SEPARATING A POLYHEDRON BY ONE TRANSLATION FROM A SET OF OBSTACLES

(Extended Abstract)

by Otto Nurmi[*] and Jörg-R. Sack[**]

SCS-TR-134

December 1987

School of Computer Science
Carleton University
Ottawa, Ontario
CANADA K1S 5B6

This work was supported by Deutsche Forschungsgemeinschaft and by NSERC.

This report also appears in the technical report series published at the University of Freiburg.

[*]On leave from the University of Helsinki, Finland.
[**]The work was done in part during a visit at the Universität Karlsruhe and the Universität Freiburg.

Separating a Polyhedron by One Translation from a Set of Obstacles[*]
(Extended Abstract)

Otto Nurmi[†]
Institut für Informatik, Universität Freiburg
Rheinstraße 10–12, D-7800 Freiburg i.Br.
Fed. Rep. of Germany

Jörg-R. Sack[‡]
School of Computer Science, Carleton University
Ottawa, Ontario, Canada K1S 5B6

We present efficient algorithms for several problems of movable separability in 3-dimensional space. The first algorithm determines *all* directions in which a convex polyhedron can be translated through a planar convex polygon. The algorithm runs in $O(n)$ time. This is a considerable improvement over the previous $O(n^2 \log n)$ time algorithm of [17]. The second algorithm computes, in $O(n)$ time, *all* directions in which a convex polyhedron can be translated to infinity without collisions with a convex obstacle (n is the number of vertices of the polyhedra). A generalization of the plane-sweep technique, called "sphere-sweep", is given and provides an efficient algorithm for the last problem which is: determine all directions in which a convex polyhedron can be separated from a set of convex obstacles. Our results are obtained by avoiding the standard technique of motion planning problems, the $O(n^2)$ time computation of the Minkowski differences of the polyhedra.

1. Introduction

Motion plannig problems arise in areas like computer-aided manufacturing, computer graphics, and robotics (see the survey in [18]). One class of such problems are the "separability" problems (see, e.g., [2,17]). Given a complex object consisting of several parts, one is asked to loosen one of them by motions such as translations and/or rotations.

In this paper we consider separability problems in which the objects are convex polyhedra and one of them must be moved by one translation motion arbitrarily far away. The object that must be separated is not allowed to collide with other objects during the translation. Such problems are often called collision avoiding problems.

The *Minkowski difference* of two sets, A and B, is $A - B = \{ a - b \,|\, a \in A, b \in B \}$. Lozano-Perez and Wesley [6] introduced an algorithmic technique based on Minkowski differences of the obstacles and the object to be moved. They called the space containing the differences "configuration space". Since then, the configuration space technique has become a standard tool for motion planning algorithms (see, e.g., [4,16,18]). Given an object B and a set of obstacles A_i, $i = 1, \ldots, k$, Lozano-Perez and Wesley first construct the "grown" obstacles $A_i - B$. An object can be translated by a vector without collisions if and only if the origin can be translated by the same vector without

[*]The work was supported by Deutsche Forschungsgemeinschaft and by NSERC.
[†]On leave from the University of Helsinki, Finland
[‡]The work was done in part during a visit at the Universität Karlsruhe and the Universität Freiburg.

collisions occuring between it and the grown obstacles. The Minkowski difference of two simple polygons as well as the difference of two polyhedra may have $\Omega(n^2)$ vertices, where n is the number of vertices of the objects. An example of such polyhedra is given in Section 4.

In this paper we develop algorithms for 3-dimensional translation problems. We avoid the construction of Minkowski differences in order to obtain more efficient solutions.

Given a convex polyhedron A and a planar polygon W, called the *window*, our first algorithm computes all directions in which A can be translated through W. The problem was posed by Toussaint [17] who gives an $O(n^2 \log n)$ time algorithm for it (n is the number of vertices of A and W). Our algorithm, based on the merge-step of the well-known 3-dimensional convex hull algorithm of Preparata and Hong [11], improves the time bound to $O(n)$ which is optimal. The algorithm is presented in Section 3. Related problems for 2-dimensional space have been discussed by Maddila and Yap for moving a chair through a door [19] and a polygon around the corner in a corridor [7].

Our second algorithm, presented in Section 4, determines all directions of collision-free translations of a polyhedron in the presence of one convex polyhedral obstacle. The algorithm needs $O(n)$ time, where n is the total number of vertices of the object and the obstacle. It is based on a fast construction of the *extreme separating planes* of the object and the obstacle. Davis [1] defines a transformation of the convex hull of polyhedra which gives these planes. We show that the planes can be obtained in $O(n)$ time, without using a transformation ([1] does not contain a time analysis of the transformation).

Given a convex polyhedron A and a set of convex polyhedral obstacles $B_1, B_2, \ldots, B_k$, our third algorithm computes *all* directions of a translation that moves A arbitrarily far away such that no collisions occur between A and the obstacles. The problem has been posed by H. Seeland, Daimler Benz ([14]), in the following context. In the design of a car, the CAD-system should allow queries of the form: Can a given faulty part be removed easily, i.e., by means of one single translation, or not. If it is possible, direction queries posed by the designer should be answered fast while preprocessing time may be longer (an overnight job).

The simple algorithm determines all directions in which A passes one obstacle. When this is done for all obstacles, the results are combined by "sphere sweep". The valid directions are obtained in a form that enables them to be stored in a data structure for planar point location. The data structure supports queries whether A can be separated into a given direction by means of one single translation. The algorithm is presented in Section 5.

Finally, Section 6 contains conclusions and some further remarks.

2. Basic Terms

Throughout this paper, we assume that polyhedra are given by doubly-connected edge lists (see [12]). Polygons are given by doubly-connected circular lists of their vertices. Each link of the list contains an information on which side of the corresponding edge the interior of the polygon lies.

Our algorithms output sets of directions. Because we want to translate the objects to infinity, an obvious way to represent a direction is to give a halfline whose endpoint is the origin. Such a halfline intersects the unit sphere at one point, and, conversely, each point of the unit sphere corresponds to one halfline. Thus, a set of directions can be represented as a collection of regions on the unit sphere. In our case, the regions are spherical polygons. Their sides are parts of "great circles" of the sphere. We represent such a polygon, analogously to a planar polygon, by a doubly-connected list of its vertices.

The set of halflines that corresponds to a spherical polygon forms a half-open polyhedron. It has only one vertex, the origin. We call such a polyhedron a (polygonal) *cone*. Our cones may be non-convex. The intersection of a cone and the unit sphere may be larger than a semisphere. The representation of a cone as a polygon on the unit sphere is readily constructed from its any other reasonable representation.

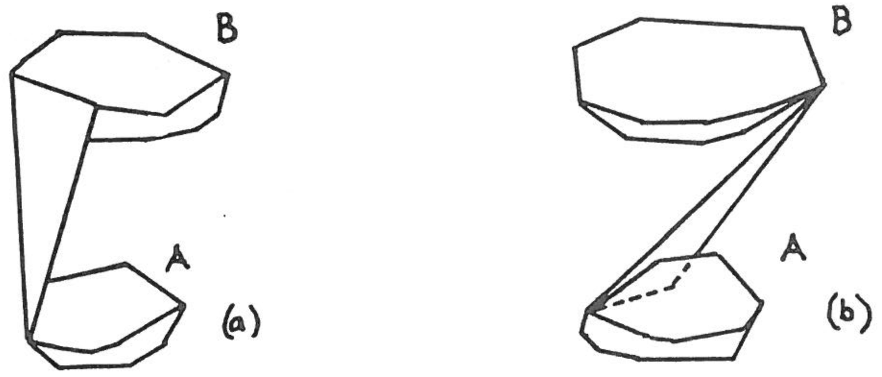

FIG. 1. (a) An extreme supporting plane of A and B. (b) An extreme separating plane of A and B.

Let A and B be disjoint polyhedra, and P a plane in 3-dimensional space. We say that P is *extreme* if it contains an edge of A (B) and a vertex of B (A), and A and B are contained in *one* of the closed halfspaces determined by P. (Note that, in contrast to [1], the faces of A and B need not be extreme.) A plane P *supports* A and B if $A \cap P \neq \emptyset$, $B \cap P \neq \emptyset$, and A and B are contained in the same closed halfspace determined by P. If P is, in addition, an extreme plane then it is an *extreme supporting plane* of A and B (see Fig. 1 (a)). A plane P *separates* A and B if they are contained in the different closed halfspaces determined by P. It is an *extreme separating plane* of A and B if it is, in addition, an extreme plane (see Fig. 1 (b)).

3. Passing a Polyhedron through a Convex Window

Given a convex polyhedron A and a planar convex polygon W (the *window*), such that $A \cap W = \emptyset$, we want to compute all directions in which A can be translated through W. The set of such directions may, of course, be empty.

Let us assume that A can be translated through W in directions $d_1, d_2, \ldots, d_k$. We call them *valid* directions. Because W is convex all directions that belong to the minimal convex cone containing $d_1, d_2, \ldots,$ and d_k, are valid directions. Our problem is to compute the maximal such cone in the sense that all directions inside the cone are valid but no direction outside the cone is valid. Following the terminology of [2], we call such a cone, $C_W(A)$, the *movability cone* of A with respect to W.

We propose the following algorithm for computing $C_W(A)$:

1. Compute the extreme supporting planes P_i ($i = 1, 2, \ldots, l$) of A and B. Let H_i denote the halfspace which is determined by P_i and which contains A and B.

2. For $i = 1, 2, \ldots, l$, translate P_i to the origin; denote the corresponding translated halfspace H_i^0.

3. Compute the intersection $C(A, W) = \bigcap_{i=1}^{l} H_i^0$.

The following lemma states the validity of the algorithm.

Lemma 1. $C(A, W) = C_W(A)$.

Proof. First, let $d \in C(A, W)$ be a direction. We claim that d is a direction of a valid translation. Let $a \in A$ be an arbitrary point. Translate $C(A, W)$ in such a way that its apex is a. By the construction, the faces of the translated cone intersect W. Since d belons to $C(A, W)$ and W is convex, d intersects W, too. But if a is translated in the direction of d then it encounters the

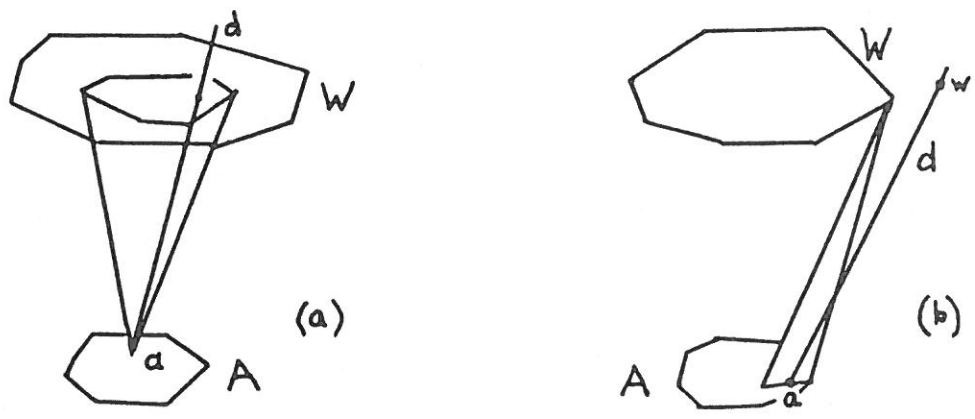

FIG. 2. Proof of Lemma 1.

intersection point of d and W (see Fig. 2 (a)). Since a was chosen arbitrarily, we can conclude that all points of A encounter a point of W when translated in the direction d. Thus, d is a valid translation direction and $d \in C_W(A)$.

Conversely, let $d \in C_W(A)$ be a valid direction. We show that $d \in C(A, W)$. Assume the countrary. Then there exists, for some $1 \leq i \leq l$, a halfspace H_i^0 that does not contain d. Choose a point $a \in A \cap H_i$ and translate the halfline d in such a way that its endpoint is a. Now d is not contained in H_i. Since $d \in C_W(A)$, a will encounter a point $w \in W$ if it is translated in the direction d (see Fig. 2 (b)).

But since d is not contained in H_i, w cannot be contained in H_i. This is a contradiction since H_i is a supporting plane. □

The merge step of the 3-dimensional convex hull algorithm of Preparata and Hong [11] computes the "cylinder faces" of the convex hull of two convex polyhedra. The cylinder faces determine the extreme supporting planes of the polyhedra. A precondition of the step is that the orthogonal projections of the polyhedra on a plane are available, and that these projections do not intersect.

We can use the merge-step of the algorithm of [11] once its preconditions are fulfilled. To do this, we first search for a plane S that separates A and W. It is found in $O(n)$ time by solving a 3-variable linear program with n constraints, where n is the number of vertices of A and W (see [9,12]). Then we choose a projection plane P which is perpendicular to S. The orthogonal projections of A and W on P are separated by the line $P \cap S$. The linked list representing the contour of A's projection is easy to construct in $O(n)$ time. The pre-conditions of the merge-step of Preparata and Hong's algorithm are now fulfilled. The merge-step takes $O(n)$ time. Thus the first step of our algorithm also takes $O(n)$ time.

The second step of our algorithm is linear since we have $O(n)$ planes to translate.

The computation of the intersection of n halfspaces takes, in general, $\Omega(n \log n)$ time (see, e.g., [12]). The bounding planes of our halfspaces have one common point, the origin, and the first step of our algorithm renders them in a special "cylindrical" order. We shall show that the intersection of our halfspaces can be computed in linear time.

Let $a \in A \cap H_1$, and $w \in W \cap H_1$ be two points, and R a plane whose normal is $w - a$ and which contains w. The intersections $H_i^0 \cap R$, $(i = 1, \ldots, k)$, are halfplanes. The cone can now be constructed by computing the common intersection of these halfplanes. The "cylindrical" order in which the halfspaces are obtained yields the halfplanes in the order of the slopes of their bounding lines, or, more specifically, in the order of the directions of their normal vectors that point outwards from the interior of the halfspaces. The intersection is constructed by considering the halfplanes in that order. This can easily be accomplished in $O(n)$ time.

Since each step of the algorithm takes $O(n)$ time the cone $C(A, W)$ can be computed in $O(n)$

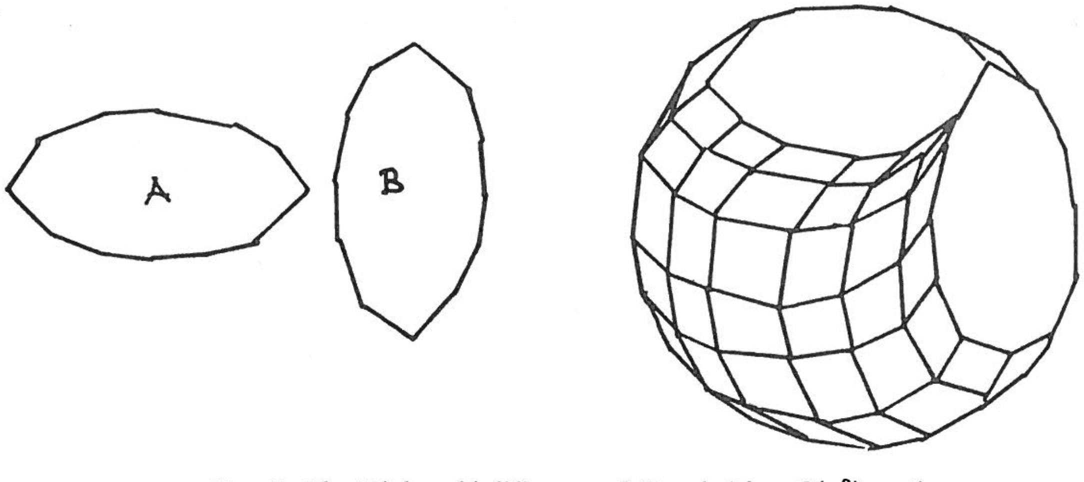

FIG. 3. The Minkowski difference of B and A has $O(n^2)$ vertices.

time. From this and Lemma 1 follows:

Theorem 1. *All directions in which a convex polyhedron can be translated in such a way that it passes through a planar convex polygon can be computed in $O(n)$ time, where n is the number of vertices of the polyhedron and the polygon.*

A (non-convex) polyhedron can be passed through a convex window by one translation if and only if its convex hull can pass the window. The convex hull can be computed in $O(n \log n)$ time by the algorithm of Preparata and Hong [11]. Thus we have the corollary:

Corollary 1. *All directions in which a (non-convex) polyhedron can be passed through a planar convex polygon can be computed in $O(n \log n)$ time, where n is the number of vertices of the polyhedron and the polygon.*

As pointed out by Toussaint [17], it is sufficient that all vertices of the polyhedron go through the window. This means that some of our supporting planes are superfluous. It is easy to modify the algorithm in such a way that these planes are not computed. This does, however, not improve its asymptotic time bound.

A related problem, discussed in [17], is to find the "smallest" window W on a plane H through which a convex polyhedron P can be passed with a single translation, orthogonal to H, after prepositioning P with arbitrary translations and a rotation. The problem can be solved using the result of McKenna and Seidel [8] who give an $O(n^2)$ algorithm for computing the minimum area shadow of a convex polyhedron with n vertices.

4. Translations in the Presence of One Convex Obstacle

Given two convex polyhedra, an *object* A, and an *obstacle* B, $A \cap B = \emptyset$, we want to translate A arbitrarily far away without collisions occuring between A and B. We say that such a direction is a *valid* direction of translating A. That is, a direction d is valid if and only if, for each vector $\mathbf{v}$ whose direction is d, the intersection of B and $A + \mathbf{v} = \{ a + \mathbf{v} \mid a \in A \}$ is empty. An *invalid* direction is a direction that is not valid.

Our problem is: compute all valid directions of translating A in the presence of B. We solve the problem by computing first all *invalid* directions of translating A. As in Section 3, they form a polygonal cone. The cone is denoted by $C_B(A)$. By the definitions, a direction d is invalid if and only if there exist points $a \in A$ and $b \in B$ such that the direction of the vector $b - a$ is d. Thus a direction d is invalid if and only if the Minkowski difference $B - A = \{ b - a \mid b \in B, a \in A \}$ contains

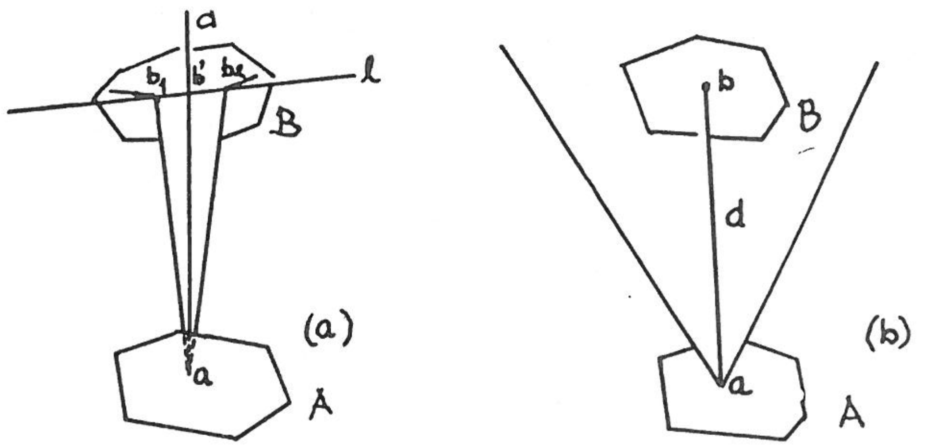

FIG. 4. Proof of Lemma 2.

a vector whose direction is d. A standard technique of motion planning problems constructs the Minkowski difference of the obstacle and the object.

If A and B are convex polyhedra with in total n vertices then $B - A$ may have $\Omega(n^2)$ vertices. An example of such polyhedra can be described as follows. Let C be a disc in the (x_1, x_2)-plane and D a disc in the (x_2, x_3)-plane. Approximate the discs by planar regular $n/2$-gons B and A whose vertices touch the bounding circles of the discs. It is easy to see that each vertex of B introduces about $n/4$ vertices to $B - A$. Thus $B - A$ has about $n^2/8$ vertices (see Fig. 3). Thus, any algorithm that computes the differences of 3-dimensional polyhedra needs $\Omega(n^2)$ time in the worst case. (In 2-dimensional space, the Minkowski difference of two convex polygons with in total n vertices can be computed in $O(n)$ time (see [3])).

The cone $C_B(A)$ is the smallest one that contains the difference $B - A$. Notice that only a few of $(B - A)$'s edges introduce a face to $C_B(A)$. The edges form a polygonal line with $O(n)$ corners. We shall show that the following algorithm computes the cone in $O(n)$ time.

1. Compute the extreme *separating* planes P_i $(i = 1, 2, \dots, l)$ of A and B. Let H_i denote the halfspace which is determined by P_i and which contains B.

2. For $i = 1, 2, \dots, l$ translate P_i to the origin; denote the corresponding translated halfspace H_i^0.

3. Compute the intersection $C(A, B) = \bigcap_{i=1}^{l} H_i^0$.

Lemma 2. $C(A, B) = C_B(A)$.

Proof. We first show that $C(A, B) \subset C_B(A)$. Since $B - A$ is convex it suffices to show that a direction d is contained in $C_B(A)$ if it is contained in a face of $C(A, B)$. Thus, let d be contained in the face F of $C(A, B)$. For some i, $1 \leq i \leq l$, the extreme separating plane P_i is parallel with F. Let us first assume that P_i contains A's vertex a and B's edge (b_1, b_2). Consider the halfline d' which is parallel with d and whose endpoint is a. Let b' be the intersection point of d' and the line l which contains (b_1, b_2) (see Fig. 4 (a)).

Since d is contained in the intersection of the separating halfspaces, b' must lie in the interval $[b_1, b_2]$ of l. (Otherwise d was cut off from $C(A, B)$ by one of the other (translated) separating planes that contain b_1 or b_2.) Thus a collides with b' if it is translated in direction d'. Hence $d \in C_B(A)$. The case where P_i contains B's vertex and A's edge is similar.

Conversely, let $d \in C_B(A)$ be a direction. Then there exist points $a \in A$ and $b \in B$ such that a collides with b when it is translated in the direction d. Translate $C(A, B)$ in such a way that its vertex is a. By the construction, B is now contained in the translated cone (see Fig. 4 (b)).

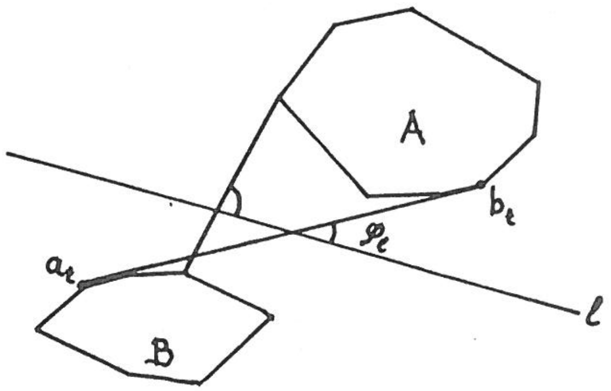

FIG. 5. The "separating tangent" of $p(A)$ and $p(B)$ has the minimal acute angle φ_l with l.

Thus b belongs to the translated cone and the direction d also belongs to it. Since the translation does not rotate the cone the direction $d \in C(A, B)$. □

Davis [1] defines, without giving a time analysis, a transformation of two polyhedra in such a way that their extreme separating planes are obtained from the extreme supporting planes of the transformed polyhedra. We shall show how the extreme separating planes can be constructed without a transformation and in $O(n)$ time, where n is the number of the vertices of the polyhedra.

Our algorithm is very similar to the one for computing the extreme supporting planes. The only difference is that the *initial plane* is chosen in a different way. The plane must be a separating plane that contains a vertex from both polyhedra.

We assume that the polyhedra A and B are projected orthogonally onto a plane P, and that their projections $p(A)$ and $p(B)$ are separated in P by a line l. These projections are computed as explained in Section 3.

We now turn l in such a way that it becomes a "separating tangent" of $p(A)$ and $p(B)$. Let S denote all lines that contain a vertex of $p(A)$ and a vertex of $p(B)$. It is clear that the "tangents" are contained in S. If a is a vertex of $p(A)$ and b is a vertex of $p(B)$ then by $\varphi_l(a, b)$ we denote the size of the acute angle of l and the line containing a and b. Now the line which contains a_t of $p(A)$ and b_t of $p(B)$ such that $\varphi_l(a_t, b_t) = \min\{\varphi_l(a, b) \mid a \text{ is a vertex of } p(A) \text{ and } b \text{ is a vertex of } p(B)\}$ is a "tangent" (see Fig. 5).

Thus, we must determine the line $t \in S$ for which the acute angle is minimal. Let a be a fixed vertex of $p(A)$. If the vertices of $p(B)$ are traversed circularly then the sequence of angles $\{\varphi_l(a, b) \mid b \text{ is a vertex of } p(B)\}$ has at most two local minima. Similarly, if a vertex of $p(B)$ is fixed and the vertices of $p(A)$ are visited circularly then the corresponding sequence of angles is bimodal. This observation yields the following algorithm for determining the "tangent".

1. Choose a vertex a_0 of $p(A)$ and a vertex b_0 of $p(B)$.
   Set $a \leftarrow a_0$ and $b \leftarrow b_0$.

2. Starting at b, traverse the vertices b_i of $p(B)$ circularly in the direction of decreasing $\varphi_l(a, b_i)$ until a local minimum is found. Let $b_{\min}$ denote that node.

3. Starting at a, traverse the vertices a_i of $p(A)$ circularly in the direction of decreasing $\varphi_l(a_i, b_{\min})$ until a local minimum is found. Let $a_{\min}$ denote that node.

4. If $a \neq a_{\min}$ or $b \neq b_{\min}$ then set $a \leftarrow a_{\min}$ $b \leftarrow b_{\min}$, and go to Step 2.

5. The "separating tangent" is the line which contains $a_{\min}$ and $b_{\min}$.

The algorithm halts at Step 5 when no new (local) minima were found. The vertices of $p(A)$ are scanned in only one direction. The vertices of $p(B)$ may be scanned in two directions. The direction can, however, change only after the first traversal. From the observation above, it follows that the vertices of $p(B)$ and $p(A)$ are visited at most twice. Thus the algorithm takes $O(n)$ time.

The *initial plane* is now the plane which contains the "tangent" and which is perpendicular to P.

All extreme separating planes of A and B can be found by "wrapping" planes around the polyhedra as in the merge-step of [11]. The "wrapping" begins from the *initial plane* which was computed above and takes $O(n)$ time. The planes are obtained in a "cylindrical" order (see Section 3). As in Section 3, steps 2 and 3 of the algorithm for computing $C_B(A)$ take $O(n)$ time. Thus the cone $C_B(A)$ is computed in $O(n)$ time.

In order to compute the valid directions of translating A, we traverse the edges of the spherical polygon corresponding to $C_B(A)$ and, during the traversal, we change at each edge the information that tells us on which side of the edge the interior of the polygon lies. The traversal takes $O(n)$ time. The resulting cone contains all directions of valid translations. Thus we have:

Theorem 2. *Given convex polyhedra A and B with in total n vertices, all directions in which A can be translated without collisions occuring between A and B, can be computed in $O(n)$ time.*

5. A Polyhedron amidst a Set of Obstacles

Given a convex polyhedron A, the *object*, and a set of convex polyhedral *obstacles* B_i, $i = 1, 2, \ldots, m-1$, we determine all directions in which A can be translated to infinity without collisions occuring between A and any of the obstacles. For simplicity, we assume that each of the m polyhedra has p vertices.

A can be translated in a direction d if and only if d does not belong to any cone $C_{B_i}(A)$, $(i = 1, \ldots, m-1)$ (see Section 4). We can thus compute all invalid directions by constructing the cones $C_{B_i}(A)$ for each $i = 1, \ldots, m-1$, and subsequently computing their union $\bigcup_{i=1}^{m-1} C_{B_i}(A)$. The desired valid directions are obtained, as in Section 4, by traversing the edges of the spherical polygon that correspond to the union.

The union of a set of m planar p-gons is easy to compute by a plane-sweep algorithm in $O((mp+k)\log(mp))$ time where k is the intersection of the sides of the polygons (see [10]).

Our aim is to modify the plane-sweep algorithm [10] in such a way that it will compute the union of a set of spherical polygons. We call this modified technique *"sphere-sweep"*.

The edges of the polygons are parts of great circles on the unit sphere. The intersection of two edges can be computed in constant time. In constant time, one can also determine on which side of a great circle a given point of the sphere lies.

Our "sweep line", the *sweep arc* is a half circle whose endpoints are fixed in the "poles" of the sphere, i.e., the intersection points of the x_2-axis and the sphere. The sweep begins at the 0-meridian and advances 360° around the sphere.

Initially, the 0-meridian is tested for intersection with all edges of the polygons. The intersecting edges are stored in a balanced search tree according to the latitude values of the intersection points. The vertices of the polygons are sorted by their longitudes. They are stored in that order in a heap. The heap is the initial list of the halting points of the sweep. It is later completed by inserting intersection points of the edges of the polygons.

After these initializations, two data structures are used and the contour edges of the union are output as in [10]. The x-structure of [10] corresponds to our "latitude structure". The y-structure of [10] corresponds to our "longitude structure".

The angle between an edge and the sweep arc is, in general, not constant. But it does not have considerable effect to our algorithm because only the order of the intersections of the edges and the sweep arc is important. The order can change only at the intersections of the edges. However, we need the following observation:

Lemma 3. *The sweep arc encounters an edge always at one of its endpoints.*

Proof. The lemma follows from the fact that the sweep arc as well as the edges are parts of great circles. □

The sweep takes $O((mp+k)\log(mp))$ time where m is the number of the spherical p-gons and k is the number of intersections of their edges.

The cones $C_{B_i}(A)$, $i = 1, \ldots, m-1$ and their corresponding polygons on the unit sphere can be computed in $O(mp)$ time (see Section 4). Their union can be computed in $O((mp+k)\log(mp))$ time where k is the number of the intersections the edges of polygons. The last step, the traversal along the contour of the union, takes $O(mp+k)$ time. Thus we have:

Theorem 3. *Given a polyhedron A and a set of polyhedral obstacles, all directions in which A can be translated to infinity without collisions occuring between A and the obstacles, can be computed in $O((mp+k)\log(mp))$ time (m is the number of given p-vertex polyhedra, and k is the number of lines along which the faces of the translation cones of A with respect to the obstacles intersected).*

The time bound of the algorithm cannot be improved by the method of [4] although the spherical polygons can be considered as Minkowski differences of convex polygons. The result of [4] requires that A is the same polygon in all differences $B_i - A$. This does not hold in our case because the intersections of A and its extreme planes depend on the obstacles.

The output of the algorithm can be stored in a data structure developed for point location in a straight line planar subdivision (see, e.g., [11]). The "lines" of the subdivision are parts of great circles. Thus, for a given direction d, we can answer efficiently queries whether A can be translated in direction d without collisions occuring between it and the obstacles. The query time depends on the chosen data structure. Without difficulty we can use, for example, the data structure of Kirkpatrick [5] which is based on a triangulation technique. Then the query time is $O(\log(mp))$. The structure needs $O(mp+k)$ space.

6. Conclusions

We have shown that several 3-dimensional motion planning problems can be solved more efficiently if the standard technique of computing the "grown" obstacles, i.e., the Minkowski differences of the obstacles and the object, is not used. Our results apply only to separation motions by means of one translation. They can be considered as the first step in solving 3-dimensional problems in which several translations or rotations are allowed. Some related results already exist, for example, for the problem of computing shortest paths amidst polyhedral obstacles (see, e.g., [15]). There exist efficient solutions to the planar variants of several such problems (see, e.g., [2,17,18]). Most of them cannot be directly generalized to 3-dimensional space without an unacceptable loss of efficiency.

The representation of the output of the algorithms as polygonal regions on the unit sphere gave us the possibility to apply conventional techiques of planar algorithms.

We have used several of the few algorithmic tools for 3-dimensional problems, including the convex hull algorithm of [11] and the algorithm of [9] for 3-variable linear programming. An important and difficult problem is to develop more such tools for 3-dimensional space.

Acknowledgments. Both authors have discussed the topics of the paper with Rolf Klein and the second author with Emo Welzl. The authors are grateful for these discussions.

References

1. G. Davis, Computing separating planes for a pair of disjoint polytopes. *Proc. 1st ACM Symp. Computational Geometry*, 1985, 8–14.

2. F. Dehne, and J.-R. Sack, Translation separability of sets of polygons. *Visual Computer* 3 (1987), 227–235.

3. L. Guibas, L. Ramshaw, and J. Stolfi, A kinetic framework for computational geometry. *Proc. 24th IEEE Symp. Foundations of Computer Science* 1983, 100–111.

4. K. Kedem, and M. Sharir, An efficient algorithm for planning collision-free translational motion of a convex polygonal object in 2-dimensional space amidst polygonal obstacles. *Proc. 1st ACM Symp. Computational Geometry*, 1985, 75–80.

5. D. G. Kirkpatrick, Optimal search in planar subdivisions. *SIAM J. Comput.* **12** (1983), 28–35.

6. T. Lozano-Perez, and M. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* **22** (1979), 560–570.

7. S. Maddila, and C.K. Yap, Moving a polygon around the corner in a corridor. *Proc. 2nd ACM Symp. Computational Geometry*, 1986, 187–192.

8. M. McKenna, and R. Seidel, Finding the optimal shadows of a convex polytope. *Proc. 1st ACM Symp. Computational Geometry*, 1985, 24–28.

9. N. Megiddo, Linear time algorithm for linear programming in R^3 and related problems. *SIAM J. Comput.* **12** (1983), 759–776.

10. Th. Ottmann, P. Widmayer, and D. Wood, A fast algorithm for boolean masking problem. *Comput. Vision Graphics Image Process.* **30** (1985), 249–268.

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

12. F. P. Preparata, and M. I. Shamos, *Computational Geometry. An Introduction*. Springer-Verlag, New York etc., 1985.

13. J. T. Schwartz, and M. Sharir, On the piano movers' problem: V. The case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Comm. Pure Appl. Math.* Vol. XXXVII (1984), 815–848.

14. H. Seeland, CAD-Abteilung, Daimler Benz AG, Stuttgart, Personal communication, 1987.

15. M. Sharir, and A. Baltsan, On shortest path amidst convex polyhedra. *Proc. 2nd ACM Symp. Computational Geometry*, 1986, 193–206.

16. M. Sharir, R. Cole, K. Kadem, D. Leven, R. Pollack, Geometric appications of Davenport-Schinzel sequences. *Proc. 27th IEEE Symp. Foundations of Computer Science*, 1986, 77–86.

17. G. T. Toussaint, Movable separability of sets. *In:* G. T. Toussaint (ed.), *Computational Geometry*, North-Holland, Amsterdam etc., 1985, 335–376.

18. S. H. Whitesides, Computational geometry and spatial planning. *In:* G. T. Toussaint (ed.), *Computational Geometry*, North-Holland, Amsterdam etc., 1985, 377–428.

19. C.K. Yap, How to move a chair through a door. Techical Report, Courant Institute, 1984.