviously degenerates.

In theory, the ray shooting method can distinguish this degeneration. Specifically, when the problem is transformed into a linear programming problem, the degenerate case yields an indeterminate result. However, this is always true only when the exact arithmetic is used. Under a limited precision floating point implementation, the linear programming could overlook this indeterminate case.

Many numerical errors encountered by each method stem from the boundary case where the grasp is on the borderline of achieving or not achieving force closure. For several objects, including the

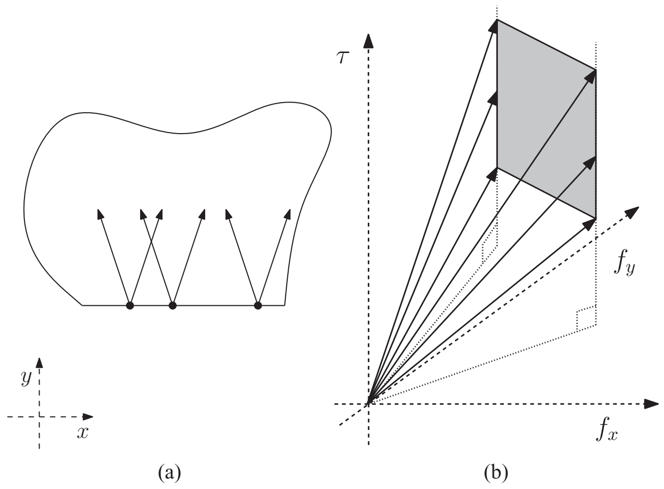

Fig. 8. (a) Degeneration case commonly occurred in a linear face of an object. (b) The associated wrench form a 2D polygon not containing the origin.

polygons with parallel edges such as the test objects (h) and (i), the boundary cases may be frequently encountered. This is because the object is likely to yield groups of three coplanar wrenches (three collinear points in the force-dual representation).

Our method, the BM method, and the Q Distance method do not suffer from such boundary cases because they are deliberately tailored to prevent the problem. For Q Distance, one entire instance of linear programming is derived to eliminate the case of equilibrium and non-equilibrium. This proves to be useful from our experiment because the problematic case of objects (h) and (i) is easily detected. For our method and the BM method, the issue is handled by the intersection detection describe in Section 3.3. Merely checking whether two line segments intersect cannot distinguish the case in Figs. 5(d) and (f), while checking whether endpoints lie inside the convex hull might miss the case in Fig. 5(b). These cases are more likely to occur when three points are collinear. Case (c) in Section 3.3 and the use of middle point instead of the boundary points reduce the chance on these boundary cases. From our preliminary experiment, using a simple checking instead of the method in Section 3.3 yields a significant number of errors. Careful attention to the geometric intricacy of the problem is crucial to achieving a robust implementation.

We also set up another experiment to demonstrate the effect of larger friction coefficient on each method, especially the errors of LLC under larger friction coefficient. The experiment is repeated using a fairly large friction coefficient of 3/4, resulting in a friction cone with half angle at approximately 36.87^{o}. Table III shows the actual time used by each algorithm, and Table IV reports the number of errors of each algorithm in the same manner as Tables I and II, respectively.

A larger friction coefficient yields more force-closure grasps. This reduces the benefit of the disposition method that can speedily rejects certain class of non-force-closure grasps. A similar slowdown also applies to the ZC method. The difference between the time used by our method and the disposition method is increased. Our method uses approximately half the time used by the disposition method and less than one-third of the GJK method. Moreover, when the friction cone gets larger, it is more likely that the case in Fig. 7(b) occurs. This can be seen from a significantly larger number of errors of LLC.

5. 3D Three-Finger Grasp

Our method mentioned above can be applied for testing three-finger grasp in 3D. The key idea can be credited to Proposition 5 in Li *et al.*,[15] where it is proven that three frictional 3D contact points achieve force closure if and only if there exists a planar force-closure grasp on the plane through the three contact points such that the three contact forces achieving force closure are constrained to lie in the intersection between the 3D friction cones and the plane (see Fig. 9).

Table III. Actual running time. The friction coefficient is 3/4. The running time of the comparing methods is given as percentages with respect to that of NS.

		Running time (%)					
Test objs.	Time of NS (s.)	BM	LLC	GJK	Liu	ZC	ZW
(a)	0.283	144	343	1086	2402	3139	8148
(b)	0.280	135	246	906	2589	3067	5186
(c)	0.293	134	255	931	2153	3112	6366
(d)	0.319	122	284	945	2287	2847	5475
(e)	0.327	117	266	882	2224	2723	5000
(f)	0.350	116	260	864	2100	2535	4643
(g)	0.315	125	274	899	2239	2895	5307
(h)	0.294	145	306	1081	2330	3235	7185
(i)	0.301	139	290	993.2	2435	3160	6223
Avg.	0.307	131.4	280.9	954.5	2307.0	2968	5948.6

Table IV. Number of errors. The friction coefficient is 3/4. The column labeled "QHULL" gives the total number of force-closure grasps computed using arbitrary precision arithmetic system.

		Number of errors						
Test objs.	QHULL	NS	BM	LLC	GJK	Liu	ZC	ZW
(a)	971,717	-	-	-	-	-	-	-
(b)	443,162	-	-	-	6385	-	-	-
(c)	798,576	-	-	-	160	-	-	-
(d)	626,034	-	-	429	2717	-	-	-
(e)	549,611	-	-	4400	4853	-	-	-
(f)	543,061	-	-	4100	5077	5	-	-
(g)	607,532	-	-	1602	2081	-	-	-
(h)	859,814	-	-	-	679	11,323	-	-
(i)	713,168	-	-	-	-	77,077	-	-
Total	6,112,675	-	-	10,531	21,952	88,405	-	-

This proposition is general, and can be applied to any frictional 3D three-finger grasp. The only assumption is that the three contact points are not collinear. When the three contact points are collinear, a torque around the collinear line cannot be countered by any force from the three contact points.

Intuitively speaking, the key observation of this proposition can be stated as follows: If the three 3D contact points form a 2D force-closure grasp on the plane through the three contact points, to achieve 3D force closure we only have to show that the contact points can produce force and torque along the axis perpendicular to the plane. The proof of the proposition simply shows that when its conditions are satisfied, positive spanning along the perpendicular axis is automatically guaranteed. With this proposition, we can compute the corresponding planar grasp through the three contact points and directly apply our method to test whether the planar grasp achieves force closure.

We propose a method for computing all combinations of three-finger grasps, given a set of contact points in 3D. Our method exploits a necessary condition for force closure in ref. [15] that the plane passing through the three contact points has to intersect the three associated friction cones. Our idea is to use this necessary condition to prune off pairs of contact points whose friction cones do not allow intersection with any possible plane through these points. How this idea is translated into an algorithm is described as follows.

Let us consider two contact points $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$, and their friction cones F_a and F_b. Clearly, the force $\boldsymbol{f}_a \in F_a$ uniquely defines the plane through $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ that contains the force (Fig. 10(a)). Therefore, all planes through $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ that are possible to intersect F_a (i.e., to contain some force of F_a) are the ones limited by the two tangent planes of the friction cone F_a (Fig. 10(b)). Let us denote this set of planes by $\Pi_{a,b}$. The set of planes $\Pi_{b,a}$ is also defined in the same manner with respect to $\boldsymbol{p}_b$, $\boldsymbol{p}_a$,

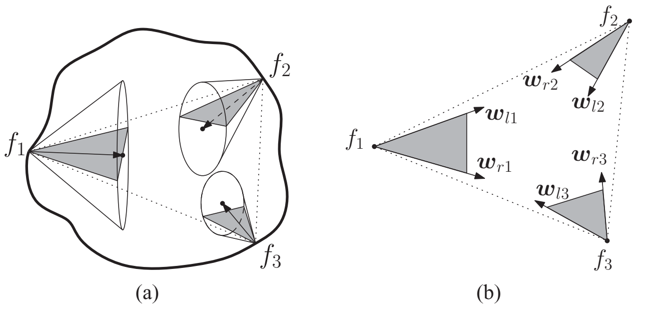

Fig. 9. (a) Three 3D friction cones from three contact points, f_1, f_2, and f_3. The 3D objects are represented by the thick lines. The dotted line is the plane defined by the three contact points. The shaded region is the intersection between the plane and the friction cones. (b) The corresponding planar force-closure grasp problem. It is to be noted that each friction cone in this case might have different half angle. Nevertheless, the wrenches associated with each contact point form a fan in the wrench space and can be described by positive combinations of two primitive wrenches.

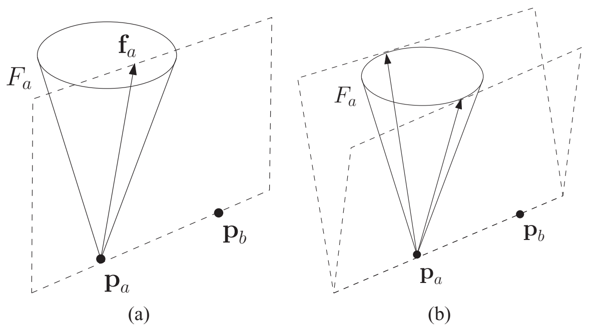

Fig. 10. A set of planes formed by two contact points and one friction cone.

and F_b. Note the special case that all possible planes cover the entire $\mathbb{R}^3$ when $\boldsymbol{p}_b$ lies in F_a, or $\boldsymbol{p}_a$ lies in F_b.

With $\Pi_{a,b}$ and $\Pi_{b,a}$ identified, the set of feasible planes formed by $\boldsymbol{p}_a$, $\boldsymbol{p}_b$, F_a, and F_b is simply $\Pi_{a,b} \cap \Pi_{b,a}$. If this intersection is empty, the pair $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ can be safely rejected from force-closure test. Clearly, for the third contact point $\boldsymbol{p}_c$ to be feasible, it has to lie in a plane from $\Pi_{a,b} \cap \Pi_{b,a}$. Since a plane from this set is guaranteed to intersect F_a and F_b, we only need to check whether the friction cone F_c intersect the plane through the three contact points. If the intersection is not empty, we compute the intersection between this plane and the three friction cones so that the problem can be reduced into a 2D three-finger force-closure test, which can be solved by our proposed implementation in Section 3.

6. Numerical Example in 3D

The objective of this section is to demonstrate the efficiency of our proposed force-closure test in 3D setting and the efficiency of our pruning method. All experiments are conducted on an Intel Core i7 3.4 GHz machine with 8-GB RAM. Half angle of a friction cone is set at 10°.

The first example compares our force-closure test with the method of Gilbert *et al.*,[9] which is known to be relatively fast, and the methods by Zhu and Wang[7] and Zheng and Chew,[11] which are more precise than that given by Gilbert *et al.*[9] In this first experiment, no pruning is performed. Given three contact points, our method requires as inputs the intersections of their friction cones and the plane they form whereas the other methods for force-closure test directly use the given contact points as inputs. We randomly pick 10^7 triples of contact points from the objects in Fig. 11 and test whether each triple achieves force closure. We measure the total time needed to test all triples. For our method, the time spent for intersecting the friction cones and the plane is also taken into account. For the methods in refs. [9 and 11] and ref. [7], a quadratic friction cone is linearized by a 16-side

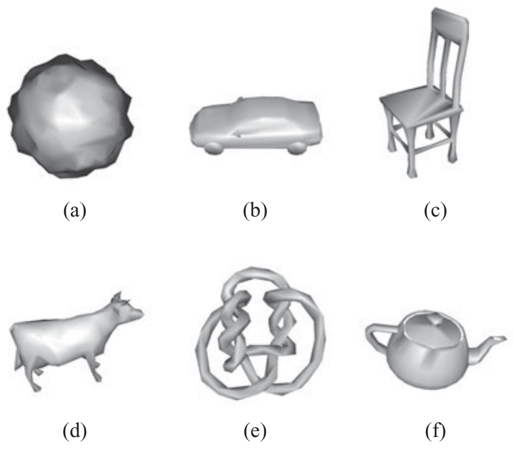

Fig. 11. Test objects.

Table V. Results of three-finger grasp computation without pruning.

	Time (s)			
Fig.	GJK	ZC	ZW	NS
(a)	167.67	391.11	1098.96	37.54
(b)	90.46	300.99	995.03	35.09
(c)	65.84	287.94	973.84	34.14
(d)	68.07	303.33	994.90	33.77
(e)	67.02	290.84	969.43	33.80
(f)	134.18	349.67	1072.47	36.61

Table VI. Results of three-finger grasp pruning.

		With pruning		Without pruning	
Fig.	#FC	# Exec	Time (s)	# Exec	Time (s)
(a)	133,150	402,136	6.82	10,586,800	33.99
(b)	22,370	140,789	3.08	10,586,800	30.76
(c)	8525	79,500	2.24	10,586,800	30.38
(d)	11,331	74,492	2.12	10,586,800	30.41
(e)	7517	64,359	2.10	10,586,800	30.72
(f)	118,177	36,8286	6.09	10,586,800	33.40

pyramid. The results of this experiment given in Table V obviously show that our method is much faster, by at least a factor of 2.

The next comparison is to demonstrate the efficiency of our pruning technique. Approximately 500 contact points are randomly generated from each test object. Every triple of these contact points has to be tested for whether it achieves force closure. We compare the performance of our method with and without pruning. As described above, we expect our pruning method to reject a large number of couples or triples of contact points that are not able to satisfy the necessary condition of force closure. The results given in Table VI confirms that when pruning is applied, the number of force-closure test executions is greatly reduced (approximately by a factor of 25) and the running time is reduced at least by a factor of 5.

7. Conclusion and Discussion

We have introduced an efficient implementation for force-closure testing of three frictional contact points based on the condition derived from the graphical representation of frictional contacts originally given by Brost and Mason.[16] Our focus is on developing an efficient implementation that properly deals with the geometric intricacy of the problem. Performance of the test has been exhibited from both theoretical and empirical points of view. Besides computational speed, the comparison with the existing methods takes into account the validity of results. This is necessary because the implementation of each method using native data type of a modern computer, which is of limited precision by its nature, inevitably suffers from the problem of roundoff errors. Our verification is based on reference results computed using exact arithmetic. This verification scheme has never been presented before in the literature. It identifies errors and reveals true robustness of force-closure testing.

Although the method introduced in this paper is described explicitly for planar grasps, it is shown that the method can easily be applied to 3D grasps as well. A method for computing all 3D three-finger grasps is proposed utilizing the necessary condition that quickly rejects several non-force-closure grasps and test only the remaining ones. It is shown that this pruning method along with our proposed implementation greatly reduces the time used to test 3D three-finger force-closure grasps.

References

1. J. Salisbury, "Kinematic and Force Analysis of Articulated Hands." *PhD thesis* (Stanford University, Stanford, CA, 1982).
2. J. Salisbury and B. Roth, "Kinematic and force analysis of articulated hands," *ASME J. Mech. Transm. Autom. Des.* **105**, 33–41 (1982).
3. A. Bicchi and V. Kumar, "Robotic Grasping and Contact: A Review," **In**: *IEEE International Conference on Robotics and Automation*, San Francisco, CA (2000) pp. 348–353.
4. A. Bicchi, "Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity," *IEEE Trans. Robot. Autom.* **16**(16), 652–662 (2000).
5. C. Ferrari and J. Canny, "Planning Optimal Grasps." **In**: *IEEE International Conference on Robotics and Automation* (1992) pp. 2290–2295.
6. Y.-H. Liu, "Qualitative test and force optimization of 3-D frictional form-closure grasps using linear programming," *IEEE Trans. Robot. Autom.* **15**(1), 163–173 (1999).
7. X. Zhu and J. Wang, "Synthesis of force-closure grasps on 3-D objects based on the q distance," *IEEE Trans. Robot. Autom.* **19**(4), 669–679 (2003).
8. X. Zhu and H. Ding, "Computation of force-closure grasps: An iterative algorithm," *IEEE Trans. Robot.* **22**(1), 172–179 (2006).
9. E. G. Gilbert, D. W. Johnson and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimentional space," *IEEE Trans. Robot. Autom.* **4**, 193–203 (1988).
10. X. Zhu, H. Ding and S. K. Tso, "A pseudodistance function and its applications," *IEEE Trans. Robot. Autom.* **20**(2), 344–352 (2004).
11. Y. Zheng and C.-M. Chew, "A Numerical Solution to the Ray-Shooting Problem and Its Applications in Robotic Grasping," *IEEE International Conference on Robotics and Automation* (2009).
12. Y. Zheng and C.-M. Chew, "Distance between a point and a convex cone in n-dimensional space: Computation and applications," *IEEE Trans. Robot.* **25**(6), 1397–1412 (2009).
13. V.-D. Nguyen, "Constructing force-closure grasps," *Int. J. Robot. Res.* **7**(3), 3–16 (1988).
14. J. Ponce and B. Faverjon, "On computing three-finger force-closure grasps of polygonal objects," *IEEE Trans. Robot. Autom.* **11**(6), 868–881 (1995).
15. J.-W. Li, H. Liu and H.-G. Cai, "On computing three-finger force-closure grasps of 2-D and 3-D objects," *IEEE Trans. Robot. Autom.* **19**(1), 155–161 (2003).
16. R. C. Brost and M. T. Mason, "Graphical analysis of planar rigid-body dynamics with multiple frictional contacts," **In**: *International Symposium on Robotics Research*, Tokyo, Japan (MIT Press, Cambridge, MA, 1989) pp. 293–300.
17. M. T. Mason, *Mechanics of Robotic Manipulation*, Intelligent Robotics and Autonomous Agents series (MIT Press, Cambridge, MA, 2001).
18. B. Mishra, J. Schwartz and M. Sharir, "On the existence and synthesis of multifinger positive grips," *Algorithmica Spec. Issue Robot.* **2**(4), 541–558 (1987).
19. R. Bix, *Topics in Geometry* (Academic Press, Waltham, MA, 1994).
20. J. E. Goodman and J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry* (CRC Press, Boca Raton, FL, 1997). ISBN 0-8493-8524-5.
21. L. Han, J. C. Trinkle and Z. X. Li, "Grasp analysis as linear matrix inequality problems," *IEEE Trans. Robot. Autom.* **16**(6), 663–674 (2000).
22. J. Cornella and R. Suarez, "Fast and Flexible Determination of Force-Closure Independent Regions to Grasp Polygonal Objects," **In**: *IEEE International Conference on Robotics and Automation*, Barcelona, Spain (2005) pp. 778–783.

23. J. Cornella and R. Suarez, "A New Framework for Planning Three-Finger Grasps of 2D Irregular Objects," **In**: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China (2006) pp. 5688–5694.
24. C.-P. Tung and A. C. Kak, "Fast construction of force-closure grasps," *IEEE Trans. Robot. Autom.* **12**(4), 615–626 (1996).
25. B. Faverjon and J. Ponce, "On Computing Two-Finger Force-Closure Grasps of Curved 2D Objects," **In**: *IEEE International Conference on Robotics and Automation*, Sacramento, CA (1991) pp. 424–429.
26. J. Ponce, S. Sullivan, J.-D. Boissonnat and J.-P. Merlet, "On Characterizing and Computing Three- and Four-Finger Force-Closure Grasps of Polyhedral Objects," **In**: *IEEE International Conference on Robotics and Automation*, Atlanta, GA (1993) pp. 821–827.
27. N. S. Pollard and A. Wolf, "5 Grasp Synthesis from Example: Tuning the Example to a Task or Object," **In**: *Springer Tracts in Advanced Robotics*, Vol. 18 *Multi-Point Interaction with Real and Virtual Objects* (F. Barbagli, D. Prattichizzo and K. Salisbury, eds.) (Springer, Berlin, Germany, 2005) pp. 77–90. ISBN 978-3-540-26036-3.
28. J.-S. Cheong and A. F. van der Stappen, "Output-Sensitive Computation of All Form-Closure Grasps of a Semi-Algebraic Set," **In**: *IEEE International Conference on Robotics and Automation*, Barcelona, Spain (2005) pp. 772–778.
29. J.-S. Cheong, H. J. Haverkort and A. F. van der Stappen, "On computing all immobilizing grasps of a simple polygon with few contacts," *Algorithmica* **44**, 117–136 (2006).
30. G. van den Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graph Tools* **4**(2), 7–25 (1999).
31. C. B. Barber, D. P. Dobkin and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Soft.* **22**(4), 469–483 (1996).
32. C. J. Ong and E. Gilbert, "The Gilbert-Johnson-Keerthi Distance Algorithm: A Fast Version for Incremental Motions," **In**: *Proceedings of 1997 IEEE International Conference on Robotics and Automation*, Vol. 2 (1997) pp. 1183 –1189. doi:10.1109/ROBOT.1997.614298.
33. D. Ding, Y.-H. Liu, M. Y. Wang and S. Wang, "Automatic selection of fixturing surfaces and fixturing points for polyhedral workpieces," *IEEE Trans. Robot. Autom.* **17**(6), 833–841 (2001).
34. 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 Trans. Robot.* **20**(5), 805–816 (2004).
35. The CGAL Project, *CGAL User and Reference Manual* (2013) (CGAL Editorial Board), available at http://doc.cgal.org/4.3/Manual/packages.html.
36. T. Granlund, *The GNU Multiple Precision Arithmetic Library*, 4.2.1 edition (Free Software Foundation, Boston, MA, 2006).
37. Y. Zheng and W.-H. Qian, "An enhanced ray-shooting approach to force-closure problems," *J. Manuf. Sci. Eng.* **128**(4), 960–968 (2006).