A Near-Quadratic Algorithm for Planning the Motion of a Polygon in a Polygonal Environment*

Dan Halperin† Micha Sharir‡

December 19, 1995

Abstract

We consider the problem of planning the motion of an arbitrary k-sided polygonal robot B, free to translate and rotate in a polygonal environment V bounded by n edges. We present an algorithm that constructs a single component of the free configuration space of B in time $O((kn)^{2+\varepsilon})$, for any $\varepsilon > 0$. This algorithm, combined with some standard techniques in motion planning, yields a solution to the underlying motion planning problem, within the same running time.

1 Introduction

Let B be an arbitrary polygonal object with k sides, and let V be an open planar polygonal region bounded by n edges. The *configuration space* $\mathcal{C}$ of B is a 3-dimensional parametric space, each point of which represents a possible placement of B by the parameterization (x, y, θ), where (x, y) are the coordinates of some fixed

*Work on this paper by Dan Halperin has been supported by a Rothschild Postdoctoral Fellowship, by a grant from the Stanford Integrated Manufacturing Association (SIMA), by NSF/ARPA Grant IRI-9306544, and by NSF Grant CCR-9215219.. Work on this paper by Micha Sharir has been supported by NSF Grants CCR-91-22103 and CCR-93-11127, by a Max-Planck Research Award, and by grants from the U.S.-Israeli Binational Science Foundation, the Israel Science Fund administered by the Israeli Academy of Sciences, and the G.I.F., the German-Israeli Foundation for Scientific Research and Development. A preliminary and extended version of the paper has appeared as: D. Halperin and M. Sharir, Near-quadratic bounds for the motion planning problem for a polygon in a polygonal environment, *Proc. 34th IEEE Symp. on Foundations of Computer Science* (1993), 382–391.

†Computer Science Robotics Laboratory, Stanford University. Part of the work on the paper was carried out while this author was at Tel-Aviv University.

‡School of Mathematical Sciences, Tel Aviv University, and Courant Institute of Mathematical Sciences, New York University

reference point on B, and θ is the orientation of B (the angle between some fixed ray attached to B and the positive x-axis). We call a placement of B a *free placement* if at this placement B does not intersect the complement V^c of V. The *free configuration space* of B, denoted FP, is the set of all free placements of B, and is clearly an open subset of $\mathcal{C}$.

The boundary of FP consists of so-called *semi-free placements*, where B makes one or more contacts with V^c but the interior of B remains disjoint from V^c. We can describe FP by defining in $\mathcal{C}$ a collection Σ of *contact surfaces*, each being either the locus of all placements of B at which some specific corner of B touches some specific edge of V, or the locus of placements at which some side of B touches some vertex of V. Clearly, each contact surface is a 2-dimensional manifold with boundary (a "surface patch"), and, if we replace θ by $\tan\frac{\theta}{2}$, the contact surfaces, as well as their bounding curves, are all algebraic of small (constant) maximum degree. The number of contact surfaces is clearly $O(kn)$.

If B is placed at a free placement Z and moves continuously from Z, then it remains free as long as the corresponding path traced in $\mathcal{C}$ does not hit any contact surface. Moreover, once this path crosses a contact surface, B becomes non-free. (For this we need to assume, as is customary in other treatments of this problem, *general position* of B and V; see [11] for a more precise definition of this notion.) It follows that the connected component of FP that contains Z is the cell that contains Z in the arrangement $\mathcal{A}(\Sigma)$ of the contact surfaces. (The entire FP is the union of a collection of certain cells in this arrangement.)

The first problem that arises is to obtain a sharp upper bound on the combinatorial complexity of a single connected component of FP, that is, of a single cell of $\mathcal{A}(\Sigma)$. The combinatorial complexity of such a cell is defined as the number of vertices, edges, and faces of $\mathcal{A}(\Sigma)$ that appear on the boundary of the cell. The problem has been studied in [11] (see also [3, 9, 10, 15]) in the case where B is convex. It was shown there that the complexity of the *entire* free configuration space FP is $O(kn\lambda_6(kn))$, where $\lambda_q(m)$ is the maximum length of *Davenport-Schinzel sequences* of order q composed of m symbols, and is nearly linear in m for any fixed q (see [14] for more details). In other words, the complexity of FP is only nearly quadratic in kn, as opposed to a naive bound $O((kn)^3)$, which is a (worst-case tight) bound on the overall number of vertices in any 3-D arrangement of $O(kn)$ algebraic surface patches of constant maximum degree.

Unfortunately, if B is not convex, the entire free configuration space of B can have $\Theta((kn)^3)$ vertices in the worst case, as is illustrated in Figure 1. Hence, to obtain a sub-cubic bound, it makes sense to focus on just a single cell of the arrangement, as we have indeed indicated above. After the original submission of this paper, the authors have shown in [8] that the complexity of a single cell in any arrangement of N algebraic surface patches in $\mathbb{R}^3$ of constant maximum degree, bounded by algebraic arcs which also have constant maximum degree, is $O(N^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε and on the maximum degree and shape of the given surfaces and of their boundaries. Prior to this, slightly better bounds have

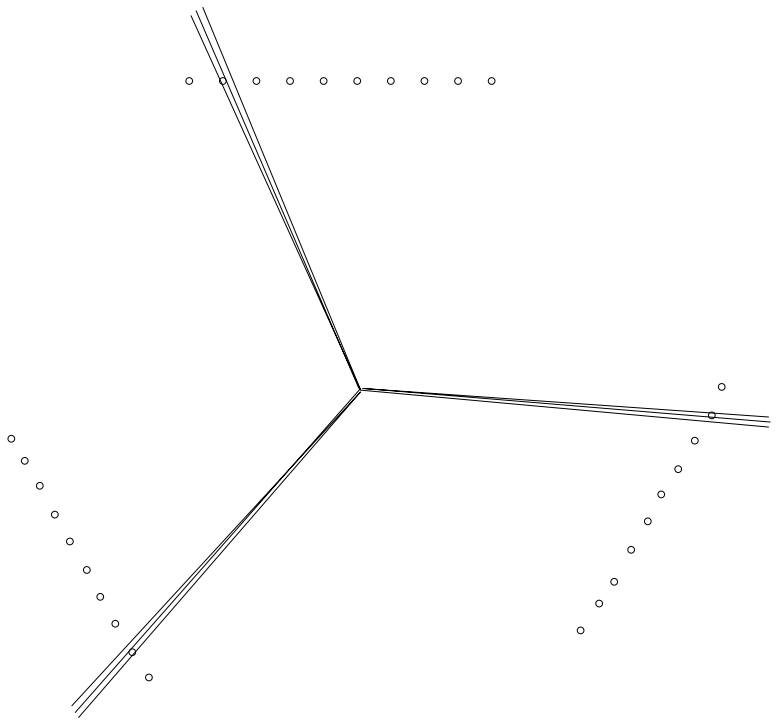


Figure 1: An example where the entire free configuration space of a non-convex polygon has cubic complexity

been obtained for certain special cases, including the case of spheres, where a (worst-case tight) quadratic bound is known [13], the case of triangles, where an $O(N^2 \log N)$ bound is known [2], and several special cases that arise in motion planning for various robot systems B with three degrees of freedom, including several restricted cases of the polygon motion planning problem that we consider here, where the shape of B and/or the shape of V is further restricted; these latter bounds are also all close to quadratic, and are reported in [7]. See also two recent surveys [5, 6] for more details concerning motion planning problems and arrangements of surfaces.

In this paper we exploit the new bounds derived in [8], introduce a special cell decomposition scheme for the cell arising in our motion planning problem, and obtain an efficient algorithm for constructing such a cell. The algorithm runs in time $O((kn)^{2+\varepsilon})$, for any $\varepsilon > 0$, where k and n are as above. The new cell decomposition technique that we develop here for the algorithm may be useful for other applications as well.

We also mention that, in the preliminary version of this paper, which appeared before the bounds of [8] were obtained, we showed that the complexity of a single cell in the arrangement that arises in our motion planning problem is $k^3 n^2 2^{O(\log^{2/3} n)}$. When k is constant, this is slightly better than the general bound of [8]. In fact, the analysis of [8] adapted and extended the technique that we used in the earlier version

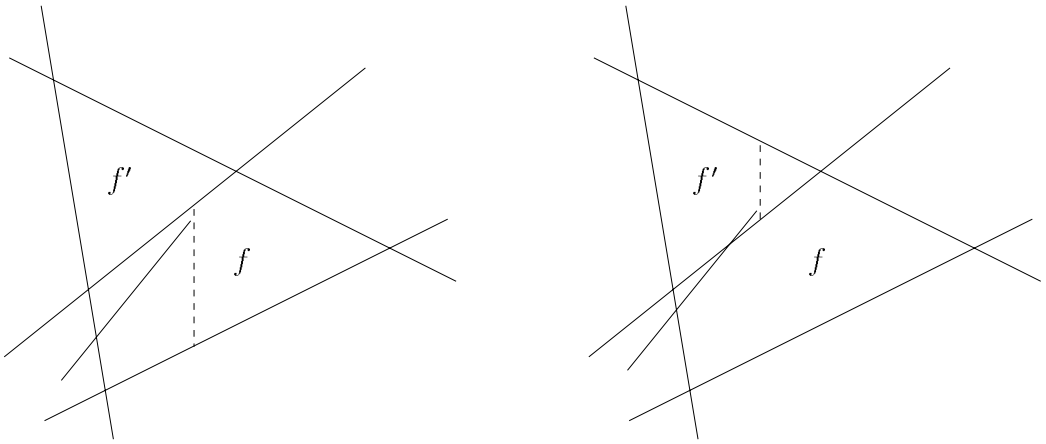
of this paper.

2 Efficient Construction of a Single Cell

In this section we obtain an efficient randomized algorithm (which can also be made deterministic) for constructing a single cell of the free configuration space of a moving k-sided polygon B. The general approach is similar to that of [1, 2]. That is, let Σ denote the collection of contact surfaces induced by the problem, and let Z be a given free placement of B; our goal is to compute the cell containing Z in $\mathcal{A}(\Sigma)$. We choose a random sample $\mathcal{R}$ of r surfaces of Σ, where r is some sufficiently large integer constant. We construct (e.g. by brute force) the cell C_0 containing Z in $\mathcal{A}(\mathcal{R})$, and decompose C_0, in a manner to be described shortly, into subcells, each having 'constant description complexity' (meaning that each cell is defined by a constant number of polynomial equalities and inequalities of constant maximum degree). The standard theory of ϵ-nets and finite VC-dimension implies that, with high probability, none of the subcells in the decomposition is crossed by more than $O(\frac{n}{r}\log r)$ surfaces of Σ (see, e.g., Appendix 7.2 of [14]). For each subcell ξ we find the subcollection Σ_ξ of surfaces that cross ξ, and compute recursively the cell C_ξ containing Z in the arrangement of these surfaces. We then form the desired cell C containing Z in $\mathcal{A}(\Sigma)$ by 'gluing' together pieces of these cells. Specifically, we start with the subcell ξ containing Z, and take the connected component K of $\xi \cap C_\xi$ that contains Z (note that $\xi \cap C_\xi$ need not be connected). If that component is disjoint from $\partial\xi$ then this is the entire desired cell C. Otherwise, let f be a connected face of $C_\xi \cap \partial\xi$. We find the other subcell(s) ξ' whose boundary contains or overlaps f (since our decomposition will not necessarily be a cell complex, there might be several such subcells ξ'). We find the connected component of $\xi' \cap C_{\xi'}$ whose boundary contains (or overlaps) f, and glue that component to K along f. We continue this gluing procedure in, say a breadth-first style, across all subcells of C_0, until no more gluing is possible, in which case we have obtained the desired cell C. We refer the reader to [1, 2] for more details. (We note that C can also be constructed using the recent randomized incremental technique of [4]. Both methods, however, rely on the existence of an efficient cell decomposition scheme, like the one about to be described.)

The performance of this algorithm crucially depends on the number of subcells ξ in the decomposition of C_0. We describe such a decomposition that has only $O(r^{2+\varepsilon})$ subcells, for any $\varepsilon > 0$. A standard calculation them implies that the expected running time of the algorithm is $O((kn)^{2+\varepsilon})$, for any $\varepsilon > 0$; see below for details.

The decomposition proceeds as follows. Any θ-cross-section of $\mathcal{A}(\mathcal{R})$ consists of r line segments. Indeed, when θ is fixed, B can only translate, and the locus of all translated placements of B at which it makes some specific edge-vertex contact is a line segment. Moreover, if we sweep a plane parallel to the xy-plane through the arrangement, the motion of the segments on the sweep plane is rather simple and has the following properties (which are easy to verify): For any pair of segments s, s', an endpoint of s has the same x-coordinate as an endpoint of s' at most a constant



cross-section at $\theta_0 - \varepsilon$ cross-section at $\theta_0 + \varepsilon$

Figure 2: The change in the extension segment as p crosses another segment at θ_0; both faces f and f' at θ_0 are added to the arrangement, each split into three subfaces by the extension segment

number of times; and an endpoint of s intersects the interior of s' at most a constant number of times.

Now, fix a segment endpoint and, at every θ, extend a vertical segment (parallel to the y-axis) up and down from that endpoint until it hits another segment, or else extends to infinity. We consider the union of these extensions, over all values of θ, as a collection of patches on an additional surface. A similar collection of patches is obtained for every other segment endpoint, so we obtain a total of at most $2r$ additional 'surfaces'.

Arguing as in [7, Lemma 6.12], one can show that at most $O(r^2)$ new faces are added to $\mathcal{A}(\mathcal{R})$ by inserting these 'extension surfaces'. Specifically, as long as a segment endpoint p remains in the same face f of the θ-cross-section of $\mathcal{A}(\mathcal{R})$, the extension segment from p traces a single face ϕ of the corresponding extension surface. Vertices of ϕ arise when the extension segment hits a vertex of f, which may also be another segment endpoint vertically visible from p (in the y-direction). Note that the collection of extension segments within a face f of a θ-cross-section partition it into several subfaces, and that as θ varies some of these subfaces can shrink and disappear, and be replaced by new subfaces, when pairs of extension segments overlap within f. Suppose that at some θ_0 the point p crosses into another face f' of the θ-cross-section. Then ϕ terminates at θ_0, and a new face ϕ' begins at this orientation along the extension surface. In this case we also add to our 3-dimensional arrangement $\mathcal{A}(S)$ all the 'horizontal' subfaces of f and of f' within the θ_0-cross-section, which are adjacent to the respective extension segments through p. See Figure 2 for an illustration of this process.

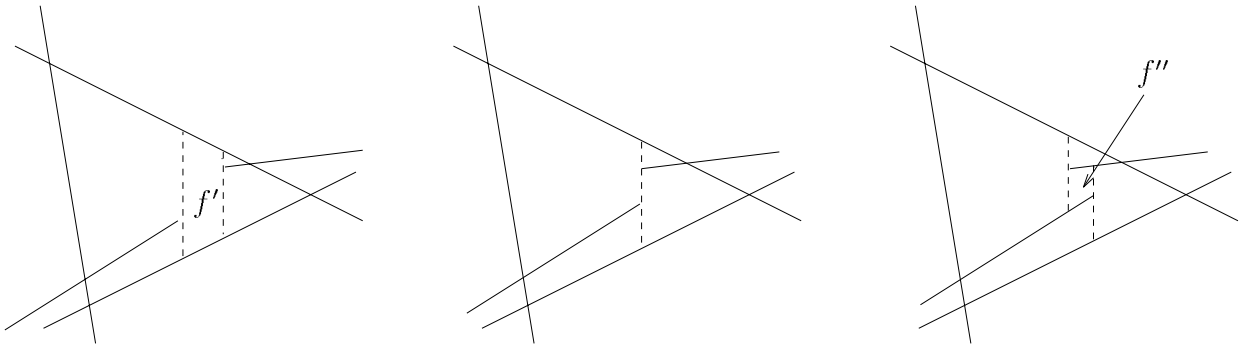


Figure 3: A subcell whose cross-section is f' terminates at θ_0, and a new subcell whose cross-section is f'' begins at θ_0

It thus follows that the overall number of faces added is $O(r^2)$; indeed, each crossing of a segment endpoint through another segment induces only a constant number of new faces, and the number of such crossings is $O(r^2)$. Note also that new subcells may start and end at θ-cross-sections in which two segment endpoints become vertically visible (in the y-direction) within a face of the cross-section (see Figure 3 for an illustration), but the number of such events is also only $O(r^2)$. Consequently, a single cell is divided by these extra surfaces into at most $O(r^2)$ 3-D subcells.

As is easily verified, each of the resulting subcells τ has the property that its θ-cross-section is always a *convex* polygon, and it varies continuously (in the Hausdorff metric of sets) with θ. This is easily seen to imply that every such subcell has a unique minimum and maximum in θ. As discussed in [7], the minimum need not be restricted to a single point, and might be attained by a line segment or a 2-dimensional face on the boundary of the cell, but this will not affect our analysis. There is also the special case of minima lying on the plane $\theta = 0$ (containing the marking point of the cell), since we want to consider this plane as another surface in our arrangement. This way we may have added up to $O(r^2)$ additional local θ-minima of cells.

One can also show that the combinatorial complexity of the faces that we have added to $\mathcal{A}(\mathcal{R})$ by the extensions from segment endpoints is $O(r\lambda_6(r))$. To see this, note that a vertex of such a face ϕ arises when the corresponding endpoint p sees a vertex of the face f containing p in some θ-cross-section, in the vertical y-direction. We can therefore define, for each of the given N surfaces σ, a (partial) function $F_\sigma(\theta)$ which is equal to the y-vertical distance from p to the segment σ_θ, the θ-cross-section of σ, whenever this distance is defined and p lies, say, below that segment (in the y-direction). It follows that each vertex of any face ϕ associated with p corresponds to a breakpoint in the lower envelope of the functions $F_\sigma(\theta)$, or in the upper envelope of a symmetric collection of functions, each defined when p lies above the corresponding segment σ_θ. Using the analysis of [11], one can show that any pair of these functions intersect in at most 4 points, so the number of breakpoints of the envelopes defined

for each endpoint p is $O(\lambda_6(r))$, from which the claim follows easily.

In contrast, we do not have equally sharp bounds for the complexity of the 'horizontal' faces that are also added to $\mathcal{A}(\mathcal{R})$ in the above analysis. We suspect that their overall complexity is also roughly quadratic in r, but so far we were not able to show this. Our decomposition scheme will finesse this issue.

To recap, we have decomposed C_0 into $O(r^2)$ subcells, each of which has the property that all its θ-cross-sections are convex and vary continuously with θ. Moreover, it follows from the above analysis, and from the general bound of [8] on the complexity of (the undecomposed) C_0, that the total combinatorial complexity of all these subcells, *excluding the complexity of the horizontal faces added in the decomposition*, is $O(r^{2+\varepsilon})$, for any $\varepsilon > 0$.

We next further decompose each of these subcells as follows. Imagine that we sweep (the decomposed) C_0 with a plane P parallel to the xy-plane, in the direction of increasing θ. Let $C_0(\theta)$ denote the cross-section $P \cap C_0$ when P is at height θ. We maintain a *balanced* triangulation of each convex face f of $C_0(\theta)$ and update it whenever P sweeps over a vertex of f, or when faces of $C_0(\theta)$ disappear, newly appear, split or merge. The triangulations are balanced in the sense that the dual graph of each triangulation is a balanced binary tree whose depth is thus only logarithmic. For specificity, we use red-black trees, as described in [17, Chapter 4]. (As defined, this dual tree is unrooted, but we root it at some arbitrary triangle incident to at least one edge of f.) In particular, no vertex of any face f is incident to more than $O(\log r)$ triangles, and the intersection of any line with a face f meets no more than $O(\log r)$ triangles (in both cases, the triangles form a path in the dual tree). For a discussion on the relation between triangulations and binary trees, see, e.g., [16]. See also Figure 4 for an illustration of a tree corresponding to a balanced triangulation. Note that a balanced binary tree with n nodes corresponds to a triangulation of a convex polygon with $n+2$ vertices, having therefore n triangles. We choose an appropriate edge of the polygon to be the *root* edge (so that the resulting rooted tree is balanced), and label all the vertices not incident to the root edge, with an increasing sequence of integers in counterclockwise order. The triangle incident to the root edge corresponds to the root of the tree, and the *key* attached to the root of the tree is the number of the vertex of the root triangle not incident to the root edge. Each of the two non-root edges of the root triangle may have a child triangle incident to that edge; its key is the number of the vertex of that triangle not incident to the parent (root) triangle. The labeling of tree nodes continues recursively in this manner; see Figure 4.

(Before proceeding, it is instructive to note that the need to maintain *balanced* triangulations is forced on us by the fact that we do not have a near-quadratic bound on the overall complexity of the horizontal faces added in the first decomposition step. If we had such a bound, we could have afforded to use any triangulation of the faces of the cross-section, because the overall number of triangles would have also been near-quadratic. We still need to triangulate these faces, to ensure that we get subcells with constant description complexity.)

For each triangle Δ we compute two critical orientations $\theta_1 < \theta_2$ at which Δ is

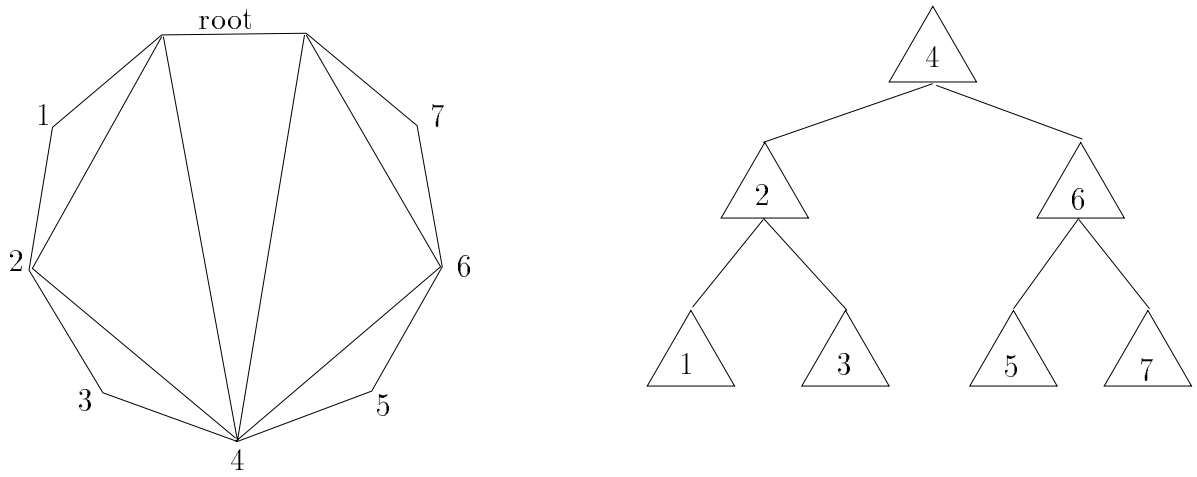


Figure 4: A balanced triangulation of a convex polygon and the corresponding balanced binary tree; each triangle is labeled by the vertex opposite to the side it shares with its parent

respectively 'opened' (newly added to the triangulation) and 'closed' (removed from the triangulation). At any time during the sweep, we store with each triangle in the current triangulation the critical orientation θ_1 at which it was opened. Such a triangle Δ induces a subcell

$$\xi_\Delta = \{(x, y, \theta) \mid \theta_1 \leq \theta \leq \theta_2,\ (x, y) \in \Delta(\theta)\}$$

where $\Delta(\theta)$ is the set of points occupied by Δ at the cross-section $C_0(\theta)$. It is clear that ξ_Δ has constant description complexity,[1] and that the collection of these subcells forms a decomposition of C_0 (which is a refinement of the first decomposition, obtained above). The main goal of the foregoing analysis is to estimate the number of subcells ξ_Δ that are created by the sweeping process. As we will see, the fact that the triangulations that we maintain are all balanced is crucial for the analysis. We also note that, even though the subcell decomposition is described below algorithmically, we are only concerned with its output size (namely with the number of triangles being created), and not with its running time, since we are dealing with a constant-size problem.

A new face of $C_0(\theta)$ is formed either when a connected component of $C_0(\theta)$ newly appears, or when a pair of adjacent convex faces of $C_0(\theta)$ merge into a new (convex) face (when a y-vertical segment separating them is removed), or when a face of $C_0(\theta)$ is split into two subfaces (when a y-vertical segment separating them is added). When a new component of $C_0(\theta)$ appears, as θ slightly increases, the component becomes

[1]Strictly speaking, since this property requires that the subcell be represented by *polynomial* equalities and inequalities, we should have replaced the third coordinate θ by $\tan\frac{\theta}{2}$; however, for convenience of presentation, we continue to denote this coordinate by θ.

a triangle; thus initializing the triangulation for a new component is trivial. To initialize the whole structure at $\theta = 0$, we simply triangulate each convex face of (the vertically decomposed) $C_0(0)$ in a balanced manner, and open all the resulting triangles at $\theta = 0$.

When the sweep plane P reaches a vertex u of some subcell τ of C_0 at an orientation θ_u (excluding vertices that lie on an added horizontal face which delimits τ from above or from below), one of several types of combinatorial changes can occur at u: an edge of the θ-cross-section $\tau(\theta)$ may shrink to a point and disappear, or a new edge of $\tau(\theta)$ may appear, or, when we encounter at θ_u an edge of τ parallel to the xy-plane, an edge of $\tau(\theta)$ may be replaced by another edge, or the entire face $\tau(\theta)$ may shrink to a long and thin trapezoid which is finally 'squashed' at θ_u. Nevertheless, such a change at u affects only a constant number of edges and vertices of $\tau(\theta)$, and thus affects only $O(\log r)$ triangles in the current triangulation of $\tau(\theta)$. Each of these triangles is closed at θ_u (so the subcells corresponding to these triangles are now fully defined), and $O(\log r)$ new triangles are formed as appropriate, replacing the old affected triangles; the new triangles are opened at θ_u.

Of course, after each such update we need to re-balance the dual tree of the triangulation of $\tau(\theta)$, if necessary. For this, we can use any of the known techniques for maintaining balanced binary trees; for specificity, we use the red-black tree technique, as described in [17, Chapter 4]. We observe that, since the triangulation that we maintain is of a convex polygon, any rotation that we want to apply to the dual tree, as an abstract structure, can be achieved by a straightforward retriangulation, in which the few triangles whose corresponding nodes have to be rotated in the tree are replaced by a few other triangles that represent these nodes after the rotation. Specifically, a single rotation in the binary tree corresponds to an *edge flip* in the triangulation; that is, for a pair of triangles sharing a diagonal in the triangulation, an edge flip is carried out by removing that diagonal—temporarily obtaining a (convex) quadrangle—and inserting the other diagonal of that quadrangle. Figure 5 shows how single and double rotations are implemented by edge flips. We will denote the balanced tree corresponding to the triangulated polygon f by $T(f)$.

We begin by describing in detail the operations that need to be performed when removing a vertex from the boundary of a convex face f. Adding a vertex to f can be done in a similar fashion. (The actual updating of f, occurring when an edge of f shrinks and disappears, or newly appears, can be implemented by removing two or one vertices from f, and then adding one or two new vertices to f, respectively.) We then describe how to update the triangulations when faces are merged or split.

Let u be the vertex of f that we are about to remove. By the above discussion, there are at most $O(\log r)$ triangles of the triangulation of f that are incident to u, and the nodes corresponding to these triangles in the tree form a path π of the tree. As is easily seen, the triangles incident to u form a convex polygon, all of whose vertices are vertices of f. If the path π contains the root node, then we denote this polygon by g. Otherwise, we follow a path π' from the root until it hits a node of π, add the triangles corresponding to the nodes of π' to those incident to u, and

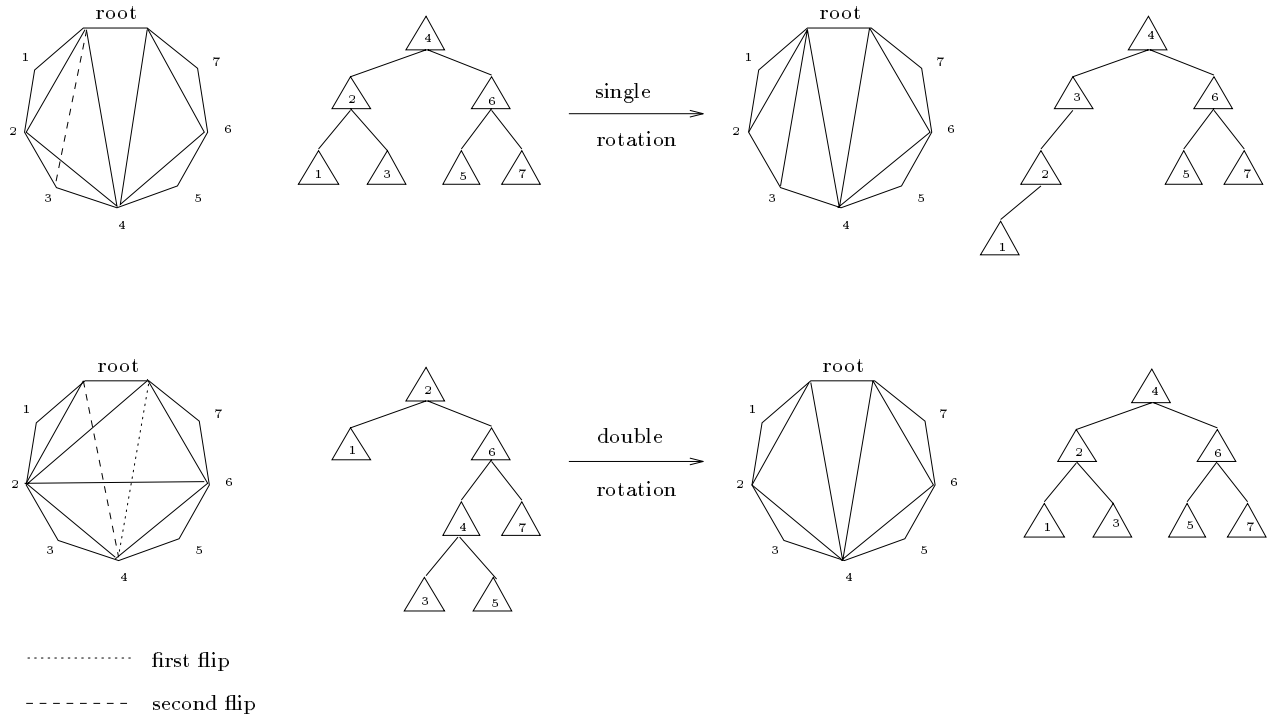


Figure 5: Realization of a rotation by re-triangulation of a face

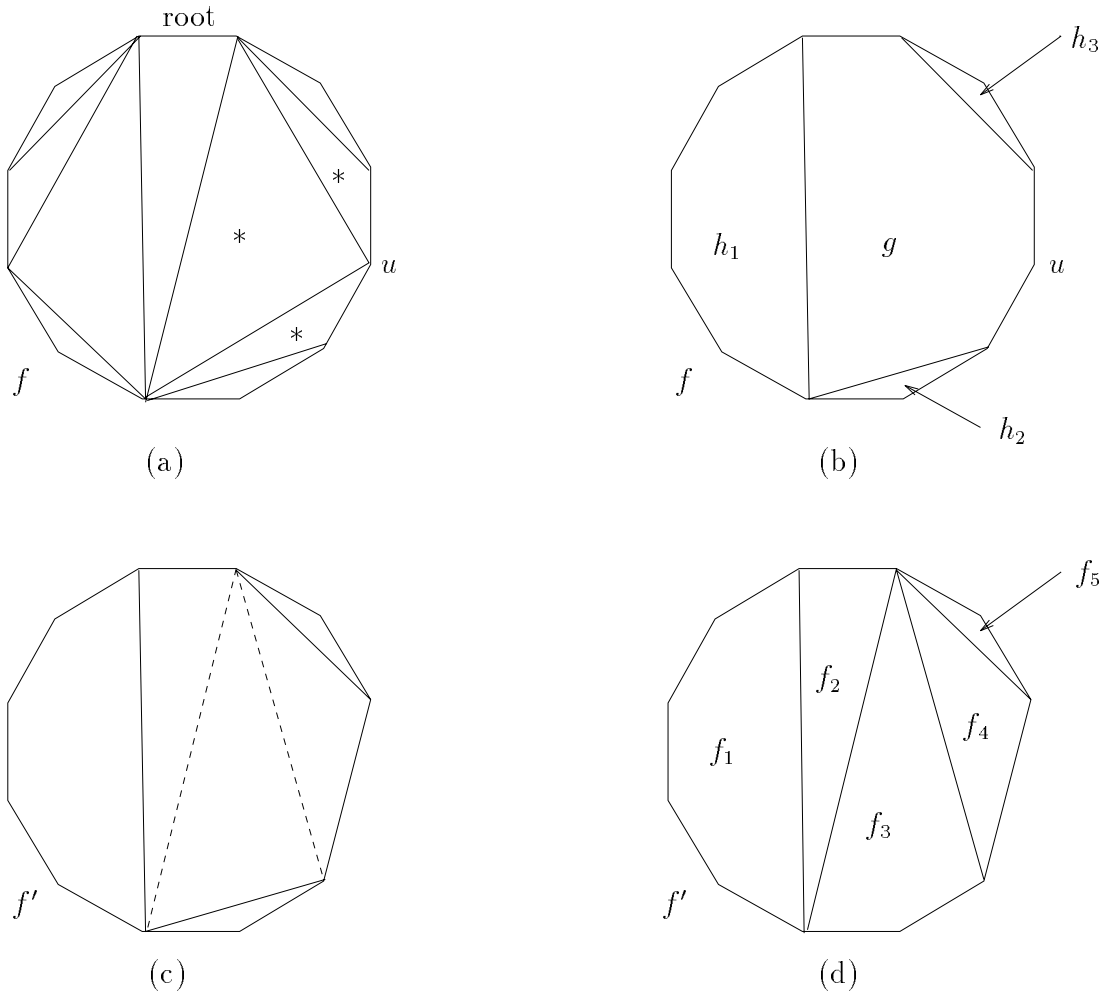


Figure 6: Removing a vertex from a convex face

let g denote the convex polygon covered by all these triangles; again, the number of original triangles contained in g is at most $O(\log r)$. Let $h_1, h_2, \ldots, h_m$ be the $O(\log r)$ connected components of $f \setminus g$. See Figure 6(a,b) for an illustration (the asterisks in Figure 6(a) denote the triangles incident to u). In the example depicted in Figure 6, there is only one node on the path π', namely the root node, which is thus added to the triangles incident to u in order to form g. Clearly, each of the h_i's is represented by a subtree of $T(f)$, and hence their corresponding subtrees $T(h_i)$ are all balanced.

We now remove the vertex u from the boundary of f and of g to obtain a new face f' and a new portion g' thereof, respectively. Note that the polygons $h_1, h_2, \ldots, h_m$ are unaffected by this removal, and only g changes (into g'). Next, we retriangulate g' into $O(\log r)$ triangles such that each diagonal in this triangulation is incident to one of the endpoints of the root edge to obtain $T(g')$; see Figure 6(c). For each triangle in $T(g')$ that shares with some h_i an edge not incident to the root edge, we add that triangle to h_i (making it the new root of $T(h_i)$), and remove it from the collection

$T(g')$ (thus the remaining triangles no longer form a triangulation of g'). It is easily verified that this process has split the face f' into subfaces $f_1, \ldots, f_q$, such that the numbers attached to the vertices of each subface form a contiguous interval of the numbers in f', and that these intervals have pairwise disjoint interiors. We number these faces in increasing order of the corresponding intervals, which corresponds to their counterclockwise order around f'; see Figure 6(d) for an illustration.

We now proceed to perform a series of join operations, as in [17, Chapter 4], taking care that when we join two polygons p_1 and p_2, the numbers attached to vertices of p_1 are all smaller than those attached to vertices of p_2. Thus, we join $T(f_1)$ to $T(f_2)$, then we join the resulting tree to $T(f_3)$, and so on, until all the trees $T(f_i)$ have been joined. This way we obtain a balanced triangulation $T(f')$ of the updated face f'. The cost (i.e., the number of tree operations) of each join operation is $O(\log r)$, and we repeat it $O(\log r)$ times. Thus the overall cost of this deletion is $O(\log^2 r)$; in other words, only $O(\log^2 r)$ new triangles are created.

The addition of a new vertex to a face f is performed in a similar manner; it is somewhat simpler, because it calls for adding just one new triangle to f.

Remarks. (1) The join operation $join(s_1, i, s_2)$ described in [17, Chapter 4] is defined for two trees s_1 and s_2, and for an additional element i, such that all the keys in s_1 are smaller than key(i) and all the keys in s_2 are greater than key(i). To conform to this notation, we note that the diagonal shared by the two subfaces that we join can play the role of the element i, by "thickening" it into a very thin triangle.

(2) An additional technical issue is that $join(s_1, i, s_2)$ proceeds by 'hanging' i as the right child of some node w lying on the rightmost path of s_1, and by making the original right child w' of w (if any) the left child of i. To implement this step via a re-triangulation, let $w_1 = w', w_2, \ldots, w_t$ be the nodes on the rightmost path of s_1 from w' downwards. As is easily seen, these nodes correspond to triangles all of which are incident to a common vertex u and arranged around u in counterclockwise order. Let $v_1, v_2, \ldots, v_{t+1}$ be the other vertices of these triangles, arranged in counterclockwise order around the face, and let v_{t+2} be the other vertex of (the 'thickened') i; see Figure 7. We now perform one 'giant' edge flip: connect v_1 with v_{t+2} by a diagonal, let i now denote the new triangle spanned by u and by this diagonal, and form new triangles representing $w_1, \ldots, w_t$ by connecting v_{t+2} with $v_1v_2, \ldots, v_tv_{t+1}$, respectively; see Figure 7 for an illustration. The remaining steps of the join are easy to implement by standard edge flips, as above.

Things are not much different when the sweep reaches an event at which two faces of $C_0(\theta)$ merge into a new face, or a face is split into two subfaces. Consider first the latter situation, let f denote the face before splitting, and let f_1 and f_2 denote the subfaces formed by the split. A naive way of handling this configuration is to close all triangles in the current triangulation of f, compute balanced triangulations of f_1 and f_2 from scratch, and open all these triangles. However, the cost of this approach would be proportional to the complexity of f, and, as noted above, we do not have a near-quadratic bound on the overall complexity of all such faces f. We thus use the following more refined procedure. Let e denote the y-vertical segment that splits f.

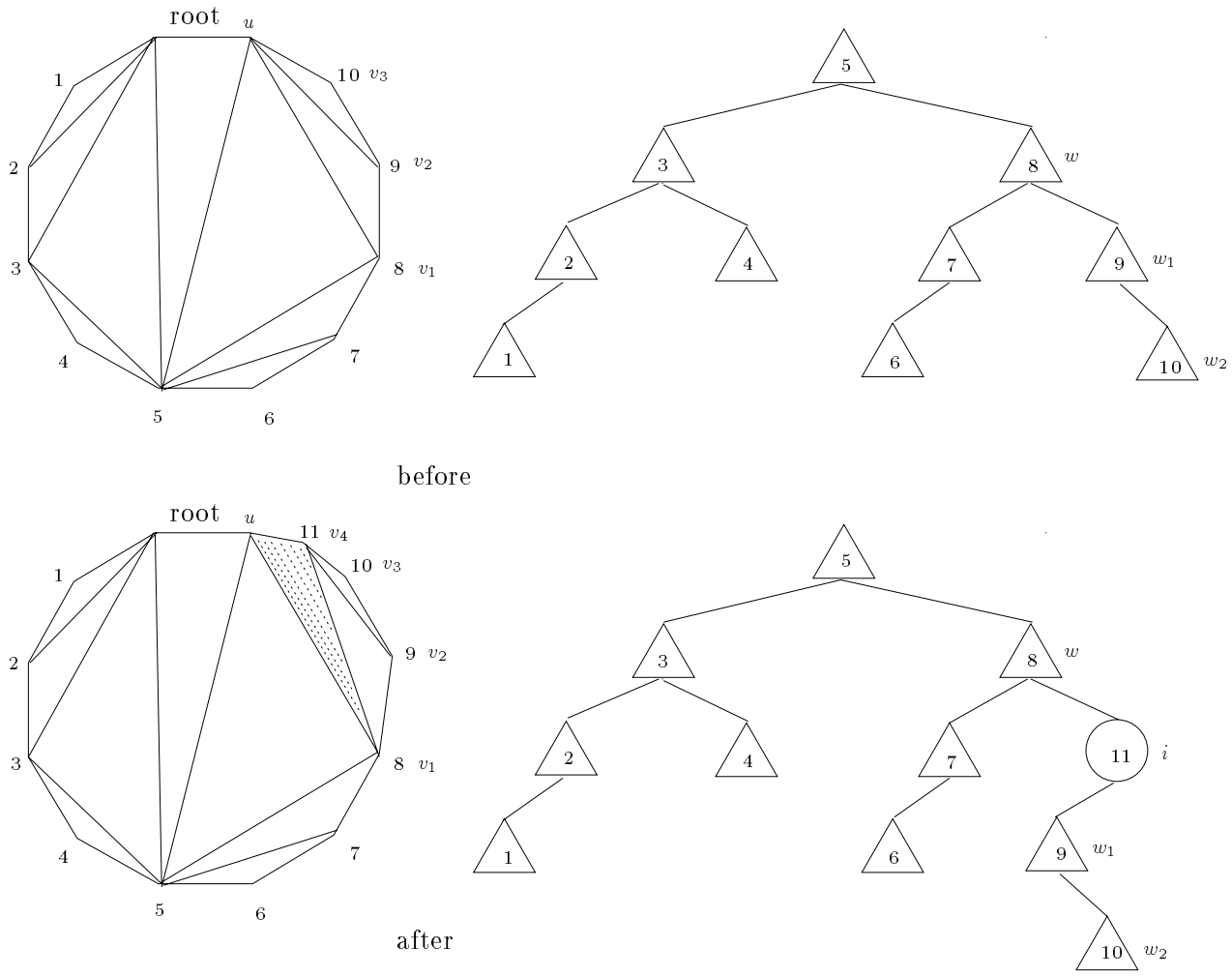


Figure 7: The re-triangulation corresponding to the hanging of a node i on a node w in the rightmost path of a tree

Since e cuts the boundary of any triangle in at most two points, it easily follows, as noted above, that the set of triangles in the triangulation of f which are crossed by e form a single path π in the dual (unrooted) tree of the triangulation, or at most two paths in its rooted version $T(f)$. Thus the number of such triangles is only $O(\log r)$. Let g denote the polygon covered by these triangles. As before, let $h_1, \ldots, h_m$ denote the connected components of $f \setminus g$, each of which is clearly a convex region. Some of the h_i's are contained in f_1 while the others are contained in f_2. We split g into two subfaces g_1 and g_2, using the y-vertical edge e.[2] The edge e splits the edges of f into two chains, one of which contains the root edge, say the boundary of f_1. In this case we extend g_1 as before until it contains the root edge, and update the h_i's accordingly. Note that in f_1 the root edge is the same root edge as in f, and in f_2 the newly added y-vertical edge e is the root edge. From this point, we proceed to handle each subface f_1, f_2 as in the case of deleting a vertex. A similar, though slightly different, procedure is used if the root edge is split by e.

Next consider a merge of two faces f_1, f_2, into a common convex face f, caused by removing a y-vertical edge e separating between f_1 and f_2. Let π_1 (resp. π_2) denote the path in $T(f_1)$ (resp. in $T(f_2)$) from the root to the triangle incident to e in f_1 (resp. in f_2). The union of all $O(\log r)$ triangles in these two paths is a convex polygon g contained in f. As above, let $h_1, \ldots, h_m$ denote the connected components of $f \setminus g$, each of which is a convex polygon. Let r_1 be the root edge of $T(f_1)$. We renumber the vertices of f so that they form an increasing sequence of integers in their counterclockwise order along f, going from r_1 all the way around. (This is an expensive operation, but, as noted above, we are only concerned with the number of triangles that the algorithm produces, and not with its running time.) We now re-triangulate g and perform a series of join operations, exactly as in the case, described above, of deleting a vertex, where the joins are performed according to the new order of the h_i's.

The final type of update occurs when a connected component of $C_0(\theta)$ shrinks to a point and disappears. In this case we simply close the single triangle of its current triangulation, and remove its singleton dual tree from our forest. This completes the description of our triangulation of the cell C_0.

Note that each vertex of C_0 (excluding those on the added horizontal faces) causes the generation of only $O(\log^2 r)$ subcells, and the same holds for each added horizontal face (whose number is only $O(r^2)$). It follows that the number of subcells in the decomposition of C_0 is $O(r^{2+\varepsilon} \cdot O(\log^2 r) = O(r^{2+\varepsilon})$, for any $\varepsilon > 0$ (with a slightly larger ε in the second exponent), as claimed.

The analysis of the expected running time of the algorithm is now straightforward, and similar to that in [1, 2]. That is, if $T(N)$ denotes the maximum expected running time of the algorithm for a collection of N contact surfaces, then we have (recalling

[2] Note that the endpoints of e need not necessarily be vertices of f. To be precise, we first have to add the endpoints of e as new vertices of f, and then proceed with the splitting operation as described above.

that r is assumed to be a constant):

$$T(N) \leq c_1 r^{2+\varepsilon} \cdot T(\frac{cN}{r} \log r) + O(N^{2+\varepsilon}) ,$$

for appropriate constants c and c_1 (independent of r). With an appropriate choice of r as a function of ε, one easily verifies that the solution of this recurrence is $T(N) = O(N^{2+\varepsilon'})$, for any $\varepsilon' > \varepsilon > 0$, where the constant of proportionality depends on ε and ε'. We have thus shown:

Theorem 2.1 *A single connected component of the free configuration space of a k-sided polygon, moving in a polygonal region bounded by n edges, can be computed by a randomized algorithm with expected running time $O((kn)^{2+\varepsilon})$, for any $\varepsilon > 0$, with the constant of proportionality depending on ε.*

Remark. The algorithm can be made deterministic, using the deterministic (though rather complicated) construction of ε-nets given by Matoušek [12]. The asymptotic running time of the algorithm remains the same.

It is interesting to note that our algorithm constructs a representation of the cell C which is suitable for *point location*. For this we maintain the entire recursive structure that the algorithm computes, in the form of a rooted tree. That is, at the root of that tree we store the decomposition of the cell C_0 into the subcells ξ_Δ, as described above. Each such subcell becomes a child of the root, and stores a similar cell decomposition in the arrangement of the random sample constructed for the corresponding subproblem, and so on. Now, given a query point q, we find, by brute force, the subcell ξ_Δ of the top cell C_0 containing q. If no such subcell is found, we conclude that q is not in our cell. Otherwise, we continue the search recursively at the child corresponding to ξ_Δ. Thus, we can determine whether q lies in our cell in $O(\log kn)$ time. In terms of the motion planning application, this means that we can determine in logarithmic time (using $O((kn)^{2+\varepsilon})$ preprocessing and storage, for any $\varepsilon > 0$) whether a given placement of the robot B is free and can be reached from Z via a collision-free path (this is the so-called *reachability problem*). It is also easy to enhance our data structure with additional information, so that actual motion planning between Z and a query placement can be performed in $O((kn)^{2+\varepsilon})$ time.

Acknowledgements

The authors wish to thank Jirka Matoušek and Leo Guibas helpful discussions concerning the problem studeid in this paper.

References

[1] B. Aronov and M. Sharir, Triangles in space, or building (and analyzing) castles in the air, *Combinatorica* 10 (1990), 137–173.

[2] B. Aronov and M. Sharir, Castles in the air revisited, *Discrete Comput. Geom.* 12 (1994), 119–150.

[3] L.P. Chew and K. Kedem, A convex polygon among polygonal obstacles: placement and high-clearance motion, *Comput. Geom. Theory Appls.* 3(2) (1993), 59–89.

[4] M. de Berg, K. Dobrindt, and O. Schwarzkopf, On lazy randomized incremental construction, *Discrete Comput. Geom.* 14 (1995), 261–286.

[5] L. Guibas and M. Sharir, Combinatorics and algorithms of arrangements, in *New Trends in Discrete and Computational Geometry,* (J. Pach, Ed.), Springer-Verlag, 1993, 9–36.

[6] D. Halperin and M. Sharir, Arrangements and their applications in robotics: Recent developments, in *The Algorithmic Foundations of Robotics*, K. Goldberg, D. Halperin, J.C. Latombe and R. Wilson, Eds., A.K. Peters, Boston, MA, 1995, 495–511.

[7] D. Halperin, *Algorithmic Motion Planning via Arrangements of Curves and of Surfaces,* Ph.D. Dissertation, Computer Science Department, Tel Aviv University, July 1992.

[8] D. Halperin and M. Sharir, Almost tight upper bounds for the single cell and zone problems in three dimensions, *Discrete Comput. Geom.* 14 (1995), 385–410.

[9] K. Kedem and M. Sharir, An efficient motion-planning algorithm for a convex polygonal object in two-dimensional polygonal space, *Discrete and Computational Geometry* 5 (1990), 43–75.

[10] K. Kedem, M. Sharir and S. Toledo, On critical orientations in the Kedem-Sharir motion planning algorithm for a convex polygon in the plane, *Proc. 5th Canadian Conference on Computational Geometry* (1993), 204–209.

[11] D. Leven and M. Sharir, On the number of critical free contacts of a convex polygonal object moving in 2-D polygonal space, *Discrete Comput. Geom.* 2 (1987), 255–270.

[12] J. Matoušek, Approximations and optimal geometric divide-and-conquer, *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, 506–511.

[13] J.T. Schwartz and M. Sharir, On the two-dimensional Davenport Schinzel problem, *J. Symbolic Computation* 10 (1990), pp. 371–393.

[14] M. Sharir and P.K. Agarwal, *Davenport Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[15] M. Sharir and S. Toledo, Extremal polygon containment problems, *Computational Geometry — Theory and Applications* 4 (1994), 99–118.

[16] D.D. Sleator, R.E. Tarjan and W.P. Thurston, Rotation distance, triangulations, and hyperbolic geometry, *J. Amer. Math. Soc.* 1 (1988), pp. 647–681.

[17] R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.