Fast Grasp Quality Evaluation with Partial Convex Hull Computation

Shuo Liu Stefano Carpin

Abstract— **We present Partial Quick Hull (PQH), an algorithm to efficiently compute one of the most commonly used grasp quality metrics. The metric relies on the computation of the convex hull of a set of points in a six dimensional space. Built on top of widely used QuickHull algorithm, PQH exploits the relationship between the convex hull and the grasp quality metric to avoid computing the whole convex hull. PQH determines at run time when the computation can be ended because the grasp quality metric can be already determined from a partially computed convex hull – hence the name of the algorithm. This improvement greatly accelerates grasp quality evaluation for force closure grasps. When the grasp is not force closure, the partial computation does not apply and PQH then behaves exactly like QuickHull. A large set of experimental tests show how PQH largely outperforms QuickHull and better scales with the size of the input.**

I. Introduction

Grasping with multifingered hands continues to be one of the areas of great interest in robotics and automation, and to date the problem cannot be considered to be solved, neither from a hardware nor from an algorithmic perspective. Form closure grasps immobilize an object by implementing an enveloping grasp preventing motion in any direction. These grasps can be useful in many situations, but cannot be used to solve every problem. Force closure grasps instead are implemented by a finger placement capable of resisting an arbitrary external wrench (i.e., a combination of force and torque). Force closure grasps are often needed when the robot is supposed to manipulate a restrained object, e.g., using a tool. Continuous progress in the development of multifingered hands (like the Barrett Hand, the Shunk hand, and the Robonaut hand, just to name a few) is intertwined with advancements in algorithms planning force closure grasps. Informally speaking, the challenge is to decide where to position the fingers in order to achieve force closure on a given object. Given that in general the problem has multiple solutions, grasp quality metrics have been developed since quite some time (see Section II for more details). In many instances, the planner performs a search in the grasp space guided by the quality measure. Since the search space is large and the planner often evaluates numerous grasps, it is essential to expedite quality evaluation. The most common grasp quality measures require the computation of a convex hull in six dimensions, and it has become customary to use the freely available QuickHull software library solve this problem. Using QuickHull requires to compromise between accuracy and speed because in many implementations the friction cones at the contact points are approximated using a finite discretization. On the one hand, one would prefer a coarse discretization to reduce the size of the input set and expedite the computation of the convex hull. On the other hand, one would favor a fine grain discretization to reduce the approximation error, but this would slow down the computation of the convex hull.

In this paper we show how in many cases it is possible to greatly expedite quality evaluation by considering the relationship between the measure sought and the convex hull being computed. We dub our method Partial Quick Hull (PQH) because it determines on the fly when the computation of the full convex hull can be terminated because the quality measure can be already determined with the partial result. Our improved method provides large performance gains when evaluating the quality of force closure grasps, whereas its computation coincides with QuickHull when the grasp is not force closure. Therefore, it is always convenient to use PQH instead of QuickHull. Numerous experiments show that the performance gap between Quick Hull and PQH increases with the size of the input. Therefore, PQH allows to use a high resolution discretization for the friction cone because it scales well with the size of the input. Our implementation is freely available to the scientific community[1] and builds upon the QuickHull implementation, thus taking advantage of a highly optimized and debugged code base.

The remainder of this paper is organized as follows. Related work is discussed in Section II and Section III defines the problem and introduces relevant notation. The PQH algorithm is discussed in Section IV, and a thorough experimental comparison is given in Section V. Section VI summarizes the lesson learned and outlines directions for further improvements.

II. Related work

In this section we discuss related work from computational geometry and robotics related to grasp quality evaluation.

A. Convex hull algorithms

Convex hull computation is a core problem in computational geometry [5]. Three parameters are used to characterize the computational complexity of convex hull algorithms,

S. Liu and S. Carpin are with the School of Engineering, University of California, Merced, CA, USA.

This work is supported by the National Institute of Standards and Technology under cooperative agreement 70NANB12H143. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the funding agencies of the U.S. Government.

[1]Code is available on `http://robotics.ucmerced.edu`.

namely the number of points n, the size of the resulting convex hull h, and the dimensionality of the space d. For the planar case ($d = 2$), various algorithms matching the known $\Omega(n \log n)$ lower bound [16] have been proposed. Among these, the Graham's scan algorithm is perhaps the most well known [7]. Since the convex hull is typically defined by just a subset of the input points, methods whose complexity depends on the size of the result (*output sensitive* complexity) have also been investigated. Algorithms with complexity $\mathcal{O}(n \log h)$ have been discovered [8], [2]. Note that the $\mathcal{O}(n \log h)$ complexity is optimal [8] in $\mathbb{R}^2$. Algorithms to compute convex hulls of points in $\mathbb{R}^d$ with $d > 2$ have been investigated as well. For $d = 3$, the convex hull can still be computed in $\mathcal{O}(n \log n)$ [14]. In fact, in $\mathbb{R}^3$ the complexity of the convex hull, defined as the number of facets or edges, is linear in the number of points n (see e.g., [5], chapter 11). However, for $d > 3$ the complexity of the convex hull h is no longer linear in the number of input points n, so output sensitive algorithms are no longer necessarily the best choice. Chazelle has shown [3] that if d is constant then the convex hull can be computed in $\mathcal{O}(n \log n + n^{\lfloor d/2 \rfloor})$. From a practical standpoint the QuickHull algorithm [1] is perhaps the most common choice when computing convex hulls in higher dimensions, also because of a widely used freely available implementation.[2] Quickhull builds upon ideas introduced in the randomized algorithm presented in [4], but it differs because rather than extending the convex hull by picking a random vertex, it always extends the partial result by picking an extremal point (these ideas will be further expanded in Section IV). Its performance is evaluated in empirical terms and is observed to be $\mathcal{O}(n \log r)$ where r is the number of vertices processed during the computation. So far, an analytic characterization of its computational complexity has not been determined.

B. Grasp metrics for force closure

Given two distinct grasps achieving force closure, a natural question to ask is which one should be preferred. Various metrics have been developed to answer this question [15]. A criterion meanwhile largely accepted suggests to prefer the grasp that can resist an arbitrary external wrench while exerting limited effort. This idea, originally proposed in [9] and then further elaborated in [6], has become the *de facto* metric used to rank grasps, and is also used to inform the behavior of numerous grasp planning algorithms. As further discussed in Section III, this property can be quantitatively measured using the distance between the origin of $\mathbb{R}^6$ and the convex hull of a set of elementary wrenches. This distance is often referred to Q distance or using similar names. The connection between grasp metrics and convex hull computation is therefore evident. Many grasp planners or grasp evaluation systems rely on QuickHull for the computation of this measure, e.g., [11]. However, because of the computational cost associated with QuickHull, there have been various attempts to replace it with alternative, faster techniques. Recently, Zheng proposed an algorithm that iteratively approximates the convex hull by growing a polytope guaranteed to be inside the convex hull [17]. His algorithm, however, cannot be applied when evaluating grasps that do not achieve force closure, whereas PQH can be applied in both cases. Note that when a grasp is not force closure the origin is outside the convex hull, but the distance between the hull and the origin is still useful to guide the planning process [18]. Pokorny and Kragic [12] provide the first accurate study of the analytical properties of the Q distance and of its approximations. In particular they introduce an efficient algorithm to reject unstable grasps, but their algorithm does not rank stable ones, i.e., they do not compute the grasp quality metric. Importantly, [12] offers an analytic bound for the approximation error introduced by the discretization of the friction cone. In the following, we make use of this result. Zheng and Qian propose one of the few methods not relying on the discretization of the convex hull [18]. The problem formulation however leads to a nonlinear optimization problem that is computationally demanding.

[2] http://www.qhull.org

III. Problem definition and notation

We here define the grasp quality evaluation problem and introduce relevant notation. The reader is referred to [13] and references therein for an in depth discussion. We assume a rigid body $\mathcal{B}$ is grasped using a multifingered hand with n fingers. Three different models for friction are commonly used, namely frictionless contact, contact with friction (also known has hard finger model), and the soft finger model. For sake of simplicity, in the following we consider just the contact with friction model, but our findings are independent of the underlying model. A grasp $\mathcal{G}$ is represented by n contact points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$ and n contact forces $\mathbf{f}_1, \ldots, \mathbf{f}_n \in \mathbb{R}^3$. The coordinates of the points are expressed with respect to a frame whose origin coincides with the center of mass or $\mathcal{B}$. The effect of every force is a wrench, i.e., the combination of a force and a torque. The wrench $\mathbf{w}_i$ generated by the i-th force is the six dimensional vector $\mathbf{w}_i = [\mathbf{f}_i \quad \mathbf{p}_i \times \mathbf{f}_i]^T$. At each contact point we define an orthogonal reference system with three unit vectors $\mathbf{n}_i$, $\mathbf{o}_i$, $\mathbf{t}_i$, where $\mathbf{n}_i$ is the unit inward normal, and $\mathbf{o}_i$, $\mathbf{t}_i$ are the unit tangent vectors. Each contact force $\mathbf{f}_i \in \mathbb{R}^3$ is written as $\mathbf{f}_i = [f_{i1} \; f_{i2} \; f_{i3}]^T$ where f_{i1} is the component along $\mathbf{n}_i$ and f_{i2} and f_{i3} are the components along $\mathbf{o}_i$ and $\mathbf{t}_i$. To prevent slippage at contact $\mathbf{p}_i$, $\mathbf{f}_i$ must lie within the friction cone $F(\mathbf{p}_i)$ defined at $\mathbf{p}_i$. The opening of cone $F(\mathbf{p}_i)$ is determined by the friction coefficient μ_i between the ith finger and the object. For simplicity, in the following we assume all these coefficients are equal to μ. The set of contact forces $F(\mathbf{p}_i)$ that do not cause slippage or separation at contact point $\mathbf{p}_i$ is defined as:

$$F(\mathbf{p}_i) = \left\{ \mathbf{f}_i \in \mathbb{R}^3 | f_{i1} \geq 0, \sqrt{f_{i2}^2 + f_{i3}^2} \leq \mu f_{i1} \right\}.$$

where $f_{i1} \geq 0$ implies no separation and $\sqrt{f_{i2}^2 + f_{i3}^2} \leq \mu f_{i1}$ implies no slippage. The set $F(\mathbf{p}_i)$ defines a cone in $\mathbb{R}^3$, hence the name friction cone. In most cases the cone is

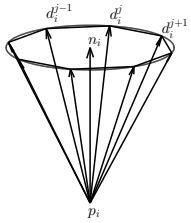


Fig. 1: The friction cone is almost always approximated using a pyramid with k edges.

represented using an approximation consisting of a regular pyramid with k edges (see Fig. 1). Using this approximation, every contact force can be written as $\mathbf{f}_i = \sum_{j=1}^{k} \alpha_{i,j}\mathbf{f}_{i,j}$ where $\mathbf{f}_{i,j}$ are the component vectors along the edges of the pyramid approximating the friction cone, and $\alpha_{i,j}$ are non negative factors adding up to 1. Based on this decomposition, the wrench generated by the ith force can then be written as

$$\mathbf{w}_i = \sum_{i=1}^{k} \alpha_{i,j}\mathbf{w}_{i,j} = \sum_{i=1}^{k} \alpha_{i,j} \begin{bmatrix} \mathbf{f}_{i,j} \\ \mathbf{p_i} \times \mathbf{f}_{i,j} \end{bmatrix}$$

Following the notation introduced in [6], let us define

$$W_1 = \text{CH}(\bigoplus_{i=1}^{n} \{\mathbf{w}_{i,1}, \dots \mathbf{w}_{i,k}\})$$

$$W_2 = \text{CH}(\bigcup_{i=1}^{n} \{\mathbf{w}_{i,1}, \dots \mathbf{w}_{i,k}\})$$

where CH indicates the convex hull and $\bigoplus$ is the Minkovski sum. A grasp is force closure if and only if the origin is inside W_1 or W_2. For the force closure case, two grasp quality metric are defined as [6]:

$$Q_i = \min_{\mathbf{x} \in W_i} ||\mathbf{x}||_2.$$

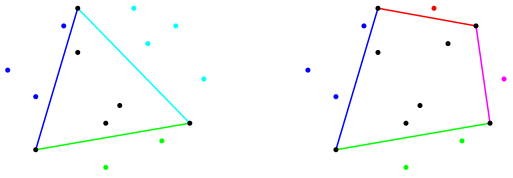
Q_i is the distance from the origin to the convex hull defining W_1 or W_2, i.e., it is the radius of the largest ball centered at the origin and completely inside the convex hull. The two different sets W_i correspond to two different physical conditions. The first one aims at minimizing the maximum finger force, whereas the second minimizes the sum of all the forces from all the fingers. In either case, a convex hull computation is needed. The reader is referred to [6] for a thorough discussion about why these are a good metrics. As a matter of fact, the Q_i metrics have become the most used used grasp quality metric.

Remarks. The definition of W_1 relies on the Minkovski sum of a finite set of elements. It is known that this is a finite set itself. Therefore for both W_1 and W_2 it is necessary to compute the convex hull of a finite set of points in $\mathbb{R}^6$. The algorithm we present in the following therefore can be used in both cases.

IV. Fast distance computation

A. QuickHull

To present our contribution, we briefly review the QuickHull algorithm. The reader is referred to [1] for a more detailed discussion. We recall that the convex hull $\mathcal{C}$ is represented by its vertices (a subset of the input set), and a set of $(d-1)$-dimensional hyperplanes called *facets*. Each facet is a convex subset of an hyperplane in $\mathbb{R}^d$ called the hyperplane supporting the facet. Given n points in $\mathbb{R}^d$ in general position,[3] QuickHull proceeds as follows. An initial simplex with $d+1$ points is created. To create the initial simplex, QuickHull picks extremal points. This expedient further accelerates the computation by reducing the number of successive iterations. Every facet is supported by an hyperplane splitting $\mathbb{R}^d$ in two half spaces, one of which includes the simplex. For each facet, a set of *outside* points is created. The outside set of a face consists of all input points lying on the half space not including the simplex (see Figure 2.a). Note that in general a point may belong to the outside set of more than one face.



(a) (b)

Fig. 2: a) QuickHull initialization. In $\mathbb{R}^2$ the initial simplex includes 3 points and every facet is a segment. The outside set of each facet is displayed using the same color of the facet. Black points are inside the simplex and not assigned to any outside set. b) QuickHull expansion. The cyan facet is expanded and replaced by two new facets (red and purple), and their respective outside sets are computed.

From the initial simplex, the convex hull is iteratively grown. If there exists a facet with a non empty outside set, the convex hull is expanded by growing the convex hull to include the farthest point in the outside set. This process, called *expansion*, eliminates one facet (the one being expanded), and generates new facets for which their respective outside sets are created (see Figure 2.b). Once the outside sets of all facets are empty, the process terminates and the simplex is equal to the convex hull. By *greedily* growing the convex hull towards the farthest point at each iteration, it was shown that QuickHull outperforms other algorithms utilizing different criteria to expand the current hull, e.g., [4].

B. Partial QuickHull

Let us now consider the specific case where the convex hull is computed to determine the grasp quality metric. Note

[3]As usual in computational geometry, this assumption can be enforced using a small perturbation of the input.

that in this case we are operating in $\mathbb{R}^6$, as per the definition of W_i and $\mathbf{w}_{i,j}$. Two distinct situations may arise, i.e., the origin of $\mathbb{R}^6$ lies inside the convex hull, or not.

Origin inside the convex hull. If the origin is inside, then the grasp is force closure and the distance of the convex hull from the origin provides the needed grasp quality measure. To greatly accelerate the computation in this case, it is essential to observe that the computation can be interrupted as soon as the closest face to the origin is determined, because the grasp quality metric is given by the distance between this face and the origin, and the computation of the remaining part of the convex hull carries no additional information.

Origin outside the convex hull. In this case the grasp is not force closure and the grasp quality measure is not defined. However, the distance between the origin and the convex hull still provides valuable information because it indicates how far a grasp is from achieving force closure [18]. In this case it is necessary to compute the whole convex hull.

It is important to notice that when the computation starts one does not know if the origin lies inside the convex hull or not. Therefore this decision must be made online. Moreover, we want to compute the distance between the origin and the convex hull both for force closure and non-force closure grasps, because it is useful in both cases. The conditions to identify the correct case are given by two lemmas building upon the following definitions.

Definition 1: Let $\mathcal{P}$ be a hyperplane in $\mathbb{R}^d$ and $\mathbf{y}$ a point in $\mathbb{R}^d$. We define $d(\mathcal{P}, \mathbf{y})$ as the Euclidean distance between $\mathcal{P}$ and $\mathbf{y}$.

Definition 2: An oriented hyperplane is an hyperplane $\mathcal{P}$ associated with a unary vector $\mathbf{n}_\mathcal{P}$ orthogonal to $\mathcal{P}$.

The role of $\mathbf{n}_\mathcal{P}$ is to identify one of the two halfspaces defined by $\mathcal{P}$. As QuickHull and PQH incrementally build the resulting convex hull, both algorithms maintains for each facet an orthogonal unit-length vector pointing outside the convex hull.

Assumption: Let $\mathcal{F}$ be a facet of a convex hull $\mathcal{C}$ and let $\mathbf{n}_\mathcal{F}$ be a unit length vector orthogonal to $\mathcal{F}$ and oriented towards the outside of $\mathcal{C}$. The oriented hyper-plane associated to $\mathcal{F}$ is obtained by setting $\mathbf{n}_\mathcal{F} = \mathbf{n}_\mathcal{P}$ for the oriented hyperplane supporting $\mathcal{F}$.

Definition 3: Let $\mathcal{P}$ be an oriented hyper-plane in $\mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$. We define $o(\mathcal{P}, \mathbf{y})$ (offset from $\mathbf{y}$ to hyperplane $\mathcal{P}$) as

$$o(\mathcal{P}, \mathbf{y}) = d(\mathcal{P}, \mathbf{y})\mathrm{sgn}((\mathbf{p}_\mathcal{P} - \mathbf{y}) \cdot \mathbf{n}_\mathcal{P})$$

where $\mathbf{p}_\mathcal{P}$ is any point on the hyperplane $\mathcal{P}$, $\cdot$ is the dot product, and sgn is the signum function.

From the above definitions it follows that $d(\mathcal{P}, \mathbf{y})$ is always non-negative, whereas $o(\mathcal{P}, \mathbf{y})$ can be positive, negative, or zero. The offset $o(\mathcal{P}, \mathbf{y})$ can be interpreted as a *signed distance*, i.e., it is $d(\mathcal{P}, \mathbf{y})$ when $\mathbf{y}$ is not in the halfplane identified by $\mathbf{n}_\mathcal{P}$ and it is $-d(\mathcal{P}, \mathbf{y})$ otherwise.

Definition 4: Let $\mathcal{F}$ be a facet on the convex hull, $\mathcal{P}$ be the hyperplane supporting $\mathcal{F}$, and $\mathcal{P}(\mathbf{y})$ the projection[4] of $\mathbf{y}$ onto $\mathcal{P}$. We define $o(\mathcal{F}, \mathbf{y})$ (offset from point $\mathbf{y}$ to facet $\mathcal{F}$) as

$$o(\mathcal{F}, \mathbf{y}) = \begin{cases} o(\mathcal{P}, \mathbf{y}) & \text{if } \mathcal{P}(\mathbf{y}) \in \mathcal{F} \\ \mathrm{sgn}(o(\mathcal{P}, \mathbf{y})) \cdot \min\limits_{\mathbf{x} \in \mathcal{V}_\mathcal{F}} ||\mathbf{x} - \mathbf{y}||_2 & \text{otherwise.} \end{cases}$$

where $\mathcal{V}_\mathcal{F}$ is vertex set on $\mathcal{F}$ and $\cdot$ is the dot product.

Starting from these definitions we can state the two lemmas supporting the PQH algorithm (their proofs are omitted for lack of space).

Lemma 1: Let $\mathcal{C}$ be the convex hull of n points in $\mathbb{R}^d$. If $\mathbf{x}$ is inside $\mathcal{C}$, then the offset from point $\mathbf{x}$ to all the oriented hyperplanes supporting all facets of $\mathcal{C}$ is larger than 0. In contrast, if $\mathbf{x}$ is outside $\mathcal{C}$, there exists at least one facet of $\mathcal{C}$ for which $\mathbf{x}$ has negative offset from the supporting hyperplane.

Lemma 2: Let $\mathbf{x}$ be inside the convex hull $\mathcal{C}$. Then the distance from the closest facet $\mathcal{F}$ of $\mathcal{C}$ to $\mathbf{x}$ is equal to the offset of $d(\mathcal{F}, \mathbf{x})$ and the distance of every other facet of $\mathcal{C}$ to $\mathbf{x}$ is larger than $d(\mathcal{F}, \mathbf{x})$.

Algorithm 1 sketches the algorithmic details of PQH. For every facet $\mathcal{F}$ of the convex hull, we indicate with $\mathcal{P}(\mathcal{F})$ the associated supporting oriented hyperplane. The algorithm creates the initial simplex as QuickHull (line 1). Inside the main loop the algorithm looks for the next facet to expand, $\mathcal{F}_e$. The first for loop (lines 4 to 8) builds upon lemma 1. If the origin is outside $\mathcal{C}$, then the algorithm behaves like QuickHull and $\mathcal{F}_e$ is selected using QuickHull's selection method. However, as soon as the origin is determined to be inside $\mathcal{C}$, the facet selection criterion changes, and the algorithm tries the expand the facet closest to the origin (line 10). Building upon Lemma 2, if the outside set of the facet closest to the origin is empty (line 11), the algorithm terminates and returns the distance from the origin (line 12).

Algorithm 1 Partial Quick Hull algorithm

```
1:  Create initial simplex C with d + 1 points
2:  loop
3:    F_e ←nil
4:    for all F ∈ C do
5:      if o(P(F), 0) < 0 then
6:        F_e ←QuickHullSelectFacet
7:      end if
8:    end for
9:    if F_e = nil then
10:     F_e ← facet with smallest offset from 0
11:     if outside set of of F_e = ∅ then
12:       return o(P(F_e), 0)
13:     end if
14:   end if
15:   Expand F_e
16: end loop
```

[4]The projection of $\mathbf{y}$ into $\mathcal{P}$ is the point obtained by the intersection between $\mathcal{P}$ and the line through $\mathbf{y}$ parallel to $\mathbf{n}_\mathcal{P}$.

In our current implementation the search for the facet with the smallest offset (line 10) uses a brute force approach, i.e., all faces are analyzed. As it will be shown in Section V, despite this suboptimal choice the gain in performance is still significant. However, in the future we will further refine the algorithm and replace the complete search with a priority queue, thus further enhancing PQH.

C. Computational complexity

QuickHull is observed to be competitive in practice, but its computational complexity has not been formally determined and its performance analysis is mostly empirical. Consequently, no accurate computational complexity analysis is available for PQH, although one can say that by construction QuickHull's (conjectured) computational complexity $\mathcal{O}(n \log r)$ is an upper bound for PQH.

V. Evaluation

In order to perform a fair comparison between PQH and QuickHull, our implementation is based on the public available QuickHull implementation written in C. As per Algorithm 1, PQH is obtained modifying the main cycle governing the expansion of the simplex. Our code is freely available on our website, together with the datasets used to generate the results presented in this section.

A. Comparison

We start comparing QuickHull with PQH over sets of randomly generated points in $\mathbb{R}^6$. The objective of these first tests is to compare the two algorithms for large input sets and contrast how their performance scales with the size of the input. In the first study we compute the convex hull of 100 sets of 10000 points uniformly distributed inside a cube. In each case the origin lies inside the resulting convex hull, and therefore PQH always operates in its most favorable condition. For both PQH and QuickHull we record the number of points added to the convex hull and the number of hyperplanes generated during the computation. As pointed out in [1], these two quantities are meaningful indicators of the computational effort. In particular, the most expensive part is the creation and removal of the hyperplanes. In addition, to give a better sense of the relative efficiency of the algorithms, for this first test we also measure the time spent on a 2.8GHz Intel i7 with 8 Gb RAM running Linux. Figure 3 shows the results. The top panel displays a comparison between the two algorithms, whereas the bottom figure presents the ratio between the three performance indices. It can be seen that for each of the three measures there is a speedup always exceeding a factor of 10. Note that in this case the average time spent by QuickHull is 2.06 seconds, whereas PQH spends on average 160 milliseconds. In the successive test we vary the number of points from 10000 to 40000 to show how the performance gain scales with the size of the input. In this set of tests the origin is again always inside the resulting convex hull. In this case we just plot the number of points and the number of hyperplanes. For each size, we provide the average of 100 different runs

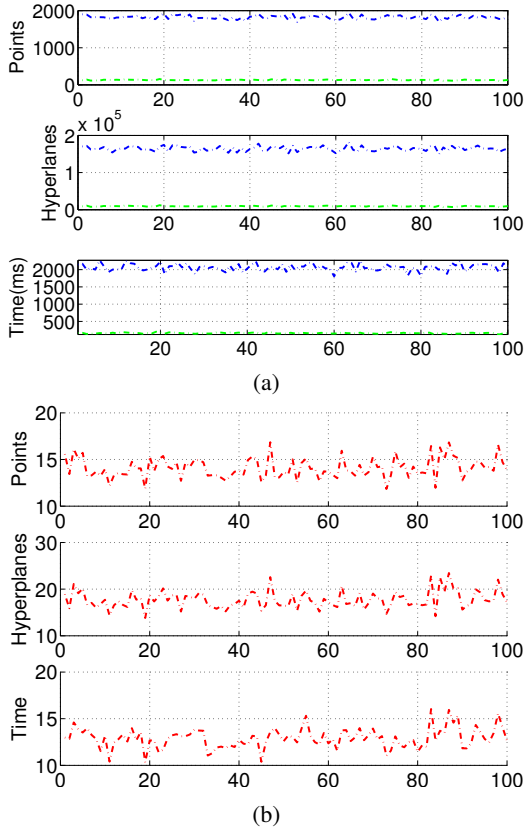


Fig. 3: a) Performance of QuickHull (blue) and PQH (green) on 100 different input sets consisting of 10000 points in which the origin is always inside the resulting hull. b) Ratio between the two performances.

(standard deviations are not shown since they are negligible). Figure 4 shows the results. Figure 4.b shows that as the input size grows the performance gap between QuickHull and PQH widens, thus showing that when operating in its most favorable regime PQH scales better with the size.

Having assessed that PQH significantly outperforms QuickHull when the origin is inside the convex hull, in the next test we compare the two algorithms over sets of 10000 points where the origin may or may not be inside the convex hull. The results are shown in Figure 5 and should be contrasted with Figure 3. The spiky trend for PQH (green line) confirms that PQH behaves like QuickHull when the origin is not inside the convex hull, and is instead much faster when it is inside. Therefore when the two lines overlap it means that the origin is not inside the convex hull, whereas there is a gap when the origin is inside. The examples shown so far dealt with randomly generated input sets uniformly distributed inside a cube in $\mathbb{R}^6$. Next, we repeat the last experiment, but with input sets generated from actual grasps. A set of 100 random grasps with 4 contact points and an approximation of the friction cone with 32 edges is used. The object being grasped is a bar. Since the grasps are randomly generated, some of them are force closure and some are not. The results shown in figure 6 are therefore similar to Figure

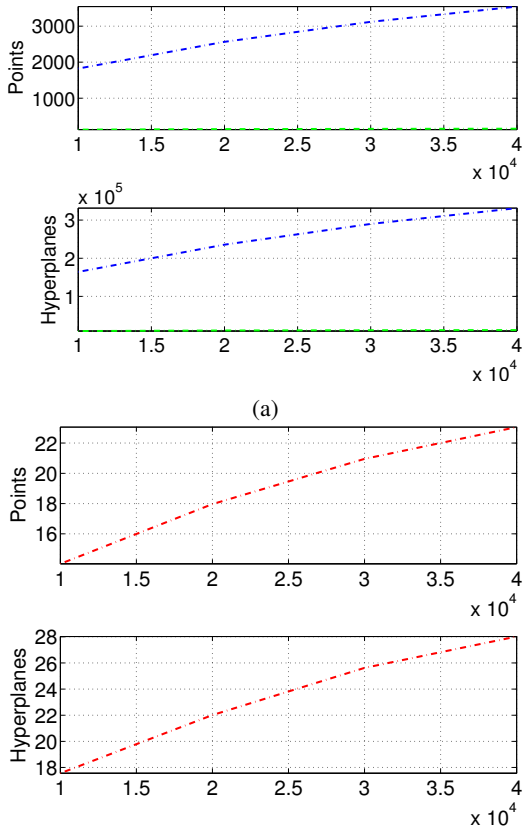


Fig. 4: a) Performance of QuickHull (blue) and PQH (green) for different sizes of the input set. b) Ratio between the two performances.

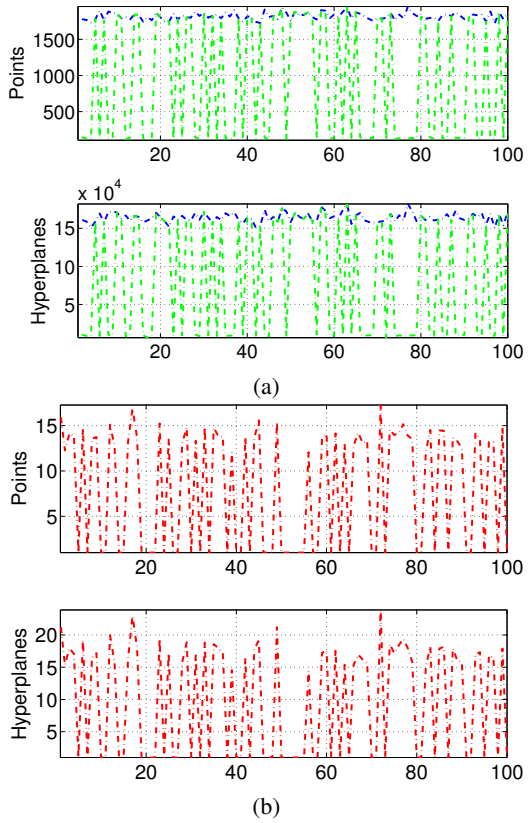


Fig. 5: a) Performance of QuickHull (blue) and PQH (green) random configurations that may or may not include the origin. b) Ratio between the two performances.

5 in the sense that PQH behaves like QuickHull when the grasp is not force closure and is much faster when the grasp is force closure. However, figure 6.b shows that for the case where the grasp is force closure the ratio between the number of generated hyperplanes is much more favorable for PQH (compare with figure 5.b). As stated in the beginning, in a practical scenario one is interested in approximating the friction cone using a large number of edges to approximate its circular section. There is evidently a linear relationship between the number of edges and the number of points for which the convex hull will be computed. Figure 7 shows the ratio between QuickHull and PQH's performance as a function of the number of edges approximating the friction cone. Figures 4 and 7 are particularly important in light of the findings presented in [12]. Pokorny and Kragic have shown that when the grasp is force closure the approximation error introduced by discretizing the friction cone with k edges is $M(1 - \cos(\pi/k))$ where M is a constant accounting for geometric aspects unrelated to k. Using PQH one can afford to use large k values to approximate the cone. For example, for $k = 64$ PQH is still extremely fast and $(1 - \cos(\pi/k)) \approx 0.0012$. A final consideration is in order. PQH expedites the computation of the grasp metric when the origin is inside the convex hull. Its performance instead coincides with QuickHull when this is not the case. When used within a grasp planner, the effective performance gain will then depend on the types of grasps for which the metric is computed. In particular, the speedup depends on the ratio between the number of evaluations of force closure grasps and the number of evaluations of non-force closure grasps. This ratio depends on the planning process. When the grasp planner explores the space of possible grasps and guides its search using the grasp metric, one can anticipate that in the initial stages the planner will test grasps of both types, but as the search progresses it will concentrate on the subspace of force closure grasps, i.e., the regime favorable to PQH. To put this consideration into context, Table I shows the results we obtained with one planner we developed that implements a search similar to the one we described (the planner is described in a separate paper [10]). Each row describes the result of a planning problem and shows the number of force closure grasps evaluated (FC) versus the number of non force closure grasps (NFC). The last column shows the percentage of cases in which PQH operates in its favorable condition. Similar trends can be expected with comparable planners[5].

[5]Unfortunately, most grasp planners presented in literature are not released as open source to the community, thus making an experimental comparison impossible.

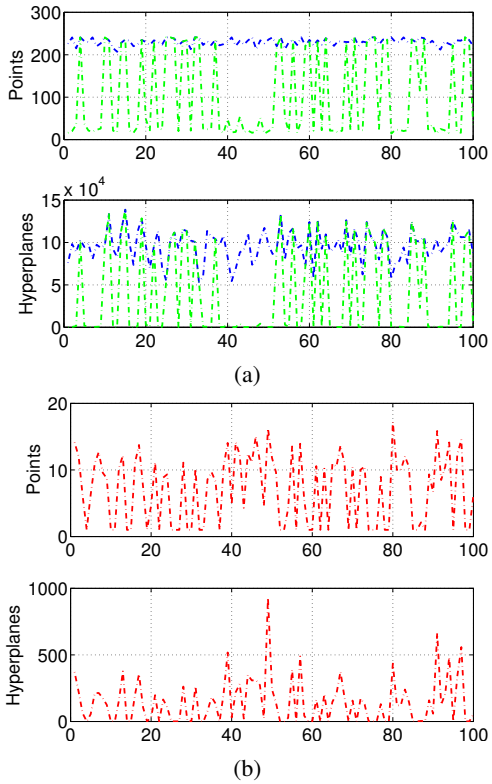


Fig. 6: a) Performance of QuickHull (blue) and PQH (green) for randomly generated grasps that may or may not be force closure. b) Ratio between the two performances.

FC	NFC	FC Percentage
369150	94056	79.7
377277	95115	79.9
435475	98459	81.5
373505	87823	80.9
361924	95870	79.0

TABLE I: Number of force closure grasps evaluated (FC) vs number of non force closure grasps (NFC) for five different grasp problems.

VI. Conclusions

We have presented PQH, a modification of the QuickHull algorithm tailored for the computation of grasp quality metrics. Our algorithm builds upon the insight that for the force closure case most of the computation of the convex hull is irrelevant to the definition of the metric, and can therefore be omitted, thus yielding large performance gains. When the grasp is not force closure PQH behaves exactly like QuickHull. Therefore, from a practical point of view PQH is always convenient and one can safely substitute QuickHull with PQH when computing grasp quality metrics. In the future we will refine our current implementation to obtain further performance improvements, e.g., using priority queues to quickly determine the face closest to the origin. In addition, we will use PQH to develop efficient planners using the grasp quality metric to guide the search process.

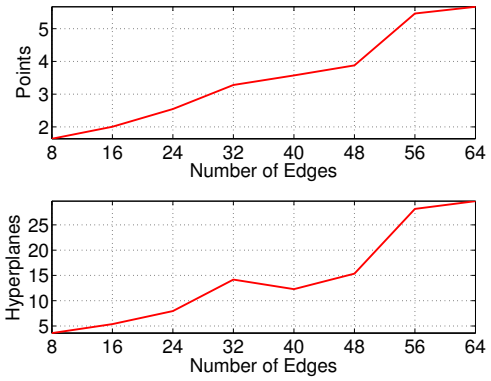


Fig. 7: Ratio between the performance of QuickHull and PQH as a function of the number of edges used to approximate the friction cone.

References

[1] C. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[2] T. M. Chan. Output-sensitive results on convex hulls, extreme points and related problems. *Discrete Computational Geometry*, 16:369–387, 1996.

[3] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Computational Geometry*, 10:377–409, 1993.

[4] K. Clarkson and P. Shor. Applications of random sampling in computational geometry. *Discrete Computational Geometry*, 4:387–421, 1989.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, 3rd edition, 2008.

[6] C. Ferrari and J. Canny. Planning optimal grasps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2290–2295. IEEE, 1992.

[7] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(132-133), 1972.

[8] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal of Computation*, 15:287–299, 1986.

[9] D.G. Kirkpatrick, B. Mishra, and C.K. Yap. Quantitative Steintz's theorems with applications to multifingered grasping. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 341–351, 1990.

[10] S. Liu and S. Carpin. Global grasp planning exploiting triangular meshes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2015 (submitted).

[11] A.T. Miller and P.K. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122, 2004.

[12] F.T. Pokorny and D. Kragic. Classical grasp quality evaluation: New theory and algorithms. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3493– 3500, 2013.

[13] D. Pratticchizzo and J.C. Trinkle. Grasping. In B. Siciliano and O. Khatib, editors, *Handbook of robotics*, chapter 28, pages 671–700. Springer, 2008.

[14] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20:87–93, 1977.

[15] R. Suárez, M. Roa, and J. Cornella. Grasp quality measures. Technical Report IOC-DT-P-2006-10, Universitat politècnica de Catalunya, March 2006.

[16] A. C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28(780-787), 1981.

[17] T. Zheng. An efficient algorithm for a grasp quality measure. *IEEE Transactions on Robotics*, 29(2):579–585, 2013.

[18] Y. Zheng and W.-H. Qian. Improving grasp quality evaluation. *Robotics and Autonomous Systems*, 57:665–673, 2009.