DISASSEMBLING TWO-DIMENSIONAL COMPOSITE PARTS VIA TRANSLATIONS+

Doron Nussbaum and Jörg-R. Sack

SCR-TR-160, JUNE 1989

+ Research supported by Natural Science and Engineering Research Council of Canada.

[1] Research on this paper was carried out while the author was at Carleton University.

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

DISASSEMBLING TWO-DIMENSIONAL COMPOSITE PARTS VIA TRANSLATIONS+

by

Doron Nussbaum[1]
Tydac Technologies Inc.
1600 Carling Avenue, Ottawa
and
Jörg-R. Sack
School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6

Abstract

This paper deals with the computational complexity of disassembling 2-dimensional composite parts (comprised of simple polygons) via collision-free translations. The first result of this paper is an $O(Mn + M \log M)$ algorithm for computing a sequence of translations (performed in a common direction) to disassemble composite parts. The algorithm improves on the $O(Mn \log Mn)$ bound previously established for this problem and is easily seen to be optimal. The algorithm solves the problem posed by Nurmi and by Toussaint.

The second result of this paper is an $\Omega(Mn + M \log M)$ lower bound proof for the problem of detecting whether a composite part can be disassembled, or contains interlocking subparts. Thus, detecting the existence of a disassembly sequence is as hard as computing one. As a consequence, the algorithm for computing a disassembly sequence is optimal also for the detecting problem.

+ Research supported by Natural Science and Engineering Research Council of Canada.

1 Research on this paper was carried out while the author was at Carleton University.

1. MOTIVATION AND PROBLEM DESCRIPTION

The rapid development in the fields of computer graphics, CAD-CAM systems, and robotics has attracted considerable attention to the problem of moving objects, e.g. line segments or polygons, in two and three dimensional space. The types of motion (such as single translation, k translations, sequences of translations and rotations,...) and the types of objects (e.g. line segments, convex polygons, polyhedra,...) determine a variety of motion problems.

Here we are interested in a motion planning problem which appears in the following "equivalent" formulations: (a) in the context of separability motions, where one wishes to separate each object from a collection of objects via collision-free motions, and (b) in the context of machine assemblies where a robot is to assemble/disassemble a composite machine part out of/into "simple" parts by moving the parts one by one.

In its most general form this problem has been shown to be P-space hard [R]. One computationally feasible variant of this problem has received considerable attention (see e.g. [COSW], [DS], [GY], [N], [Na], [OW], [NS], [ST], [T]). It assumes that all disassembly motions are translations which are performed in a common direction (typically specified by the user) and one translation motion per object. More formally, the problem is expressed as follows: Can a given composite part $P = \{P_1,P_2,\ldots,P_M\}$ composed of M n-vertex (sub)parts be disassembled, by translating one part at a time in a specified direction over an arbitrarily large distance and only once for each object, in such a way that no object (when translated) will collide with any of the objects not yet moved? If a disassembly of the composite part is possible then, to actually perform it, one must determine a sequence in which the parts are to be moved. We refer to these problems as disassembly detection and disassembly (sequence) determination. (We refrain from defining conceptually clear notions like e.g. collide, translate, etc.). Next, we state some terminology and review results which of immediately relevance to our work.

2. RELEVANT RESULTS

2.1 Terminology

We say that a composite part $P = \{P_1,P_2,\ldots,P_M\}$ is *simple* if each of the M n-vertex (sub)parts P_i, is a simple polygon and $P_i \cap P_j = \emptyset$ for $i \neq j$. Any polygon P_i is represented as a list of its vertices as they appear on the boundary of P_i in clockwise order. Unless otherwise stated, all composite parts considered in this paper are assumed to be simple. Such a composite part is *convex* if each P_i is convex. Similarly, a composite disassembly is *rectilinear* if each P_i is rectilinear (also called orthogonal). A composite part P is *monotone in a direction d* if each of its parts P_i is monotone in d. A composite part P

is *monotone* if there exists a direction d such that P is monotone in d. (The reader is referred to [PS] for basic definitions and techniques from the field of computational geometry.)

2.2 Existing and New Results

Guibas and Yao [GY] have shown that a disassembly sequence for a composite parts of M rectangles (convex n-gons) can be determined in $O(M \log M)$ time (respectively, $O(Mn + M \log M)$ time). Ottmann and Widmayer [OW] gave an algorithm for translating a set of line segments. Their algorithm provides an alternate solution for solving the problems studied in [GY] achieving the same asymptotic time bound, but reducing the two-pass algorithm by Guibas and Yao to one-pass.

While for convex composite parts (as well as line segments) a disassembly sequence exists for any direction specified, an arbitrary simple composite part may not admit disassembly, i.e. may contain a subset of the polygons that are interlocking.

For rectilinear composite parts, Chazelle, Ottmann, Soisalon-Soininen, and Wood [COSW] showed that the problem of detecting whether a disassembly exists can be solved in $O(Mn \log Mn)$ time, while computation of a disassembly sequence is done in $O(Mn \log^2 Mn)$ time. Nurmi [N] proved that for arbitrary simple composite parts both problems can be solved in $O(Mn \log Mn)$ time. Nurmi posed the question of whether an $O(Mn + M \log M)$ algorithm exists for solving these problems. Toussaint gave alternate solutions for these problems. Also in his solutions, the complexities for detection and computation differ; he conjectures that faster algorithms may exist for either problem.

In the following table we give a succinct summary of the existing results together with those results obtained in this paper. It is assumed that $P = \{P_1, P_2, \ldots, P_M\}$ is a composite part and each P_i has n vertices. The type of the assembly is determined by the type of the parts.

The first result proposed in this paper is an $O(Mn + M \log M)$ algorithm for disassembling arbitrary simple composite parts. The algorithm improves on the $O(Mn \log Mn)$ bound previously established for this problem [N] and is optimal. This solves the problem posed in [N] and [T]. In addition, it turns out that the algorithm is simpler than the one presented in [N].

The second result of this paper is an $\Omega(Mn + M \log M)$ lower bound for the problem of detecting whether a composite part can be disassembled or not. Thus, detecting the existence of a disassembly

sequence is as hard as computing one. As a consequence, the algorithm for computing a sequence is optimal also for the detection problem. (The lower bound does not depend on the time it takes to read the data.)

Authors	Type of Parts	Computing a Disassembly Sequence	Detecting the Existence of a Disassembly Sequence
Guibas, Yao	rectangles	$O(M \log M)$	exists always
Guibas, Yao; Ottmann, Widmayer	convex polygons	$O(Mn + M \log M)$	exists always
Chazelle, Ottmann, Soisalon-Soininen, Wood	rectilinear polygons motion parallel to axes	$O(Mn \log^2 Mn)$	$O(Mn \log Mn)$
Toussaint	simple polygons	$O(\min(M^3 n, M^2 n \log Mn))$	$O(\min(M^2 n, Mn \log Mn))$
Nurmi	simple polygons	$O(Mn \log Mn)$	$O(Mn \log Mn)$
here	**simple polygons**	**$\Theta(Mn + M \log M)$**	**$\Theta(Mn + M \log M)$**

Table 1 Existing and new results

3. ALGORITHMS FOR DISASSEMBLING 2-DIMENSIONAL COMPOSITE PARTS

To determine a disassembly sequence for a composite part $P = \{P_1, P_2, \ldots, P_M\}$ of M n-vertex polygons clearly $\Omega(Mn + M \log M)$ time is required. This is true since the input size is $\Omega(Mn)$ and sorting reduces to finding a disassembly sequence for a set of M distinct points on the x-axis. This does, however, not answer the question whether detecting the existence of an assembly sequence is easier than computing one. Indeed, some of the existing algorithms exhibit running times which are different for the two problem instances (see Table 2.1). In this section, we give lower bounds and matching upper bounds for both problems.

When simple composite parts are to be disassembled in the positive x-direction the visibility hulls of the individual parts are found. The *visibility hull* in the x-direction of a polygon P_f is obtained by augmenting P_f by all points lying on a horizontal line segment connecting two points in P. The visibility hulls (in the x-direction) are polygons monotone in the y-direction. If the direction of disassembly, d, is not the positive x-axis then the composite part is rotated until the direction is the positive x-direction.

We recall the following relations introduced earlier by Guibas, Yao [GY] and Ottmann, Widmayer [OW]: *dominance* (*dom*), *immediate dominance* (*idom*), its transitive closure *idom*$^+$ and *below*. Let A and B be two objects. The *idom* (immediate dominance) relation - A *idom* B if, and only if, there exists a point p_A on A and a point p_B on B such that the line segment from p_A to p_B, parallel to the direction of translation, does not intersect any other object. The transitive closure of idom is denoted by *idom*$^+$. When the direction of translation is the positive x-axis, then *idom* is denoted as *ixdom*, which refers to an immediate dominance in the x-direction; similarly *ixdom*$^+$ is defined. If two objects A and B cannot be related by using *ixdom*$^+$ then there exists a line L parallel to the x-axis such that A (B) is above L and B (A) is below L. In such a case, A (B) is *below* B (A). Objects A and B are related by the *dom* relation as A *dom* B if and only if (A *ixdom*$^+$ B) or (not (B *ixdom*$^+$ A) and (A *below* B)) denoted by *dom*=*ixdom*$^+$|*below*. The relation *dom* is totally defined whereas *ixdom*$^+$ and *below* may only be partially defined.

For the following lemma, let $P = \{P_1, P_2, \ldots, P_M\}$ be a simple composite part of M n-vertex polygons and let d (equal to the x-axis) be the direction of disassembly. Next we will observe that the relation *dom* defines a linear order if for each pair P_i, P_j, for which i≠j, holds $VH(P_i, d) \cap VH(P_j, d) = \emptyset$. Lemma 1 and Theorem 1 are easy to prove by using the results established in [ST] and [T].

Lemma 1 (a) The relation *ixdom* is asymmetric if, and only if, the $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \le i, j \le M$ and $i \ne j$.

(b) Let P_i, P_j and P_k be three polygons of P such that their visibility hulls in d are pairwise non-intersecting. P_k *ixdom* P_j and P_j *ixdom* P_i implies that either P_k *ixdom* P_i, or P_k and P_i are incomparable in *ixdom*, when P_k and P_i are examined in isolation.

Theorem 1 Let $P = \{P_1, P_2, \ldots, P_M\}$ be as above and let d (the direction of translation) be the positive x-axis. $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \le i, j \le M$ and $i \ne j$ if and only if the relation *dom* defines a linear ordering.

Thus, to find a disassembly sequence for P, if one exists, we first need to ensure that P contains no pair of polygons whose visibility hulls intersect. Then, given its existence, a disassembly sequence for P

is computed by finding a disassembly sequence for the monotone part comprised of the (pairwise non-intersecting) visibility hulls of $P_1, P_2, \ldots, P_M$. This is done by computing a linear extension of *dom* whose optimal computation is addressed next.

3.1 Monotone Composite Parts

Let $S = \{P_1, P_2, \ldots, P_M\}$ be a set of M simple n-vertex polygons monotone with respect to the positive y-axis. The task is to (1) determine whether S contains a pair of intersecting polygons, and (2) to find a linear extension of the relation *dom* defined on S. Next, we sketch the underlying idea of the algorithm.

Our algorithm uses the plane-sweep paradigm as employed in [OW]. A linear list called *LinExt* contains a linear extension of *dom*. A balanced tree maintains the set of all polygons intersected by the sweep-line such that an in-order traversal of the tree yields the intersection points of the polygons with the sweep-line, sorted by x-coordinate.

Whenever the scan line stops at an upper endpoint, for example vertex v of polygon A, A is inserted into the balanced binary search tree. During the insertion into the tree, v is examined with each of the nodes on the path (from the root to the position of A in the tree) whether it is to the left of, right of, or contained in the polygon stored at the node. If it is contained inside the polygon (stored at the node) then an intersection occurs and the algorithm terminates. If v is not contained in any of the polygons then the successor A'' and the predecessor A' of A in the tree are found. A is then tested for an intersection with A' and with A''. If no intersection occurs between these polygons then A is inserted into a linked list, called *LinExt*, before A'' (its successor) looking from left to right. Whenever a lower endpoint, for example u of B, is encountered the polygon B ceases to be *active*, i.e. whose lower endpoint had been encountered earlier. The predecessor B' and the successor B'' of B are found and B' and B'' are tested for an intersection. If no intersection occurs between them then B is deleted from the tree, but it remains in the linked list. If an intersection has occurred in any one of the above cases the algorithm reports that an intersection has occurred and terminates by concluding that the composite cannot be disassembled in the specified direction. (See Figure 1 for an example). We give the pseudo-code description of the algorithm.

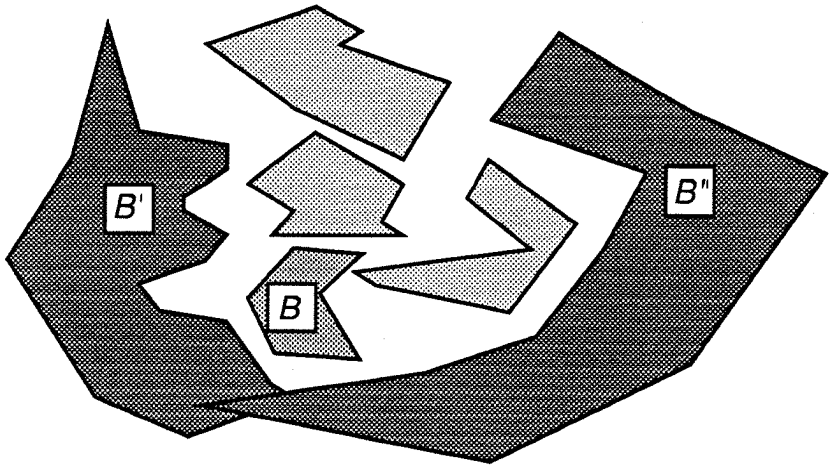

Figure 1 The intersection of the monotone polygons B' and B'' is detected when B ceases to be active.

Algorithm 1: Translating a set of simple monotone polygons.

{The direction of translation is the positive x-axis}

Input: A set $S = \{P_1, P_2, \ldots, P_M\}$ of M simple n-vertex polygons monotone with respect to the positive y-axis.

Output: A translation order of the polygons.

begin

Find the vertices of the polygons with the highest and the lowest y-value (denote these vertices as the endpoints of the polygons);

Initiate the balanced binary search tree and the doubly linked list;

For all endpoints of the polygons in descending order do:

if the endpoint is an upper endpoint of polygon P_f then

1. Insert P_f into the tree using the procedure Insert_polygon_Into_Tree(Root, P_f);
2. Find the predecessor P' and the successor P'' of P in the tree;
3. if P_f and P' intersect* then exit program; {NO TRANSLATION ORDER EXISTS ! }
4. if P_f and P'' intersect* then exit program; {NO TRANSLATION ORDER EXISTS ! }

if (P'' exists) then insert P_f as the new predecessor of P'' in the linked list;

else insert P_f as the new final element in the linked list;

if the endpoint is a lower endpoint of polygon P_f then begin

find the predecessor P' and the successor P'' of P_f in the tree;

5. if P' and P'' intersect* then exit program; {NO TRANSLATION ORDER EXISTS ! }

else delete P_f from the tree; {But not from the linked list}

end; {algorithm}

* If P' or P'', respectively does not exist then the polygons are assumed to be non-intersecting.

Procedure Insert_Polygon_Into_Tree (Root, P_f);

{This procedure inserts a polygon P into a binary balanced search tree. The relation between the polygon is $ixdom^+$.

Input: Root - the root of the tree.

P_f - a polygon.}

begin

While (P_f is not inserted into the tree) do begin

Find the two vertices, one each of the left and right chains (denoted as v_{Li} and v_{Rj} respectively), of the Root such that the y-value of the upper vertex of P_f (denoted as p) is less then the y-value of v_{Li} (v_{Rj}) and greater than the y-value of v_{Li-1} (v_{Rj+1}) (see Figure 2);

{the scan for the new vertices v_{Li} and v_{Rj} starts with the last v_{Lu} and v_{Rw}, of the Root, that have already been found}

If p is to the right of edge(v_{Li}, v_{Li-1}) and to the left of edge (v_{Rj}, v_{Rj+1}) then

exit program; {NO TRANSLATION ORDER EXISTS ! }

elseif p is to the left of edge(v_{Li}, v_{Li-1}) then

Insert_Polygon_Into_Tree (Left_Son_Of_Root, P_f);

else Insert_Polygon_Into_Tree (Right_Son_Of_Root, P_f); {p is to the right of edge (v_{Rj}, v_{Rj+1}) }

end;

Theorem 2 Algorithm 1 detects in $O(Mn + M \log M)$ time whether a set of M n-vertex polygons monotone in a common direction contains a pair of polygons which are intersecting.

Proof:

Complexity Analysis

For each polygon P_k of the composite part two pointers are maintained: (i) v_{Li} - that points to vertex i at the left chain of P_k; (ii) v_{Rj} - that points to vertex j at the right chain of P_k. (The left polygonal chain consists of the vertices between the bottom vertex and the upper vertex). Each time the algorithm inserts a new polygon P_f into the tree a search for the new vertices v_{Li} and v_{Rj} of the root is done. Pointer v_{Li} (v_{Rj}) points to vertex i (j) of the root such that the y-value of the upper vertex of P_f is less then the y-value of v_{Li} (v_{Rj}) and greater than the y-value of v_{Li-1} (v_{Rj+1}). The search starts from the previous v_{Lu} and v_{Rw} that have already been found. If the new vertex v_{Li} (v_{Rj}) is the same as the previous vertex v_{Lu} (v_{Rw}) then O(1) time was involved in the computation of finding the new vertex v_{Li} (v_{Rj}). If the new vertex v_{Li} (v_{Rj}) differs from the previous vertex v_{Lu} (v_{Rw}) then there exists a chain of vertices v_{Li+1}, ..., v_{Lu} (v_{Rj-1}, ..., v_{Rw}) that have been examined and discarded (as their y-coordinates are too large). Let C_i

denote the number of vertices that have been discarded at the i-th search. Then the total cost of a search is less then $C_i + O(1)$.

The time complexity of the i-th search is $C_i + O(1)$, where $0 < C_i$. The algorithm inserts M polygons into the tree. Each time at most log M additional recursive searches are done and, thus, the maximum number of times that a search for vertices is done is M log M. Hence, the cost of the algorithm, during all its execution, is $\sum_{i=1..MlogM}(C_i + O(1)) = \sum_{i=1..MlogM}C_i + \sum_{i=1..MlogM}O(1)$. $\sum_{i=1..MlogM}C_i$ is the total number of edges that have been discarded. Each vertex is discarded at most once and, therefore, $\sum_{i=1..MlogM}C_i \leq Mn$. Thus, the total cost for all insertions (line 1) is $\sum_{i=1..MlogM}C_i + \sum_{i=1..MlogM}O(1) \leq Mn + M \log M$.

After the predecessor P' and the successor P'' of P_f have been identified in the tree structure (line 2), at a cost of O(log M), P_f is tested for intersection with P' and P'' at a cost of O(n) each (lines 3-4). For the deletion (line 5) one pair of polygons is tested for intersection. Since the total number of insertions and deletions is $2*M$, the total cost of this step and thus of the entire algorithm is O($Mn + M \log M$).

Correctness

If there is no pair of intersecting polygons then the algorithm reports no intersection which is correct. Otherwise, we will show that the algorithm finds a highest intersection point (highest being defined as having maximum y-coordinate). Let u be a highest intersection point which occurs say between polygons P_i and P_j. Without loss of generality we may assume that the y-coordinate of the upper vertex of P_i is less than the y-coordinate of the upper vertex of P_j. Two cases arise: (a) the upper vertex of P_i is the intersection point u. (b) the intersection point is any other point of the boundary of P_i.

(a) When the algorithm encounters P_i at u all polygons encountered so far (i.e. appearing in *LinExt*) are either pairwise non-intersecting or have an intersection point below u. No polygons which has ceased to be active, intersects P_i. The binary search tree, maintains a sorted list of all active polygons (i.e. the in-order traversal of the tree yields the polygons in order of increasing x-values of points intersected by the sweep line). Thus the first if-statement of Procedure *Insert_Polygon_Into_Tree* finds that u is contained in P_j and the algorithm correctly reports that no translation order exists (which means that a pair of intersecting polygons has been found).

(b) Assume now that the intersection point u is a boundary point of P_i other than the highest vertex. W.l.o.g. assume that when P_i is inserted into the tree P_i and P_j are related as P_j *ixdom*$^+$ P_i. Two cases to be examined.

(i) P_j *ixdom* P_i Then, since P_j is successor of P_i, P_i and P_j are tested for intersection (line 4 of Algorithm 1), the intersection point is correctly found.

(ii) P_j *ixdom* $^+$ P_i but not(P_j *ixdom* P_i). P_i and P_j are active and u has not been found. Hence one or more polygons $P_{k'}$ exists such that P_j *ixdom* P_k' and P_k' *ixdom* P_i. Among these polygons let P_k be the polygon with lowest lower vertex. The y-coordinate of the lower vertex of P_k is greater than the y-coordinate of u and, therefore, P_k is deleted at which point polygon P_i becomes predecessor of P_j in the

tree (before the algorithm reaches u). In line 5 of the algorithm, P_i and P_j are then tested for intersection and point u is correctly determined. q.e.d.

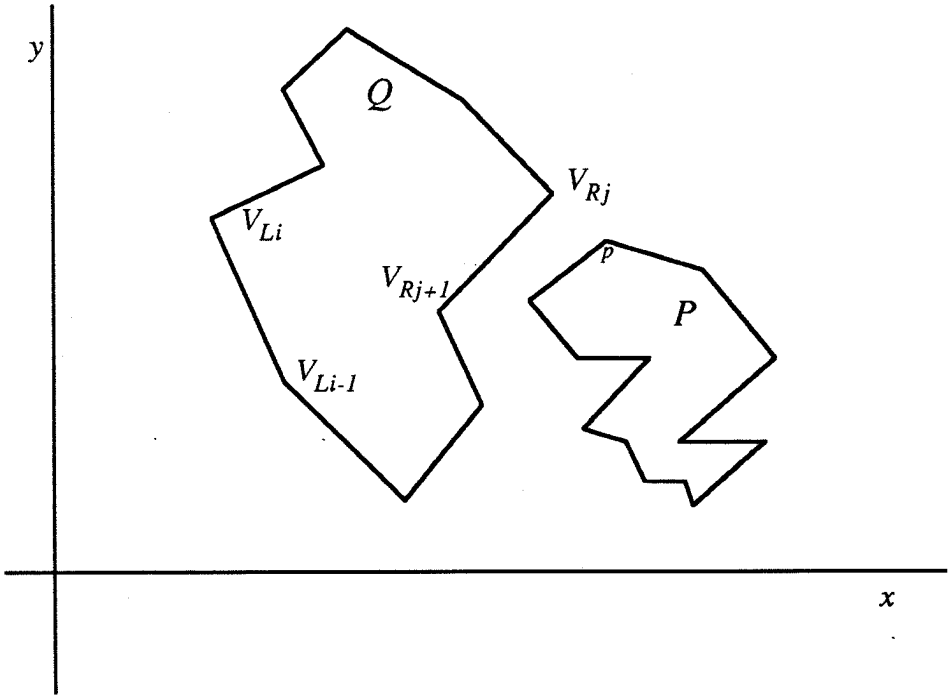

Figure 2 The vertices v_{Li} and v_{Rj} of Q are found when the vertex p of P_f is inserted.

Theorem 3 Let $P = \{P_1, P_2, \ldots, P_M\}$ be a composite part (possibly non-simple) of M simple n-vertex polygons, monotone with respect to the y-direction. In $O(Mn + M \log M)$ time it can be determined whether a disassembly sequence for P exists, and, if so, a disassembly sequence for P can be computed.This is worst-case optimal.

Proof

By Theorem 1 and Theorem 2 the existence of a disassembly sequence can be determined in $O(Mn + M \log M)$ time. Thus we assume that a translation ordering for P exists and show that the algorithm correctly determines a disassembly sequence. The claim is that at any time during the execution of the algorithm the linked list *LinExt* maintained by the algorithm represents a linear extension of the dominance relation for all polygons encountered so far. This will imply in particular that upon termination of the algorithm a translation ordering for P is described in *LinExt*.

We establish the claim inductively assuming in addition that at any time an inorder traversal of the tree produces a subsequence of *LinExt*. Assume that this is true before vertex v of P_f is encountered. Two cases arise depending on whether v is a lower endpoint (case (a)) or an upper endpoint (case (b)).

(a) P_f is deleted then clearly the claim remains true.

(b) P_f is inserted into the tree. The inorder traversal of the tree gives the sweep line status which is the relation *ixdom* of all polygons in the tree, i.e. of all *active* polygons. If a successor P'' for P_f exists then P'' *ixdom* P_f and by transitivity of *dom*, P_f is also in the correct position (using *dom*) with respect to all polygons succeeding P'' in *LinExt*. If no successor P'' for P_f exists then P_f is appended to *LinExt* which is correct since for no P_i holds P_i *ixdom* P_f. If a predecessor P' for P_j exists then P_f *ixdom* P' and again P_f is in the correct position (using *dom*) with respect to all polygons preceding P' in *LinExt*. All other polygons P_j between P' and P'' are no longer active, thus P_f *below* P_j holds, implying P_f *dom* P_j. q.e.d.

3.2 Simple Composite Parts

As remarked earlier, we disassemble simple composite parts in the positive x-direction by first finding the visibility hulls of the individual parts. Each visibility hull can be determined in linear time by using any of the algorithms developed in [DSs], [EA], or [L]; thus the total time taken for this step is $O(Mn)$. Then the visibility hulls are given as input to the Algorithm 1, the output of Algorithm 1 is a disassembly sequence for the simple composite part P.

Theorem 4 Let $P = \{P_1, P_2, \ldots, P_M\}$ be a (simple) composite part of M n-vertex polygons. In $O(Mn + M \log M)$ time it can be determined whether a disassembly sequence for P exists, and, if so, a disassembly sequence for P can be computed.

Clearly computing a translation ordering for a composite part is at least as hard as sorting and thus the following Corollary holds.

Corollary 1 For the problem of computing a disassembly sequence of simple composite part, the above algorithm is worst case optimal.

3.3 Lower Bound for Detecting the Existence of a Disassembly Sequence

It remains to see whether detecting the existence of a disassembly sequence is easier than computing one. For a variety of problems detecting a certain geometric property is easier than computing it; even sublinear solutions may exist, if the input is already stored in memory. These problems include

computing lines of support for convex polygons, see e.g. [PS], detecting whether two convex polygons intersect [CD], etc.

The lower bound presented here for the disassembly detection problem will be independent on the time required to read in the polygons. We assume further that the visibility hull of each polygon in P has been previously computed and is stored in memory.

Chazelle and Dobkin [CD] have established a tight logarithmic lower bound for the Intersection Detection Problem for two convex polygons (stored in memory). Here, we first establish a linear lower bound on the corresponding problem for polygons monotone in a known direction. Given two polygons P and Q, monotone with respect to the same direction, detect whether P and Q intersect (where m and n are the number of vertices in P and Q respectively). A program that can solve this problem will be called ***IntersectionDetector***.

Lemma 2 At least $m+n$ -4 time is required to test whether two polygons monotone in a specified direction intersect, or not, where m, n are the number of vertices in the each of the polygons, respectively. The bound is independent on the time required to read the polygons into memory.

Proof We give an adversary-based argument. Let P and Q be two polygons monotone with respect to the y-axis. Let the vertices $q_2, q_3, \ldots, q_{n-1}$ of Q face the edge(p_m, p_1) of P. Let the vertices $p_3, p_4, \ldots, p_m$ of P face the edge(q_{n-1}, q_n) of Q, let the x-coordinate of p_1 (p_2) be less than the x-coordinate of q_1 (q_n) and let the y-coordinate of p_1 (p_2) be equal to the y-coordinate of q_1 (q_n) (see Figure 3).
Assume the contrary of Lemma 3.9, i.e. that there exists an IntersectionDetector that requires less then $m+n$-4 steps. To solve the problem, IntersectionDetector must be able to detect whether P and Q intersect. Whenever IntersectionDetector accesses vertex p_i (q_j) the adversary will give IntersectionDetector, for free, the information which edge(q_u, q_v) of Q (edge(p_r, p_s) of P) p_i (q_j) is facing and if p_i (q_j) intersects Q (P). The adversary maintains a list of all vertices that have not yet been examined by IntersectionDetector. If IntersectionDetector examines less than $m+n$-4 vertices then there exist at least one vertex among $p_3, p_4, \ldots, p_m$ and $q_2, q_3, \ldots, q_{n-1}$ that was not examined by IntersectionDetector. Assuming that vertex p_k, $2 < k \leq m$, $(q_r, 2 \leq r < n)$ was not examined. If IntersectionDetector outputs that the polygons intersect then IntersectionDetector fails because the polygons do not intersect. If IntersectionDetector outputs that the polygons do not intersect then just before the detector outputs the answer, that the polygons do not intersect, the adversary "moves" p_k (q_r), by increasing (decreasing) the x-coordinate of p_k (q_r), until p_k (q_r) intersect Q (P). "Moving" p_k (q_r) does not change the property of monotonicity of P (Q) with respect to the y-axis, as the x-values of the vertices are independent, and thus IntersectionDetector fails to give a correct answer. The adversary can find in $O(m+n)$ steps which vertex should be moved by traversing all the vertices. q.e.d.

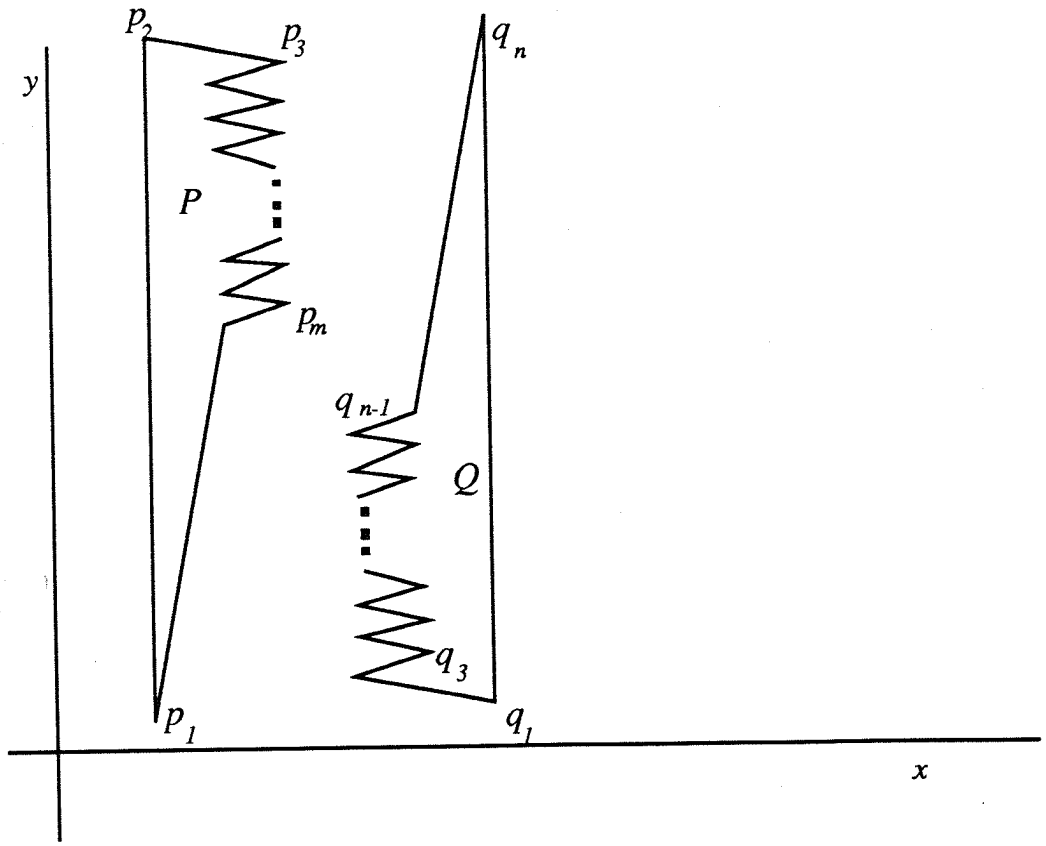

Figure 3 Construction of polygons P and Q for adversary argument.

It is easy to see that this construction generalizes to sets of monotone polygons. A similar argument can then be made for set of visiblity hulls instead of monotone polygons. Note however that it must be ensured that the input polygons are pairwise non-intersecting while the visibility hulls may, or may not, be intersecting.

Lemma 4 Detecting whether a simple composite part $P = \{P_1, P_2, \ldots, P_M\}$ of M n-vertex polygons can be disassembled in a given direction d requires at least $\Omega(Mn + M \log M)$ time.The bound is independent on the time to read the input.

Proof (a) "$\Omega(Mn)$" Follows from Lemma 3 and the previous discussion.

(b) "$\Omega(M \log M)$" The proof is based on a reduction from integer element uniqueness. The *Integer Element Uniqueness Problem* is defined as: given M integers N_i, $1 \le i \le M$, decide if any two are equal and has been shown to require $\Omega(M \log M)$ time [SP]. The construction needs to ensure that the input polygons are non-intersecting and that the visibility polygons intersect if, and only if, the M numbers are not all distinct. The transformation from the Integer Element Uniqueness to the Separability Detecting

Problem is as follows. For each integer N_i, $1 \le i \le M$, of the Integer Element Uniqueness Problem a V-shaped simple 4-vertex polygon P_i, $1 \le i \le M$, is constructed (see Figure 4). Each polygon P_i, $1 \le i \le M$, is constructed such that its width is less than 1 (x-coordinate of $p_{i,4}$ - x-coordinate of $p_{i,2}$), also y-coordinate of $p_{i,3}$ - y-coordinate of $p_{i,1}$ < 1. Therefore, there is no pair of intersecting polygons.
Two polygons P_i and P_j are nested inside each other if, and only if, their visiblity hulls are intersecting. Two polygons P_i and P_j are nested if, and only if, $N_i = N_j$. Therefore, if the output of the intersection test is "YES" then no two integers are equal in the Integer Element Uniqueness Problem. If the output of the intersection test is is "NO" then there exists at least one pair of polygons that are nested and, therefore, there exist two equal integers in the Integer Element Uniqueness Problem that are equal.
Transforming the Integer Element Uniqueness Problem to the Separability Detecting Problem requires $O(M)$ time because each polygon consists of only four vertices. Let $f(M,n)$ denote the time required for testing the M polygons for separability.Converting the output of the intersection detection algorithm into a correct output for the Integer Element Uniqueness Algorithm requires O(1) time.

Hence, the Separability Detecting Problem requires at least $\Omega(M \log M) - O(M) - O(1)$, which yields $f(M,n) \ge \Omega(M \log M)$. q.e.d.

Theorem 5 Detecting the existence of a disassembly sequence for a simple composite part is as hard as computing one. Both problems can be solved in $\Theta(Mn + M \log M)$ time.

Proof: Follows from Theorem 3.7 and Lemma 3.11. q.e.d.

REMARKS

This paper presents lower bounds and matching upper bounds for 2-dim. disassemblies via translations in a common direction. Related variants of the assembly problems allow e.g. multiple directions of translations for a composite 2-d part as discussed in [DS] and in [Na], or disassembly of a pair of polygons by multiple translations [PSS]; see also [T] for additional references. In [Nu] it is shown how to efficiently disassemble 3-dimensional composite parts and how to compute directions for disassembly.

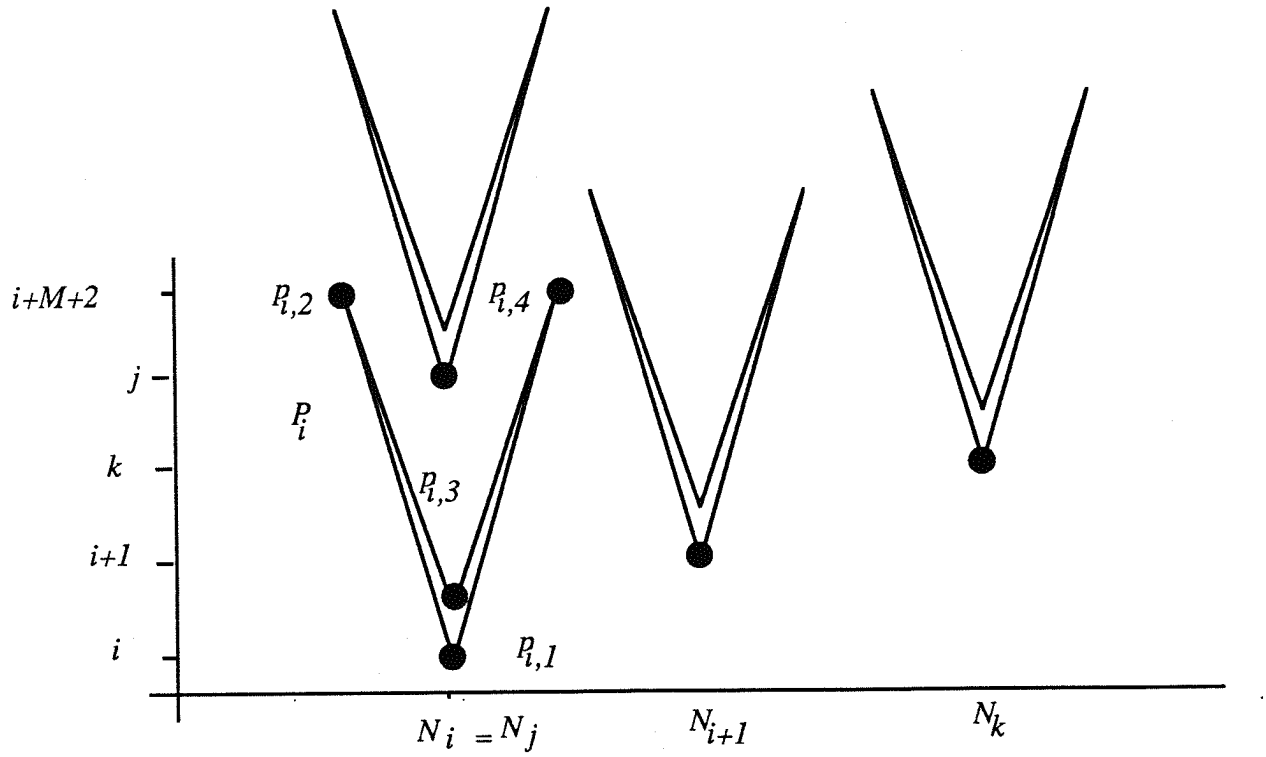

Figure 4 The reduction from element uniqueness.

REFERENCES

[CD] B. Chazelle and D. P. Dobkin, "Intersection of Convex Objects in Two and Three Dimensions", Journal of the ACM, Vol. 34, No. 1, January 1987, pp. 1-27.

[COSW] B. Chazelle, T. Ottmann, E. Soisalon-Soininen and D. Wood, "The Complexity and Decidability of Separation", Technical Report no. CS-83-34, University of Waterloo, Waterloo, Ontario, November 1983.

[D] R. Dawson, "On Removing a Ball Without Disturbing the Others", Mathematics Magazine, Vol. 57, No. 1, January 1984, pp. 27-30.

[DSa] J. Dean and J.-R. Sack, "Efficient Hidden-Line Elimination by Capturing Winding Information", Proceedings 23rd Allerton Conference on Communication, Control and Computing, Ill., Oct. 1985, pp. 496-505.

[DS] F. Dehne and J.-R. Sack, "Translation Separability of Sets of Polygons", The Visual Computer, No. 3, 1981, pp. 227-235.

[EA] H. ElGindy, D. Avis, "A Linear Algorithm for Computing the Visiblity Polygon from a Point", Journal of Algorithms, Vol. 2, No. 3, 1983, pp. 191-202.

[GY] L. J. Guibas and F. F. Yao, "On Translating a Set of Rectangles", in Advances in Computing Research Volume I: Computational Geometry, Ed. F. P. Preparata, JAI Press Inc., Greenwich, CO, 1983, pp. 61-77.

[L] D.T. Lee, "Visiblity of a Simple Polygon", Computer Vision, Graphics and Image Processing, Vol. 22, No. 2, 1983, pp. 207-221.

[Na] B. K. Natarajan, " On Planning Assemblies", Proceedings of the Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, Illinois, June 1988, pp. 299-308.

[N] O. Nurmi, "On Translating a Set of Objects in 2- and 3- Dimensional Space", Computer Vision, Graphics, and Image Processing, Vol. 36, 1986, pp. 42-52.

[NS] O. Nurmi and J.-R. Sack, "Separating a Polyhedron by One Translation from a Set of Obstacles", Proceedings Workshop on Graph Theory, Amsterdam 1988, Lecture Notes in Computer Science, Vol. 344, 1988, pp. 202-212.

[Nu] D. Nussbaum, "Directional Separability in two and three Dimensional Space", School of Computer Science, Carleton University, 1988.

[OW] T. Ottmann and P. Widmayer, "On Translating a Set of Line Segments", Computer Vision, Graphics, and Image Processing Vol. 24, 1983, pp. 382-389.

[PSS] R. Pollack, M. Sharir and S. Sifrony, "Separating Two Simple Polygons by a Sequence of Translations", Technical Report No. 59/87, Eskenasy Institute of Computer Science, School of Mathematical Science, Tel-Aviv University, Tel-Aviv Israel, January 1987.

[PS] F. P. Preparata and M. I. Shamos, *Computational Geometry An Introduction*, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

[R] J. Reif, "Complexity of the Mover's Problem and Generalizations", Proceedings 20th Symposium on the Foundations of Computer Science, 1979, pp. 560-570.

[ST] J.-R. Sack and G.T. Toussaint, "Separability of Pairs of Polygons Through Single Translations", Robotica, Vol. 5, 1987, pp. 55-63.

[T] G. T. Toussaint, "Movable Separability of Sets", Computational Geometry Ed. G. T. Toussaint, North Holland Amsterdam, New York, Oxford, 1985.

[Ta] R. E. Tarjan, "Depth First Search and Linear Graph Algorithms", SIAM Journal on Computing, Vol. 1, 1972, pp. 146-160.