On Computing All Immobilizing Grasps of a Simple Polygon with Few Contacts

Jae-Sook Cheong

Herman J. Haverkort

A. Frank van der Stappen

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-013

www.cs.uu.nl

On Computing All Immobilizing Grasps of a Simple Polygon with Few Contacts

Jae-Sook Cheong Herman J. Haverkort
A. Frank van der Stappen
Institute of Information and Computing Sciences,
Utrecht University, P.O.Box 80089, 3508 TB Utrecht, the Netherlands
{jaesook,herman,frankst}@cs.uu.nl

Abstract

We study the output-sensitive computation of all the combinations of edges and concave vertices of a simple polygon that allow an immobilizing grasp with less than four frictionless point contacts. More specifically, if n is the number of edges, and m is the number of concave vertices of the polygon, we show how to compute:

- in $O(m^{4/3} \log^{1/3} m + K)$ time, all K combinations that allow a *form-closure* grasp with *two* contacts;
- in $O(n^2 \log^4 m + K)$ time, all K combinations that allow a *form-closure* grasp with *three* contacts;
- in $O(n \log^4 m + (nm)^{2/3} \log^{2+\delta} m + K)$ time, all K combinations of one concave vertex and one edge that allow a grasp with one contact on the vertex and one contact on the interior of the edge, satisfying *Czyzowicz's* weaker conditions for immobilization;
- in $O(n^2 \log^3 n + K)$ time, all K combinations of three edges that allow a grasp with one contact on the interior of each edge, satisfying *Czyzowicz's* weaker conditions for immobilization.

1 Introduction

Many applications such as robot hand grasping and manufacturing operations require an object to be *immobilized*, such that any motion of the object violates the rigidity of the object or the contacts. An attractive theoretical model for immobility was formulated by Reuleaux in 1876 [21]. He defines a rigid body to be in *form closure* if a set of contacts along its boundary constrains all finite and *infinitesimal* motions of the body. This notion is stronger than immobility, as for instance an equilateral triangle with a point contact in the middle of each edge is immobilized, but is not in form closure (it permits an infinitesimal rotation around its center). Form closure depends only on the position of the contacts and their normals, and is invariant with respect to the curvature of body and contacts. This is not true for immobility in general: if we replace the equilateral triangle in the example by its inscribed circle, contacts and normals remain identical, but the body is no longer immobilized.

Markenscoff et al. [14] and Mishra et al. [16] independently showed that, with the exception of a circle, any two-dimensional body can be put in form closure with four frictionless point contacts, and that almost any three-dimensional body can be put in form closure with seven such contacts. We will call a configuration of frictionless point contacts that put an object in form closure a *form-closure grasp*.

We consider the problem of computing all form-closure grasps of a polygonal part. The availability of *all* grasps of a certain part allows a user—usually a machinist—to select the grasps that best meet specific additional requirements, such as accessibility, which may vary from one operation to another. As the computation of all grasps along a given combination of edges and vertices can be accomplished in constant time [17, 26], the algorithmic challenge is to efficiently report all combinations of edges and vertices that yield at least one grasp. Since placing a contact at a convex vertex may damage the part, convex vertices are not generally accepted as contact positions, so we restrict the search to combinations of edges and/or concave vertices.

An algorithm to compute, for a simple polygon with n edges, all the edge combinations that have a form-closure grasp with four frictionless contacts was presented by van der Stappen et al. [26]. The algorithm runs in $O(n^{2+\varepsilon} + K)$ time, where K is the number of edge quadruples reported. Brost and Goldberg [3] studied the same problem in modular settings, where the contact positions are restricted to a grid.

Fewer than four point contacts may suffice for form closure if the object has concave vertices.[1] Informally speaking, such vertices allow to have two contacts at the price of one, as a contact at a concave vertex can be regarded as lying on both incident edges or arcs. Form-closure grasps involving concave vertices were first studied by Gopalakrishnan and Goldberg [10], who gave an $O(m^2)$ time algorithm to find all K concave vertex pairs that allow a two-contact form-closure grasp, where m is the number of concave vertices of the polygon. In Section 2, we improve this to $O(m^{4/3} \log^{1/3} m + K)$. All combinations of one concave vertex and two edges can be reported using a generalization of the algorithm by van der Stappen et al. We improve on that by presenting an $O(n^2 \log^4 m + K)$ time algorithm. Furthermore, we show how to report all combinations of two concave vertices and one edge in $O(n^2 \log^\delta m + m^2 \log^4 n + K)$ time, and finally, all combinations of three concave vertices in $O(m^2 \log^3 m)$ time. In total, we find all K combinations of three concave vertices and/or edges that allow a three-contact form-closure grasp in time $O(n^2 \log^4 m + K)$.

In Section 3, we turn our attention away from form closure to a different condition for immobility. Czyzowicz et al. [6] provided a necessary and sufficient geometric condition for a simple polygon to be immobilized by three point contacts. Analogous to the above, we call a configuration of frictionless point contacts that immobilizes a rigid body according to this geometric condition an *immobility grasp*. A more general analysis applicable to arbitrary objects was given by Rimon and Burdick [22, 23, 24], who define the term *second-order immobility*, as it not only takes position and normal, but also curvature of object, contacts, and possible motions into account. Note that all form-closure grasps are also immobility grasps.

For a simple n-vertex polygon without parallel edges, an algorithm that reports all the edge triples that yield at least one immobility grasp was given by van der Stappen et al. [26]. Its running time is $O(n^2 \log^2 n + K')$, where K' is the number of triples considered according to some criterion. This criterion is necessary, but not sufficient, so the algorithm may return triples that do not yield immobility grasps. We resolve this shortcoming by giving a truly output-sensitive algorithm with a running time of $O(n^2 \log^3 n + K)$, where K is the number of edge triples such that we can immobilize the object with a contact on the interior of each edge. Furthermore, we give an $O(n \log^4 m + (nm)^{2/3} \log^{2+\delta} m + K)$-time algorithm to report all combinations of an edge and a concave vertex that yield an immobility grasp.

Note that in general, a polygon is not guaranteed to allow any two- or three-point immobility

[1]Fewer than four point contacts also suffice for form closure if there is friction between the contacts and the object. Several researchers studied what is—in the presence of friction—often referred to as force-closure grasps. Non-output-sensitive algorithms exist for computing (subsets of) all such grasps of planar objects with two (see e.g. [5, 11, 19]) and three (see e.g. [18, 13]) contacts, and with parallel-jaw grippers [20].

grasps. For example, a square cannot be immobilized without a contact on each side. However, for polygons without parallel edges, Czyzowicz et al. [6] showed that there must be at least one three-point immobility grasp—and it will be reported by our algorithms.

2 On Form-Closure Grasps with Less than Four Contacts

In this section we will explain how to find all combinations of concave vertices and (possibly) edges that allow a two- or three-point form-closure grasp. We will first explain how we can characterize such combinations in general, and introduce a few data structures that we need for our algorithms. It will become clear that there are four cases to distinguish:

- pairs of concave vertices;
- triples of one concave vertex and two edges;
- triples of two concave vertices and one edge;
- triples of concave vertices.

For each of these cases, we will give an efficient algorithm to report all combinations that yield a form-closure grasp.

2.1 Characterization of Form-Closure Grasps

When a force F is applied to an object P at position $p = (p_x, p_y)$, it will make the object translate and/or rotate. The translation is determined by the force vector $f = (f_x, f_y)$. The rotation is determined by the torque $f \times p$. The full effect is a *wrench*, which is described as a three-dimensional vector $w_F = (f_x, f_y, f \times p)$ in *wrench space*. A contact C_i at position p_i can apply force in the inward normal direction $n(p_i) = (n_x(p_i), n_y(p_i))$ of the boundary of P at p_i. This results in a wrench $\lambda_i w_i$, where $w_i = (n_x(p_i), n_y(p_i), n(p_i) \times p_i)$ is the “unit wrench” of contact C_i, and $\lambda_i \geqslant 0$ is the magnitude of the force applied by C_i. Our results on form-closure grasps are based on the following characterization of form closure [9, 16, 17, 25].

Theorem 2.1 *Given a set of $k \geqslant 4$ wrenches $w_1, w_2, \cdots w_k$ on an object P. Then the following three conditions are equivalent:*

(i) P is in form closure.

(ii) Any wrench w_F can be written as $-w_F = \lambda_1 w_1 + \cdots + \lambda_k w_k$, with $\lambda_i \geqslant 0$.

(iii) The origin O lies in the interior *of the convex hull of $w_1, w_2, \cdots, w_k$.*

The equivalence of (i) and (ii) relies on the fact that the contacts together can be seen to apply any wrench that is a non-negative combination of the individual contact wrenches. Intuitively, P is in form closure if and only if any wrench applied to P can be cancelled by such a non-negative combination of contact wrenches. The theorem implies that the magnitudes of wrench vectors are not important; only the direction matters. Hence, from here on, we only work with wrenches with unit force vectors.

The equivalence of (ii) and (iii) can be verified easily. Note that if the origin lies on the *boundary* of the convex hull of $w_1, w_2, \cdots, w_k$, the object is not in form-closure, but it may still satisfy Czyzowicz’s conditions for immobility—such cases are treated in Section 3.

When we place a point contact at a concave vertex p incident to edges e_1 and e_2, it provides a range of wrenches. On one extreme, there is the wrench w_1 determined by p and the inward normal n_1 of e_1. On the other extreme, there is the wrench w_2 determined by p and the inward normal n_2 of e_2. The other wrenches provided by the concave vertex are found as we turn the direction of the force from n_1 towards n_2. These wrenches can all be expressed as a positive linear combination $\lambda_1 w_1 + \lambda_2 w_2$, with $\lambda_1 \geqslant 0$ and $\lambda_2 \geqslant 0$. Therefore these intermediate wrenches can be ignored: they do not contribute anything to the satisfiability of condition (ii) in Theorem 2.1, so we can view a concave vertex as a contact that provides a pair of contact wrenches w_1 and w_2.

We may achieve form closure with less than four point contacts by taking advantage of concave vertices: thus, two or three contacts can provide up to six wrenches. We will compute all form-closure grasps using concave vertices by finding all combinations of single wrenches, provided by edge contacts, and pairs of wrenches, provided by concave vertices, that satisfy the condition in Theorem 2.1. Observe that there are four cases to distinguish:

- *two point contacts* can only provide enough wrenches if both of them are in a concave vertex;
- *three point contacts that provide four wrenches*, that is two edge contacts and one contact in a concave vertex;
- *three point contacts that provide five wrenches*, that is one edge contact and two contacts in concave vertices;
- *three point contacts that provide six wrenches* by means of three contacts in concave vertices.

Note that the fact that four wrenches suffice to keep an object in form closure does not imply that there must be redundant wrenches in the latter cases. For example, if we take three lines through the origin in wrench space and place, on each line, a wrench on each side of the origin, their convex hull contains the origin in its interior. But no subset of four or five of these wrenches contains the origin in the interior of its convex hull.

2.2 Pairs of Concave Vertices

In the following, we define the segment $\overline{pq}$ to be the *relatively open* segment connecting p and q, that is, the set $\overline{pq} := \{\lambda p + (1-\lambda)q \mid 0 < \lambda < 1\}$. We need a simple geometric lemma.

Lemma 2.1 *Let w_1, w_2, w_3, w_4 be four points in $\mathbb{R}^3$. The origin O lies in the interior of the convex hull of $w_1, \ldots, w_4$ if and only if there are points $p_{12} \in \overline{w_1 w_2}$ and $p_{34} \in \overline{w_3 w_4}$ such that $O \in \overline{p_{12} p_{34}}$.*

Proof: The “if” direction is trivial, so let us only prove the “only if” direction.

Suppose the origin O lies strictly inside the tetrahedron formed by w_1, w_2, w_3 and w_4—see Figure 1. Consider the plane Π containing w_1, w_2, and O. It intersects the segment $\overline{w_3 w_4}$ in a point p_{34}. The intersection of the tetrahedron with Π is the triangle $\triangle w_1 w_2 p_{34}$. The point O lies in the interior of this triangle, so the line through O and p_{34} intersects $\overline{w_1 w_2}$ in the required point p_{12}. $\square$

We now build a *screen* Γ in wrench space. In wrench space, the horizontal dimensions x and y represent the direction of the force applied by a wrench, while the vertical dimension z represents the torque that is caused by a wrench. The screen Γ consists of two vertically infinite slabs, namely $\Gamma_1 := \{(x, 1, z) \mid -1 < x < 1 + \varepsilon, z \in \mathbb{R}\}$ and $\Gamma_2 := \{(-1, y, z) \mid -1 < y < 1 + \varepsilon, z \in \mathbb{R}\}$, where ε is an arbitrarily small constant, which we will explain later.

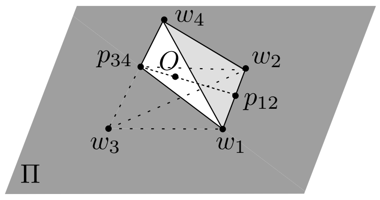

Figure 1: The origin O lies in the interior of the convex hull of $w_1, \ldots, w_4$ if and only if there are points $p_{12} \in \overline{w_1 w_2}$ and $p_{34} \in \overline{w_3 w_4}$ such that $O \in \overline{p_1 p_2}$.

We will project wrenches w that do not lie on the z-axis onto Γ as follows. The projection $\pi_i(w)$ of w on Γ_i is the intersection, if it exists, of Γ_i with the line through w and the origin O. If w lies between O and $\pi_i(w)$, we color $\pi_i(w)$ *black*. If O lies between w and $\pi_i(w)$, we color $\pi_i(w)$ *gray*. It is easy to see that for each wrench w, at least one of $\pi_1(w)$ and $\pi_2(w)$ exists. Note that we do not need to consider wrenches that lie on the z-axis, as the point contacts are unable to apply torque to the object without pushing it.

A segment $\overline{w_1 w_2}$ is projected onto Γ by projecting each point $w \in \overline{w_1 w_2}$. Note that a segment $\overline{w_1 w_2}$ representing a concave vertex will never intersect the z-axis, since the angle between the direction of the forces in w_1 and w_2 must be less than π. The complete projection $\pi(\overline{w_1 w_2})$ consists of at most four segments on Γ: one of each color on each part of the screen—see Figure 2. This is were ε has its purpose in the construction of the screen: by making the screens just a little bigger than two units wide, we do not have to worry about interior points of edges becoming end points in the projection. They will always be interior points in the projection on at least one screen.

Lemma 2.2 *Given an object with four contact wrenches* w_1, w_2, w_3, w_4. *The object is in form-closure if and only if the interior of a gray part of* $\pi(\overline{w_1 w_2})$ *intersects the interior of a black part of* $\pi(\overline{w_3 w_4})$, *or vice versa.*

Proof: By Theorem 2.1 and Lemma 2.1, the object is in form closure if and only if there exist $p_{12} \in \overline{w_1 w_2}$ and $p_{34} \in \overline{w_3 w_4}$ such that $O \in \overline{p_{12} p_{34}}$. This implies that neither $\overline{w_1 w_2}$ nor $\overline{w_3 w_4}$ passes through the origin. Furthermore, on a screen Γ_i where $\pi_i(p_{12})$ exists (which must be true for at least one of the screens Γ_1 and Γ_2), we must have $\pi_i(p_{12}) = \pi_i(p_{34})$ (since they lie on the same line through the origin) and the colors of these projections differ (since they lie on different sides of the origin)— see Figure 3. ◲

We now have all the ingredients for an efficient algorithm that reports all pairs of concave vertices that allow a form-closure grasp by placing two frictionless point contacts at these vertices. We assume that the concave vertices have already been identified. For each concave vertex p, we compute the two wrenches $w_1(p)$ and $w_2(p)$ corresponding to it, and project the segment $\overline{w_1(p) w_2(p)}$ onto Γ. Let $s(p) := \pi(\overline{w_1(p) w_2(p)})$ be the projection. By Lemma 2.2, two concave vertices p and q have a form-closure grasp if and only if the interiors of $s(p)$ and $s(q)$ form a gray-black intersection in Γ.

The family $\{s(p)\}$ consists of at most $4m$ gray and black segments in Γ, where m is the number of concave vertices of P. It remains to compute all gray-black intersections in this set, a problem that can be solved in time $O(m^{4/3} \log^{1/3} m + K)$, using the solution by Agarwal [1] with the improvement by Chazelle [4] (Chazelle's description mentions colorblind intersections only, but his approach also works for gray-black intersections).

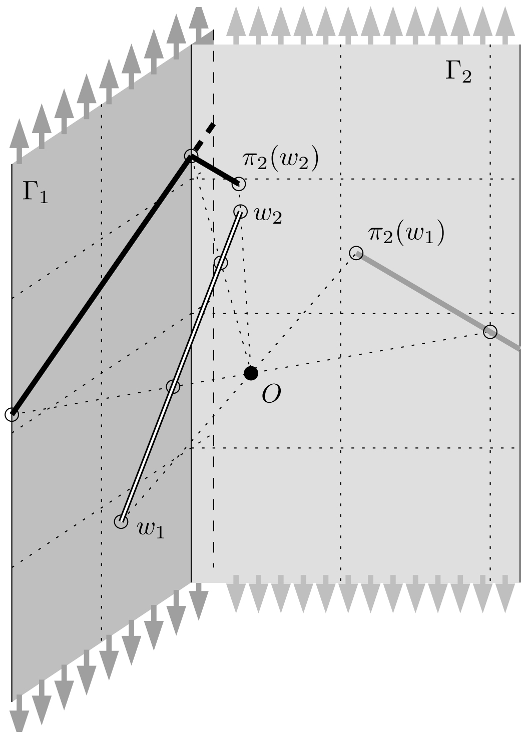

Figure 2: The screen Γ, with the projection of $\pi(\overline{w_1 w_2})$ of $\overline{w_1 w_2}$. In this case, the projection consists of three segments: one black segment on Γ_1 (partially hidden behind Γ_2 in the figure), and a black and a gray segment on Γ_2.

Theorem 2.2 *Given a polygon with m concave vertices, all K form-closure grasps formed by two concave vertices can be computed in time $O(m^{4/3} \log^{1/3} m + K)$.*

2.3 Triples of One Concave Vertex and Two Edges

Form closure may also be achieved by placing three frictionless point contacts, one at a concave vertex p, and one each on two edges e_1 and e_2. We now give an algorithm to report all such triples (p, e_1, e_2). Again, we have four wrenches: $w_1 \in \hat{e_1}$, $w_2 \in \hat{e_2}$, and the two wrenches $w_1(p)$ and $w_2(p)$ corresponding to the concave vertex p. Here, $\hat{e}$ is the set of wrenches corresponding to the possible placement of contacts on the interior of edge e. All such wrenches have a common force vector, so $\hat{e}$ is a vertical segment in wrench space.

Let $r(e_1, e_2) := \bigcup\{\pi(\overline{w_1 w_2}) \mid w_1 \in \hat{e_1}, w_2 \in \hat{e_2}\}$. The region $r(e_1, e_2)$ is the union of at most four relatively open trapezoids with two vertical sides, where each trapezoid is either all gray or all black. For a concave vertex p, let $s(p)$ be as above in Section 2.2. By Lemma 2.2, a triple (p, e_1, e_2) allows a form-closure grasp if and only if a black part of $s(p)$ intersects a gray trapezoid of $r(e_1, e_2)$, or vice versa. Observe that the edges of P define $O(n^2)$ trapezoids: at most four trapezoids for each pair of edges.

It remains to solve the following problem: given a set of m line segments and a set of $O(n^2)$ trapezoids, find all intersections between a line segment and a trapezoid. We observe that a segment s intersects a trapezoid r if and only if the midpoint of s lies in r, or s intersects one of the sides of r. We test the two cases separately.

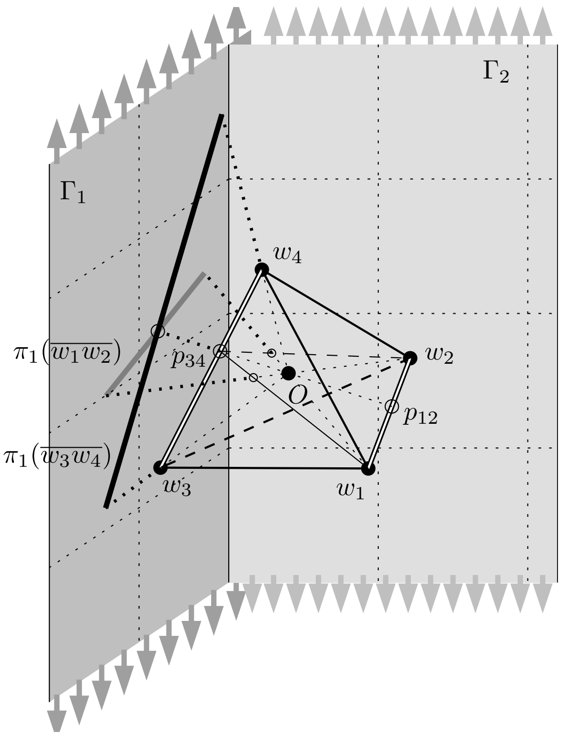

Figure 3: $\pi_1(\overline{w_1w_2})$ and $\pi_1(\overline{w_3w_4})$ intersect, and their colors differ. Therefore, the origin O must lie in the interior of the convex hull of $w_1, \ldots, w_4$.

For the first we are going to use a *triangle search structure*, which stores a set P of points in $\mathbb{R}^2$ and supports queries of the following form: given a query triangle Δ, report the points in $\Delta \cap P$. For the second we will use a *segment intersection structure* that stores a set S of line segments in $\mathbb{R}^2$, and supports queries of the form: given a query segment q, report all segments $s \in S$ with $s \cap q \neq \emptyset$. For both data structures, there are several alternatives. However, in this paper, we will stick to data structures based on Matoušek's hierarchial cuttings [15], because of the good bounds on the preprocessing time. This works well for our application, but if the number of concave vertices m is very small in comparison to the number of edges n, slightly better solutions may be possible by choosing data structures with slightly more preprocessing time—see, for example, the survey by Agarwal and Erickson [2].

Matoušek explains how we can build, for any set P of m points in the plane, and a prescribed parameter t such that $\log m \leqslant t \leqslant m$, a tree of height $O(\log m)$ with the following properties:

- the number of nodes at depth i is $O(\rho^{2i})$, for some constant ρ; each node v at depth i has an associated subset P_v of P of size $O(m/\rho^i)$;
- there are $O((m/t)^2)$ leaves v, and their sets P_v have size $O(t)$;
- for any halfplane H, the points in $P \cap H$ are exactly the points in the sets associated with a set of non-leaf nodes (one node at each depth in the tree), plus some or all of the points in a single leaf. The set of non-leaf nodes and the leaf can be identified in $O(\log m)$ time.

The tree can be built in $O(m^2/t)$ time.[2] If we just store the sets P_v explicitely, this tree can obviously

[2]Matoušek's theorems are in fact far more general than this, but to keep it simple, we rephrased it with restrictions to

be used to answer halfplane range reporting queries in $O(\log m + t + k) = O(t + k)$ time: find the leaf, check its complete contents, and find the non-leaf nodes, and just report their complete contents.

To extend this approach to a triangle search or segment intersection structure, we proceed as follows. We generalize the above tree a little bit. Instead of points p, we store tuples of points $(p_1, ..., p_h)$. The halfplane property of the tree will now read as: "For any halfplane H, the tuples $\{(p_1, ..., p_h)|p_1 \in H\}$ are exactly the tuples...". We call such a tree an order-1 tree. A tree of order j, for $j > 1$, will be just like an order-1 tree, with two exceptions. First, and most important: each set P_v for a node v of the order-j tree will be stored as a tree of order $(j-1)$ on the tuples in P_v. Second, the halfplane property now reads as: "For any halfplane H, the tuples $\{(p_1, ..., p_h)|p_j \in H\}$ are exactly the tuples...".

Lemma 2.3 *A tree of order $(j-1)$:*

- *can be built in time $O(m^2(\log^{j-1} m)/t)$, and*
- *can be used to report, for any set of j halfplanes $(H_1, ..., H_j)$, all tuples $\{(p_1, ..., p_h)|\forall_{1 \leqslant i \leqslant j} \quad p_i \in H_i\}$, in time $O(t \log^{j-1} m + k)$, where k is the number of tuples reported.*

Proof: We proof the lemma by induction on j.

For $j = 1$, it is obviously true.

The construction of a tree of order $j > 1$, consists of the construction of the main structure, in $O(m^2/t)$ time, and the construction of the associated trees of order $(j-1)$. By the induction hypothesis, the construction of an order-$(j-1)$ tree at depth i in the order-j tree takes $O((m/\rho^i)^2(\log^{j-2} m)/t)$ time. The construction times thus add up to:

$$O\left(\frac{m^2}{t}\right) + \sum_{i=0}^{O(\log m)} O(\rho^{2i}) O\left(\left(\frac{m}{\rho^i}\right)^2 \frac{\log^{j-2} m}{t}\right) = O\left(\frac{m^2 \log^{j-1} m}{t}\right)$$

The search in an order-j tree with H_j yields $O(\log m)$ nodes whose order-$(j-1)$ trees have to be searched. By the induction hypothesis, searching the order-$(j-1)$ trees costs $O(t \log^{j-2} m + k)$ for each tree, which adds up to $O(t \log^{j-1} m + k)$. Furthermore, one leaf of size t has to be searched, for a cost of $O(t)$, so that the total time spent searching is $O(t \log^{j-1} m + k)$. ◻

Setting t to $\log m$, we find that an order-j tree can be built in $O(m^2 \log^{j-2} m)$ time and answers queries in $O(\log^j m + k)$ time.

We can use this structure as a triangle search structure by storing each point p as a tuple (p, p, p) in a tree of order 3. The construction time is $O(m^2 \log m)$, and we can search for points p that lie in the intersection of three halfplanes (such as triangles) in $O(\log^3 m + k)$ time.

We can also use it as a segment intersection structure, as follows (following the same transformation as, for example, in [2]). Assume that there are no vertical segments (if there are vertical segments, we must turn everything just a little bit to prevent degeneracies). We build an order-4 tree, storing each line segment $s = \overline{s_0 s_1}$ as a tuple $(l^*(s), l^*(s), s_0, s_1)$, where $l^*(s) = (a, b)$ is the dual of the supporting line $l(s) : y = ax + b$ of s. Observe that a query segment $q = \overline{q_0 q_1}$ intersects s if and only if the following two conditions are met:

- s_0 lies above $l(q)$ while s_1 lies below $l(q)$ (or the other way around), and

match our purposes.

- q_0 lies above $l(s)$ while q_1 lies below $l(s)$, or equivalently: $l^*(s)$ lies below the dual line q_0^* of q_0 and above the dual line q_1^* of q_1 (or the other way around).

An intersection query with a line segment can thus be formulated as a query with four halfplanes, bounded by q_0^*, q_1^*, and $l(q)$ (twice), in the order-4 tree storing tuples $(l^*(s), l^*(s), s_0, s_1)$. Such a tree is built in time $O(m^2 \log^2 m)$, and a query is answered in $O(\log^4 m + k)$ time. [3]

To return to solving our original problem: we build, in $O(m^2 \log m)$ time, a triangle search structure on the set of midpoints of the segments: this permits queries with a trapezoid (by decomposing it into triangles), identifying the k points inside the trapezoid in $O(\log^3 m + k)$ time. Furthermore, we build, in $O(m^2 \log^2 m)$ time, a segment intersection structure for segment intersection queries on the segments that represent the concave vertices. Finding all k segments intersecting a given trapezoid boundary takes $O(\log^4 m + k)$ time.

To find all the k segments intersecting a given trapezoid, we query both data structures, in total time $O(\log^4 m + k)$. Since there are $O(n^2)$ trapezoids, this takes time $O(n^2 \log^4 m + K)$.

Theorem 2.3 *Given a polygon with m concave vertices and n edges, all K combinations of one concave vertex and two edges, such that one contact on the vertex and one each on the interior of each edge can put the object in form closure, can be computed in time $O(n^2 \log^4 m + K)$.*

2.4 Triples of Two Concave Vertices and One Edge

Placing two point contacts at a pair of concave vertices p, q may not achieve form closure. Placing one more contact in the interior of an appropriate edge e, however, can achieve form closure with p and q. Here, we present an algorithm to report all such triples (p, q, e).

Consider a pair of concave vertices p, q that does not achieve form closure. Let w_1, w_2 and w_3, w_4 be the wrenches induced by p and q, respectively, and let $W := \{w_1, w_2, w_3, w_4\}$. By Theorem 2.1 the origin O does not lie in the interior of the convex hull of W. An additional edge contact achieves form closure if and only if O lies in the interior of the convex hull of $W \cup \{w\}$, where w is the wrench induced by the contact.

Let $W' := W \cup \{O\}$. The convex hull of W' is a convex polytope with four or five vertices,[4] one of which is O. Consider a facet f_i incident to O, and let H_i be the open half-space bounded by the supporting plane of f_i not containing W'. If O lies in the interior of the convex hull of $W \cup \{w\}$, for some w, then $w \in H_i$ for all i's. Conversely, if this is true for every facet incident to O, then O does lie in the interior of the convex hull of $W \cup \{w\}$.

It follows that an edge e can achieve form closure together with p and q if and only if the wrench space segment $\hat{e}$ intersects the intersection of three or four half-spaces. The bounding planes of these half-spaces pass through O, so we can again project everything onto a two-dimensional screen. Here, we do not wish to identify wrenches that are symmetric around the origin, so we use a screen Γ' enclosing the origin as follows:

$$\Gamma' := \{(x, y, z) | \max(|x|, |y|) = 1, z \in \mathbb{R}\}.$$

[3]Matoušek improves the construction time for the triangle search structure to $O(m^2 \log^\delta m)$ (with the same query time); the same technique could be used to improve the construction time for the segment intersection structure to $O(m^2 \log^{1+\delta} m)$. However, these improvements don't affect our final bounds, so we will ignore them for simplicity.

[4]If W' has four vertices, one of the wrenches is redundant. This means that form closure could also be achieved by placing point contacts on e, at one of the vertices p or q, and on one of the edges incident to the other vertex. This triple will be reported by the algorithm given above for finding all combinations of one concave vertex and two edges that yield a form-closure grasp.

To prevent degeneracies, we would turn the screen a little so that no segment is projected onto an edge of the screen. We project the n segments $\hat{e}$ onto Γ', build a triangle search structure on their endpoints, and a segment intersection structure on the segments themselves.

For the triangle search structure we use a structure by Matoušek again; however, this time we use the variant that allows us to balance preprocessing and query time. More precisely, we can choose a parameter M (in fact the size of the structure) such that $n \leqslant M \leqslant n^2$, and will get a preprocessing time of $O(n^{1+\delta} + M \log^\delta n)$, for an arbitrarily small constant $\delta > 0$, and a query time of $O((n/\sqrt{M}) \log^3 \frac{M}{n} + k)$. We choose $M = n^2 / \log_m^\delta n$, to get a preprocessing time which is, in any case, $O(n^2 \log^\delta m)$, and a query time of $O((\log_m^{\delta/2} n) \log^3 n + k)$. With $\delta \leqslant 2$ this is certainly $O(\log^4 n + k)$.

Before we choose the segment intersection structure, observe that all segments to be stored are vertical. A query segment $q = \overline{q_0 q_1}$, where $q_i = (x(q_i), y(q_i))$, intersects a stored segment $s = \overline{s_0 s_1}$, where $s_i = (x(s), y(s_i))$, if and only if the following two conditions are met:

- s_0 lies above $l(q)$ while s_1 lies below $l(q)$ (or the other way around), and
- $x(s)$ lies between $x(q_0)$ and $x(q_1)$.

Therefore, we can solve our query problem with an order-2 structure, as explained in the previous section. The structure stores tuples (s_0, s_1), and stores the sets P_v associated with internal nodes in order-1 trees sorted by x-coordinate. We can pre-sort all segments by x-coordinate as an initialization step, and keep them sorted while distributing and copying them to subtrees, so that no further sorting is necessary. Thus, the complete structure can be constructed in the same time bound as a normal order-2 structure, that is, $O(n^2)$. The query time of an order-2 structure is normally $O(\log^2 n + k)$, but in this case, we cannot just report all contents of the internal nodes found: we have to do a binary search to report only those segments with x-coordinates between $x(q_0)$ and $x(q_1)$. This increases the query time to $O(\log^3 n + k)$.

In total, both data structures are built in $O(n^2 \log^\delta m)$ time. We now consider each pair (p, q) of concave vertices in turn. We compute the wrenches W induced by the two vertices, the convex hull of $W \cup \{O\}$, and the intersection R of the three or four relevant half-spaces. We then compute $R' := R \cap \Gamma'$, a polygonal area of constant complexity. We triangulate R', and find the k segment endpoints inside R' by triangle range queries in time $O(\log^4 n + k)$. Furthermore, we find all k segments intersecting the boundary of R' in time $O(\log^3 n + k)$ by a constant number of segment intersection queries. Since there are $\Theta(m^2)$ pairs of concave vertices, the total running time is $O(n^2 \log^\delta m + m^2 \log^4 n + k)$.

To list *all* triples of two concave vertices and one edge that yield a form-closure grasp, we should also run the algorithm of Section 2.2, to get, in time $O(m^{4/3} \log^{1/3} m + k)$, all k pairs of concave vertices that yield a form-closure grasp, and combine the result with every edge of the polygon.

Theorem 2.4 *Given a polygon with m concave vertices and n edges, all K combinations of two concave vertices and one edge, such that one contact each on each vertex and one contact on the interior of the edge can put the object in form closure, can be computed in time $O(n^2 \log^\delta m + m^2 \log^4 n + K)$.*

It would be possible to trade some of the dependency on n in this bound for dependency on m, by exploiting the trade-off between preprocessing and query time for triangle search and intersection search structures. However, in the end it would not affect the final bounds for describing all three-point form-closure grasps, as that requires running the $O(n^2 \log^4 m + K)$-time algorithm from the previous section anyway. The latter will dominate the bound on the total running time.

2.5 Triples of Concave Vertices

A triple of concave vertices (p, q, r) induces six wrenches $w_1, ..., w_6$. Three point contacts in these vertices put an object in form closure if the convex hull of the six wrenches contains the origin in its interior. We can distinguish two cases:

1. a subset of five wrenches already contains the origin in the interior of its convex hull, and thus achieves form closure;

2. no subset of five wrenches contains the origin in the interior of its convex hull.

In the first case, only two of the concave vertices contribute two wrenches to the convex hull. The contact in the third vertex contributes only one wrench: it could just as well have been placed close by on the corresponding edge which is incident to that vertex. The first case is thus very similar to the case discussed in Section 2.4. The algorithm of that section can easily be adapted to list all such cases. We will just use the triangle search structure only, not the segment intersection structure, and store only the edge end points that are actually concave vertices. Building the data structure takes $O(m^2 \log^\delta m)$ time; we do $O(m^2)$ queries in $O(\log^3 m + k)$ time each; thus, we can list all triples of concave vertices of the first case in time $O(m^2 \log^3 m + K)$.

For the second case, we will make use of the following lemma from the theory of positive bases [7, 12]:

Lemma 2.4 *Let S be any set of six points in $\mathbb{R}^3$ such that the convex hull of S contains the origin in its interior, but no subset of five points of S contains the origin in the interior of its convex hull. It follows that S consists of six points on three lines through the origin: on each line, one point to each side of the origin.*

It follows that the wrenches induced by the three concave vertices must form three pairs of opposite wrenches. Since no vertex contact could induce opposite wrenches itself, it follows that we are looking for triples (p, q, r) where $w_1(q) = -w_2(p)$, $w_1(r) = -w_2(q)$, and $w_1(p) = -w_2(r)$.

A straightforward algorithm is now as follows. We sort all wrenches induced by concave vertices lexicographically. For every concave vertex p, we search in the sorted list for matching vertices q, that is, vertices q with $w_1(q) = -w_2(p)$. For each vertex q found, we do another search for a vertex r such that $w_1(r) = -w_2(q)$ and $w_2(r) = -w_1(p)$. If such a vertex r is found, we report the triple (p, q, r).

The sorting is done in $O(m \log m)$ time. The query for q, and testing for a matching r, takes $O(\log m)$ time per candidate-q which is tested, which amounts to $O(m \log m)$ in the worst case. Searching for matching q and r for each vertex p thus takes $O(m^2 \log m)$ time.

In total, both cases can be dealt with in $O(m^2 \log^3 m + K)$ time.

Theorem 2.5 *Given a polygon with m concave vertices and n vertices in total, all K form-closure grasps formed by three concave vertices can be computed in time $O(m^2 \log^3 m + K)$.*

3 Computing All Immobility Grasps

In this section we will explain how to find combinations of three edges and combinations of a concave vertex and an edge, that allow a three- or two-point immobility grasp, respectively.

We first introduce some notations and definitions used in this section. Let the edges of the simple polygon P be oriented counter-clockwise around P, that is, P lies locally to the left of each edge. We denote the line orthogonal to an edge e through the start and end point of e by $s_0(e)$ and $s_1(e)$,

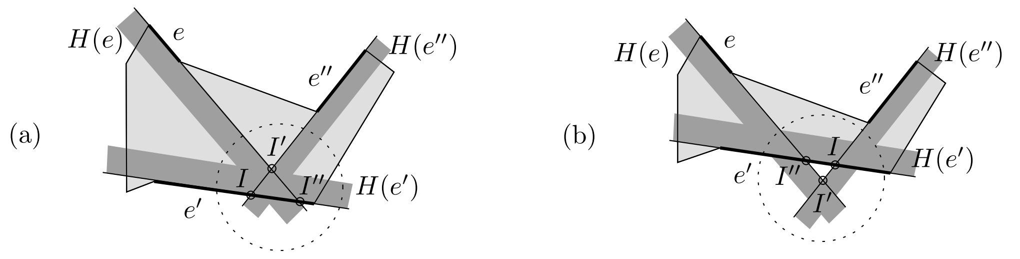

Figure 4: The edges e, e', e'' in (a) are a triangular triple, while those in (b) are not.

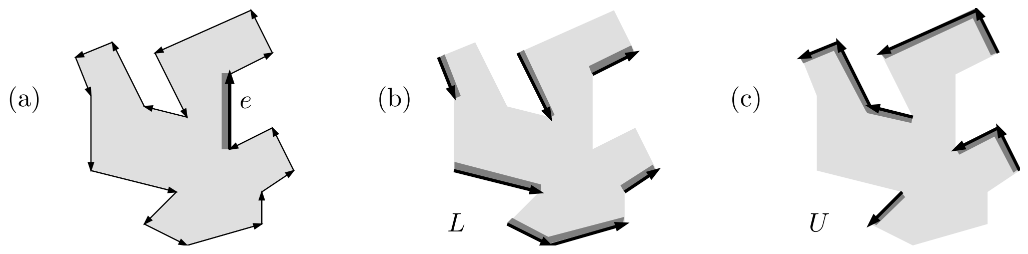

Figure 5: (a) A polygon with directed edges, oriented such that e is on the y-axis, pointing upward. (b) The edges in L, and (c) in U.

respectively. Let $\hat{s}(e)$ be the relatively open infinite slab bounded by $s_0(e)$ and $s_1(e)$, that is, the union of all lines that are orthogonal to and intersect the interior of e (see Figure 6). Let $l(e)$ be the supporting line of e, and let $H(e)$ be the open half-plane bounded by $l(e)$ lying locally to the left of e, that is, locally containing P (see Fig. 4). When the intersection of $H(e), H(e')$ and $H(e'')$ forms a (finite) triangle, then e, e', and e'' are said to be a *triangular triple*. (Compare Fig. 4 (a) with (b).)

3.1 Immobility Grasps with Three Contacts

A necessary and sufficient condition for three edges to have a configuration of three point contacts that immobilizes a simple polygon is provided by Czyzowicz et al. [6].

Lemma 3.1 (Czyzowicz et al. [6]) *There are three point contacts on the interior of three edges e, e', and e'' that immobilize a polygon if and only if:*

(i) $\hat{s}(e) \cap \hat{s}(e') \cap \hat{s}(e'') \neq \emptyset$ *(common normal intersection condition), and*

(ii) $H(e) \cap H(e') \cap H(e'')$ *is a triangle (triangular triple condition).*

To find all such edge triples, we take a similar approach as in [26]. We find all the edge triples that have a common normal intersection; among these, triangular triples will be filtered out. The sketch of the global approach is as follows. For each edge e of P, we build a data structure. It will be queried with each of the remaining $n-1$ edges e', to report all edges e'' such that the triple (e, e', e'') satisfies the conditions of Lemma 3.1.

From now on, we focus on building and searching the data structure for a fixed edge e. We choose a coordinate system such that $l(e)$ is the y-axis, oriented in upward direction. We divide the remaining edges into two groups L(ower) and U(pper); when an edge forms an angle between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ with the positive x-axis, it is in L, otherwise it is in U (see Fig. 5 (b) and (c)). We omit all vertical edges from L and U, since they could never make a triangular triple with e and a third edge. For $i \in \{0, 1\}$,

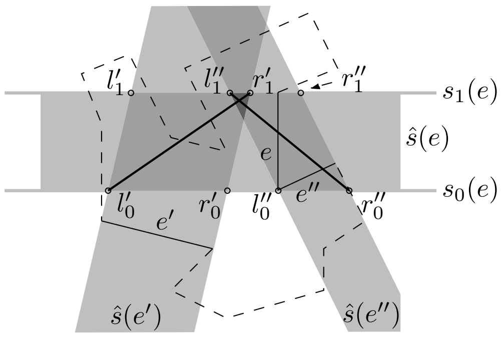

Figure 6: Notation for Lemma 3.2.

we define l'_i and r'_i as the x-coordinates of the left and right intersection points of $s_i(e)$ and the slab boundaries of $\hat{s}(e')$. We define l''_0, r''_0, l''_1 and r''_1 for edge e'' likewise—see Figure 6. The following is a necessary and sufficient condition for three edges to have a non-empty common normal intersection region.

Lemma 3.2 *Two slabs $\hat{s}(e')$ and $\hat{s}(e'')$ have a common intersection with $\hat{s}(e)$ if and only if one of the following is true:*

(i) $l'_0 < r''_0 \wedge l''_1 < r'_1$

(ii) $l''_0 < r'_0 \wedge l'_1 < r''_1$

Proof: We will first prove that if one of the conditions (i) or (ii) is met, there is a common intersection. After that we will prove that if neither of the conditions is fulfilled, there cannot be a common intersection.

The “if” direction: the two cases are identical except for e' and e'' switching roles, so without loss of generality, we restrict ourselves to the first case. Condition (i) implies that the line segments $\overline{l'_0 r'_1}$ and $\overline{r''_0 l''_1}$ intersect (see Figure 6 for an example). The first line segment lies completely inside $\hat{s}(e)$ and $\hat{s}(e')$; the second lies completely inside $\hat{s}(e)$ and $\hat{s}(e'')$. Hence, their intersection lies in all three slabs, which means that the intersection of $\hat{s}(e)$, $\hat{s}(e')$ and $\hat{s}(e'')$ is not empty.

The “only-if” direction: suppose neither condition (i) nor condition (ii) is true, i.e. the following *is* true:

$$(l'_0 \geqslant r''_0 \vee l''_1 \geqslant r'_1) \wedge (l''_0 \geqslant r'_0 \vee l'_1 \geqslant r''_1)$$

Because by definition, $r'_0 > l'_0$ and $r''_0 > l''_0$, we cannot simultaneously have $l'_0 \geqslant r''_0$ and $l''_0 \geqslant r'_0$. Likewise, we cannot simultaneously have $l'_1 \geqslant r''_1$ and $l''_1 \geqslant r'_1$. It follows that the proposition above is equivalent to:

$$(l'_0 \geqslant r''_0 \wedge l'_1 \geqslant r''_1) \vee (l''_0 \geqslant r'_0 \wedge l''_1 \geqslant r'_1)$$

In other words: the left boundary of one slab of $\hat{s}(e')$ and $\hat{s}(e'')$ lies to the right of the right boundary of the other slab, and the situation is the same both at the intersection with the lower boundary of $\hat{s}(e)$, and at the intersection with the upper boundary of $\hat{s}(e)$. It follows that the intersections of $\hat{s}(e')$ and $\hat{s}(e'')$ with $\hat{s}(e)$ are disjoint.

So if there is a common intersection, at least one of the conditions must be fulfilled, and if at least one of the conditions is fulfilled, there must be a common intersection. ◻

Suppose $l(e')$ is the line defined by $y = a'x + b'$, and $l(e'')$ is the line defined by $y = a''x + b''$.

Lemma 3.3 *$H(e) \cap H(e') \cap H(e'')$ is a triangle if and only if one of the following is true:*

(i) $a' < a'' \wedge b' < b'' \wedge e' \in L \wedge e'' \in U$

(ii) $a'' < a' \wedge b'' < b' \wedge e' \in U \wedge e'' \in L$

Proof: Let I' be the intersection $(0, b'')$ of $l(e)$ and $l(e'')$; let I'' be the intersection $(0, b')$ of $l(e)$ and $l(e')$, and let I be the intersection (I_x, I_y) of $l(e')$ and $l(e'')$, where $I_x = (b'' - b')/(a' - a'')$ and $I_y = a'I_x + b' = a''I_x + b''$ (see Figure 4). Observe that $H(e) \cap H(e') \cap H(e'')$ is a triangle if and only if $I \in H(e)$, $I' \in H(e')$ and $I'' \in H(e'')$.

We will first prove that if one of the conditions (i) or (ii) holds, $H(e) \cap H(e') \cap H(e'')$ is a triangle, and then, that if the latter is a triangle, one of the conditions must be fulfilled.

The “if” direction: the two cases are identical except for e' and e'' switching roles, so without loss of generality, we restrict ourselves to the first case. Condition (i) (as well as (ii)) implies that $I_x < 0$, so $I \in H(e)$. Furthermore, $e' \in L$ means that $H(e')$ is the halfplane above $l(e')$; since $b' < b''$, we have that $I' = (0, b'')$ lies above $l(e')$, and thus, inside $H(e')$. Likewise, from $e'' \in U$ and $b' < b''$ it follows that $I'' \in H(e'')$. Hence, $H(e) \cap H(e') \cap H(e'')$ is a triangle.

The “only-if” direction: suppose $H(e) \cap H(e') \cap H(e'')$ is a triangle, then $I = l(e') \cap l(e'') \in H(e)$, that is: $I_x = (b'' - b')/(a' - a'') < 0$. This implies that one of the following is true:

(1) $a' < a'' \wedge b' < b''$

(2) $a'' < a' \wedge b'' < b'$

In the first case, $I' = (0, b'')$ lies above $I'' = (0, b')$, so the triangle formed by the $l(e')$, $l(e'')$, and the y-axis $l(e)$, is bounded by $l(e')$ from below and by $l(e'')$ from above. From the fact that this triangle lies inside $H(e')$ and $H(e'')$, it follows that $e' \in L$ and $e'' \in U$, fulfilling condition (i) of the Lemma. In the same manner, we can derive that the second case implies that $e' \in U$ and $e'' \in L$, fulfilling condition (ii) of the Lemma. $\square$

From Lemmas 3.1, 3.2 and 3.3 it follows that e, e' and e'' allow a three-point immobility grasp if and only if one of the following conditions is satisfied:

(i) $l''_1 < r'_1 \wedge r''_0 > l'_0 \wedge a'' > a' \wedge b'' > b' \wedge e' \in L \wedge e'' \in U$

(ii) $l''_1 < r'_1 \wedge r''_0 > l'_0 \wedge a'' < a' \wedge b'' < b' \wedge e' \in U \wedge e'' \in L$

(iii) $l'_1 < r''_1 \wedge r'_0 > l''_0 \wedge a'' > a' \wedge b'' > b' \wedge e' \in L \wedge e'' \in U$

(iv) $l'_1 < r''_1 \wedge r'_0 > l''_0 \wedge a'' < a' \wedge b'' < b' \wedge e' \in U \wedge e'' \in L$

Since the roles of e' and e'' are interchangeable, we only need to search for triples satisfying condition (i) or (ii). We can do this with two four-dimensional orthogonal range trees [8] as follows. In the first tree, store every edge $e'' \in U$ as a four-dimensional point (l''_1, r''_0, a'', b''). Query this tree with every edge $e' \in L$ for all points in $\langle -\infty, r'_1 \rangle \times \langle l'_0, \infty \rangle \times \langle a', \infty \rangle \times \langle b', \infty \rangle$. In the second tree, store every edge $e'' \in L$ as a four-dimensional point (l''_1, r''_0, a'', b''). Query this tree with every edge $e' \in U$ for all points in $\langle -\infty, r'_1 \rangle \times \langle l'_0, \infty \rangle \times \langle -\infty, a' \rangle \times \langle -\infty, b' \rangle$. Every edge e'' reported forms a triple with e and e' such that three point contacts on e, e' and e'' will immobilize the polygon.

Now we analyze the time complexity of this algorithm. A four-dimensional orthogonal range tree can be built in $O(n \log^3 n)$ time using $O(n \log^3 n)$ space, and can be queried in $O(\log^4 n + k)$ time

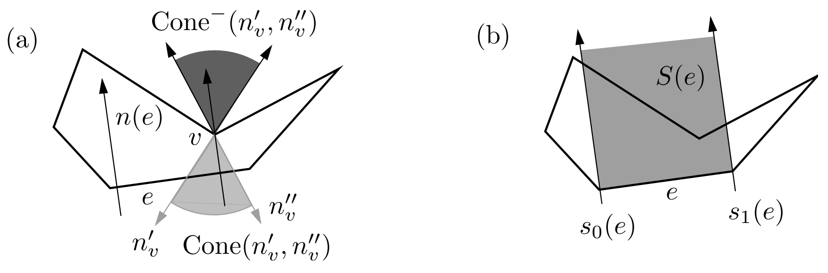

Figure 7: (a) Notations of normals of edges and concave vertices. (b) Vertex p is in the simplex $S(e)$ of edge e.

(see Chapter 5.4 in [8]). This can be improved to $O(\log^3 n + k)$ query time, with the same building time, using fractional cascading (see Chapter 5.6 in [8]).

We query each tree with $O(n)$ edges e', for a total building and query time of $O(n \log^3 n + k)$ per tree, and we do this for every edge e, so that the complete search takes $O(n^2 \log^3 n + k)$ time.

Theorem 3.1 *Given a polygon with n edges, all K edge triples (e, e', e'') such that the polygon can be immobilized by three frictionless point fingers on the interiors of e, e' and e'', can be computed in time $O(n^2 \log^3 n + K)$.*

3.2 Immobility Grasps with Two Contacts

If we exploit concave vertices, two contacts can immobilize a simple polygon: one at a concave vertex p and the other in the interior of an edge e. When a polygon has n edges and m concave vertices, all such pairs can be reported in time $O(mn)$ by simply checking all vertex-edge pairs. Obviously we want a more efficient algorithm. We could adapt the algorithm in Section 3.1 to find and report only triples of edges where two edges are in fact reduced to points that coincide on a concave vertex. But this would cost even more than $O(mn)$ time. Therefore we develop a specialized algorithm based on the lemma below.

First, we introduce some notations used in this section. Let e' and e'' be the edges incident to p. Let n'_p be the inward normal to e', and let n''_p be the inward normal to e''. Let Cone(n'_p, n''_p) be $\{\lambda' n'_p + \lambda'' n''_p | \lambda', \lambda'' > 0\}$, that is, the set of all positive linear combinations of n'_p and n''_p. In the same way, let Cone$^-(n'_p, n''_p)$ be the set of all positive linear combinations of $-n'_p$ and $-n''_p$ (see Figure 7(a)). For each edge e, let n_e be the inward normal to e, and let the open simplex $S(e)$ be $\hat{s}(e) \cap H(e)$ (see Figure 7 (b).)

Lemma 3.4 *Placing two point contacts at a concave vertex p and on an edge e immobilizes a polygon if and only if:*

(i) $n_e \in$ *Cone*$^-(n'_p, n''_p)$*, and*

(ii) $p \in S(e)$.

Proof: Let e' and e'' be the adjacent edges to p, shrunk onto the vertex p, so that $\hat{s}(e')$ is the line orthogonal to e' through p, and $\hat{s}(e'')$ is the line orthogonal to e'' through p. We will first show that any three edges e, e' and e'' satisfying the above statement must satisfy Lemma 3.1. Since $p = \hat{s}(e') \cap \hat{s}(e'') \in S(e) \subset \hat{s}(e)$, we must have $\hat{s}(e) \cap \hat{s}(e') \cap \hat{s}(e'') \neq \emptyset$. Furthermore, since $p \in S(e) \subset H(e)$, the intersection $H(e) \cap H(e') \cap H(e'') \neq \emptyset$. In fact, $H(e) \cap H(e') \cap H(e'')$ is a triangle, because $n_e \in$ Cone$^-(n'_p, n''_p)$, i.e., the normals of e, e' and e'' span the plane positively.

Let us now show that any three edges e, e' and e'', such that e' and e'' are both adjacent to and shrunk onto a common concave vertex p, and e, e' and e'' satisfy Lemma 3.1, must also satisfy the two conditions above. The common normal intersection condition assures that $p \in \hat{s}(e)$. The triangular triple condition, that $H(e) \cap H(e') \cap H(e'') \neq \emptyset$, implies that the normals of the edges span the plane positively, which proves that $n_e \in \text{Cone}^-(n'_p, n''_p)$. ◻

For any edge e and any concave vertex p, let θ_e, θ'_p and θ''_p be the angles that n_e, $-n'_p$ and $-n''_p$, respectively, make with the positive x-axis. Let θ'_p be the smaller angle of θ'_p and θ''_p, i.e. $\theta'_p < \theta''_p$. Observe that $n_e \in \text{Cone}^-(n'_p, n''_p)$ if and only if one of the following is true:

(i) $\theta''_p - \theta'_p < \pi \wedge \theta_e \in \langle \theta'_p, \theta''_p \rangle$

(ii) $\theta''_p - \theta'_p > \pi \wedge \theta_e \in [-\pi, \theta'_p\rangle \cup \langle \theta''_p, \pi]$.

For a given edge e, we find all concave vertices that have a two-point immobilizing grasp with e. For this, we store the concave vertices in a data structure that stores pairs of the form (I_p, p), where I_p is a one-dimensional interval and p is a point in the plane. Each vertex p with $\theta''_p - \theta'_p < \pi$ is stored once, as a pair $(\langle \theta'_p, \theta''_p \rangle, p)$. Each vertex p with $\theta''_p - \theta'_p > \pi$ is stored twice: once as a pair $(\langle -\infty, \theta'_p \rangle, p)$ and once as $(\langle \theta''_p, \infty \rangle, p)$. We query this data structure with each edge e of P, to report all vertices p stored as a pair (I_p, p) such that $\theta_e \in I_p$ and $p \in S(e)$.

The data structure we use is a two-level data structure. The top level is a segment tree [8] on the intervals I_p. Let X be the set of all begin and end points of intervals I_p to be stored in the tree, in order of increasing value. A segment tree is a balanced binary tree on the intervals between consecutive values from X: each leaf is associated with one such interval. Each internal node v is associated with an interval $I(v)$, which is the union of the intervals of its descendants. With each node v, we associate a data structure $\mathcal{T}(v)$ that stores all pairs (I_p, p) such that I_p contains $I(v)$ but not $I(\text{parent}(v))$. In our case, the data structures $\mathcal{T}(v)$ will be triangle search structures on the points p in the pairs (I_p, p). Again, we use the triangle search structure by Matoušek [15], using $O(m^d)$ space to store $O(m)$ points, for a certain constant d. We will explain later how d is chosen, but in any case, we will choose it such that $1 < d \leqslant 2$.

Let us first analyse the time needed to construct the data structure. A simplex range searching structure can be built in time $O(m^d \log^\delta m)$, where m is the number of points stored, m^d is the amount of storage used for them, and δ is any small positive constant [15]. A node v at depth i in a segment tree stores intervals I_p that completely contain $I(v)$, but not $I(\text{parent}(v))$, which means that all intervals I_p stored in v must have an endpoint in $I(\text{brother}(v))$. Since the segment tree is balanced, there are at most $2m/2^i$ such intervals. Thus, at each depth i in the segment tree, we have at most 2^i nodes storing at most $2m/2^i$ intervals each. The time needed to build the complete tree thus becomes $O(m \log m)$ (for the segment tree itself) plus, for the associated triangle search structures, $O(\sum_{i=0}^{\log m} 2^i O((m/2^i)^d \log^\delta (m/2^i)))$. Since $d > 1$, the larger triangle search structures dominate, making a total construction time of $O(m^d \log^\delta m)$.

A query with an edge e for matching p in this multi-level data structure proceeds as follows. We walk down the segment tree, finding all $O(\log m)$ nodes v (one at each depth) such that $I(v)$ contains θ_e. Together, these nodes contain all pairs (I_p, p) such that I_p contains θ_e. For each of these nodes, we search the associated triangle search structure, and report the answers. The query time in a simplex range searching structure on m points with m^d storage is $O(m(\log^3 m^{d-1})/\sqrt{m^d} + k)$. The total time for a query in our data structure is therefore

$$O\left(\sum_{i=0}^{\lceil \log m \rceil}\left(1+O\left(\frac{m}{2^i}\frac{\log^3\left(\frac{m}{2^i}\right)^{d-1}}{\sqrt{\left(\frac{m}{2^i}\right)^d}}\right)\right)+k\right)$$

If $d = 2$, this is $O(\log^4 m + k)$, otherwise it is $O(m(\log^3 m)/\sqrt{m^d} + k)$.

Since we do n queries, the time for building and querying the data structure adds up to $O(m^d \log^\delta m + n\log^4 m + K)$ (for $d = 2$) or $O(m^d \log^\delta m + nm(\log^3 m)/\sqrt{m^d} + K)$ (for $1 < d < 2$).

Let's now choose d. If $m/\log^{3/2} m \leqslant \sqrt{n}$, we choose $d = 2$, and the algorithm runs in $O(n \log^4 m + K)$ time. Otherwise, we have $\sqrt{n} < m/\log^{3/2} m$, so $n^{2/3}\log^2 m < m^{4/3}$, and thus $(mn)^{2/3}\log^2 m < m^2$. Furthermore we have $n > m$, so certainly $n^{2/3}\log^2 m > m^{1/3}$, and thus $(mn)^{2/3}\log^2 m > m$. Hence we can choose d such that $1 < d < 2$ and $m^d = (mn)^{2/3}\log^2 m$, resulting in a total running time of $O(m^d \log^\delta m + nm(\log^3 m)/\sqrt{m^d} + K) = O((mn)^{2/3}\log^{2+\delta} m + K)$.

Theorem 3.2 *Given a polygon with n edges and m concave vertices, all K pairs of an edge e and a concave vertex p such that the polygon can be immobilized by two frictionless point fingers on e and at p, can be computed in time $O(n \log^4 m + (mn)^{2/3}\log^{2+\delta} m + K)$.*

Acknowledgments

We thank Günter Rote, Marc van Kreveld, Mark de Berg and Otfried Cheong for helpful discussions.

References

[1] P. K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete & Computational Geometry*, 5:533–573, 1990.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] R. Brost and K. Goldberg. A complete algorithm for designing planar fixtures using modular components. *IEEE Transactions on Robotics and Automation*, 12:31–46, 1996.

[4] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(1):145–158, 1993.

[5] I.-M. Chen and J.W. Burdick. Finding antipodal point grasps on irregularly shaped objects. *IEEE Transactions on Robotics and Automation*, 9(4):507–512, 1993.

[6] J. Czyzowicz, I. Stojmenovic, and J. Urrutia. Immobilizing a shape. *International Journal of Computational Geometry and Applications*, 9(2):181–206, 1999.

[7] C. Davis. Theory of positive linear dependence. *American Journal of Mathematics*, pages 733–746, 1954.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997.

[9] A. J. Goldman and A. W. Tucker. Polyhedral convex cones. *Linear Inequalities and Related Systems*, pages 19–40, 1956.

[10] G. Gopalakrishnan and K. Goldberg. Gripping parts at concave vertices. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1590–1596, May 2002.

[11] Y.-B. Jia. Curvature-based computation of antipodal grasps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1571–1577, May 2002.

[12] R. M. Lewis and V. Torczon. Rank ordering and positive bases in pattern search algorithms. Technical report, crpc-tr96674, Center for Research on Parallel Computation, Rice University, Nov 1996.

[13] J.-W. Li, M.-H. Jin, and H. Liu. A new algorithm for three-finger force-closure grasp of polygonal objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1800–1804, 2003.

[14] X. Markenscoff, L. Ni, and C. H. Papadimitriou. The geometry of grasping. *International Journal of Robotics Research*, 9(1):61–74, 1990.

[15] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(2):157–182, 1993.

[16] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

[17] V.-D. Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3):3–16, 1988.

[18] J. Ponce and B. Faverjon. On computing three-finger force-closure grasps of polygonal objects. *IEEE Transactions on Robotics and Automation*, 11:868–881, 1995.

[19] J. Ponce, D. Stam, and B. Faverjon. On computing force-closure grasps of curved two-dimensional objects. *International Journal of Robotics Research*, 12(3):263–273, 1993.

[20] A. Rao and K. Y. Goldberg. Friction and part curvature in parallel-jaw grasping. *Journal of Robotic Systems*, 12(6):365–382, 1995.

[21] F. Reuleaux. *The Kinematics of Machinery*. Macmilly and Company, 1876. Republished by Dover in 1963.

[22] E. Rimon and J. W. Burdick. New bounds on the number of frictionless fingers required to immobilize planar objects. *J. of Robotic Systems*, 12(6):433–451, 1995.

[23] E. Rimon and J. W. Burdick. Mobility of bodies in contact—part I: A second-order mobility index for multiple-finger graps. *IEEE Transactions on Robotics and Automation*, 14:696–708, 1998.

[24] E. Rimon and J. W. Burdick. Mobility of bodies in contact—part II: How forces are generated by curvature effects. *IEEE Transactions on Robotics and Automation*, 14:709–717, 1998.

[25] K. Salisbury. *Kinematic and force analysis of articulated hands*. PhD thesis, Stanford University, 1982.

[26] A. F. van der Stappen, C. Wentink, and M. H. Overmars. Computing immobilizing grasps of polygonal parts. *International Journal of Robotics Research*, 19(5):467–479, 2000.