Chapter 1
Efficient Generation of k-Directional Assembly Sequences[*]

Pankaj K. Agarwal[†] Mark de Berg[‡] Dan Halperin[§] Micha Sharir[¶]

Abstract

Let S be a collection of n rigid bodies in 3-space, and let D be a set of k directions in 3-space, where k is a small constant. A k-directional assembly sequence for S, with respect to D, is a linear ordering $\langle s_1, \ldots, s_n \rangle$ of the bodies in S, such that each s_i can be moved to infinity by translating it in one of the directions of D and without intersecting any s_j, for $j > i$. We present an algorithm that computes a k-directional assembly sequence, or decides that no such sequence exists, for a set of polyhedra. The algorithm runs in $O(km^{4/3+\varepsilon})$ time, where m is the total number of vertices of the polyhedra. We also give an algorithm for 'k-directional' rotational motions.

1 Introduction.

Let S be a set of objects in 3-space. A *depth order* for S in a given direction $\vec{d}$ is a linear ordering $\langle s_1, s_2, \ldots, s_n \rangle$ of the objects in S such that each s_i can be translated to infinity in direction $\vec{d}$ without colliding with any of the objects s_j, for $j > i$. The study of depth orders was mainly motivated by computer graphics, where the existence of a depth order, with respect to the viewing direction, of the objects in a 3D scene gives a simple hidden surface removal procedure, the so-called *painter's algorithm*: the objects are drawn one after the other on top of each other, in the reverse depth order. Notice that a depth order does not always exist; even in a set of three triangles there can be cyclic overlap. De Berg et al. [6] gave an $O(m^{4/3+\varepsilon})$-time algorithm, for any $\varepsilon > 0$, that computes a depth order in a given direction (or decides that no such order exists) for a set of polygons in 3-space with m vertices in total. Faster algorithms are known for several special cases [1, 6].

The interest in depth orders is not limited to computer graphics. Suppose we want to manufacture a product consisting of several parts. If a depth order for the set of parts in a direction $\vec{d}$ exists, then the product can be assembled by translating the constituent parts one after another to their target positions along $\vec{d}$. Thus one robot arm that can pick up the parts and move them in one fixed direction—the vertical direction for instance—suffices to do the assembly. (We ignore here and in the rest of the paper the issue of how the parts are grasped and manipulated, focusing on the inherent assemblability of the product.) Wilson and Latombe [18] call products that can be assembled in this manner *stack products*. The simplicity of the assembly process makes stack products attractive to manufacture. See the collection edited by Homem de Mello and Lee [11] for a survey of problems and techniques in computer-aided mechanical assembly planning.

Many products, however, are not stack products, that is, a single direction in which the parts must be moved is not sufficient to assemble the product. One solution, proposed by Wilson and Latombe [17, 18], is to search for an assembly sequence that allows to move a subcollection of parts as a rigid body in *any* direction. Although their solution has polynomial running time (in contrast with many techniques in mechanical assembly planning), its running time is rather high in the worst case: it may require $\Omega(m^4)$ time in the worst case for a collection of m tetrahedra in 3-space. Dehne and Sack [8] study a similar problem in the plane: They allow motions in arbitrary directions, but only one part can be moved at a time. For a set of m constant-complexity polygons, their algorithm runs in

[*]Work on this paper by Pankaj Agarwal has been supported by NSF Grant CCR-93-01259, an NYI award, and matching funds from Xerox Corporation. Work on this paper by Mark de Berg has been supported by the Netherlands' Organization for Scientific Research (NWO) and partially by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*). Work on this paper by Dan Halperin has been supported 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.

[†]Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA.

[‡]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

[§]Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305, USA.

[¶]School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA.

$O(m^2 \log m)$ time. See Toussaint's survey [16] for an extensive (albeit ten-year-old) overview of assembly problems in the plane.

We propose the following alternative approach. We are looking for a sequence that allows for assembling the product by moving the parts one after the other, along a small number of prescribed directions. This is attractive from a practical point of view, where some directions of motion are preferable to others. The products that can be assembled in this manner can be considered a simple generalization of stack products. The problem of finding such an assembly sequence, which we call the *k-directional assembly sequence problem*, can be stated formally as follows: Given a collection $S = \{s_1, s_2, \ldots, s_n\}$ of n rigid bodies in 3-space, and a set $D = \{\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_k\}$ of k directions in 3-space, where k is a small constant, find a linear ordering of the bodies in S of the form $\langle (s_{\pi_1}, \vec{d}_{i_1}), (s_{\pi_2}, \vec{d}_{i_2}), \ldots, (s_{\pi_n}, \vec{d}_{i_n}) \rangle$, where $\langle s_{\pi_1}, \ldots, s_{\pi_n} \rangle$ is a permutation of the objects in S, and, for each j, the direction $\vec{d}_{i_j}$ is in D, and the object s_{π_j} can be moved to infinity by translating it in the direction $\vec{d}_{i_j}$ so that its interior does not intersect any of the objects s_{π_l}, for $l > j$, throughout the motion. Otherwise, determine that no such ordering is possible.

Note that we allow an object to slide along another object during its motion. We call such a linear sequence a *k-directional assembly sequence* of S relative to D, and denote it by $\mathcal{AS}(S, D)$. The case $k = 1$ is the well-known depth order problem discussed at the beginning of this introduction, and can be solved using the algorithm by de Berg et al. [6]. Their approach, however, does not work for $k > 1$.

The above order actually induces a *dis*assembly sequence, that is, a sequence of translations that will take the product apart. The desired assembly sequence is simply the disassembly sequence in reverse order. This approach is common practice in mechanical assembly planning when the objects are rigid. For this reason, problems of finding an assembly sequence are sometimes referred to as *separability problems.*

Our approach is more restrictive than that of Wilson and Latombe in two ways: we prescribe a small number of directions of allowed translations, and we only allow one part to be moved at a time. There are products that cannot be assembled in our model, but that can be assembled in their model. Nevertheless, the assembly process resulting from our approach is relatively simple, which is advantageous in practice. Moreover, our algorithm for the k-directional assembly sequence problem runs in $O(m^{4/3+\varepsilon})$ time, for any $\varepsilon > 0$, for a set of polyhedra with m vertices in total. (The constant of proportionality in the bound depends on k and ε.) Thus it is significantly faster than their algorithm.

Our approach works incrementally: For each direction $\vec{d}_i$ we maintain the subset of the objects that are maximal in $\vec{d}_i$; an object is called *maximal* in $\vec{d}_i$ if it can be moved to infinity in direction $\vec{d}_i$ without collisions. We then take one of the maximal objects, separate it from the remaining objects, and update the set of maximal objects. This process is repeated until all objects are separated, in which case we have found the desired k-directional assembly sequence, or until we are stuck, in which case no k-directional assembly sequence exists. The heart of our solution is a new data structure for maintaining maximal objects under deletions. We present efficient maximality-maintenance structures for a set of segments and for a set of polyhedra in $\mathbb{R}^3$, which, in turn, rely on a dynamic data structure for the following problem, which we refer to as the *maximality-reporting* problem. Given two sets A, B of objects in $\mathbb{R}^d$, preprocess them into a data structure that supports efficient deletions from A and B; after each deletion of an object b from B the deletion procedure should return the objects of A that become maximal (in the $+x_d$-direction) after deleting b. We first present a maximality-reporting structure for the case when A is a set of points in $\mathbb{R}^d$ and B is a set of hyperplanes in $\mathbb{R}^d$. Using this data structure as a substructure, we present a maximality-reporting structure for segments and for polyhedra.

We extend our algorithm to rotational motions, where the $\vec{d}_i$'s specify axes of rotation for the motions. For this case, the running time of our algorithm becomes $O(m^{8/5+\varepsilon})$, for any $\varepsilon > 0$. Prior to our work, most of the study of separability problems deals with translational separation; rotations are often handled by resorting to infinitesimal rotations [10, 19]. García-López and Ramos-Alonzo [9] study true rotational motions, but in a very limited setting: They consider separating a single point from a polygon. Another recent paper dealing with rotations is by Schömer and Thiel [14]. Given a stationary polyhedron and a rotating polyhedron, they can find the first collision point in $O(m^{5/3+\varepsilon})$ time, for any $\varepsilon > 0$, where m is the total number of vertices of the polyhedra. This has recently been improved to $O(m^{8/5+\varepsilon})$ [15]. In our solution to the rotational ordering, we use the data structure devised in these papers.

The paper is organized as follows. In Section 2 we present the global approach for computing a k-directional assembly sequence. Section 3 presents the maximality-reporting structure for points and hyperplanes. Since this data structure is somewhat involved, we first give the basic idea and then describe the data structure in detail. Next, we describe in Section 4 a maximality-maintenance structure for a set of segments in $\mathbb{R}^3$, and in Section 5 for a set of polyhedra. We consider rotational motions in Section 6.

2 The Global Method.

Let S be an arbitrary collection of rigid bodies in $\mathbb{R}^3$, and let $D = \{\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_k\}$ be a set of k directions. We describe a general method to compute a k-directional assembly sequence for S with respect to D. The method works by maintaining the so-called maximal elements in the set S. Given a pair s, s' of objects, we say that s lies *above* s' in direction $\vec{d}$ if, for every oriented line ℓ in the direction $\vec{d}$, all points of $s' \cap \ell$ lie before any point of $s \cap \ell$. An object s is called *maximal* relative to a set S in direction $\vec{d}$ if s lies above all objects of $S \setminus \{s\}$ in direction $\vec{d}$.

Our algorithm for computing a k-directional assembly sequence uses the following "greedy" approach. Suppose we have already computed the sequence

$$\langle (s_{\pi_1}, \vec{d}_{i_1}), (s_{\pi_2}, \vec{d}_{i_2}), \ldots, (s_{\pi_{j-1}}, \vec{d}_{i_{j-1}}) \rangle.$$

In the j-th step, we either compute a pair $(s_{\pi_j}, \vec{d}_{i_j})$ such that s_{π_j} is maximal relative to $S^j = S \setminus \{s_{\pi_1}, \ldots, s_{\pi_{j-1}}\}$ in direction $\vec{d}_{i_j} \in D$, or conclude that no such pair exists, in which case we stop and report that S does not admit a k-directional assembly sequence relative to D. It is easy to see that this greedy approach is correct. In order to apply this approach efficiently, we maintain for each direction $\vec{d}_i \in D$ a data structure $T_{\vec{d}_i}$ that supports efficient execution of the following operations on a set S of objects:

max: Return an object of S that is maximal in the direction $\vec{d}_i$, if there exists one. Otherwise, return NULL.

delete(s): Delete the object s from S, and update the data structure accordingly.

We call such a data structure a *maximality-maintenance structure.* At the start of the j-th step each $T_{\vec{d}_i}$ stores the set S^j. Searching for a maximal element can now be done by performing a **max**-query on each $T_{\vec{d}_i}$; updating the structures for the $(j+1)$-st step (if there was a maximal element) can be done by deleting the maximal element from each $T_{\vec{d}_i}$.

Let $f(n)$ denote the maximum time required to build the maximality-maintenance structure $T_{\vec{d}_i}(S)$ and to perform n operations (either **max** or **delete**) on it, over all possible sets S of n objects of a fixed type and over all possible directions in 3-space. Then a k-directional assembly sequence for a set S of n objects with respect to a set D of k directions can be computed in time $O(kf(n))$. We have thus reduced the problem of computing a k-directional assembly sequence to that of designing a maximality-maintenance structure.

3 A Maximality-Reporting Structure.

Before we present maximality-maintenance structures for segments and polyhedra, we present a maximality-reporting structure, defined below, for points and hyperplanes, which will be used as a substructure for maximality-maintenance structures. Let P be a set of m points in $\mathbb{R}^d$ and H a set of n hyperplanes in $\mathbb{R}^d$. A point is called *maximal* if it lies above all hyperplanes of H (with respect to the x_d-axis). A *maximality-reporting structure* for P and H, denoted by $\Psi(P, H)$, supports deletions of elements from P and H, where the deletion procedure for a hyperplane $h \in H$ reports and deletes all points in P that become maximal upon the deletion of h. Furthermore, besides constructing the data structure, the preprocessing procedure should also report and delete the points of P that are maximal with respect to H. Since we delete the points whenever we report them, a point is reported only once.

We present the data structure in three stages. In Section 3.1 we present the main idea and quote results that will be needed later. In Section 3.2 we describe a data structure that works well when $m > n^{\lfloor d/2 \rfloor}$, and then in Section 3.3 we present another structure that works well for all values of m and n.

3.1 Overview and auxiliary results.

We first briefly sketch the main idea. We preprocess H into a dynamic data structure, proposed by Agarwal and Matoušek [3], for the *upper-envelope* searching problem, which supports deletions of hyperplanes and queries that can quickly determine whether a query point lies above all the hyperplanes of H. This data structure is a tree T, where each node v stores a subset H^v of H along with some auxiliary information.

For a query point p, the query procedure visits a set of nodes of the tree, and, at each visited node v, it checks whether p lies above all hyperplanes in H^v. A naive approach for reporting and deleting all points of P that become maximal upon deleting a hyperplane of H is to query T with every point of P that has not been deleted so far. It will, however, be expensive to repeat this step after every deletion. However, since the data structure supports efficient deletion of hyperplane, the procedure for deleting a hyperplane h visits only a few nodes of T. The deletion of h can affect the outcome of only those queries that visit one of the nodes visited by the deletion procedure while deleting h.

To exploit this observation, we query T with each point $p \in P$ only once, immediately after constructing T. For each node $v \in T$ visited by the query procedure for p, we store p at v if it is not maximal with respect to H^v. Let P^v be the set of points stored at v. When we delete a hyperplane h, for each node v visited by the deletion procedure, we also delete the points of P^v that become maximal with respect to H^v. We report the points that were deleted from the root. We will show that, by proceeding in a bottom-up manner, the points can be deleted efficiently (in an amortized sense). This is the general idea; the data structure is described in detail in the following subsections.

We now state some results that will be useful for our data structure. The *arrangement* of H, $\mathcal{A}(H)$, is the decomposition of $\mathbb{R}^d$ induced by the hyperplanes in H. The *level* of a point p with respect to H is the number of hyperplanes of H lying strictly above p. Let $\mathcal{A}_{\leq k}(H)$ denote the (closure of the) set of all points whose level with respect to H is at most k. For a parameter $r \leq n$, a $(1/r)$*-cutting* of $\mathcal{A}_{\leq k}(H)$ is a set Ξ of simplices with pairwise disjoint interiors covering $\mathcal{A}_{\leq k}(H)$, so that the interior of each simplex intersects at most $|H|/r$ hyperplanes of H. The number of simplices in Ξ is called the *size* of the cutting. Finally, for a set P of m points in $\mathbb{R}^d$, a family $\Pi = \{(P_1, \Delta_1), (P_2, \Delta_2), \ldots, (P_l, \Delta_l)\}$ is called a *simplicial partition* of size l for P if $P_1, \ldots, P_l$ form a partition of P and each Δ_i is a (relatively open) simplex containing P_i.

Theorem 3.1. (Matoušek [12]) *Let H be a set of n hyperplanes and P a set of m points in $\mathbb{R}^d$. Let $r \leq n^\alpha$ be a parameter, where $0 \leqslant \alpha \leqslant 1$ is a suitable constant (depending on d).*

(i) *For any $k = O(n/r)$, a $(1/r)$-cutting Ξ of $\mathcal{A}_{\leq k}(H)$ of size $C_1 r^{\lfloor d/2 \rfloor}$ can be computed in $O(n \log r)$ time; here C_1 is an absolute constant.*

(ii) *There exists a simplicial partition Π of size l for P, with $l \leqslant r$, such that, for $1 \leq i \leq l$,*

   (ii.a) *$m/r \leq |P_i| \leq 2m/r$; and*

   (ii.b) *each hyperplane of H that lies below at most m/r points of P intersects $\kappa(r)$ simplices of $\{\Delta_1, \ldots, \Delta_l\}$, where $\kappa(r)$ is $O(\log r)$ for $d = 2, 3$, and $O(r^{1-1/\lfloor d/2 \rfloor})$ for $d \geq 4$.*

   Moreover, Π can be computed in $O(m \log r + nr)$ time.

Let Δ be a simplex. We say that a hyperplane h is *relevant* for Δ if either h lies above Δ or h intersects Δ. Let $H_\Delta \subseteq H$ denote the set of hyperplanes relevant for Δ. For any $(1/r)$-cutting Ξ of $\mathcal{A}_{\leq k}(H)$, we can assume that every simplex $\Delta \in \Xi$ has at most $k + n/r$ relevant hyperplanes, because otherwise Δ does not intersect $\mathcal{A}_{\leq k}(H)$ and can be discarded from Ξ.

3.2 The first data structure. We can now describe the desired maximality-reporting structure. Let $m = |P|$ and $n = |H|$. We first describe a data structure that works well when $m > n^{\lfloor d/2 \rfloor}$, and in the following subsection present another structure that works well for all values of m and n.

The first maximality-reporting structure $\Psi(P, H)$, a variant of the dynamic halfspace-range-searching structure by Agarwal and Matoušek [3], is a tree based on $(1/r)$-cuttings. It can be constructed in $O(n^{\lfloor d/2 \rfloor + \varepsilon} + m \log n)$ time and can delete all points and hyperplanes in time $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$.

Each node v of Ψ is associated with a pair $P^v \subseteq P$ and $H^v \subseteq H$. The root u is associated with P and H. We will use Ψ^v to denote the subtree rooted at v. Ψ^v is reconstructed periodically after deleting some hyperplanes of H^v. Let m_v (resp. n_v) denote the size of P^v (resp. H^v) when Ψ^v was reconstructed the last time; put $m = m_u$ and $n = n_u$. We set $r = n^\delta$, for some sufficiently small constant $\delta > 0$. The value of r is the same for all nodes of Ψ, and it is updated only when the entire tree is reconstructed. We reconstruct the subtree Ψ^v after deleting $n_v/2r$ hyperplanes from H^v since the last time it was reconstructed.

If $n \leq r$ then $\Psi(P, H)$ consists of a single node. Otherwise, the root of $\Psi(P, H)$ is an internal node and, apart from P and H, it stores the following information:

(i) A partition of H into subsets $H_1, \ldots, H_t$, where $t \leq \lceil 1/\delta^2 \rceil$.

(ii) For every $1 \leq i \leq t$, we store a $(1/r)$-cutting Ξ_i. Initially, Ξ_i is a $(1/r)$-cutting of $\mathcal{A}_{\leq k}\left(\bigcup_{j \geq i} H_j\right)$, where $k = n/r$. Each hyperplane of H_i is relevant for at most

$$\mu = 2C_1 r^{\lfloor d/2 \rfloor - 1 + \delta} \tag{3.1}$$

simplices of Ξ_i, where C_1 is the constant arising in Theorem 3.1(i).

(iii) For each point $p \in P$, we store a bit *shallow*(p), which is 1 if, for every $i \leq t$, p lies in some simplex of Ξ_i, and 0 otherwise. If *shallow*$(p) = 0$, then we ignore p from this version of the structure because it cannot become maximal during the next $n/2r$ deletions.

(iv) For every i and for every $\Delta \in \Xi_i$, we store a pointer to a subtree $\Psi(P_{i,\Delta}, H_{i,\Delta})$, where

$$P_{i,\Delta} = \{p \in P \mid p \in \Delta \text{ and } \textit{shallow}(p) = 1\},$$

and $H_{i,\Delta} \subseteq H_i$ is the set of hyperplanes relevant for Δ. By construction, $|H_{i,\Delta}| \leq 2n/r$ and, for every $1 \leq i \leq t$, $\sum_{\Delta \in \Xi_i} |P_{i,\Delta}| \leq m$.

(v) A counter *count*$_u$, which is initially set to $n/2r$.

We first describe how to compute Ξ_i and H_i for $1 \leq i \leq t$. Suppose we have already computed $H_1, \ldots, H_{i-1}$. Let $\overline{H}_i = H \setminus (H_1 \cup \cdots \cup H_{i-1})$, and let $n_i = |\overline{H}_i|$. If $n_i \leq n/r$, then Ξ_i consists of just one sufficiently large simplex, and $H_i = \overline{H}_i$. Otherwise, set

$$r_i = r\frac{n_i}{n} \leqslant r \quad \text{and} \quad k = \frac{n}{r} = \frac{n_i}{r_i},$$

and compute a $(1/r_i)$-cutting Ξ_i of size $C_1 r_i^{\lfloor d/2 \rfloor}$ for $\mathcal{A}_{\leq k}(\overline{H}_i)$, in time $O(n_i \log r_i)$, using Theorem 3.1. For every simplex $\Delta \in \Xi_i$, let $\overline{H}_{i,\Delta} \subseteq \overline{H}_i$ be the set of hyperplanes relevant for Δ. We have

$$|\overline{H}_{i,\Delta}| \leq \frac{n_i}{r_i} + k = \frac{2n_i}{r_i}\,.$$

Therefore

$$(3.2)\quad \sum_{\Delta \in \Xi_i} |\overline{H}_{i,\Delta}| \leq \frac{2n_i}{r_i} C_1 r_i^{\lfloor d/2 \rfloor} = 2C_1 n_i r_i^{\lfloor d/2 \rfloor - 1}\,.$$

We call a hyperplane 'good' if it is relevant for at most μ simplices (see Equation (3.1)). Then the number of 'bad' hyperplanes is at most $n_i/r^\delta = n_i/n^{\delta^2}$. Let H_i be the set of good hyperplanes of $\overline{H}_i$. We now repeat the construction for $\overline{H}_{i+1} = \overline{H}_i \setminus H_i$. The above analysis implies that $|\overline{H}_i| \leq n^{1-i\delta^2}$, so the process terminates after at most $\lceil 1/\delta^2 \rceil$ steps. Next, by testing each point p of P with every simplex of Ξ_i, for $1 \leq i \leq t$, we determine *shallow*(p) and the set $P_{i,\Delta}$ for every $\Delta \in \Xi_i$. If $P_{i,\Delta}$ is not empty, we recursively construct the data structure $\Psi(P_{i,\Delta}, H_{i,\Delta})$ and store a pointer at the root to this subtree.

Here is a brief informal description of how this data structure functions. The cuttings Ξ_i cover the 'shallow' portion of $\mathcal{A}(H)$. If a point $p \in P$ lies below such a cutting Ξ_i, then there are at least $n_i/r_i = n/r$ hyperplanes of H passing above p, so p can be ignored in the current version of the tree, because at most $n/2r$ hyperplanes will be deleted before the tree is reconstructed, so p will remain nonmaximal while these deletions take place. This is the purpose of the bit *shallow*(p), and the reason for recursive processing of only points with *shallow*$(p) = 1$. The reason for maintaining t different cuttings is that we want all hyperplanes to be 'good' with respect to the relevant cutting, so that deleting a hyperplane can be done efficiently, by traversing and updating at most μ simplices.

The total time spent in computing the information stored at the root of $\Psi(P,H)$ is $O((m+n)tr^{\lfloor d/2 \rfloor})$. Hence, it can be shown that the total time spent in constructing $\Psi(P,H)$ is $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$, for any $\varepsilon > 0$. Note that, by the choice of r, the depth of $\Psi(P,H)$ is constant (depending on δ).

After we have constructed the tree $\Psi(P,H)$, we change the sets P^v stored at each node v so that the following condition is met:

($\star$) No point of P^v is maximal with respect to H^v.

Deleting the points of P^v that are maximal with respect to H^v can be accomplished in time $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$ in a bottom-up manner. Finally, we return all the points deleted at the root of $\Psi(P,H)$.

Deleting a point $p \in P$**.** We visit Ψ in a top-down fashion, starting from the root. Suppose we are at a node v. If $p \notin P^v$, there is nothing to do, and we return. Otherwise, we delete p from P^v. If v is an interior node, we also delete p recursively from its children. Since a point of P is stored at $O(1)$ nodes (with the constant of proportionality depending on δ), the total time spent is $O(1)$.

Deleting a hyperplane $h \in H$**.** We again visit Ψ in a top-down fashion, starting from the root. Suppose we are at a node v. We delete h from the set H^v. If v is a leaf, for each point $p \in P^v$, we determine whether p lies above all the remaining hyperplanes of H^v. If so, we delete p from P^v. We return all the deleted points.

If v is an interior node, we decrease $count_v$ by 1. If $count_v$ becomes zero, we reconstruct the subtree Ψ^v and return those points of P^v that are maximal with respect to H^v (recall that these points are deleted from the set P^v). Next, assume that $count_v > 0$ and $h \in H_i^v$. For each simplex Δ in the $(1/r)$-cutting Ξ_i^v, for which h is relevant, we delete h recursively from the substructure $\Psi(P_{i,\Delta}^v, H_{i,\Delta}^v)$. The recursive call returns the subset of those points of P^v that are now maximal with respect to $H_{i,\Delta}^v$. Using the information returned by the recursive calls, we delete and return all those points of P^v which have become maximal with respect to H^v (after deleting h). Specifically, a point is deleted from P^v if *shallow*$(p) = 1$ and if it has been deleted from all the children of v where it was stored.

Following an argument similar to (but more involved than) the one used by Agarwal and Matoušek in [3], we can prove that the total time spent in deleting all n hyperplanes of H and all m points of P is $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε.

Lemma 3.1. *Let P be a set of m points in $\mathbb{R}^d$ and let H be a set of n hyperplanes in $\mathbb{R}^d$. A maximality-reporting structure for P and H can be constructed in $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$ time, for any $\varepsilon > 0$, so that the deletions of all hyperplanes and all points can be performed in $O((m + n^{\lfloor d/2 \rfloor})n^\varepsilon)$ time.*

3.3 The second data structure.

We now describe another maximality-reporting structure for P and H, which combines Lemma 3.1 with a structure that works well for the case when m is much smaller than n. (Here we follow a standard technique used in range searching to obtain space/query-time tradeoff.) Following the notation of the previous subsection, we will use $\Psi(P,H)$ to denote the new tree, Ψ^v to denote the subtree rooted at the node v, and P^v, H^v to denote the subsets of points and hyperplanes, respectively, associated with v. The root is again associated with P and H. The subtree rooted at v is reconstructed after deleting $|H^v|/2r$ hyperplanes from H^v.

We now choose r to be a constant. $\Psi(P,H)$ consists of a single node for $n \le r$, and $\Psi(P,H)$ is constructed using Lemma 3.1 for $m > n^{\lfloor d/2 \rfloor}$. So, assume that $m \le n^{\lfloor d/2 \rfloor}$ and $n \ge r$. The data structure is constructed recursively as follows.

Let $\tilde{p}$ be the hyperplane dual to a point p in $\mathbb{R}^d$ and let $\tilde{h}$ be the point dual to a hyperplane h in $\mathbb{R}^d$, so that p lies above h if and only if $\tilde{h}$ lies below $\tilde{p}$. Let $\tilde{P} = \{\tilde{p} \mid p \in P\}$ and $\tilde{H} = \{\tilde{h} \mid h \in H\}$. A point of p is maximal if the halfspace lying above $\tilde{p}$ does not contain any point of $\tilde{H}$. We construct a simplicial partition $\Pi = \{(\tilde{H}_1, \Delta_1), (\tilde{H}_2, \Delta_2), \ldots, (\tilde{H}_l, \Delta_l)\}$ of $\tilde{H}$ in $O(n)$ time, using Theorem 3.1. The root of $\Psi(P,H)$ stores the following information. For each $p \in P$, we store a bit *shallow*(p), which is 1 if the hyperplane $\tilde{p}$ intersects at most $\kappa(r)$ simplices of Π. For each $1 \le i \le l$, let

$$P_i = \{p \in P \mid \mathit{shallow}(p) = 1 \text{ and } \tilde{p} \text{ intersects } \Delta_i\}.$$

We recursively construct $\Psi(P_i, H_i)$ and attach a pointer from the root of $\Psi(P,H)$ to the data structure $\Psi(P_i, H_i)$.

It can be shown that the total time spent in constructing $\Psi(P,H)$ is $\Phi(m,n) = O(m^\gamma n^{\gamma+\varepsilon} + mn^\varepsilon + n \log n)$, where

$$\gamma = \lfloor d/2 \rfloor / (\lfloor d/2 \rfloor + 1). \tag{3.3}$$

Again, we want to satisfy condition $(\star)$, so we delete all points of P^v that are maximal with respect to H^v in a bottom-up manner. Finally, points of P and hyperplanes of H are deleted using appropriate variants of the procedures described in the previous subsection. Once again, with analysis similar to the one in [3], we can prove that the total time spent in deleting all n hyperplanes of H and all points of P is $O(m^\gamma n^{\gamma+\varepsilon} + mn^\varepsilon + n \log^2 n)$ for any $\varepsilon > 0$, with γ being the same as in (3.3).

THEOREM 3.2. *Let P be a set of m points in $\mathbb{R}^d$ and let H be a set of n hyperplanes in $\mathbb{R}^d$. A maximality-reporting structure for (P,H) can be constructed in time $O(m^\gamma n^{\gamma+\varepsilon} + mn^\varepsilon + n \log n)$, for any $\varepsilon > 0$, such that the deletions of all points and hyperplanes can be performed in $O(m^\gamma n^{\gamma+\varepsilon} + mn^\varepsilon + n \log^2 n)$ time.*

4 A Maximality-Maintenance Structure for Segments.

We now present a maximality-maintenance structure for a set S of segments in $\mathbb{R}^3$. The time to build this structure and to perform n **max** and **delete** operations on it will be $f(n) = O(n^{4/3+\varepsilon})$, for any $\varepsilon > 0$, see Lemma 4.1. This implies the following theorem.

THEOREM 4.1. *Computing a k-directional assembly sequence for a set of n segments in 3-space relative to a given set of k directions (or determining that no such order exists) can be done in $O(kn^{4/3+\varepsilon})$ time, for any $\varepsilon > 0$.*

Without loss of generality we describe a maximality-maintenance structure for the $+z$-direction. It turns out to be simpler not to tackle this data structuring problem directly, but to first apply the following decomposition step. For a segment $s \in \mathbb{R}^3$, let s^* denote the xy-projection of s, and for a set S define $S^* = \{s^* \mid s \in S\}$. Our goal is to construct a family

$$\mathcal{F} = \{(A_1, B_1), (A_2, B_2), \ldots, (A_t, B_t)\},$$

of pairs of sets that satisfy the following conditions for some given $\varepsilon > 0$ and a sufficiently large constant integer r. Let $a_i = |A_i|$ and $b_i = |B_i|$.

(C1) For $1 \le i \le t$, $A_i, B_i \subseteq S$, $A_i \cap B_i = \emptyset$; each segment of A_i^* intersects every segment of B_i^*. Also, the slope of every segment in A_i^* is smaller than that of any segment in B_i^*, or the slope of every segment in A_i^* is larger than that of any segment in B_i^*.

(C2) For every pair of distinct segments $s_i, s_j \in S$ such that s_i^*, s_j^* intersect, there are two indices $k, l \le t$ with $s_i \in A_k, s_j \in B_k$ and $s_j \in A_l, s_i \in B_l$.

(C3) For $1 \le k \le \lceil \log_r n \rceil$, let $\mathcal{F}^{(k)} = \{(A_i, B_i) \mid r^k \le b_i < r^{k+1}\}$ and $t_k = |\mathcal{F}^{(k)}|$. Then

$$t_k = O\left(\frac{n^{4/3+\varepsilon}}{r^{2k}}\right), \sum_{(A_i,B_i)\in\mathcal{F}^{(k)}} a_i = O(n^{4/3+\varepsilon}) \quad \text{for} 1 \le k \le \left\lceil \tfrac{1}{3} \log_r n \right\rceil,$$

$$t_k = O\left(\frac{n^{1+\varepsilon}}{r^{k}}\right), \sum_{(A_i,B_i)\in\mathcal{F}^{(k)}} a_i = O\left(\frac{n^{3/2+\varepsilon}}{r^{k/2}}\right) \quad \text{for } \left\lceil \tfrac{1}{3} \log_r n \right\rceil < k \le \lceil \log_r n \rceil.$$

In the next subsection we show that, using the known range-searching data structures [4, 13], the family $\mathcal{F}$ can be computed in $O(n^{4/3+\varepsilon})$ time. We now explain how $\mathcal{F}$ can be used to construct the desired data structure.

For each of the pairs (A_i, B_i), we construct a maximality-reporting structure $\Psi(A_i, B_i)$, as described below in Section 4.2. More precisely, $\Psi(A_i, B_i)$ maintains the segments from A_i that are maximal with respect to B_i. It allows for the deletion of segments from A_i and from B_i. In the latter case, all segments from A_i that become maximal upon the deletion of the segment from B_i are reported. Moreover, the preprocessing algorithm for $\Psi(A_i, B_i)$ should report all segments from the set A_i that are initially maximal with respect to B_i.

Besides the data structures $\Psi(A_i, B_i)$, we maintain the following information. For every segment $s \in S$, we maintain three lists of indices:

$$A(s) = \{i \mid s \in A_i\}, \qquad B(s) = \{i \mid s \in B_i\},$$
$$NM(s) = \{i \mid s \in A_i \wedge s \text{ is not maximal w.r.t. to } B_i\}.$$

Finally, we maintain the set

$$\mathcal{M} = \{\, s \in S \mid NM(s) = \emptyset\}$$

of maximal segments in S. It is easy to verify that $\mathcal{M}$ is indeed the set of all maximal segments: If s is not maximal, because it is 'blocked' by some segment s', then there is an index i such that $s \in A_i$, $s' \in B_i$, so the non-maximality of s will be 'captured' by the pair (A_i, B_i). All these lists will be updated as segments are being deleted from S.

Initializing the lists $A(s)$ and $B(s)$ is trivial once we have computed $\mathcal{F}$. Furthermore, the preprocessing algorithm for the structures $\Psi(A_i, B_i)$ gives us all the information to initialize the lists $NM(s)$. Initializing $\mathcal{M}$ is easy once we have the lists $NM(s)$ available. Hence, the time for initializing the lists $A(s)$, $B(s)$, $NM(s)$, and $\mathcal{M}$ is subsumed by the time spent in constructing $\mathcal{F}$ and all structures $\Psi(A_i, B_i)$.

We can now describe how to perform the **max** and **delete** operations. A **max** query is easy to answer: if $\mathcal{M} \neq \emptyset$ we return an arbitrary segment $s \in \mathcal{M}$, otherwise we return NULL. Now consider a **delete**(s) operation. First we delete s from $\mathcal{M}$ (if necessary) and for each $i \in A(s)$ we delete s from $\Psi(A_i, B_i)$. Next, for each $i \in B(s)$ we delete s from $\Psi(A_i, B_i)$. The deletion procedure for $\Psi(A_i, B_i)$ returns all segments $s' \in A_i$ that become maximal with respect to B_i. For each such segment s', we delete i from $NM(s')$ and we add s' to $\mathcal{M}$ if $NM(s')$ becomes empty. It can be verified that conditions (C1)–(C3) are invariant under **max** and **delete** operations.

Let $\varphi(a_i, b_i)$ denote the total time spent in constructing $\Psi(A_i, B_i)$ and in deleting all the segment of $A_i \cup B_i$. Theorem 4.2 below states that

$$\varphi(a_i, b_i) = O(a_i^{2/3} b_i^{2/3+\varepsilon} + a_i^{1+\varepsilon} + b_i^{1+\varepsilon}),$$

for any $\varepsilon > 0$. As already promised, the time needed to compute the family $\mathcal{F}$ will be $O(n^{4/3+\varepsilon})$, for any $\varepsilon > 0$. Hence, summing $\varphi(a_i, b_i)$ over all pairs in $\mathcal{F}$ and using (C3), one can show that the total running time of the algorithm is bounded by $O(n^{4/3+\varepsilon'})$ for any $\varepsilon' > 3\varepsilon$. We thus obtain the following result.

LEMMA 4.1. *Given a collection of n segments in 3-space, a direction $\vec{d}$, and a parameter $\varepsilon > 0$, we can construct a maximality-maintenance structure for direction $\vec{d}$, such that the preprocessing time and the time to perform n operations on it is $O(n^{4/3+\varepsilon})$.*

4.1 Computing the family $\mathcal{F}$.

Let S be a set of n segments in $\mathbb{R}^3$. We describe a divide-and-conquer algorithm to compute a family $\mathcal{F} = \{(A_1, B_1), \dots, (A_t, B_t)\}$ of pairs of subsets of S satisfying the conditions (C1)–(C3) stated in the beginning of Section 4. Recall that s^* denotes the projection of a segment s and that $S^* = \{s^* \mid s \in S\}$. Let ε be fixed. We sort the segments of S by the slopes of S^* and partition S into two subsets S_1 and S_2, where S_1 consists of the first $\lfloor n/2 \rfloor$ segments and S_2 consists of the next $\lceil n/2 \rceil$ segments. We recursively construct families $\mathcal{F}_1$ and $\mathcal{F}_2$ for S_1 and S_2, respectively. We also construct, in $O(n^{4/3+\delta})$ time, for $\delta = \varepsilon/2$, a family

$$\mathcal{F}_{12} = \{(A_1, B_1), \dots, (A_u, B_u)\}$$

that satisfies the following conditions.

(C1′) For every i, we have either $A_i \subseteq S_1$, $B_i \subseteq S_2$ or $A_i \subseteq S_2, B_i \subseteq S_1$, and each segment of A_i^* intersects every segment of B_i^*.

(C2′) For every pair $s_i, s_j \in S_1 \times S_2$ such that s_i^*, s_j^* intersect, there are two indices $k, l \leq u$ with $s_i \in A_k, s_j \in B_k$ and $s_i \in B_l, s_j \in A_l$.

(C3′) Same as (C3) with ε replaced by δ.

It is easy to verify that $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_{12}$ satisfies conditions (C1)–(C3), and that the total running time of the algorithm is $O(n^{4/3+\varepsilon})$.

It remains to describe how to compute $\mathcal{F}_{12}$. We construct the segment-intersection-searching data structure [4] on the set S_2^*, which can report the set of segments of S_2^* intersecting a query segment in the plane, as a union of few pairwise disjoint subsets of S_2^*. This segment-intersection-searching data structure is a multi-level partition tree, each of whose nodes is associated with a so-called *canonical subset* of S_2^*. The total size of all canonical subsets in the tree is $O(n^{4/3+\delta})$. For a query segment e in the plane, the query procedure selects $O(n^{1/3+\delta})$ pairwise disjoint canonical subsets whose union consists of exactly those segments of S_2^* that intersect e. Using this structure, we can construct the family $\mathcal{F}_{12}$ as follows. We query the data structure with all segments of S_1^*. For each canonical subset $B_i^* \subseteq S_2^*$, let $A_i^* \subseteq S_1^*$ be the set of segments whose query output contained the canonical subset B_i^*. If $A_i^* \neq \emptyset$, we add the pair (A_i, B_i) to $\mathcal{F}_{12}$.

Next, we repeat the same procedure with roles of S_1 and S_2 being interchanged. That is, we construct a similar segment-intersection-searching data structure for S_1^*, query it with all segments of S_2^*, and add the pairs (A_i, B_i), with $A_i \subseteq S_2, B_i \subseteq S_1$, to $\mathcal{F}_{12}$ as above.

It is immediate that the family $\mathcal{F}_{12}$ satisfies conditions (C1$'$) and (C2$'$). Finally, exploiting the distribution of the sizes of the canonical subsets in the data structure and in a query output, one can show that $\mathcal{F}_{12}$ satisfies (C3$'$).

Lemma 4.2. *Given a set S of n segments in $\mathbb{R}^3$ and a parameter $\varepsilon > 0$, a family $\mathcal{F}$ satisfying* (C1)–(C3) *can be constructed in $O(n^{4/3+\varepsilon})$ time.*

4.2 The maximality-reporting structure.

Let A and B denote two sets of segments in $\mathbb{R}^3$ satisfying condition (C1), that is, every segment in A^* intersects all segments in B^*, and, say, the slope of every segment in A^* is smaller than the slope of any segment in B^*. Recall that the data structure $\Psi(A, B)$ should support deletions of segments from A and B; the procedure for deleting a segment e from B should report all segments from A that become maximal with respect to B upon the deletion of e. Furthermore, the preprocessing algorithm should return all segments in A that are maximal with respect to B.

For a segment e in $\mathbb{R}^3$, let $\pi(e)$ and $\varpi(e)$ denote, respectively, the Plücker point and the Plücker hyperplane, represented in $\mathbb{R}^5$, of the line supporting e and oriented in the increasing x-direction [7]. We will use the following observation made by Chazelle et al. [7].

Observation 4.3. *Let ℓ and ℓ' be two lines in $\mathbb{R}^3$ oriented from left to right such that the slope of ℓ^* is smaller (resp. greater) than that of ℓ'^*. Then ℓ lies above ℓ' in the $+z$-direction if and only if the Plücker point of ℓ lies above (resp. below) the Plücker hyperplane of ℓ'.*

Since the slope of every segment in A^* is smaller than that of any segment in B^* and each segment of A^* intersects every segment of B^*, a segment e of A is maximal with respect to B if and only if $\pi(e)$ lies above all hyperplanes $\varpi(e')$, for $e' \in B$. Hence, the problem reduces to constructing the maximality-maintenance data structure, described in Section 3, for the set of points $P = \{\pi(e) \mid e \in A\}$ and the set of hyperplanes $H = \{\varpi(e') \mid e' \in B\}$. Plugging $d = 5$ into Theorem 3.2, we obtain the following result.

Theorem 4.2. *Let A and B be two sets of segments in $\mathbb{R}^3$ satisfying condition (C1), and let $a = |A|$ and $b = |B|$. These sets can be preprocessed in time $O(a^{2/3}b^{2/3+\varepsilon} + ab^\varepsilon + b\log b)$ into a maximality-reporting structure $\Psi(A, B)$, such that all $a + b$ deletions take $O(a^{2/3}b^{2/3+\varepsilon} + ab^\varepsilon + b\log^2 b)$ time.*

5 The Case of Polyhedra.

We now extend our solution for a set of (possibly nonconvex) polyhedra. Let S be a collection of n polyhedra in $\mathbb{R}^3$ with m vertices in total. As above, we will develop a maximality-maintenance structure for S, relative to the $+z$-direction.

For a polyhedron $\mathcal{P}$, let $F(\mathcal{P})$, $E(\mathcal{P})$, and $V(\mathcal{P})$ denote the set of faces, edges, and vertices of $\mathcal{P}$, respectively. Furthermore, let $F(S)$, $E(S)$, and $V(S)$ denote the set of faces, edges, and vertices of all polyhedra in S. Recall that an object is maximal in the $+z$-direction with respect to a set of objects if the object can be translated to infinity in the $+z$-direction without colliding with any of the objects in the set. A polyhedron $\mathcal{P} \in S$ is maximal if and only if every face $f \in F(\mathcal{P})$ is maximal with respect to $V(S) \setminus V(\mathcal{P})$, every edge $e \in E(\mathcal{P})$ is maximal with respect to $E(S) \setminus E(\mathcal{P})$, and every vertex $v \in V(\mathcal{P})$ is maximal with respect to $F(S) \setminus F(\mathcal{P})$.

Using this observation, our maximality-maintenance structure for polyhedra works as follows. For each polyhedron $\mathcal{P}$ we have a counter that counts the number of features (vertices, edges, faces) of $\mathcal{P}$ that are maximal with respect to the corresponding features (faces, edges, vertices) of the other polyhedra. When this counter becomes equal to the total number of features of $\mathcal{P}$, then $\mathcal{P}$ itself becomes maximal. Next we describe how to maintain this counter.

First consider the edges of a polyhedron. In the previous section we have seen how to maintain maximal elements in a set of segments in $\mathbb{R}^3$. However, we cannot use this structure directly for the set $E(S)$, because we are now interested in determining whether an edge of a polyhedron $\mathcal{P}$ is maximal with respect to $E(S) \setminus E(\mathcal{P})$. Fortunately, the needed modification is relatively easy. We reformulate conditions (C1) and (C2) for the family $\mathcal{F}$ stated in Section 4.

(C1$''$) For $1 \leq i \leq t$, we have $A_i, B_i \subseteq E(S)$, $A_i \cap B_i = \emptyset$, each segment of A_i^* intersects every segment of B_i^*; either the slope of every segment in A_i^* is smaller than that of any segment in B_i^*, or the slope of every segment in A_i^* is larger than that of any segment in B_i^*; and no polyhedron has edges in both A_i and B_i.

(C2$''$) For every pair $s_k, s_l \in E(S)$ such that s_k^* and s_l^* intersect and belong to different polyhedra, there are two indices $i, j \leq t$ with $s_k \in A_i, B_j$ and $s_l \in B_i, A_j$.

With these new conditions on $\mathcal{F}$ we can apply our previous solution. To compute the new family $\mathcal{F}$ we use a divide-and-conquer strategy and a segment-intersection-searching structure, as before. The only difference is that there is an additional restriction in the segment-intersection searching: given a segment e^* in the plane, the structure should report all intersected segments whose label is different from the label of e^*, where the label of a segment e^* is the polyhedron that contains the edge e.

Next, we construct a data structure for maintaining maximal vertices and maximal faces. We describe the structure only for maintaining maximal vertices; the structure for maximal faces is similar and has the same performance bounds. We may assume, with no loss of generality, that all faces are triangles. As in the case of edges, we want to determine whether a vertex of $\mathcal{P}$ is maximal with respect to $F(S) \setminus F(\mathcal{P})$. To this end we construct a family $\mathcal{F} = \{(V_1, F_1), (V_2, F_2), \ldots, (V_t, F_t)\}$, satisfying the following conditions.

(D1) For $1 \le i \le t$, we have $V_i \subseteq V(S)$, $F_i \subseteq F(S)$, each point $p^* \in V_i^*$ lies in every triangle $f^* \in F_i^*$, and there is no polyhedron $\mathcal{P}$ such that V_i contains a vertex of $\mathcal{P}$ and F_i contains a face of $\mathcal{P}$.

(D2) For every pair $(v, f) \in V \times F$ such that $v^* \in f^*$ and v and f belong to different polyhedra, there is an $i \le t$ with $v \in V_i$ and $f \in F_i$.

(D3) Similar to (C3).

Computing the family $\mathcal{F}$ can be done in essentially the same way as before: we construct a triangular-range-searching structure for the set V^* and we query with each triangle in T^*. The canonical subsets in the structure are the sets V_i; the corresponding set T_i is formed by all triangles for which that canonical subset is reported by the query procedure. A triangular range query on a set of m points in the plane can be answered in $O(m^{1/3+\varepsilon})$ time after $O(m^{4/3+\varepsilon})$ preprocessing [4].

For each pair (P_i, T_i) in $\mathcal{F}$ we need a maximality-maintenance structure. In view of conditions (D1) and (D2), we can replace triangles of F_i by their supporting planes. We have therefore reduced the problem to a maximality-reporting problem for points and planes in $\mathbb{R}^3$. Applying Theorem 3.2 with $d = 3$, observing that the resulting time complexity is subsumed by that of the above planar range searching, and putting everything together, we obtain the following theorem.

THEOREM 5.1. *Computing a k-directional assembly sequence for a set of polyhedra in 3-space with m vertices in total, relative to a given set of k directions (or determining that no such sequence exists), can be done in $O(km^{4/3+\varepsilon})$ time, for any $\varepsilon > 0$.*

6 Rotational Order.

So far we have discussed assembly sequences where the motion of each part is translational. Next we extend our method to the case where some, or all, of the directions $\vec{d_i} \in D$ are *rotational* around some fixed axis in 3-space, with different axes of rotation for different $\vec{d_i}$'s.

We define a rotational direction of motion $\vec{d_i}$ in the following way. Now $\vec{d_i}$ stands for a directed line in 3-space, and we define the motion *clockwise* around the line when looking in the direction $\vec{d_i}$. We make the following assumption: all the objects in S lie on one side of a plane through $\vec{d_i}$. We denote by H_i the halfspace bounded by this plane and containing all the objects. We say that the object $s \in S$ can be separated from the objects in $S \setminus \{s\}$ by a rotation around $\vec{d_i}$, if the interior of the volume swept by s, as it is rotated clockwise around $\vec{d_i}$ until it lies completely outside H_i, does not intersect the interior of any object in $S \setminus \{s\}$.

Our goal is to construct a maximality-maintenance structure for a set S of objects lying in the halfspace H_i, for a given direction of rotation $\vec{d_i}$. Without loss of generality, we describe such a structure for the case where $\vec{d_i}$ is the z-axis oriented upwards, and all the objects in S lie in the positive y-halfspace.

We say that object s_j is *below* object s_k if, while rotating s_j counterclockwise by angle π around the z-axis (as viewed from above), s_j hits s_k. An object in S is maximal if it is not below any other object in S.

First, consider the case when S is a set of n segments. We fix $\varepsilon > 0$ and construct a data structure for answering the following queries: Given a segment q lying in the positive y halfspace, report the segments in S that q intersects when rotated counterclockwise by π around the z-axis. We then query the structure with each of the segments in S. This can be done in $O(n^{8/5+\varepsilon})$ time [14, 15][1]. As a result we get a family of pairs of segments:

$$\{(A_1, B_1), (A_2, B_2), \ldots, (A_t, B_t)\},$$

satisfying the following conditions. Let $a_i = |A_i|$ and $b_i = |B_i|$.

(E1) For $1 \le i \le t$, we have $A_i, B_i \subseteq S$, and each segment of A_i lies below each segment of B_i.

(E2) For every pair $s, s' \in S$ such that s lies below s', there is an $i \le t$ with $s \in A_i$ and $s' \in B_i$.

(E3) $\displaystyle\sum_{i=1}^{t} a_i = O(n^{8/5+\varepsilon})$ and $\displaystyle\sum_{i=1}^{t} b_i = O(n^{8/5+\varepsilon})$.

[1]As mentioned in the Introduction, the bound in [14] is $O(n^{5/3+\varepsilon})$. It has recently been improved to $O(n^{8/5+\varepsilon})$ [15].

For each segment s, we maintain three lists of indices $A(s), B(s)$, and $NM(s)$, and a set $\mathcal{M}$ of maximal segments in S, as in the case of translational motion described in Section 4. All these lists will be updated as we delete segments from S, but this time the updating is trivially done on the lists themselves, without using any additional data structures. This is possible because of condition (E1), which requires the segments of A_i to be below those of B_i. (This condition is stronger than condition (C1), which we used for translational motions: condition (C1) only requires for each pair of segments from A_i and B_i that one of them be below the other. The stronger condition (E1) makes maintaining the lists $A(s)$, $B(s)$, $NM(s)$ and $\mathcal{M}$ much easier. Enforcing condition (E1) does, however, result in a loss of efficiency. This is why our algorithm for rotations is slower. Unfortunately, we don't know how to construct an efficient maximality-maintenance structure for rotations without enforcing (E1).)

Note that if $NM(s)$ contains index i, that is, s is below all segments in B_i, it will continue to store i until all the segments of B_i are deleted. Also note that for a pair of segments $s, s' \in S$, it can be the case that s lies below s' and at the same time s' lies below s.

Clearly the overall update time of the lists, as we delete all the segments, is proportional to the number of elements in all the sets A_i, B_i, that is $O(n^{8/5+\varepsilon})$, for any $\varepsilon > 0$. Hence we have:

THEOREM 6.1. *Given a collection of n segments in 3-space, a direction of rotation $\vec{d}$ (where all the objects lie in a halfspace bounded by a plane passing through the line defining $\vec{d}$), and a parameter $\varepsilon > 0$, we can construct a maximality-maintenance structure for direction $\vec{d}$, such that the preprocessing time and the time to perform n operations on it is $O(n^{8/5+\varepsilon})$.*

We can also extend this technique to the case of polyhedra, along the lines of Section 5, devising maximality maintenance structures for vertices with respect to faces, and for faces with respect to vertices. Omitting all further details from this version, we obtain:

THEOREM 6.2. *Computing a k-directional assembly sequence for a set of polyhedra in 3-space with m vertices in total, relative to a given set of k directions, at least one of which is rotational (or determining that no such sequence exists), can be done in $O(km^{8/5+\varepsilon})$ time, for any $\varepsilon > 0$.*

References

[1] P. Agarwal, M.J. Katz, and M. Sharir, Computing depth orders and related problems, *Proc. 4th Scandinavian Workshop on Algorithm Theory*, Denmark, 1994, pp.1–12. Also, to appear in *Comput. Geom. Theory Appls.*

[2] P. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* 22 (1993), 794–806.

[3] P. Agarwal and J. Matoušek, Dynamic half-space range searching and its applications, *Algorithmica* 13 (1995), 325–345.

[4] P. Agarwal and M. Sharir, Applications of a new space partitioning scheme. *Discrete Comput. Geom.* 9 (1993), 11–38.

[5] M. de Berg, *Ray Shooting, Depth Orders and Hidden Surface Removal*, Lecture Notes in Computer Science, Vol. 703, Springer-Verlag, Berlin, 1993.

[6] M. de Berg, M. Overmars and O. Schwarzkopf, Computing and verifying depth orders, *SIAM J. Comput.* 23 (1994), 437–446.

[7] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Stolfi. Lines in space: Combinatorics and algorithms. *Proc. 21st ACM Symp. on Theory of Computing*, 1989, pp. 382–393.

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

[9] J. García-López and P. Ramos-Alonzo, Circular visibility and separability, *Proc. 5th Canadian Conference on Computational Geometry*, 1993, pp. 18–23.

[10] L.J. Guibas, D. Halperin, H. Hirukawa, J.-C. Latombe and R.H. Wilson, A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions, *Proc. IEEE International Conference on Robotics and Automation*, 1995, pp. 2553–2560.

[11] L. S. Homem de Mello and S. Lee, editors, *Computer-Aided Mechanical Assembly Planning*, Kluwer Academic Publishers, Boston, 1991.

[12] J. Matoušek, Reporting points in half-spaces, *Comput. Geom. Theory Appls.* 2 (1992), 169–186.

[13] J. Matoušek, Geometric range searching, *ACM Computing Surveys* 26 (1994), 421–461.

[14] E. Schömer and C. Thiel, Efficient collision detection for moving polyhedra, *Proc. 11th ACM Symp. on Computational Geometry*, 1995, pp. 51–60.

[15] C. Thiel, personal communication.

[16] G.T. Toussaint, Movable separability of sets, in *Computational Geometry* (G.T. Toussaint, editor), North-Holland, Amsterdam, 1985, pp. 335–375.

[17] R. H. Wilson, *On Geometric Assembly Planning*, Ph.D. Dissertation, Computer Science Department, Stanford University, March 1992.

[18] R. H. Wilson and J.-C. Latombe, Geometric reasoning about mechanical assembly, *Artificial Intelligence* 71 (1994), 371–396.

[19] R. H. Wilson and T. Matsui, Partitioning an assembly for infinitesimal motions in translation and rotation, *Proceedings of the IEEE International Conference on Intelligent Robots and Systems* (1992), pp. 1311–1318.