Discrete Comput Geom 22:201–221 (1999)

Discrete & Computational
Geometry

Motion Planning for a Convex Polygon in a Polygonal Environment*

P. K. Agarwal,[1] B. Aronov,[2] and M. Sharir[3]

[1]Department of Computer Science, Box 90129, Duke University,
Durham, NC 27708-0129, USA
pankaj@cs.duke.edu

[2]Department of Computer and Information Science, Polytechnic University,
Brooklyn, NY 11201-3840, USA
aronov@ziggy.poly.edu

[3]School of Mathematical Sciences, Tel Aviv University,
Tel Aviv 69978, Israel
sharir@math.tau.ac.il
and
Courant Institute of Mathematical Sciences, New York University,
New York, NY 10012, USA

Abstract. We study the motion-planning problem for a convex m-gon P in a planar polygonal environment Q bounded by n edges. We give the first algorithm that constructs the *entire* free configuration space (the three-dimensional space of all free placements of P in Q) in time that is near-quadratic in mn, which is nearly optimal in the worst case. The algorithm is also conceptually simple. Previous solutions were incomplete, more expensive, or produced only part of the free configuration space. Combining our solution with parametric searching, we obtain an algorithm that finds the largest placement of P in Q in time that is also near-quadratic in mn. In addition, we describe an algorithm that preprocesses the computed free configuration space so that *reachability queries* can be answered in polylogarithmic time.

* All three authors have been supported by a grant from the U.S.–Israeli Binational Science Foundation. Pankaj Agarwal has also been supported by a National Science Foundation Grant CCR-93-01259, by an Army Research Office MURI Grant DAAH04-96-1-0013, by a Sloan fellowship, and by an NYI award and matching funds from the Xerox Corporation. Boris Aronov has also been supported by NSF Grant CCR-92-11541 and a Sloan Research Fellowship. Micha Sharir has also been supported by NSF Grants CCR-94-24398 and CCR-93-11127, by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University, and by a grant from the G.I.F., the German–Israeli Foundation for Scientific Research and Development.

1. Introduction

Problem Statement. Let P be a closed convex m-gon. We consider the problem of planning a collision-free motion for P inside a closed planar polygonal *environment* Q, bounded by a total of n edges. We allow P to translate and rotate. A (*congruent*) *placement* of P is thus any congruent copy of P (without reflections). A placement of P is *free* if it is fully contained in Q, and *semifree* if it is free and the boundary ∂Q of Q touches the boundary ∂P of P. Any placement of P can be represented by three real parameters (x, y, θ), where $(x, y) \in \mathbb{R}^2$ is the position of a reference point of P and $\theta \in [-\pi, \pi]$ is the counterclockwise angle by which P is rotated from some fixed orientation. The space of all placements of P, known as *configuration space*, is thus identified with $\mathbb{R}^2 \times \mathbb{S}$, where $\mathbb{S}$ is the unit circle. The *free configuration space* $\mathcal{C}$ of P in Q is the space of all free placements of P in Q, and the boundary $\partial\mathcal{C}$ corresponds to the set of all semifree placements. Note that $\mathcal{C}$ is a closed set. If scaling of P is also permitted, the configuration space can be identified with $\mathbb{R}^3 \times \mathbb{S}$.

We consider two types of problems in this context:

Motion planning: Construct $\mathcal{C}$, the space of all free congruent placements of P. Preprocess $\mathcal{C}$ so that one can determine efficiently whether two given placements I, F of P lie in the same connected component of $\mathcal{C}$; that is, whether there exists a collision-free motion of P inside Q from one of these placements to the other. If so, then also return a path from I to F that lies within $\mathcal{C}$. See Fig. 1.

Largest placement: Allowing scaling, find a largest similar copy of P that fits inside Q.

Previous Results. Both problems are central problems in robotics and manufacturing, and have been studied intensively in computational geometry during the past two decades. Some of the initial results on this problem can be found in [14], [26], [34], and [38]; these algorithms are either inefficient or consider only special cases (e.g., where P is assumed to be a line segment). See recent surveys for a summary of known results in motion planning [22], [35]. The first significant progress was made by Leven and Sharir [27], who analyzed the *combinatorial complexity* of $\mathcal{C}$ when no scaling is allowed, which can be measured by the number of free *critical placements* of P. A placement Z of P is called *critical* if there exist three distinct pairs (e_1, v_1), (e_2, v_2), and (e_3, v_3), so that, for each $i = 1, 2, 3$, either e_i is an obstacle edge and v_i is a vertex of P or e_i is an

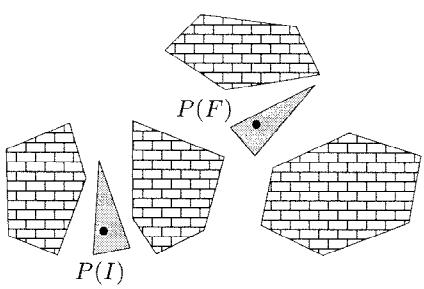

Fig. 1. Motion planning for a convex polygon inside a polygonal environment

edge of P and v_i is an obstacle vertex, and so that the vertex v_i touches the edge e_i at placement Z. Leven and Sharir showed that this quantity and thus the complexity of $\mathcal{C}$ are both $O(mn\lambda_6(mn))$. Here $\lambda_s(q)$ is the maximum length of (q, s)-Davenport–Schinzel sequences [36], which is nearly linear in q for any fixed s. They also showed that the complexity of $\mathcal{C}$ is $\Omega(m^2n^2)$ in the worst case. Thus the complexity of $\mathcal{C}$ is near-quadratic in mn. The goal then was to compute $\mathcal{C}$ in time that is also near-quadratic in mn. The first result in this direction was obtained by Kedem and Sharir [24], where an $O(mn\lambda_6(mn)\log mn)$-time algorithm was proposed. However, this algorithm turned out to have a technical difficulty. The algorithm constructs $\mathcal{C}$ in two stages. The first stage computes a *superset* of all the vertices of $\mathcal{C}$, where each such vertex is a free critical placement of P in Q, as defined above, and then aims to filter out the spurious vertices (nonfree placements). The filtering process is rather complicated, and some of the cases are not handled correctly [24].

Two subsequent papers aimed to fix Kedem and Sharir's algorithm. The first solution, given by Sharir and Toledo [37], processes Q into several range-searching data structures, and then it queries these structures with each placement of P produced by the algorithm of Kedem and Sharir [24] to discard nonfree placements. The overall running time of their algorithm is close to $O(m^3n^2)$. This is significantly more expensive for large values of m, which is what we assume here. The second solution, proposed by Kedem et al. [25], correctly computes the connected components of $\mathcal{C}$ that contain I and F, but does not always compute the entire free space. The time complexity of their algorithm is $O(mn\lambda_6(mn)\log mn)$. For other solutions to the problem, which are less efficient but also apply to the case when P is nonconvex, see [10]. These results leave open the problem of whether the entire free space can be computed in time that is near quadratic in mn.

The case in which scaling is allowed and we seek the largest placement of P inside Q has been studied in [14], [17], and [37]. Chazelle gave an $O(m^3n^3(m+n)\log(m+n))$-time algorithm to compute the largest placement of P inside Q. Using generalized Delaunay triangulations induced by P in Q, Chew and Kedem [17] gave an $O(m^4n^2\alpha(n)\log n)$-time algorithm for computing a largest free similar placement of P in Q; here $\alpha(n)$ is the inverse Ackermann's function. A variant of this algorithm also solves the motion-planning problem for P in Q, with the additional advantage of finding a "high-clearance" motion, where P aims to stay as far away from the boundary of Q as possible; see [17] for a more precise definition of high clearance. Sharir and Toledo [37] proposed another algorithm that combines parametric searching [29] with a construction of the entire configuration space for the fixed-size case, as in the preceding paragraph; the running time of their algorithm is close to $O(m^3n^2)$. If only translation and scaling are allowed, the largest homothetic placement of P inside Q can be computed in time $O(mn\log n)$, using the generalized Voronoi diagram of ∂Q induced by P [20], [28].

In [2] a much simpler situation is discussed where Q is also a convex polygon. The resulting problems are still challenging and have an interesting geometric structure. It is shown there that a largest scaled copy of P that can fit inside Q can be computed in $O(mn^2\log n)$ time. The maximum combinatorial complexity of the four-dimensional space $\mathcal{C}'$ of all *similar* placements of P inside Q is proven to be $\Theta(mn^2)$. It is shown that $\mathcal{C}'$ can be computed in $O(mn^2\log n)$ time. It is interesting that no better bounds are known for the space of all *congruent* placements.

New Results and Methods. We present a randomized divide-and-conquer algorithm for computing $\mathcal{C}$, whose expected running time is $O(mn\lambda_6(mn)\log mn\log n)$. The merge step of the algorithm is based on a line-sweep algorithm. Our technique is quite general and can be applied to other problems, as discussed in a remark at the end of Section 2.3. This is the first correct solution for computing all of $\mathcal{C}$ whose running time is near quadratic in mn. Our algorithm is rather simple, at least conceptually. It has the advantage that it is easy to parallelize, which is needed in our solution to the largest-placement problem, see Section 4. Even for the task of computing only a portion of $\mathcal{C}$, our algorithms are simpler than the ones in [25] and [37]. In addition, we can preprocess $\mathcal{C}$ in $O((mn)^{2+\varepsilon})$ time so that we can efficiently answer *reachability queries*: for any two placements of P, we can determine in $O(\log mn)$ time whether there is a collision-free motion from one to the other (i.e., whether they lie in the same connected component of $\mathcal{C}$). A variant of the algorithm can also produce a path connecting the two placements, in additional time proportional to the combinatorial complexity of the path. No claims of optimality of the resulting path are made.

Using an approach based on parametric searching, similar to that of [37], we can find the largest similar placement of P in Q, in randomized expected time $O(mn\lambda_6(mn)\log^3 mn\log^2 n)$, thus improving significantly over the previous bounds in [17] and [37]. Parametric searching requires an "oracle" procedure that has to determine, for a given size of P, whether the corresponding $\mathcal{C}$ is nonempty, which we can do using our algorithm for computing the entire $\mathcal{C}$. Notice that we can neither use the algorithm by Kedem et al. [25] here nor the one by Kedem and Sharir [24], since the former may miss some of the components of $\mathcal{C}$ and the latter may produce placements that are not free.

The paper is organized as follows. Section 2 describes the randomized algorithm for computing the free configuration space. Section 3 presents the data structures for answering reachability queries, and Section 4 describes the algorithm for computing a largest copy of P that can be placed inside Q.

2. Constructing the Free Configuration Space

Consider a convex m-gon P translating and rotating rigidly in a general polygonal environment Q bounded by n edges, without scaling. Recall that a placement of P can be parametrized by (x, y, θ), where $(x, y) \in \mathbb{R}^2$ is the position of a reference point of P and $\theta \in [-\pi, \pi]$ is the counterclockwise angle by which P is rotated from some fixed reference orientation. For the sake of convenience, we represent a placement of P by $(x, y, \tan(\theta/2))$, so that the set of all placements is $\mathbb{R}^3$. Note that, with this representation, special treatment is required for $\theta = \pm\pi$ because, for any $(x, y) \in \mathbb{R}^2$, $(x, y, -\pi)$ and $(x, y, +\pi)$ represent the same physical placement of P. In brief, even though at these points $\tan(\theta/2) \to \pm\infty$, we represent them explicitly. At the end of our construction we glue together the two cross sections $\theta = -\pi, \theta = +\pi$. This is necessary to preserve connectivity along paths that cross the surface $\theta = \pm\pi$.

We assume that P and Q are in *general position*. In our context, this means that no two obstacle vertices have the same x-coordinate and there is no placement of P

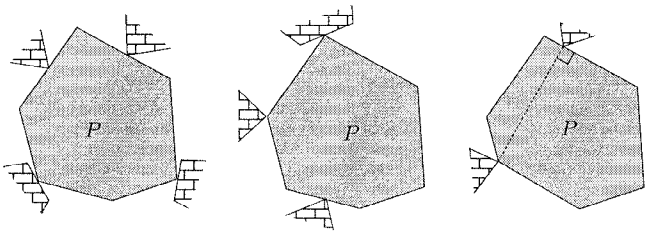

Fig. 2. Typical degeneracies.

at which four "independent" constraints imposed on P by its possible contacts with Q are simultaneously satisfied. Each constraint corresponds to a set of placements of P at which a vertex of P touches an edge of Q, an edge of P touches a vertex of Q, or the segment connecting two points of contact (between P and Q) is normal to the edge of P or Q involved in one of these contacts. See [27] for details. Fig. 2 illustrates several degeneracies (i.e., placements not in general position).

We triangulate Q^c, the complement of Q, using Steiner points if necessary, so that the degree of each vertex in the triangulation is bounded by a constant; such a triangulation can be constructed, e.g., by computing the vertical decomposition of Q^c and by triangulating each trapezoid of the vertical decomposition. From now on we assume, for technical reasons, that Q^c is the union of *n pairwise disjoint open triangular obstacles*, some of which may be unbounded. Note that the new n is larger than the original n by only a constant factor. Also observe that replacing general polygonal obstacles with disjoint open triangles adds zero-width "passages" to Q. However, it does not affect the free configuration space $\mathcal{C}$ in any significant way, as long as P has nonempty interior.

We use $P(Z)$ to denote P at a placement Z. We define a *contact* to be a triple (e, v, Δ), where e is an edge of the obstacle Δ and v is a vertex of P, or e is an edge of P and v is a vertex of Δ.[1] In the former case there is a unique contact triple corresponding to a physical contact of v and e, and in the latter case, since each vertex of Q is incident to $O(1)$ triangular obstacles, there are only $O(1)$ contact triples corresponding to such a physical contact. The total number of contacts is therefore $O(mn)$. If Δ is not important or is obvious from the context, we omit Δ. A placement Z of P *involves* a contact (e, v, Δ) if the vertex v lies on the edge e when P is placed at Z. Z is called *locally free* if $P(Z)$ does not intersect Δ.

We present a randomized algorithm for computing the boundary $\partial\mathcal{C}$ of the free configuration space $\mathcal{C}$. More precisely, we compute each connected component of $\partial\mathcal{C}$, decompose it into xy-monotone patches, and represent each patch as a planar map, using any standard representation (see, e.g., [32] and [38]). The expected running time of the algorithm is $O(mn\lambda_6(mn)\log mn \log n)$ and is thus close to the the worst-case complexity bound for $\mathcal{C}$. As usual for this type of algorithm, the expectation is over the random

[1] Unlike Leven and Sharir [27], we include Δ in the definition of a contact because a vertex may be shared by several triangular obstacles, and we prefer to regard (e, v, Δ_1) and (e, v, Δ_2), where v is a common vertex of Δ_1 and Δ_2, as two different contacts (especially while analyzing the running time of the algorithm), even though, geometrically, they correspond to the same contact.

choices made by the algorithm, for any fixed input, and not over any assumed distribution of the input data.

2.1. *Overall Approach*

Our algorithm is based on the following approach. For each (open) triangular obstacle Δ, let $K(\Delta)$ denote the set of *forbidden* placements of P at which it intersects Δ. These are open sets, and $\mathcal{C}$ is the complement of their union, so it suffices to compute the boundary of the union $K = \bigcup_\Delta K(\Delta)$. For each obstacle Δ_0, we compute the faces of ∂K that lie in $\partial K(\Delta_0)$, and then patch these faces together to construct ∂K. This leads to the following simple high-level description of our algorithm: Fix an obstacle Δ_0, and compute the intersections $A(\Delta) = K(\Delta) \cap \partial K(\Delta_0)$, for every obstacle $\Delta \neq \Delta_0$. Construct the two-dimensional union of the sets $A(\Delta)$, and form its complement within $\partial K(\Delta_0)$. This complement is exactly the portion of ∂K that is contained in $\partial K(\Delta_0)$. After applying this procedure to all obstacles Δ_0, we have computed all the two-dimensional faces of $\partial\mathcal{C}$ and the edges and vertices incident to them. Thus for each face f of $\partial\mathcal{C}$, we have the list of all faces adjacent to f. We can glue together these faces, by performing a depth-first search on the graph dual to $\partial\mathcal{C}$, to obtain an appropriate discrete representation of the entire boundary of K, and thus also of $\partial\mathcal{C}$. We omit the details concerning the gluing process, since they are straightforward and have been described earlier, see, e.g., [23] and [38]. Note that this approach does not identify which connected component of $\mathcal{C}$ is adjacent to each face of $\mathcal{C}$. We show in Section 3.2 that we can compute this information in an additional $O(mn\lambda_6(mn)\log mn)$ time.

We now describe in detail how to compute $\partial K(\Delta_0)$. Note that $\partial K(\Delta_0)$ consists of all (free or nonfree) placements of P at which its boundary makes a locally free contact with $\partial\Delta_0$. We partition $\partial K(\Delta_0)$ into $O(m)$ patches so that the same locally free contact (e, v, Δ_0) is made for all placements of P within each patch. The boundary of each patch corresponds to placements at which P makes simultaneously two (locally free) contacts with Δ_0; here we regard a vertex of P touching a vertex of Δ_0 or an edge of P overlapping an edge of Δ_0 as "double" contacts. If a patch is not xy-monotone, we further partition it into a constant number of xy-monotone patches. This allows us to use the (x, y)-coordinate system when manipulating objects contained in a patch. Such a partition is easy to obtain in $O(m)$ time. We refer to the resulting patches as *contact surfaces*. It is easily checked that a contact surface has constant description complexity, in the sense that each patch is a portion of an algebraic surface of bounded degree and its boundary consists of $O(1)$ algebraic arcs of bounded degree.

Repeating the process for every choice of Δ_0, we obtain a collection of $O(mn)$ two-dimensional contact surfaces. For each such surface $\pi \subset \partial K(\Delta_0)$, we compute the intersections $\Delta_\pi = \pi \cap K(\Delta)$, for all obstacles $\Delta \neq \Delta_0$, and construct $A_\pi = \pi \setminus (\bigcup_{\Delta\neq\Delta_0} \Delta_\pi)$, the complement of their union within π.[2] If π represents contacts of

[2] If two obstacles Δ_1, Δ_2 share a vertex v, then for an edge $e \in P$, we have two contacts (e, v, Δ_1) and (e, v, Δ_2). Let $\pi_1 \subseteq \partial K(\Delta_1)$ and $\pi_2 \subseteq \partial K(\Delta_2)$ denote the corresponding contact surfaces, and let $\overline{\pi}$ denote the set of all placements at which the edge e of P touches the vertex v. Then $\pi_1, \pi_2 \subseteq \overline{\pi}$. Although the two contact surfaces may not be identical, it is easily seen that $A_{\pi_1} = A_{\pi_2}$, so it suffices to compute only one of

an edge e and a vertex v, then A_π corresponds to placements at which v is in contact with e and P does not intersect the interior of any obstacle. (As noted earlier, this holds independently of the triangle Δ_0 containing e or v. Hence, for convenience, we denote the above contact surface simply as $\pi_{e,v}$, with the corresponding triangle Δ_0 being implicit in this notation.) Gluing these complements together will give us ∂K, as above. We refer to the sets Δ_π as *virtual π-obstacles*.

Let $\pi = \pi_{e,v}$ be a fixed contact surface. We can parametrize π by $(\rho, \tan(\theta/2))$, where ρ measures the displacement along e of its contact with v and θ is the orientation of P. For an obstacle Δ, constructing Δ_π is easy: Note that, for any fixed θ, the locus of placements contained in Δ_π with orientation θ is a line segment. (Indeed, the only motion available for P in this set is translation parallel to e; the set of such translations at which the two convex polygons P and Δ intersect is a line segment.) The combinatorial nature of an endpoint of this segment (i.e., the pair of features whose contact defines the endpoint) changes at only those orientations at which either (a) the line parallel to e through some vertex of P passes through some vertex of Δ, or (b) an edge of P becomes parallel to an edge of Δ. There are $O(m)$ such orientations and there are only $O(1)$ changes in the structure at each such orientation, so $\partial\Delta_\pi$ consists of $O(m)$ arcs. As shown in [34], each such arc is a section of an algebraic curve of degree at most 4. Δ_π can easily be computed in $O(m \log m)$ time by sorting and processing these orientations in increasing order. The total time needed to produce the sets Δ_π, over all Δ, is thus $O(n) \times O(m \log m) = O(mn \log m)$. The above arguments imply that each Δ_π is θ-monotone (in the coordinate frame representing π). However, Δ_π need not be connected. Indeed, it can have as many as $\Omega(m)$ components in the worst case; see Fig. 3.

2.2. *Computing A_π*

We fix a triangular obstacle Δ_0 and compute $A_\pi = \pi \setminus (\bigcup_{\Delta \neq \Delta_0} \Delta_\pi)$ using a randomized divide-and-conquer approach. We randomly divide the set of virtual π-obstacles into two equal subsets (so that every such partition occurs with equal probability), recursively compute the complements of their two unions in π, denoted by A_1, A_2, and compute $A_\pi = A_1 \cap A_2$ using a standard sweep-line procedure. Since the boundaries of obstacles are not disjoint, the edges of A_1 and A_2 may overlap, so extra (albeit standard) care needs to be taken to handle degeneracies while computing $A_1 \cap A_2$ by a sweep-line algorithm. We assume, as is standard, an appropriate model of computation, in which various basic operations on the arcs forming the boundaries of the virtual obstacles (such as intersecting a pair of such arcs) can be performed in $O(1)$ time. If an edge of A_1 crosses an edge of A_2, then their crossing point is a vertex of $A_1 \cap A_2$; and if an edge of A_1 overlaps an edge of A_2, then the endpoints of their overlap are vertices of A_1 or A_2, so the total time spent in the divide and merge steps is $O((|A_\pi| + |A_1| + |A_2|) \log mn)$, where $|A_\pi|$, $|A_1|$, and $|A_2|$ are the numbers of vertices of these respective sets. Let κ_π denote the total number of vertices in all the intermediate unions of all recursive subproblems produced by the

---

them. If two obstacles Δ_1, Δ_2 share an edge e and v is a vertex of P, Δ_1 and Δ_2 lie on opposite sides of e, so there are no locally free placements that realize contacts (e, v, Δ_1) and (e, v, Δ_2), so there is no need to process the corresponding contact surface.

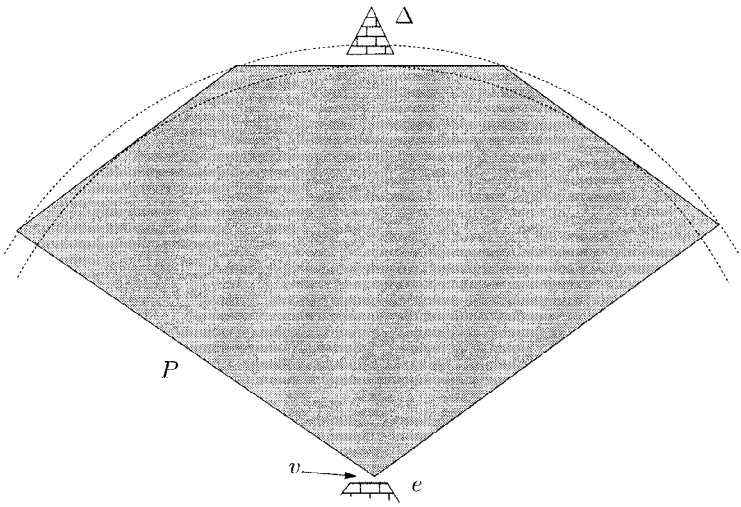

Fig. 3. Δ_π may consist of $\Omega(m)$ connected components. P is a sector of a regular polygon centered at v; Δ is placed at a distance from e which is between the radii of the inscribed and circumscribed circles of the polygon, and e is relatively short, so that P cannot be slid along e and then rotated so that a different edge is facing up, without overlapping Δ.

algorithm. (If a vertex appears in k intermediate unions, then we count it k times.) The total time to compute A_π, for a fixed π, including the time spent in computing the virtual π-obstacles, is $O((mn + \kappa_\pi)\log mn)$.

Applying this procedure to each of the $O(mn)$ contact surfaces independently and gluing the results together, we construct ∂K in time $O((m^2n^2 + \sum_\pi \kappa_\pi)\log mn)$, where the summation is taken over all contact surfaces. We prove in Section 2.3 that the expected value of $\sum_\pi \kappa_\pi$ is $O(mn\lambda_6(mn)\log n)$, which implies that the expected running time of the overall algorithm is $O(mn\lambda_6(mn)\log mn \log n)$. Hence, we can conclude:

Theorem 2.1. *Given a convex polygon P with m edges and a polygonal environment Q with a total of n edges, we can compute the boundary of the entire free configuration space $\mathcal{C}$ by a randomized algorithm in expected time $O(mn\lambda_6(mn)\log mn \log n)$.*

2.3. *Bounding the Expected Value of $\sum_\pi \kappa_\pi$*

In this section we prove that the expected value of $\sum_\pi \kappa_\pi$ is $O(mn\lambda_6(mn)\log n)$. For simplicity, assume that n, the total number of triangular obstacles, is of the form $2^h + 1$ for some integer h. Any vertex ζ that can appear on an intermediate union U produced by the algorithm, while computing A_π for some contact surface $\pi = \pi_{e,v}$, is either an endpoint of an edge of a virtual π-obstacle or an intersection of the boundaries of some pair of virtual π-obstacles. There are a total of $O(m^2n^2)$ vertices of individual virtual π-obstacles, and each of them may be counted $O(\log n)$ times in $\sum_\pi \kappa_\pi$ (once at each level of recursion). Therefore it suffices to bound the number of intersection points between

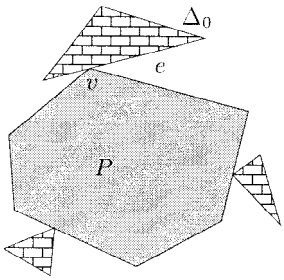

Fig. 4. A triple-contact vertex.

the boundaries of virtual obstacles. Let ζ be such an intersection point. Suppose π is a portion of $\partial K(\Delta_0)$ for some obstacle Δ_0. Then ζ represents a placement of P at which the following conditions hold:

(a) P makes three simultaneous contacts with the obstacle boundaries, one of which is the contact (e, v, Δ_0) defining π and no two contacts involve the same edge–vertex pair;
(b) P is disjoint from the union of all the obstacles Δ whose corresponding virtual π-obstacles participate in U and there is no other placement in a sufficiently small neighborhood of ζ that satisfies the same three contacts; and
(c) P is disjoint from Δ_0.

Conditions (b) and (c) imply that P is openly disjoint from the three obstacles involved in the three contacts that P makes.

A *triple-contact vertex* ζ is a quadruple (Z, C_1, C_2, C_3), where Z is a (not necessarily free) placement of P at which ∂P makes three simultaneous (vertex–edge or edge–vertex) contacts C_1, C_2, and C_3, each involving a distinct edge–vertex pair, and P is locally free in the sense that it does not intersect the obstacles corresponding to the three contacts and that no Z' in a sufficiently small neighborhood of Z satisfies the same property. See Fig. 4. Since the degree of each vertex of Q is bounded by a constant, each placement Z gives rise to $O(1)$ triple-contact vertices. If the actual contacts are not important, we will not distinguish between ζ and the corresponding placement Z of P. We say that a triple-contact vertex ζ has *level k* (with respect to the full collection of obstacles) if removal of some k other obstacles (not containing the at most three that participate in the triple contact) causes ζ to become a free placement, relative to the remaining obstacles, and no set of fewer than k obstacles has this property. Note that level-0 vertices are exactly the triple-contact vertices of $\mathcal{C}$.

If two contacts in a triple-contact vertex are formed by the same obstacle Δ_i, then an edge of P must overlap an edge of Δ_i at the corresponding placement, or a vertex of P must coincide with a vertex of Δ_i. It is easily checked that the total number of such placements, regardless of their level, is $O(m^2n^2)$ and that each of them is counted $O(\log n)$ times in $\sum_\pi \kappa_\pi$. In what follows we therefore consider only those triple-contact vertices at which each of the three contacts is made by a different obstacle.

Let F_k denote the number of level-k (triple-contact) vertices for the given P and Q. For a level-k vertex ζ, let p_k denote the expected number of recursive subproblems of any size that contain ζ in their output (we momentarily prove an upper bound on p_k that does indeed depend only on k and not on the choice of ζ). Then the expected value of $\sum_\pi \kappa_\pi$ is easily seen to be

$$\mathrm{E}\left[\sum_\pi \kappa_\pi\right] = \sum_{k=0}^{n-3} F_k p_k.$$

We first obtain a bound on p_k. Fix a triple-contact vertex ζ that appears on the boundary of free configuration space with respect to the three obstacles defining the triple contact. Suppose ζ is a vertex at level k, with respect to the full set of obstacles. Note that, throughout its execution, the algorithm encounters sets of virtual obstacles of cardinality 2^i, for $i = 0, \ldots, h$. Fix one such i. We bound the probability that ζ occurs during the execution of the algorithm, for any contact surface, while processing subproblems involving $r = 2^i$ obstacles. The previous discussion implies that ζ lies at the intersection of three contact surfaces. Fix one of these contact surfaces π. Then ζ appears in some fixed subproblem involving r obstacles in the construction carried out within π if and only if these r obstacles include the other two obstacles defining ζ and do not include any of the k obstacles that "cover" ζ. Since every set of r obstacles not containing the obstacle inducing π has the same probability of being the set of input obstacles to our fixed subproblem, the probability of ζ appearing in the output of the subproblem is $\binom{n-3-k}{r-2}/\binom{n-1}{r}$. (Recall that we ignore vertices that are determined by fewer than three obstacles; these vertices appear as vertices of some virtual π-obstacle, so we already have a bound on their number, as above.) Thus the expected contribution of a level-k vertex ζ to the output size of all subproblems during a run of the algorithm is

$$p_k \le \sum_{i=0}^{h} 3 \cdot 2^{h-i} \frac{\binom{n-3-k}{2^i-2}}{\binom{n-1}{2^i}}.$$

Here we used the fact that ζ may appear in the construction in each of the three different contact surfaces that define ζ, and that, in any fixed recursive construction within π, there are 2^{h-i} subproblems involving 2^i obstacles each. Hence,

$$\begin{aligned}\mathrm{E}\left[\sum_\pi \kappa_\pi\right] &\le \sum_{k=0}^{n-3}\left(F_k \sum_{i=0}^{h} 3 \cdot 2^{h-i} \frac{\binom{n-3-k}{2^i-2}}{\binom{n-1}{2^i}}\right) \\ &= \sum_{i=0}^{h} 3 \cdot 2^{h-i} \sum_{k=0}^{n-3} \frac{\binom{n-3-k}{2^i-2}}{\binom{n-1}{2^i}} F_k. \qquad (1)\end{aligned}$$

To bound this sum, we let $G(r)$ denote the expected number of level-0 vertices for P in an environment obtained by picking a random sample of r of the n triangular obstacles, where any subset of r obstacles is chosen with equal probability. We express $G(r)$ in terms of $F_1, F_2, \ldots, F_{n-3}$. What is the probability that a level-k vertex ζ defined by three contacts, as above, is counted in $G(r)$? In other words, what is the probability that it corresponds to a vertex of the free configuration space, in the environment defined

by r randomly selected obstacles? It is defined by three obstacles and "covered" by k other obstacles, so, arguing as before, the probability is $\binom{n-3-k}{r-3}/\binom{n}{r}$. Thus, the expected number of free triple-contact vertices arising in the r-sample is

$$G(r) = \sum_{k=0}^{n-3} \frac{\binom{n-3-k}{r-3}}{\binom{n}{r}} F_k.$$

Putting $r = 2^i + 1$, we obtain

$$G(2^i+1) = \sum_{k=0}^{n-3} \frac{\binom{n-3-k}{2^i-2}}{\binom{n}{2^i+1}} F_k = \frac{2^i+1}{n} \sum_{k=0}^{n-3} \frac{\binom{n-3-k}{2^i-2}}{\binom{n-1}{2^i}} F_k\,. \tag{2}$$

Substituting (2) into (1), we obtain

$$\begin{aligned} \mathrm{E}\left[\sum_{\pi} \kappa_\pi\right] &\le \sum_{i=0}^{h} 3 \cdot 2^{h-i} \frac{n}{2^i+1} G(2^i+1) \\ &= O(n^2) \cdot \sum_{i=0}^{h} \frac{G(2^i+1)}{2^i(2^i+1)}\,. \end{aligned}$$

Recall that each placement of P gives rise to $O(1)$ triple-contact vertices, so $G(2^i+1)$ is proportional to the combinatorial complexity of the free configuration space $\mathcal{C}$ for P moving amidst 2^i+1 obstacles, which, as noted above, is known to be $O(2^i m \lambda_6(2^i m))$ [27]. Therefore

$$\mathrm{E}\left[\sum_{\pi} \kappa_\pi\right] = O(n^2) \cdot \sum_{i=0}^{h} \frac{2^i m \lambda_6(2^i m)}{2^i(2^i+1)} \;=\; O(mn\lambda_6(mn)\log n),$$

as claimed.

Remark. As mentioned in the Introduction, our approach is quite general and can be extended to compute the union of a family of three-dimensional regions in many other cases. For example, let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a collection of k convex polyhedra in $\mathbb{R}^3$ with a total of n faces. We can use the same algorithm to compute the boundary of $\bigcup \mathcal{P}$ as follows. For each face π of a polyhedron $P_i \in \mathcal{P}$, we first compute the set $Q_\pi = \{\pi \cap P_j \mid 1 \le j \ne i \le k\}$, and then compute $\pi \setminus \bigcup Q_\pi$ using the randomized divide-and-conquer algorithm described above. Using essentially the same reasoning, the total expected running time of the algorithm is $O(k^3 \log n + nk \log k \log^2 n)$ time. This is a consequence of the fact, proven in [9], that the complexity of the union is $O(k^3 + nk \log k)$. If the polyhedra are obtained as Minkowski sums of some k disjoint convex polyhedra with a common convex polyhedron, the boundary of the union can be computed in randomized expected $O(nk \log k \log^2 n)$ time, as the complexity of the union is now only $O(nk \log k)$ [8], [9].

3. Motion-Planning Queries for P in Q

In this section we describe data structures that answer efficiently the following two types of queries involving P and the polygonal environment Q:

Free-placement query: Is a given placement Z of P free with respect to Q (i.e., does $Z \in \mathcal{C}$)? If Z is free, then, optionally, return also a placement Z' that lies on $\partial\mathcal{C}$ directly above Z in the $(+y)$-direction (i.e., return the first placement at which P touches an obstacle as we translate P from Z in the $(+y)$-direction).

Motion-planning query: Given two placements I and F of P, determine whether there is a collision-free path for P inside Q from I to F (i.e., whether I and F lie in the same connected component of $\mathcal{C}$). If the answer is "yes," then also return such a path for P from I to F. The first part of the query (to determine only whether F is reachable from I) is called a *reachability query*.

Both types of queries call for a point-location data structure in the three-dimensional space $\mathcal{C}$. Since the topology of $\mathcal{C}$ can be rather complicated, the known techniques, such as the point-location data structure by Preparata and Tamassia [33], do not seem to be directly applicable. We propose a different point-location data structure, tailored to our application. We first describe the data structure for free-placement queries and then extend it to answer reachability and motion-planning queries.

3.1. *Free-Placement Queries*

Let E be the set of obstacle edges (here we consider only the original edges of ∂Q and ignore the "inner passages" created by the triangulation of Q^c). Recall that $P(Z)$ denotes P at a placement Z. For a placement Z and for a subset $E' \subseteq E$, we define $Z' = \sigma(Z, E')$ to be the first placement at which P intersects a segment of E' as we translate P from Z in the $(+y)$-direction (Fig. 5); if $P(Z)$ itself intersects an edge in E', then $\sigma(Z, E') = Z$. For a given placement Z, we aim to determine whether Z is free, and if the answer is yes, we also want to return $\sigma(Z, E)$. To simplify the analysis, we assume that Q is bounded, so that $\sigma(Z, E)$ always exists. If Q is unbounded, we artificially clip it within a sufficiently large square, so that all placements of P at which P touches an obstacle vertex lie inside the square, and add the top edge of the square to E.

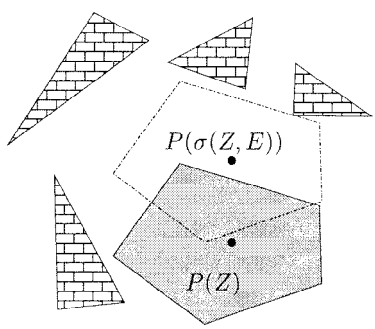

Fig. 5. A placement Z of P and $\sigma(Z, E)$.

Data Structure. We construct a segment tree T on the x-projections of the segments in E. Each node $v \in T$ is associated with an interval δ_v; let $W_v = \delta_v \times \mathbb{R}$ be the vertical strip erected on δ_v. Let $p(v)$ denote the parent of a node v. An interval I is stored at a node v if $\delta_v \subseteq I$ and $\delta_{p(v)} \not\subseteq I$. Let $E_v \subseteq E$ be the set of segments corresponding to the intervals stored at node v, and let $S_v \subseteq E$ denote the set of segments having at least one endpoint in the interior of W_v; we clip the segments of S_v and E_v within W_v. We sort the segments of E_v in the increasing order of their intercepts with any vertical line within W_v, which is a well-defined order, since the endpoints of the (clipped) segments in E_v lie on the boundary of W_v and their relative interiors are pairwise disjoint. Note that a segment can appear in sets S_v of at most $O(\log n)$ nodes v of T (the nodes lying on the two paths of T to the leaves whose strips contain the endpoints of the segment).

We construct two data structures on S_v. The first structure answers line-intersection queries, i.e., queries that determine whether a query line intersects any of the segments in S_v. We dualize each segment $e \in S_v$ to a double wedge[3] e^*, construct the arrangement of the resulting double wedges, preprocess the arrangement for planar point-location queries, and mark each face of the arrangement that is contained in at least one of the double wedges. The size of this data structure is $O(|S_v|^2)$ and it can be constructed in time $O(|S_v|^2 \log|S_v|)$; see, e.g., [1]. A line L intersects a segment of S_v if and only if the point L^* dual to L lies in a marked face. This can be determined in $O(\log|S_v|)$ time.

Next, we construct a two-level data structure on S_v. For each segment $e \in S_v$, we mark one of its endpoints; let A_v be the set of these points. We preprocess A_v into a halfplane range-searching data structure, using the algorithm by Chazelle et al. [16]. Their algorithm chooses a parameter r (greater than a constant specified by their algorithm) and constructs a family of *canonical subsets* of A_v so that there are $O((|S_v|/r^j)^{2+\delta})$ canonical subsets of size between r^j and r^{j+1}, for any integer $1 \le j \le \log_r n$; here $\delta > 0$ is an arbitrarily small constant. For a query line L, A_v can be partitioned into $O(\log_r n)$ canonical subsets so that all points within each canonical set lie on the same side of L. This partition can be computed in $O(\log n)$ time. For each canonical set, we construct the following second-level structure. Let $A_{v,i}$ be the ith canonical subset, let $S_{v,i} \subseteq S_v$ be the set of segments whose marked endpoints are in $A_{v,i}$, and let $V_{v,i}$ be the set of all endpoints of the segments in $S_{v,i}$. We use $z = \tan(\theta/2)$ to denote the parametric representation of the orientation of P. For each segment $e \in S_{v,i}$ and for every vertex $p \in P$, we define two partially defined bivariate functions $y = f_{e,p}(x, z)$ and $y = g_{e,p}(x, z)$ as follows: For a given pair (x_0, z_0) let y_0 be the y-value so that, at the placement $Z_0 = (x_0, y_0, z_0)$, the vertex p of $P(Z_0)$ lies in the relative interior of e and $P(Z_0)$ lies *below* (resp. *above*) the line containing e. If y_0 exists, then it is unique, and we put $y_0 = f_{e,p}(x_0, z_0)$ (resp. $y_0 = g_{e,p}(x_0, z_0)$); otherwise, $f_{e,p}(x_0, z_0)$ (resp. $g_{e,p}(x_0, z_0)$) is undefined; See Fig. 6(i) for an illustration to the definition of $f_{e,p}(\cdot, \cdot)$. Next, for each endpoint ξ of a segment e in $S_{v,i}$ and for every edge γ of P, we define two functions $f_{\xi,\gamma}(x, z)$ and $g_{\xi,\gamma}(x, z)$ as follows. For a given pair (x_0, z_0), let y_0 be the y-value so that at the placement $Z_0 = (x_0, y_0, z_0)$, the vertex ξ lies on γ and both

[3] In the duality that we use, the dual of a point $p(a, b)$ is the line $p^* : y = -ax + b$ and the dual of a line $L : y = \alpha x + \beta$ is the point $L^*(\alpha, \beta)$. The dual of a segment $e = pq$ is the double wedge formed by the lines p^* and q^* that does not contain the vertical line passing through the intersection point of p^* and q^* (which is the point dual to the line supporting e).

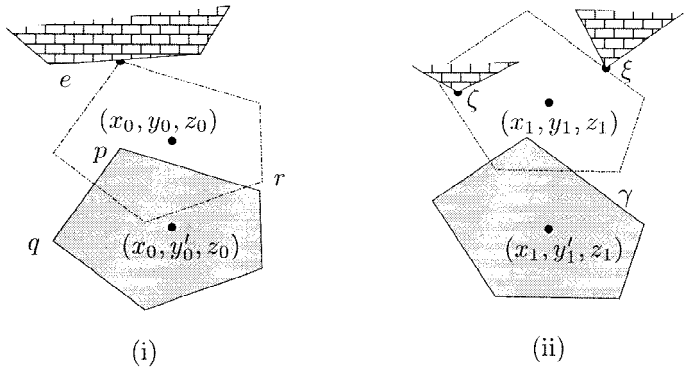

Fig. 6. (i) $y_0 = f_{e,p}(x_0, z_0)$; $f_{e,q}(x_0, z_0)$, $f_{e,r}(x_0, z_0)$ are not defined. (ii) $y_1 = f_{\xi,\gamma}(x_1, z_1)$; $f_{\zeta,\gamma}(x_1, z_1)$ is not defined.

obstacle edges incident to ξ lie *above* (resp. *below*) the line supporting γ. If y_0 exists, then it is unique and we set $y_0 = f_{\xi,\gamma}(x_0, z_0)$ (resp. $y_0 = g_{\xi,\gamma}(x_0, z_0)$); otherwise the respective functions are undefined. See Fig. 6(ii) for an illustration to the definition of $f_{\xi,\gamma}(\cdot, \cdot)$. We compute the lower envelope $F_{v,i}$ of

$$\{f_{e,p} \mid e \in S_{v,i} \text{ and } p \text{ a vertex of } P\} \cup \{f_{\xi,\gamma} \mid \xi \in V_{v,i} \text{ and } \gamma \text{ an edge of } P\}$$

and the upper envelope $G_{v,i}$ of

$$\{g_{e,p} \mid e \in S_{v,i} \text{ and } p \text{ a vertex of } P\} \cup \{g_{\xi,\gamma} \mid \xi \in V_{v,i} \text{ and } \gamma \text{ an edge of } P\}.$$

These envelopes can be computed and preprocessed in $O((m|S_{v,i}|)^{2+\delta})$ time, for any $\delta > 0$, for point location, so that, for any given pair (x_0, z_0), $F_{v,i}(x_0, z_0)$ and $G_{v,i}(x_0, z_0)$ can be computed in $O(\log mn)$ time [4]. We store these envelopes as the secondary structures of the ith canonical subset. Choosing $r = n^{\delta}$ and summing the complexity of these envelopes over all canonical subsets, the total size of the two-level data structure constructed on S_v is $O((m|S_v|)^{2+\delta})$. Summing over all nodes of the segment tree, the overall size of the data structure is $O((mn)^{2+\delta})$, for slightly larger but still arbitrarily small $\delta > 0$. The total time spent in constructing these structures is $O((mn)^{2+\delta})$.

Answering a Query. The query procedure determines whether Z_0 is a free placement. If the answer is "yes," then it also returns $\sigma(Z_0, E)$, the placement that lies on $\partial\mathcal{C}$ directly above Z_0 in the $(+y)$-direction.

Let $Z_0 = (x_0, y_0, z_0)$ be a query placement. We can determine in $O(\log m)$ time the leftmost and rightmost vertices, ℓ and r, of $P(Z_0)$. We can test in $O(\log n)$ time whether ℓ lies inside an obstacle. If so, we can conclude that Z_0 is not a free placement. We can thus assume that ℓ lies in Q. We use the following simple lemma to answer the query.

Lemma 3.1. *Let Z_0 be a placement so that $P(Z_0)$ does not lie completely inside Q^c. If $P(Z_0)$ intersects (resp. touches) an obstacle, then there exists a node v in the segment*

tree T so that at least one of ℓ and r lies in $W_{p(v)}$ and (*at least*) *one of the following two conditions is satisfied*:

(i) *$P(Z_0)$ intersects* (*resp. touches*) *a segment of E_v, or*
(ii) *ℓ and r do not lie in W_v and $P(Z_0)$ intersects* (*resp. touches*) *a segment of S_v.*

Proof. Suppose $P(Z_0)$ intersects an obstacle. Since $P(Z_0) \not\subseteq Q^c$, there must exist an obstacle edge e that intersects $\partial P(Z_0)$. Let ξ be an intersection point of e and $\partial P(Z_0)$, and let u be the leaf of T whose strip W_u contains ξ. Let w be the unique ancestor of u so that $e \in E_w$. If ℓ or r lies in $W_{p(w)}$, then condition (i) holds with $v = w$. Otherwise, let z be the lowest ancestor of w such that $W_{p(z)}$ contains one of ℓ or r. Since z is a proper ancestor of w, we have $e \in S_z$. Hence condition (ii) holds in this case with $v = z$. □

Let $V_1 = \{v \in T \mid \ell \in W_{p(v)} \text{ or } r \in W_{p(v)}\}$ and $V_2 = \{v \in V_1 \mid \ell, r \notin W_v\}$. The above lemma suggests that, to test whether $P(Z_0)$ is free, it suffices to search E_v for all $v \in V_1$ and S_v for all $v \in V_2$. Note that $|V_2| \leq |V_1| = O(\log n)$. For each node $v \in V_1$, we test whether $P(Z_0)$ intersects a segment of E_v. We first compute the left and right endpoints, ℓ^* and r^*, respectively, of the portion of the segment ℓr inside W_v. We determine in $O(\log n)$ time the segments e_ℓ, e_r of E_v lying immediately above ℓ and r, respectively. If $e_\ell \neq e_r$, then $\ell^* r^*$, and therefore $P(Z_0)$, intersects an obstacle edge and we stop (see Fig. 7 (i)). Otherwise, $\ell^* r^*$ does not intersect any segment of E_v. We put $e = e_\ell = e_r$ and determine in $O(\log m)$ time a vertex $\xi \in P(Z_0) \cap W_v$ on the top boundary of $P(Z_0) \cap W_v$ that touches the supporting line of $P(Z_0) \cap W_v$ parallel to e. If ξ lies above e, then $P(Z_0)$ intersects e and therefore Z_0 is not a free placement. Similarly we can determine in $O(\log m)$ time whether $P(Z_0)$ intersects the segment of E_v lying immediately below $\ell^* r^*$. If $P(Z_0)$ does not intersect these segments of E_v,

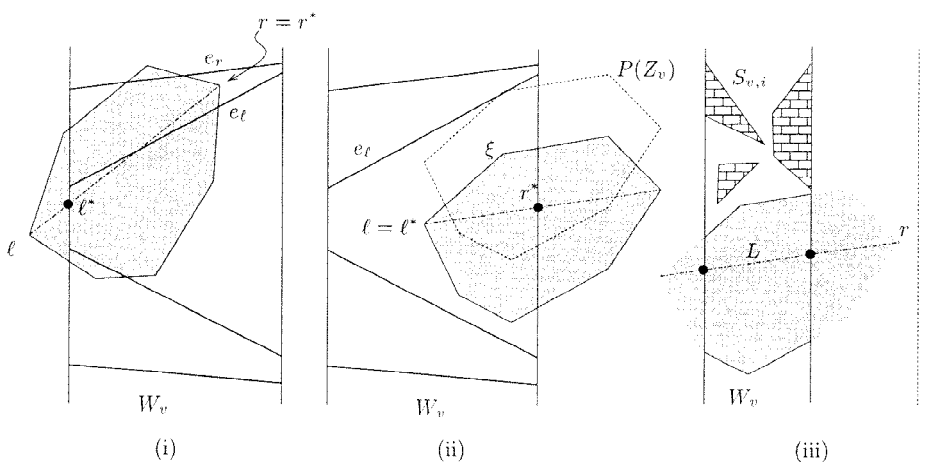

Fig. 7. (i) $e_\ell \neq e_r$ at a node $v \in V_1$. (ii) $P(Z_0)$ does not intersect any segment of E_v at a node $v \in V_1$. (iii) $P(Z_0)$ does not intersect any segment of $S_{v,i}$ at a node $v \in V_2$.

it avoids all segments of E_v (clipped within W_v). We can then determine in $O(1)$ time the placement $Z_v = (x_0, y_1, z_0)$ so that the vertex ξ of $P(Z_v)$ touches e (see Fig. 7(ii)); if there is no such placement, y_1 is set to $+\infty$. It is easily seen that $Z_v = \sigma(Z_0, E_v)$, provided $P(Z_0)$ does not intersect any (clipped) segment of E_v. Repeating this step for all nodes of V_1, we can determine in $O(\log mn \log n)$ time whether $P(Z_0)$ intersects any segment of $\bigcup_{v\in V_1} E_v$; if it does not, then we also obtain $Z_1 = \sigma(Z_0, \bigcup_{v\in V_1} E_v)$.

Next, for each node $v \in V_2$, we test whether $P(Z_0)$ intersects any segment of S_v. We first determine in $O(\log n)$ time whether the line L supporting the segment ℓr intersects any segment of S_v, using the line-intersection data structure. Since $L \cap W_v = \ell r \cap W_v \subset P(Z_0) \cap W_v$, we conclude that if L intersects S_v, then Z_0 is not free, so we stop immediately. Otherwise, we query the halfplane range-searching data structure with L. Let $S_{v,i}$ be one of the $O(1)$ canonical subsets of the query output (see Fig. 7(iii)).

Lemma 3.2. *Let Z_0 be a placement as above. If a canonical subset $S_{v,i}$ lies above (resp. below) L, then $P(Z_0)$ intersects some (clipped) segment of $S_{v,i}$ if and only $F_{v,i}(x_0, z_0) < y_0$ (resp. $G_{v,i}(x_0, z_0) > y_0$).*

Proof. Suppose $S_{v,i}$ lies above L. First assume that $P(Z_0)$ does not intersect any segment of $S_{v,i}$. Let e be a segment of $S_{v,i}$ and p a vertex of P so that their x-projections intersect at placement Z_0 of P. If $P(Z_0)$ does not intersect $S_{v,i}$, then e lies above p at placement Z_0, i.e., $f_{e,p}(x_0, z_0) \geq y_0$. Similarly, if ξ is an endpoint of $S_{v,i}$ and γ an edge of P so that their x-projections intersect at placement Z_0, then $f_{\xi,\gamma}(x_0, z_0) \geq y_0$. Hence, $F_{v,i}(x_0, z_0) \geq y_0$.

Next, assume that $P(Z_0)$ intersects a segment e of $S_{v,i}$. Then either one of the endpoints ξ of e lies inside P or a vertex p of P lies above e. In the former case, $f_{\xi,\gamma}(x_0, z_0) < y_0$, where γ is the edge of P lying vertically above ξ; while in the latter case, $f_{e,p}(x_0, z_0) < y_0$. This completes the proof of the lemma. □

In view of the above lemma, we can determine in $O(\log mn)$ time whether $P(Z_0)$ intersects any segment of $S_{v,i}$. If $P(Z_0)$ does not intersect any segment of $S_{v,i}$, then $(x_0, F_{v,i}(x_0, z_0), z_0) = \sigma(Z_0, S_{v,i})$. Repeating this procedure for all canonical sets of the query output and for all nodes $v \in V_2$, we can determine in $O(\log mn \log n)$ time whether $P(Z_0)$ intersects any segment of $\bigcup_{v\in V_2} S_v$. If it does not, we also obtain $Z_2 = \sigma(Z_0, \bigcup_{v\in V_2} S_v)$. Now $\sigma(Z_0, E)$ is the lowest of Z_1 and Z_2.

The query time can be improved to $O(\log mn)$ by constructing the segment tree with a larger fan-out, e.g., as described in [16], without increasing the asymptotic size and preprocessing time. Omitting the technical details of this improvement, we summarize the analysis in the following theorem:

Theorem 3.3. *Given a parameter $\varepsilon > 0$, a convex polygon P with m edges, and a polygonal environment Q with a total of n edges, we can preprocess P and Q in time $O((mn)^{2+\varepsilon})$ into a data structure of size $O((mn)^{2+\varepsilon})$, so that we can determine in $O(\log mn)$ time whether a given placement Z_0 is free. If Z_0 is free, we can also compute $\sigma(Z_0, E)$ within the same time bound, where E is the set of edges in Q.*

3.2. *Reachability and Motion-Planning Queries*

Returning to the original motion-planning query problem, we show that the data structure given above and the algorithm described in Section 2 can be used to answer reachability queries efficiently. The idea is to "retract" $\mathcal{C}$ onto a one-dimensional network connecting the vertices of $\mathcal{C}$ and reduce the motion-planning problem to path planning in this network. This retraction approach has been used extensively in the past for motion planning [11]–[13], [31].

We preprocess P and Q for free-placement queries, using Theorem 3.3. Next, we compute all the connected components of $\partial\mathcal{C}$, using the algorithm described in Section 2. The algorithm computes A_π for each contact surface $\pi = \pi_{e,v}$, and then glues them together. Actually, it computes a refinement $\partial\mathcal{C}^*$ of $\partial\mathcal{C}$ so that each two-dimensional face of ∂C^* is x-monotone, which implies that there is a path along the edges of $\partial\mathcal{C}^*$ between any pair of vertices of the same connected component of $\partial\mathcal{C}^*$. We preprocess (the xy-projection of) each A_π for efficient planar point-location queries.

A natural choice for constructing the one-dimensional network is the 1-skeleton of $\partial\mathcal{C}^*$, but it is not sufficiently connected to capture the connectivity of $\mathcal{C}$, because the boundary of a connected component $\mathcal{C}_i$ of $\mathcal{C}$ need not be connected. Let A_i be a connected component of $\partial\mathcal{C}_i$, and let $\zeta_i = (x_i, y_i, z_i)$ be a point on A_i with the maximum y-coordinate. We call A_i an *inner* component of $\partial\mathcal{C}_i$ if $\zeta_i^+ = (x_i, y_i + \varepsilon, z_i)$, for sufficiently small $\varepsilon > 0$, lies in $\mathcal{C}_i$. We refer to ζ_i as the *apex* of A_i; see Fig. 8. If A_i is an inner component, then we may assume that a vertex p of P touches a vertex of Q at ζ_i, so ζ_i is either a vertex of A_i or a point of locally maximum y-coordinate on an edge of A_i. Furthermore, at $P(\zeta_i)$ the reference point o lies vertically above the contact vertex p, i.e., the directed line segment po is parallel to the y-axis and oriented upward. Hence, there are only $O(mn)$ apex placements.

Define $\zeta_i' = \sigma(\zeta_i, E)$, where E is the set of all obstacle edges, and let A_j be the connected component of $\partial\mathcal{C}$ containing ζ_i'. Obviously A_j also belongs to $\partial\mathcal{C}_i$. We can compute ζ_i' in $O(mn)$ time using a naive procedure. If ζ_i is not a vertex of A_i, we add ζ_i as a vertex of A_i and split at ζ_i the edge of A_i containing ζ_i. Similarly, we add ζ_i' as a vertex of A_j and split at ζ_i' the edge, or the face, containing ζ_i'. To split a face f, we pass through

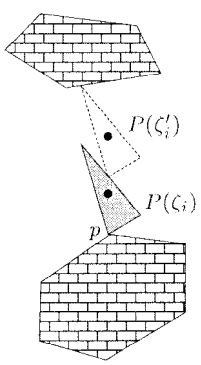

Fig. 8. An apex placement.

ζ_i' an arc γ that lies on f and is parallel to the xz-plane, extend γ in both directions until it hits ∂f, and split the edges of f hit by γ at the hitting points. We also add the edge (ζ_i, ζ_i') to the resulting network. We repeat this procedure for all inner components of $\partial\mathcal{C}$. The total time spent in this step is $O(m^2n^2)$. This step connects together all the boundary components of each connected component of $\mathcal{C}$. Let $\partial\mathcal{C}^{**}$ denote the resulting structure, and let G denote the 1-skeleton of $\partial\mathcal{C}^{**}$. The following property of G is obvious.

Lemma 3.4. *If Z, Z' are two vertices of the same connected component of $\mathcal{C}$, then Z and Z' belong to the same connected component of G.*

We perform a depth-first search on G and identify and label all vertices that lie in the same connected component of G, so that we can determine in $O(1)$ time whether two given vertices of G belong to the same connected component. Finally, we construct a spanning forest T of G, so that for any two vertices ζ, ζ' lying in the same connected component of G, we can return the path from ζ to ζ' in T in time proportional to its length.

We now answer a reachability or a motion-planning query as follows. Let $I = (x_I, y_I, z_I)$ and $F = (x_F, y_F, z_F)$ be two given placements. Using the free-placement data structure, we first determine whether both I and F are free. If so, we also compute $I' = \sigma(I, E)$ and $F' = \sigma(F, E)$, and the contact surfaces π_I and π_F containing I' and F', respectively. By locating I' in A_{π_I}, we can determine in $O(\log mn)$ time the edge e_I that lies immediately above I' in the $(+x)$-direction. Let I'' be the point on the edge e_I whose y-coordinate is $y_{I'}$, let γ_I be the arc from I' to I'' lying in A_{π_I} and parallel to the xz-plane, and let v_I be an endpoint of e_I. Similarly, we compute e_F, F'', γ_F, and v_F for the final placement F. Using Lemma 3.4, we can determine in $O(1)$ time whether v_I and v_F belong to the same connected component of $\partial\mathcal{C}$.

If v_I and v_F belong to the same component, we can also compute a path from I to F. Let Π_I be the path composed of the y-vertical segment II', the arc γ_I, and the portion of e_I from I'' to v_I. Define Π_F in an analogous manner. Finally, let Π be the path in G from v_I to v_F. Then the path obtained by concatenating Π_I, Π, and Π_F is a path in $\mathcal{C}$ from I to F. Hence, we obtain the following theorem.

Theorem 3.5. *Given a parameter $\varepsilon > 0$, a convex polygon P with m edges, and a polygonal environment Q with a total of n edges, we can preprocess, in additional $O((mn)^{2+\varepsilon})$ time, the (already computed) space $\mathcal{C}$ of all free congruent placements of P inside Q into a data structure of size $O((mn)^{2+\varepsilon})$ so that, for any two query free placements I and F of P, we can determine, in $O(\log mn)$ time, whether there exists a collision-free motion of P from I to F. If one exists, we can return such a path in time proportional to its complexity, which is at most $O(mn\lambda_6(mn))$.*

4. Finding the Largest Placement of *P* inside *Q*

As mentioned in the Introduction, we use the parametric-searching technique of Megiddo [29] (see also [5] and [7]) to compute a largest free similar placement of P inside Q. The parametric-searching paradigm requires an "oracle" procedure to determine, for a

given scaling factor $s > 0$ of P, whether $\mathcal{C}_s$, the free configuration space corresponding to sP moving within Q, is nonempty. Using Theorem 2.1, we can obtain an oracle that performs this task in expected time $O(mn\lambda_6(mn)\log mn \log n)$. An efficient implementation of the parametric search, however, also requires a *parallel* implementation of the oracle, in Valiant's comparison model [39]. Fortunately, the algorithm provided by Theorem 2.1 is easy to parallelize, because all recursive subproblems at the same depth can be performed in parallel. In fact, the only part of this algorithm that does not parallelize in a straightforward manner is the sweep-line procedure used in the merge step, because the standard implementation of line-sweeping is inherently sequential. We therefore perform the merge step in the parallel version using a different approach, based on segment trees, such as the one used in Section 5 of [7]. As argued in [7], the merge step requires $O(\log mn)$ time using $O(mn\lambda_6(mn)\log mn)$ processors, under Valiant's model of computation.

Omitting all further details, we conclude that one can compute $\mathcal{C}_s$ in $O(\log mn \log n)$ parallel steps, using $O(mn\lambda_6(mn)\log mn)$ expected number of processors, in Valiant's comparison model. Megiddo [29] showed that if the sequential algorithm for the oracle runs in time T_s and the parallel algorithm runs in time T_p using Π processors, then the parametric searching takes $O(T_p\Pi + T_sT_p \log \Pi)$ time, provided that all the control-flow decisions made by the parallel version can be expressed as sign tests of constant-degree polynomials in the parameter whose critical value is being sought (the scaling factor s, in our case), or are independent of this parameter. Since this is the case for our algorithm, we obtain the following result.

Theorem 4.1. *Given a convex polygon P with m edges and a polygonal environment Q with a total of n edges, we can compute a largest free placement of P inside Q in randomized expected time $O(mn\lambda_6(mn)\log^3 mn \log^2 n)$.*

5. Concluding Remarks

In this paper we studied the motion-planning problem for a convex m-gon P inside a polygonal environment Q with a total of n vertices. We presented an efficient algorithm for computing the entire free configuration space, whose expected time complexity is $O(mn\lambda_6(mn)\log mn \log n)$, which is near optimal in the worst case. We applied the algorithm to solve the following two problems:

(a) answering free-placement and motion-planning queries for P inside Q,
(b) finding a largest free placement of P inside Q.

We conclude with two open problems:

1. What is the combinatorial complexity of the four-dimensional configuration space of all free placements of P in Q, when scaling is also allowed? Is it also near-quadratic in mn? See the Introduction for the analogous result when Q is convex.
2. Agarwal and Sharir [6] gave a randomized algorithm, with $O(n^{3/2+\varepsilon})$ expected running time, to find a placement of a longest segment that can be placed inside a simple n-gon. Can one also obtain subquadratic algorithms for finding a largest

placement of a segment inside an arbitrary polygonal environment, or for finding the largest copy of a given triangle that can be placed inside a convex polygon?

References

1. P. K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *SIAM J. Comput*. 21 (1992), 540–570.
2. P. K. Agarwal, N. Amenta, and M. Sharir, Largest placement of one convex polygon inside another, *Discrete Comput. Geom*. 19 (1998), 95–104.
3. P. K. Agarwal, B. Aronov, and M. Sharir, Computing envelopes in four dimensions with applications, *SIAM J. Comput*. 26 (1997), 1714–1732.
4. P. K. Agarwal, O. Schwarzkopf, and M. Sharir, Overlay of lower envelopes in three dimensions and its applications, *Discrete Comput. Geom*. 15 (1996), 1–13.
5. P. K. Agarwal and M. Sharir, Algorithmic techniques for geometric optimization, in: *Computer Science Today*: *Recent Trends and Developments*, (J. van Leeuwen, ed.), Lecture Notes in Computer Science, vol. 1000, Springer-Verlag, Berlin, 1995, pp. 234–253.
6. P. K. Agarwal and M. Sharir, Efficient randomized algorithms for some geometric optimization problems, *Discrete Comput. Geom*. 16 (1996), 317–337.
7. P. K. Agarwal, M. Sharir, and S. Toledo, Applications of parametric searching in geometric optimization, *J. Algorithms* 17 (1994), 292–318.
8. B. Aronov and M. Sharir, On translational motion planning in 3-space, *SIAM J. Comput*. 26 (1997), 1785–1806.
9. B. Aronov, M. Sharir, and B. Tagansky, The union of convex polyhedra in three dimensions, *SIAM J. Comput*. 26 (1997), 1670–1688.
10. F. Avnaim, J.D. Boissonnat, and B. Faverjon, A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles, *Proc. Workshop on Geometry and Robotics*, Lecture Notes in Computer Science, vol. 391, Springer-Verlag, Berlin, 1989, pp. 67–86.
11. S. Basu, R. Pollack, and M.-F. Roy, Computing roadmaps of semi-algebraic sets, *Proc. 28th Ann. ACM Symp. Theory of Computing*, 1996, pp. 168–173.
12. J. Canny, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1987.
13. J. Canny, Computing roadmaps in general semialgebraic sets, *Comput. J*. 36 (1993), 409–418.
14. B. Chazelle, The polygon containment problem, in: *Advances in Computing Research*, vol. 1 (F. P. Preparata, ed.), JAI Press, London, 1983, pp. 1–33.
15. B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink, Computing a face in an arrangement of line segments and related problems, *SIAM J. Comput*. 22 (1993), 1286–1302.
16. B. Chazelle, M. Sharir, and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, *Algorithmica* 8 (1992), 407–429.
17. L.P. Chew and K. Kedem, A convex polygon among polygonal obstacles: placement and high-clearance motion, *Comput. Geom. Theory Appl*. 3(2) (1993), 59–89.
18. K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom*. 4 (1989), 387–421.
19. M. de Berg, K. Dobrindt, and O. Schwarzkopf, On lazy randomized incremental construction, *Discrete Comput. Geom*. 14 (1995), 261–286.
20. S. Fortune, A fast algorithm for polygon containment by translation, *Proc. 12th Internat. Colloq. Automata, Languages and Programming*, 1985, pp. 189–198.
21. L. Guibas, D. Knuth, and M. Sharir, Randomized incremental construction of Voronoi and Delaunay diagrams, *Algorithmica* 7 (1992), 381–413.
22. D. Halperin, L. E. Kavraki, and J.-C. Latombe, Robotics, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, eds.), CRC Press, Boca Raton, FL, 1997, pp. 755–778.
23. D. Halperin and M. Overmars, Spheres, molecules, and hidden surface removal, *Proc. 10th Ann. ACM Symp. Computataion Geometry*, 1994, pp. 113–122.
24. K. Kedem and M. Sharir, An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space, *Discrete Comput. Geom*. 5 (1990), 43–75.

25. K. Kedem, M. Sharir, and S. Toledo, On critical orientations in the Kedem–Sharir motion planning algorithm for a convex polygon in the plane, *Discrete Comput. Geom.* 17 (1997), 227–239.
26. D. Leven and M. Sharir, An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers, *J. Algorithms* 8 (1987), 192–215.
27. D. Leven and M. Sharir, On the number of critical free contacts of a convex polygonal object moving in two-dimensional polygonal space, *Discrete Comput. Geom.* 2 (1987), 255–270.
28. D. Leven and M. Sharir, Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams, *Discrete Comput. Geom.* 2 (1987), 9–31.
29. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mach.* 30 (1983), 852–865.
30. N. Miller and M. Sharir, Efficient randomized algorithms for constructing the union of fat triangles and of pseudo-disks, Manuscript, 1991.
31. C. Ó'Dúnlaing, M. Sharir, and C. Yap, Generalized Voronoi diagrams for a ladder: II. Efficient construction of the diagram, *Algorithmica* 2 (1987), 27–59.
32. F. P. Preparata and M. I. Shamos, *Computational Geometry*: *An Introduction*, Springer-Verlag, New York, 1985.
33. F.P. Preparata and R. Tamassia, Efficient point location in a convex spatial cell-complex, *SIAM J. Comput.* 21 (1992), 267–280.
34. J.T. Schwartz and M. Sharir, On the piano movers' problem: I. The case of a rigid polygonal body moving amidst polygonal barriers, *Comm. Pure Appl. Math.* 36 (1983), 345–398.
35. M. Sharir, Algorithmic motion planning, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, eds.), CRC Press, Boca Raton, FL, 1997, pp. 733–754.
36. M. Sharir and P. K. Agarwal, *Davenport–Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.
37. M. Sharir and S. Toledo, Extremal polygon containment problems, *Comput. Geom. Theory Appl.* 4 (1994), 99–118.
38. S. Sifrony and M. Sharir, A new efficient motion planning algorithm for a rod in two-dimensional polygonal space, *Algorithmica* 2 (1987), 367–402.
39. L. Valiant, Parallelism in comparison problems, *SIAM J. Comput.* 4(3) (1975), 348–355.

Received September 9, 1997, *and in revised form September* 24, 1998.