MicroD

Figure 3.3.6 $P_k \text{ idom } P_l$

Figure 3.3.7 P_l and P_k are incomparable in idom^+ (P_l is below P_k).

Figure 3.3.8 Lines L_1 and L_2 as y -separators.....

Figure 3.3.9 P_l and P_j are incomparable in idom^+

Figure 3.3.10 P_l and P_h are incomparable in idom^+

Figure 3.3.11 $P_{cu} \text{ idom } P_k$ and therefore $P_l \text{ idom}^+ P_k$

Figure 3.3.12 $P_k \text{ idom } P_{cu}$ and therefore $P_k \text{ idom } P_j$

1

MicroD

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

DIRECTIONAL SEPARABILITY IN TWO AND THREE DIMENSIONAL SPACE

by

Doron Nussbaum, B. Sc.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario

August 15, 1988

© copyright

1988 Doron Nussbaum

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-51160-5

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**DIRECTIONAL SEPARABILITY IN TWO AND THREE
DIMENSIONAL SPACE**

submitted by Doron Nussbaum, B. Sc.
in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chairman, School of Computer Science

Carleton University

Date

ABSTRACT

In this thesis, algorithms are presented for solving several problems arising in the collision-free separation of objects in two and three dimensional space.

In two dimensional space, an $O(Mn + M \log M)$ algorithm is given for the problem of translating a set of M simple non-intersecting n -vertex polygons, one polygon at a time in a fixed direction, such that no polygon, when translated, will collide with any of the polygons not yet moved. This algorithm improves on the previously obtained upper bound of $O(Mn \log Mn)$. This solves the open problem posed by Nurmi. By establishing a lower bound of $\Omega(Mn + M \log M)$, it is shown that this algorithm is optimal.

In three dimensional space several algorithms are given.

- (i) An $O(m^2n^2 \log mn)$ algorithm is given for determining all directions in which a simple n -vertex polyhedron (n -vertex simple polygonal face) can be translated to infinity without colliding with another simple m -vertex polyhedra (simple m -vertex polygonal face). This generalizes previously known results. This algorithm is shown to be nearly worst case optimal by establishing an $\Omega(m^2n^2)$ lower bound.
- (ii) An $O(M^2n^2)$ worst case algorithm is given for the problem of translating a set of M n -vertex polyhedra in three dimensional space, one polyhedron at a time in a fixed direction, such that no polyhedron, when translated, will collide with any of the polyhedra not yet translated. When the set of objects, to be translated, consists of simple polygonal faces, convex polygonal faces or polytopes, the worst case time complexity is $O(M^2n \log n)$, $O(M^2 \log n)$ and $O(M^2 \log n)$, respectively. These algorithms improve on the worst case time complexity of $O(M^2n^2 \log Mn)$ of the previously existing algorithm.
- (iii) It is shown that a set of M line segments in three dimensional space can always be separated by translating one line segment at a time such that no line segment will collide with those line segments not yet moved. The direction of translation may differ from line segment to line segment. An $O(M \log M)$ algorithm is given for this problem. When the direction of translation is fixed for all line segments it is shown that as few as six line segments can be arranged in such a way that no fixed direction exists in which the line segments can be separated.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor Jorg-Rüdiger Sack for suggesting this topic and for his support while writing this thesis. He spent many hours discussing this work and his suggestions, observations and comments have improved the work presented here.

I would like, also, to thank the following people: Professor Michael Atkinson for his support and encouragement in the past two years while studying at Carleton University; my friends at Carleton University for the good times and discussions; the support staff for their continuing help and assistance; Glenn Booth for proof-reading parts of this thesis; and finally, Bev for her support and encouragement in the past two years and for helping in the writing of this thesis.

TABLE OF CONTENTS

	Page
ACCEPTANCE SHEET	ii
ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER I: INTRODUCTION	1
1.1 Computational Geometry.....	1
1.2 Motivation and Problem Description.....	1
1.3 Directional Separability Definitions.....	3
1.4 Related Results.....	4
1.5 Geometric Tools and Definitions.....	6
1.5.1 Definitions.....	6
1.5.2 Plane Sweep Technique (Line Sweep).....	7
1.5.3 Priority Queue.....	8
1.5.4 Model of Computation and Computational Complexity.....	9
1.5.5 The Doubly-Connected-Edge-List (DCEL).....	11
1.5.6 The Sphere Sweep.....	13
1.6 Thesis Outline.....	13

CHAPTER II: DIRECTIONAL SEPARABILITY SURVEY.....	15
2.1 The Dominance Relation.....	15
2.2 Separability in Two Dimensional Space.....	20
2.2.1 Separability of Convex Polygons (Line Segments).....	20
2.2.1.1 Guibas and Yao's algorithm.....	20
2.2.1.2 Ottmann and Widmayer's algorithm.....	24
2.2.2 Simple Polygons in the Plane.....	26
2.3 All-Direction Separability of Two Polytopes.....	32
 CHAPTER III: TWO DIMENSIONAL SPACE.....	 36
3.1 Introduction.....	36
3.2 Preliminary Definitions and Results.....	378
3.3 The Dominance Relation.....	40
3.4 Uni-Directional Separability Algorithms.....	51
3.4.1 Algorithm for Monotone Polygons.....	51
3.4.2 Algorithm for Simple Polygons.....	56
3.4.3 Complexity Analysis.....	57
3.4.3.1 Time and Space Complexity of Algorithm 3.4.1.....	57
3.4.3.2 Time and Space Complexity of Algorithm 3.4.2.....	60
3.5 Correctness of Algorithms 3.4.1 and 3.4.2.....	60
3.6 Lower Bound.....	64
3.6.1 Lower Bound for Monotone Polygons.....	65
3.6.2 Lower Bound for Visibility Hulls.....	71
3.6.3 Lower Bound for the Uni-directional Separability Problem.....	77

3.6.3.1 Lower Bound for Determining a Translation Order	78
3.6.3.2 Lower Bound for Detecting whether a Translation Order Exists..	80
CHAPTER IV: THREE DIMENSIONAL SPACE.....	85
4.1 Introduction.....	85
4.2 Separability of a Set of Line Segments.....	86
4.2.1 Interlocking Lines.....	87
4.2.2 Translation Order in Three Dimensional Space.....	91
4.3 Polygonal Faces in Three Dimensional Space.....	95
4.3.1 Preliminary Operations and Definitions.....	95
4.3.2 The Dominance Relation.....	101
4.3.3 Simple Polygonal Faces Algorithm.....	102
4.3.3.1 The Algorithm.....	102
4.3.3.2 Complexity of Algorithm 4.3.1.....	107
4.3.4 Convex Polygonal Faces Algorithm..	110
4.3.4.1 Introduction.....	110
4.3.4.2 Preliminary Operations.....	110
4.3.4.3 The Algorithm.....	111
4.3.4.4 Complexity of Algorithm 4.3.2.....	113
4.4 Uni-directional separability Problem for Polyhedra.....	1
4.4.1 Simple Polyhedra.....	115
4.4.1.1 Preliminary Operations.....	115
4.4.1.2 Complexity.....	116
4.4.2 Convex Polyhedra (Polytopes).....	117

4.4.2.1 Preliminary Operations.....	117
4.4.2.2 The Algorithm.....	119
4.4.2.3 Complexity of Algorithm 4.4.1.....	121
4.5 All-Directions Separability Problem.....	123
4.5.1 Introduction.....	123
4.5.2 All Directions Separability Problem of Two Simple Polyhedra.....	124
4.5.2.1 The Algorithm.....	124
4.5.2.2 Complexity of the Algorithm.....	125
4.5.3 Lower Bound.....	126
4.5.4 Simple Polygonal Faces in Three Dimensional Space.....	131
4.5.4.1 Introduction.....	131
4.5.4.2 The Algorithm.....	131
4.5.4.3 Complexity of the Algorithm.....	132
CHAPTER V: APPLICATIONS.....	134
5.1 Ray Tracing.....	134
5.2 Intelligent Dishwasher.....	136
5.3 Intelligent Assembly Line.....	138
5.4 Rescue Robot.....	140
5.5 Automatic Loader (Intelligent Crane).....	140
5.6 Computer Graphics.....	140
5.7 Designing New Products.....	142
5.8 Medicine.....	142
5.9 Automatic Vacuum Cleaner.....	142

CHAPTER VI: CONCLUSIONS.....	144
APPENDIX A: PROCEDURES OF ALGORITHM 4.3.1.....	147
REFERENCES.....	156

LIST OF TABLES

Table	Description	Page
6.1	Summary of Results.....	146

LIST OF FIGURES

Figure 1.5.1 An example of DCEL.....	12
Figure 2.1.1 $A \text{ dom } B$ and $A \text{ dom } C$ but not $A \text{ dom } D$	16
Figure 2.1.2 $A \text{ idom } B$ and $A \text{ idom } D$ but not $A \text{ idom } C$	17
Figure 2.1.3 Line segments s_i and s_j are <i>incomparable</i> by idom^+	18
Figure 2.1.4 Line segment s_i is <i>below</i> line segment s_j	19
Figure 2.2.1 The relations that are produced during insertion and deletion.....	22
Figure 2.2.2 Neither A nor B can be translated as $A \text{ dom } B$ and $A \text{ dom } A$	27
Figure 2.2.3 Polygon A is represented by its edges.....	28
Figure 2.2.4 Polygons A, B, C and D of Example 2.2.3.....	29
Figure 2.2.5 The digraph produced out of the partial order of the polygons in Figure 2.2.3.....	30
Figure 2.3.1 (a) An extreme supporting plane of A and B . (b) An extreme separating plane of A and B	33
Figure 3.1.1 P will collide with Q if P is to be translated in direction d_1	36
Figure 3.2.1 The visibility hull of a simple polygon P with respect to the positive x - axis direction. The $\text{VH}(P, 0)$ is monotone with respect to $\pi/2$	37
Figure 3.2.2 Polygon P is not monotone with respect to the x axis.....	38
Figure 3.3.1 $P_k \text{ idom } P_i$	43
Figure 3.3.2 P_i and P_k are incomparable in idom	43
Figure 3.3.3 $P_k \text{ idom } P_i$	43
Figure 3.3.4 P_i and P_k are incomparable in idom	44
Figure 3.3.5 P_j and P_k are incomparable in idom	44

Figure 3.3.6 $P_k \text{ idom } P_i$	44
Figure 3.3.7 P_i and P_k are incomparable in idom^+ (P_i is below P_k).....	47
Figure 3.3.8 Lines L_1 and L_2 as y -separators.....	47
Figure 3.3.9 P_i and P_j are incomparable in idom^+	47
Figure 3.3.10 P_i and P_h are incomparable in idom^+	48
Figure 3.3.11 $P_{eu} \text{ idom } P_k$ and therefore $P_i \text{ idom}^+ P_k$	50
Figure 3.3.12 $P_k \text{ idom } P_{eu}$ and therefore $P_k \text{ idom } P_j$	51
Figure 3.4.1 The vertices v_{Ll} and v_{Rl} of Q are found when the vertex p of P is inserted.....	55
Figure 3.5.1 The upper vertex of A is the highest intersection point w	61
Figure 3.5.2 The intersection between two edges one on A and one on B forms w ..	61
Figure 3.5.3 $P_j \text{ idom } P_k$ and $P_k \text{ idom } P_i$	64
Figure 3.6.1 Vertices $p_3, \dots, p_m$ of P face edge(q_{n-1}, q_n) and vertices $q_2, \dots, q_{m-1}$ face edge(p_1, p_m).....	67
Figure 3.6.2 Polygons P and Q intersect.....	68
Figure 3.6.3 The arrangement of polygons $P_1, \dots, P_M$	70
Figure 3.6.4 An intersection between the visibility hulls of P and Q (d is the x -axis)	71
Figure 3.6.5 The polygons P and Q	73
Figure 3.6.6 The visibility hulls of P and Q	74
Figure 3.6.7 The visibility hulls of P and Q intersect.....	75
Figure 3.6.8 Two simple polygons P_i and P_k that are used in Lemma 3.6.4.....	76
Figure 3.6.9 A construction of the visibility hulls.....	77
Figure 3.6.10 The transformation of the intersecting visibility hulls problem to the Separability Detection Problem.....	79

Figure 3.6.11 The transformation of the Integer Element Uniqueness Problem to the Separability Detection Problem.....	79
Figure 3.6.12 The construction of the polygons during the transformation.....	81
Figure 3.6.13 The transformation of the Integer Sorting Problem to the Translation Order Determination Problem.....	83
Figure 3.6.14 The construction of the polygons during the transformation.....	83
Figure 4.1.1 Line segments A, B and C , d is directed from the page up.....	86
Figure 4.2.1 The placement of triangle R on the plane T	88
Figure 4.2.2 Line A, B and C are interlaced on T	88
Figure 4.2.3 The final construction of the six line segments.....	89
Figure 4.3.1 The intersection L_{PQ} between T_P and T_Q	96
Figure 4.3.2 The digraph representing the <i>idom</i> relation of A, B and C	102
Figure 4.3.3 (a) the polygon P (b) P_{above} and P_{below}	106
Figure 4.4.1 The intersection points p_L^d and p_R^d	118
Figure 4.5.1 Polyhedron P	127
Figure 4.5.2 Polyhedron Q	128
Figure 4.5.3 The translation region of Q with respect to P when Q is translated through teeth p_1, p_2, p_{n+1} and p_{n+2}	128
Figure 4.5.4 All the n^2 translation regions of Q with respect to P (from example 4.5.1).....	129
Figure 4.5.5 Teeth q_1 and q_2 are replaced each by m teeth.....	130
Figure 4.5.6 All the $m^2 n^2$ translation regions of Q with respect to P	130
Figure 5.1 The principle of ray tracing.....	136
Figure 5.2 The intelligent dishwasher.....	137

Figure 5.3 The intelligent dishwasher BDN.....	137
Figure 5.4 Multi-link assembly robot (PUMA arm).....	139
Figure 5.5 Prismatic arm assembly robot.....	139
Figure 5.6 The role of hidden lines and surfaces.....	141

CHAPTER 1

INTRODUCTION

1.1 COMPUTATIONAL GEOMETRY

Geometry has played an important role in daily living for thousands of years. From the time of the ancient Egyptians, considered to be the first to use geometry for measuring their fields, its application to a variety of areas of science has been evident. Over the past 40 years, with the rapid evolution of the computer, there has been a tremendous increase in the amount of data that needs to and can be processed. This need and the speed of computers have made them the main executioners of many algorithms.

Today computers are found in almost every field e.g. medicine, engineering, geology, et cetera. This quiet revolution of computers in our lives has not neglected the field of geometry. Today one can find geometrical problems in the fields of computer graphics, robot vision, robot motion, pattern recognition and VLSI. Classical geometrical "algorithms" and techniques, which have been used for many years, are often not suitable to be used by the computer. Many of these classical "algorithms" have been reassessed and redesigned and new tools (techniques) and algorithms have been developed. The need to develop new computer-executable algorithms has led to the development of, what is known today as, the field of computational geometry. The goal of computational geometry is to design efficient algorithms to solve present-day geometrical problems.

1.2 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 sets of objects, e.g.

line segments or polygons, in two and three dimensional space. In computer graphics a movability problem arises when objects are moved on a screen (see Guibas and Yao [GY] for a more detailed description). In robotics the problem arises when one wishes to determine whether a machine can be assembled by a robot out of a set of parts by moving the parts one by one. 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 movability problems.

One class of movability problems is the **directional separability problem**. The directional separability problem can be divided into three subproblems: the uni-directional separability problem; the multi-directional separability problem; and the all-directions separability problem.

The uni-directional separability problem can be expressed as follows: can a given set $S = \{P_1, P_2, \dots, P_M\}$ of M objects with n vertices each be moved by translating one object at a time in a fixed 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 such a separation of the object set is possible then an order, in which the objects can be translated, needs to be determined. (For more precise details of the definitions of *move*, *collide*, *translation order* see Section 1.3).

The multi-directional separability problem can be expressed as follows: can a given set $S = \{P_1, P_2, \dots, P_M\}$ of M objects with n vertices each be moved by translating one object at a time over an arbitrarily large distance and only once, in such a way that no object (when translated) will collide with any of the objects not yet moved? If such a separation of the object set is possible then an order in which the objects can be translated

needs to be determined. The direction of translation may differ from object to object and each object P_i , $1 \leq i \leq M$, is associated with direction d_i .

The **all-directions separability problem** can be expressed as follows: given an object Q , with m vertices, and a set $S = \{P_1, P_2, \dots, P_M\}$ of M objects with n vertices find all the directions in which Q can be translated, to infinity, with respect to the set S so that Q does not collide with any of the objects of S .

1.3. DIRECTIONAL SEPARABILITY DEFINITIONS

In this section some definitions and notation used in subsequent sections of this thesis are given.

Notation:

ab - a line segment connecting two points a and b .

$line(ab)$ - the infinite line that contains ab .

$direction(ab)$ - a direction parallel to $line(ab)$ and directed from a to b .

Definition: An object P can be ***moved*** from position S to position T if there exists a finite sequence of translations and rotations of P that carries P from S to T .

Definition: Let object P be moved during a time interval starting at t_1 and ending at t_2 . A ***collision*** between object P and an object Q occurs if there exists a time t , where $t_1 \leq t \leq t_2$, such that the interiors of P and Q intersect.

Definition: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M objects and let π denote a permutation on the indices $1, 2, \dots, M$ of S . A **permutation of S** , denoted as S_π , is the permutation $(P_{\pi_1}, P_{\pi_2}, \dots, P_{\pi_M})$ of the objects.

Definition: A *translation order* of S is a permutation S_π with the following property: for every i , (where $1 \leq i \leq M$), P_{π_i} does not collide with P_{π_j} , for $i < j \leq M$.

Definition: A *solution* to the uni-directional separability problem exists if there exists a permutation of S which is a translation order. A solution to the uni-directional translation order is denoted as *uni-directional translation order*.

Definition: An object P *blocks* object Q from moving in direction d if P intersects Q during this motion.

Definition: Two objects are *movably separable* if one of them can be moved, by a single translation, an arbitrarily large distance away from the other without colliding with it. If two objects A and B are not movably separable then A and B are said to *interlock*.

1.4 RELATED RESULTS

Guibas and Yao [GY] have shown that M *isothetic rectangles* (rectangles for which each side is parallel to the x or y axis) in the plane can be separated in any given direction and that this translation order can be found in $O(M \log M)$ (where M is the number of rectangles). They expanded their results and have shown that for any given direction a solution to the uni-directional separability problem in two dimensional space always exists when the objects are convex polygons. They presented an $O(M \log M)$ two-pass algorithm, where M is the number of the convex polygons. Ottmann and Widmayer [OW] described a one-pass algorithm to determine such a translation order with the same running time of $O(M \log M)$. Related work has been done by Rival and Urrutia [RU] on the correspondence between the separability problem of convex polygons in two dimensional space and ordered sets.

When simple polygons are considered, a translation order is not always guaranteed. Toussaint [T] presented an $O(\min(M^2n, Mn \log Mn))$ algorithm to detect whether a solution to the uni-directional separability problem exists and an $O(\min(M^3n, M^2n \log Mn))$ to determine a translation order. Chazelle et al. [COSW] gave an $O(Mn \log Mn)$ algorithm for detecting whether a solution to the uni-directional separability problem for rectilinear polygons in two dimensional space exists when the direction of translation is parallel to the x or y axis and an $O(Mn \log^2 Mn)$ algorithm for determining a translation order. Sack and Toussaint [ST] have observed that a set of polygons is uni-directional separable in all directions if, and only if, the convex hulls of the polygons do not intersect. Based on this observation one can determine in $O(Mn + M \log M + M \log n)$ time whether a set of M simple non-intersecting n -vertex polygons are uni-directional separable in all directions. Nurmi [N] extended Guibas and Yao's results to arbitrarily simple polygons in two-dimensional space and presented an $O(Mn \log Mn)$ algorithm for the uni-directional separability problem. Recently, Pollack, Sharir and Sifrony [PSS] gave an $O(mn \alpha(mn) \log m \log n)$ algorithm for separating two simple polygons P and Q , with m and n vertices, by a sequence of translations, where $\alpha(k)$ is the extremely slowly growing inverse Ackermann's function. The first result of this thesis is to give an $O(Mn + M \log M)$ algorithm for solving the uni-directional separability problem which is also shown to be optimal.

Not much work has been done on directional separability problems in three dimensional space in this context. Nurmi [N] has given an $O(K \log Mn)$ algorithm, where $Mn \leq K \leq M^2n^2$ for the uni-directional separability problem of simple polygonal faces and simple polyhedra. A second result of this thesis is to present a different method which reduces the complexity to $O(Mn + M \log M + K \log n + Un \log n)$ and $O(Mn)$ space, where $0 \leq K \leq M^2$ and $0 \leq U \leq K$, for polygonal faces and $O(M^2n^2)$ for polyhedra. A significant reduction in the time complexity is achieved when the polygonal faces or the polyhedra are convex.

The all-directions separability in two dimensional space was introduced by Sack and Toussaint [ST] in 1983. They gave an $O(mn)$ algorithm [ST] and later improved the result to $O((n+m) \log n)$ [ST1]. Toussaint [T1] also gave an alternative $O((n+m) \log n)$ algorithm for solving the problem. Recently, Sack [S], and Bhattacharya and Toussaint [BT], independently, gave an $O((n+m) \log \log (n+m))$ [$O((n+m) + \text{triangulation time})$] algorithm for this problem. In three dimensional space, Sack and Nurmi [SN] have given an $O(n + m)$ solution to the all-directions separability problem of two convex polyhedra.

The third result of this thesis solves the all-directions separability problem of simple polyhedra and simple polygonal faces in three dimensional space in $O(M^2 n^2)$ time. This algorithm is a worst case optimal.

1.5 GEOMETRIC TOOLS AND DEFINITIONS

In this section some geometrical definitions and tools used in subsequent sections of the thesis are given.

1.5.1 DEFINITIONS

Definition: A *polygon* is specified by a finite set of vertices $(v_1, v_2, \dots, v_n)$ listed in clockwise order. The vertices are specified by their cartesian coordinates and no three consecutive vertices are collinear. An edge, e_i , of a polygon is the line segment between the vertex v_i and the vertex $v_{(i \bmod n) + 1}$. An n -vertex polygon is an n -gon.

Definition: A polygon is *simple* if there is no pair of non-consecutive edges sharing a point. A *simple polygon* partitions the space into two distinct regions: the *interior* (bounded) and the *exterior* (unbounded) which are separated by the polygon (from The Jordan Curve Theorem). The *inside* of the polygon includes the interior of the polygon and the polygon boundary itself.

Definition: A simple polygon P is *convex* if, for any two points p_1 and p_2 in P , the segment p_1p_2 is entirely contained inside P .

Definition: A *polygonal chain* C_{ij} is a sequence of consecutive vertices $p_i, p_{i+1}, \dots, p_j$ of P . A chain is monotone if there exists a line L such that the projections of $p_i, p_{i+1}, \dots, p_j$ on L are ordered exactly the same as $p_i, p_{i+1}, \dots, p_j$.

Definition: A polygon P is *monotone* if there exists a line L such that the boundary of P can be partitioned into two chains C_{ij} and C_{ji} and each chain is monotone with respect to L .

Definition: A *polyhedron* in three dimensional space is a solid figure consisting of four or more planar faces (all polygons in two dimensional space), of which the pairs meet along an edge and three or more edges meet at a vertex.

Definition: A polyhedron P is *convex* if, for any two points p_1 and p_2 in P , the segment p_1p_2 is entirely contained inside P . A convex polyhedron is called a *polytope*.

1.5.2 PLANE SWEEP TECHNIQUE (LINE SWEEP)

The plane sweep technique is a powerful tool for solving geometrical problems, such as detecting intersections among line segments, constructing Voronoi diagrams and triangulation. In this technique a line is swept across the plane. Without loss of generality assume that the line is vertical (parallel to the y -axis) and is gradually swept from left to right.

Often, there are only a few positions which will cause a change in the information obtained by the sweep line. One example would be when the sweep line encounters an object in the plane. At each position the sweep line stops and updates the information

obtained by the line so far. The points (positions) at which the sweep line halts are called *event points*. The intersection of the sweep line with the problem data contains all the relevant information for the continuation of the sweep. This information is called the *sweep line status*.

Thus, there are two basic structures:

- (i) The *Event Point Schedule* - is a sequence of abscissae ordered from left to right, which defines the halting points (positions) of the sweep line. The Event Point Schedule may be dynamically updated. An example of this can be found in the algorithm by Bentley and Ottmann [BO] which finds all the intersection points of a set of line segments. Different data structures are used for different applications. In this thesis a priority queue is used as the data type of the Event Point Schedule.
- (ii) The *Sweep Line Status* - is a data structure which contains the relevant information about the intersection of the sweep line with the geometric structure being swept. The sweep line status is updated at each event point.

1.5.3 PRIORITY QUEUE

A priority queue is a queue in which the elements have a priority function defined for them. An example would be a case in which a doctor who, when faces with a line of people waiting in the emergency room, sees the patients according to the urgency of their medical situation rather than on a first-come-first-serve basis. The priority of an element is, in general, a member of some linearly ordered list (e.g. real numbers). Operations on a priority queue are: (i) Insert - this operation inserts a new element into its position in the queue according to its priority; (ii) DeleteMin - this operation returns the element of the queue with the minimum priority and then discards it from the queue; (iii) Construct - this operation constructs a priority queue out of a set of elements; and (iv) DeleteMax - this operation returns the element of the queue with the maximum priority and then discards it from the queue.

In this thesis the priority queue is implemented by using a Min-heap or a Max-heap. Building a priority queue as a Min-heap or Max-heap can be done in $O(n)$ time and space. Insert and DeleteMin operations are done in $O(\log n)$ time where n is the number of elements in the heap [AHU].

1.5.4 MODEL OF COMPUTATION AND COMPUTATIONAL COMPLEXITY

The model of computation used in this thesis for measuring the complexity of the algorithm is the real RAM (a random access machine) with infinite precision and real number arithmetic. In the RAM the following instructions and function are available: (i) Arithmetic operations: +, -, *, /; (ii) Comparison operations between real numbers: <, ≤, >, ≥, =, ≠; (iii) Functions: root, trigonometric functions (sin, tan et cetera), LOG and EXP. The access time and the execution time of each of the instructions and the functions mentioned above are considered to be constant ($O(1)$). For further details see the description of the RAM in [AHU1].

In this thesis, the algorithms are evaluated by comparing the rate of growth of the running time and storage space required to solve the problem as a function of n , where n is the size of input. Hence, the algorithms are compared using asymptotic time complexity. The standard "big-Oh" and "big-omega" notation, as described by Aho, Hopcroft and Ullman [AHU] are used.

Definition: $f(n) = O(g(n))$ if there exist a positive $c_0 > 0 \in \mathbb{R}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \leq c_0 g(n)$ for all $n > n_0$.

Definition: $f(n) = \Omega(g(n))$ if there exist a positive $c_0 \in \mathbb{R}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \geq c_0 g(n)$ for infinite number of values of n where $n > n_0$.

An algorithm which solves a problem of size n in $f(n)$ time proves an upper bound of $O(f(n))$ for the time complexity of the problem. If a problem cannot be solved by any algorithm in less than $f(n)$ time then $f(n)$ is the lower bound of the problem and is denoted by $\Omega(f(n))$.

Definition: An algorithm is *optimal* if, and only if, $f(n) = O(g(n)) = \Omega(g(n))$, i.e. the upper bound equals the lower bound (within a constant factor).

Definition: The *worst case complexity* of an algorithm, for a given size of the input, is the maximum complexity achievable over all possible inputs of that size.

Definition: The *average case complexity* of an algorithm, for a given size of the input, is the average complexity over all possible inputs of that size. This complexity measure presumes a probabilistic model of the input.

Two techniques, which are used in this thesis to obtain lower bounds, are: (i) *adversary argument*; (ii) *transformation of problems*.

(i) *Adversary argument* - Let U be a problem and let C be a class of incorrect programs, for solving U . Each program in C has a worst case running time of $O(f(n))$ steps for an input of size n . An *adversary* is a program α , along with a polynomial P , such that α can compute a counter example to any program V in C with $P(f(n))$ steps. Examples of steps of computation are 'accessing' or 'examining' a vertex of a polygon.

(ii) *Transformation of problems* - This technique is used to relate the difficulty of a problem to another problem of known difficulty. Specifically, suppose that the input to problem A is converted (transformed) into a suitable input to problem B . Problem B is then solved, and the output of problem B is then transformed into a correct solution of problem A . If problem A is known to require at least $\Omega(f(n))$ time and A is $O(\tau(n))$ transferable to B and the output of B is $O(\psi(n))$ transferable to a correct output of A then B

requires at least $\Omega(f(n)) - O(\tau(n)) - O(\psi(n))$. If an algorithm to solve problem B beats this bound then a method which solves problem A that runs faster than the lower bound is found - a contradiction.

A more detailed description of measuring the complexity of algorithm can be found in [AHU,PS]. In this thesis, unless otherwise stated, all the complexities are worst case complexities and are functions of the number of objects and the number of vertices in an object (given as input).

1.5.5 THE DOUBLY-CONNECTED-EDGE-LIST (DCEL)

The doubly-connected-edge-list (DCEL) is particularly suited to represent a planar graph embedded in the plane [MP]. A planar embedding of a planar graph $G = (V,E)$ is a mapping of each vertex in V to a point in the plane and each edge in E to a simple curve between the two images of its vertices, such that no two images of edges share a point except perhaps at their endpoints.

Let $V = \{v_1, v_2, \dots, v_n\}$ and $E = \{e_1, e_2, \dots, e_m\}$. The DCEL is represented by an array of edge-nodes. There is a one-to-one correspondence between an edge and an edge-node, and, therefore, the DCEL can be represented as an array of size M edge-nodes. Each edge-node is a record consisting of six fields: four information fields V_1, V_2, F_1 and F_2 and two pointers P_1 and P_2 . The meanings of these fields are as follows: V_1 contains the origin of the edge, V_2 contains the terminus of the edge. The fields F_1 and F_2 contain the names of the faces which lie on the left and on the right of the edge, respectively, when the orientation of the edge is from V_1 to V_2 . The pointer P_1 (P_2) points to the edge-node containing the first edge encountered after edge (V_1, V_2) which proceeds counter-clockwise around V_1 (V_2). An example of a fragment of the graph and the corresponding DCEL is shown in Figure 1.5.1.

Figure 1.5.1 An example of DCEL

The DCEL array easily enables one to obtain a list of all edges that incident on a vertex or all edges enclosing a given face. The following procedure is given as an example of how to obtain a list of all edges that incident on a given vertex j .

Procedure Vertex (j)

Begin

$a := HV[j];$

$a_0 := a;$

Print(a);

if (DCEL[a]. $V_1 = j$) then $a := \text{DCEL}[a].P_1$ else DCEL[a]. P_2 ;

while ($a \neq a_0$) do

begin

Print(a);

if (DCEL[a]. $V_1 = j$) then $a := \text{DCEL}[a].P_1$ else DCEL[a]. P_2 ;

end;

end;

$HV[j]$ is an array that contains an appearance of vertex j in the DCEL. In a similar way one can find all edges that enclose a face.

1.5.6 THE SPHERE SWEEP

The sphere sweep technique [SN] is a modification of the plane sweep technique discussed earlier in Section 1.5.2. The sweep line is an arc on the sphere that is a half circle whose endpoints are fixed in the poles of the sphere (the intersection of the x_2 -axis and the sphere). The sweep begins at the 0° meridian and advances 360° around the sphere. The two basic data structures, used by Sack and Nurmi, are:

(i) the 'longitude structure' which corresponds to the Event Point Schedule (see Plane Sweep); and (ii) the 'latitude structure' which corresponds to the Line Status (see Plane Sweep).

1.6 THESIS OUTLINE

This thesis is divided into six chapters.

Chapter 2 reviews and elaborates on some of the important algorithms and techniques used in this thesis.

In Chapter 3 the uni-directional separability problem of simple polygons in two dimensional space is discussed. Nurmi [N] posed as an open problem the question of whether there exists an $O(Mn + M \log M)$ algorithm for the uni-directional separability problem of M simple non-intersecting n -vertex polygons. In this chapter an optimal $O(Mn + M \log M)$ algorithm is given for solving this problem. In addition, several lower bounds are given: (i) an $\Omega(Mn + M \log M)$ lower bound for detecting whether a solution to the uni-directional separability problem and for determining a translation order of a set S of M simple n -vertex polygons; (ii) an $\Omega(Mn)$ lower bound for detecting the existence of a pair of polygons, among a set S of M simple, n -vertex polygons, monotone with respect to the same direction, that intersect.

Directional separability in three dimensional space is discussed in Chapter 4. Solutions to the uni-directional separability problem of polygonal faces and polyhedra are given. It is

also shown that as few as six line segments can be arranged such that there is no solution to the uni-directional separability problem. An $O(M \log M)$ algorithm is given for the multi-directional separability problem of a set S of M line segments. A nearly worst case optimal algorithm is given for the all-directions separability problem of two simple polyhedra.

In Chapter 5 several applications in which directional separability problems could be used are given.

Finally, Chapter 6 concludes and summarizes the results and presents a number of open problems for further study and research.

CHAPTER 2

DIRECTIONAL SEPARABILITY SURVEY

In this section selected algorithms and techniques, used in this thesis, are explained in more detail. In Section 2.1, a comparison mechanism used by Guibas and Yao, Ottmann and Widmayer, and by Nurmi is discussed. In Section 2.2, the algorithms developed by these authors for solving the uni-directional separability problem of polygons in two dimensional space, will be presented. In Section 2.3, the algorithm developed by Nurmi and Sack for solving the all-directions separability problem of convex polyhedra in three dimensional space, is explained.

2.1 THE DOMINANCE RELATION

The uni-directional separability problem can be solved as follows. First, construct a relation on the set of polygons to determine for each pair of polygons which of the two polygons must be translated first. Second, examine the transitive closure of the relation for asymmetry by testing if any *cycles* occur in the digraph representing the transitive closure of the relation. For example: let A, B, C be three polygons and assume that A must be translated before B , B before C and C before A , then a *cycle* consisting of A, B, C, A occurs. In case the transitive closure of the relation is: (i) symmetrical, then a cycle occurs and it is not possible to translate the objects; (ii) asymmetrical, then it defines a partial order among the objects. Third, if the relation is asymmetric then it is extended from a partial order to a total order.

This type of solution relies heavily on the type of relation defined on the set of polygons. Guibas and Yao [GY] define two relations: **dominance** (*dom*) and **immediate dominance** (*idom*).

The *dom* (dominance) relation - Informally, if two objects *A* and *B* are related by the *dom* relation as $A \text{ dom } B$ then object *A* must be moved before object *B*. Formally, $A \text{ dom } B$ means that if *B* is moved before *A* then either *B* will collide with *A* or there exists a sequence of objects $O_1, O_2, \dots, O_n$ such that O_i collides with O_{i+1} for $i = 1..n-1$ ($n > 2$) and *B* collides with O_1 and O_n collides with *A* (see Figure 2.1.1).

Figure 2.1.1 $A \text{ dom } B$ and $A \text{ dom } C$ but not $A \text{ dom } D$

The *idom* (immediate dominance) relation - Informally, if two objects A and B are related by an *idom* relation as $A \text{ idom } B$ then there exists a point p on B that, by looking from p in the direction of the translation, it is possible to see a point of object A . Formally, $A \text{ idom } B$ 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 (see Figure 2.1.2).

Figure 2.1.2 $A \text{ idom } B$ and $A \text{ idom } D$ but not $A \text{ idom } C$

The transitive closure of *idom* is denoted by idom^+ . If two objects A and B are related by the idom^+ relation as $A \text{ idom}^+ B$ then there exists a sequence of objects $B = O_1, O_2, \dots, O_n = A$ such that $O_{i+1} \text{ idom}^+ O_i$ for $i = 1..n-1$ ($n > 2$). $O_2, \dots, O_{n-1}$ may be empty. When the direction of translation is the positive x -axis, then idom^+ is denoted as idom^+_x , which refers to an immediate dominance in the x direction.

For a given set of objects it may be that not all objects are *comparable* by $idom^+$. Two objects A and B are *incomparable* in $idom^+$ if neither $A idom^+ B$ nor $B idom^+ A$.

Example 2.1.1: Consider the set of objects as two line segments s_i and s_j and let the direction of translation d be the positive x -axis. If the y -value of the upper endpoint of s_i is less than the y -value of the lower endpoint of s_j then s_i and s_j are *incomparable* by $idom^+$ (see Figure 2.1.3). ♦

Figure 2.1.3 Line segments s_i and s_j are *incomparable* by $idom^+$.

In cases where the objects are incomparable in $idom^+$, an additional relation is needed in order to achieve a relation between the objects. Without loss of generality the positive x -axis will be the direction of translation (see Section 2.2 for an explanation of why such an assumption is possible).

If two objects A and B are incomparable in $ixdom^+$ then there exists a line L parallel to the x -axis such that A (B) is above L and B (A) is below L . The line L is called a *y-separator*. In such a case, A and B are *y-comparable* and $A \text{ dom } B$ if A is *below* B .

Example 2.1.2: Consider the two line segments s_i and s_j of Example 2.1.1 (the direction of translation is the positive x -axis). Line segments s_i and s_j are incomparable in $ixdom^+$. A line L' parallel to the x -axis through the upper endpoint of s_i separates the plane into two half-planes. Line segment s_i is contained in the bottom half-plane (denoted as T_b) and line segment s_j is contained in the upper half plane (denoted as T_u) which yields s_i *below* s_j . Thus, $s_i \text{ dom } s_j$ if and only if $(s_i \text{ ixdom}^+ s_j)$ or $(\text{not } (s_j \text{ ixdom}^+ s_i) \text{ and } (s_i \text{ below } s_j))$ (see Figure 2.1.4). ♦

Figure 2.1.4 Line segment s_i is *below* line segment s_j

The *below* relation is used by Ottmann and Widmayer [O'V] in those cases when the relation $ixdom^+$ is not sufficient. They construct the relation dom using the relations $ixdom^+$ and *below*. Two objects A and B are related by the dom relation as $A dom B$ if and only if $(A ixdom^+ B)$ or $(\text{not } (B ixdom^+ A) \text{ and } (A below B))$. Guibas and Yao [GY] used the term *consulting* to describe this kind of relation. The dom relation reads as $ixdom^+ consulting below$ and is denoted as $ixdom^+|below$. The relation $dom=ixdom^+|below$ enables one to obtain a relation between any given pair of objects.

2.2 SEPARABILITY IN TWO DIMENSIONAL SPACE

This section reviews solutions to the uni-directional separability problem for polygonal objects in two-dimensional space. Section 2.2.1 presents two algorithms, one by Guibas and Yao and one by Ottmann and Widmayer, for translating a set of convex polygons (line segments). Section 2.2.2 presents the generalization by Nurmi to arbitrary simple polygons.

Without loss of generality, in this section, the positive x -axis will be the direction of translation. If the direction of translation is not the positive x -axis then, in time linear to the number of objects, it is possible to rotate the objects so that the direction of translation is the positive x -axis. This rotation does not affect the overall complexity of $O(M \log M)$ of the algorithms (where M is the number of objects to be translated).

2.2.1 SEPARABILITY OF CONVEX POLYGONS (LINE SEGMENTS)

2.2.1.1 GUIBAS AND YAO'S ALGORITHM

Guibas and Yao [GY] have shown that the separability problem for a set of convex objects in two dimensional space can be reduced to the following problem: given a set $S=\{s_1, s_2, \dots, s_M\}$ of M non intersecting line segments in the plane, find an ordering for the

segments such that the line segments can be translated, one by one, to the right (in the direction of the positive x -axis) without colliding with the line segments not yet translated.

They have defined the relation dom as $ixdom^+$ ($dom = ixdom^+$) in their algorithm. The digraph representing the relation $dom = ixdom^+$ is proven to be acyclic.

The algorithm uses two passes through the data. In the first pass, the algorithm computes a partial order for the set of line segments. In the second pass, the algorithm performs a topological sort to obtain a linear extension (total order) of the partial order. The algorithm performs a line sweep from top to bottom on the endpoints of the line segments. Whenever the sweep line encounters an upper (lower) endpoint of a line segment the line segment becomes (ceases to be) *active*. The data structure, used in the algorithm, is a balanced binary search tree. The nodes of the balanced binary search tree are the currently active line segments. An inorder traversal on the tree yields the $ixdom^+$ relation among the current active line segments and, therefore, a translation order of the line segments contained in the tree.

In the first pass of the algorithm, all endpoints of the line segments are sorted by decreasing y -value. A line-sweep technique, from top to bottom, is then employed on the sorted endpoints. Every time the sweep line encounters an upper endpoint of a line segment t , t becomes active and is inserted into the balanced binary search tree. During the insertion, the two immediate segments to the left and right of t , t' and t'' respectively, are found and $ixdom^+$ relations between t and t' and between t and t'' are output as $t'' ixdom^+ t$ and $t ixdom^+ t'$ (see Figure 2.2.1a). Whenever a bottom endpoint of segment s is encountered, s ceases to be active and is deleted from the binary tree. During the deletion, the two immediate segments to the left and right of s , s' and s'' respectively, are found and an $ixdom^+$ relation between s' and s'' is output as $s'' ixdom^+ s'$ (see Figure 2.2.1b). The output of the first pass is a partial order of the line segments. The partial order occurs because not all line segments are comparable in $ixdom^+$. In the second pass a topological sort [K] is done on the partial order and a total order is achieved.

a. Insertion: $t' \text{ ixdom}^+ t$ and $t \text{ ixdom}^+ t''$

b. Deletion: $s'' \text{ ixdom}^+ s'$

Figure 2.2.1 The relations that are produced during insertion and deletion

Algorithm 2.2.1: Guibas and Yao (Translating a set of line segments)

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

Input: A set of M line segments.

Output: A translation order of the set of line segments.

begin

{ first pass }

1. Construct a priority queue E which consists of all endpoints of the polygons in a descending order of their y -values;
2. Initialize the balanced binary search tree;
3. While ($E \neq \emptyset$)

begin

$p = \text{DeleteMax}(E)$;

if p is an upper endpoint of segment s then

begin

Insert s into the tree;

find the predecessor s' and the successor s'' of s in the tree;

if (s' exists) then

begin { a partial order between s and s' is output }


```

        Output an ixdom between  $s$  and  $s'$ ; {as  $s$  ixdom  $s'$ }
    end; {if}
    if ( $s''$  exists) then
    begin      { a partial order between  $s$  and  $s''$  is output }
        Output an ixdom between  $s$  and  $s''$ ; {as  $s''$  ixdom  $s$ }
    end; {if}
end; {if}
else { $p$  is a bottom endpoint of  $s$ }
begin
    find the predecessor  $s'$  and the successor  $s''$  of  $s$  in the tree;
    if ( $s'$  and  $s''$  exist) then
    begin      { a partial order between  $s'$  and  $s''$  is output }
        Output an ixdom between  $s'$  and  $s''$ ; {as  $s''$  ixdom  $s'$ }
    end; {if}
    delete  $s$  from the tree;
end; {else}
end; {while}
{ Second pass}
4. Do a topological sort on the partial order;
end;

```

The running time of Algorithm 2.2.1 is $O(M \log M)$ time and $O(M)$ space. There are $2M$ points in the queue (M upper endpoints and M lower endpoints). Hence, the construction of the priority queue can be done in $O(M)$ time and $O(M)$ space and the DeleteMax operation on the priority queue costs $O(\log M)$ time [AHU]. In each iteration of the while loop in Line 3 (the first pass) the first element of the priority queue is extracted and a line segment is either deleted from the binary balanced search tree or inserted into the balanced binary search tree. Each of the insertions and the deletions into and from the balanced binary search tree costs, in the worst case, $O(\log M)$. Hence, the cost of an iteration of the while loop is $O(\log M)$ time. There are $2M$ iterations and therefore, the total cost of the while loop is $O(M \log M)$ time and $O(M)$ space.

In every iteration of the while loop (Line 3), either an upper endpoint or a lower endpoint is extracted from the priority queue. Every time an upper endpoint is extracted from the queue at most two *ixdom* relations are produced. There are M upper endpoints and therefore, at most $2M$ *ixdom* relations are produced. Every time a lower endpoint is extracted at most one *ixdom* relation is produced. There are M lower endpoints and, therefore, at most M *ixdom* relations are produced. Thus, the output from the first pass is at most $3M$. The time complexity of topological sort is linear with respect to the number of relations between the elements $[K]$. The input to the second pass is at most $3M$, the output of the first pass and, therefore, the topological sort requires $O(M)$ time and $O(M)$ space.

2.2.1.2 OTTMANN AND WIDMAYER'S ALGORITHM

Ottmann and Widmayer [OW] have developed a one-pass algorithm instead of the two-pass algorithm by Guibas and Yao. As mentioned in Section 2.1 the relation *dom* is *ixdom*⁺|*below*. The relation *dom* = *ixdom*⁺|*below* is proven to be asymmetric.

The algorithm performs a plane-sweep from top to bottom. It stops at each endpoint of the line segments and maintains a data structure that contains all line segments encountered thus far by the sweep line. Whenever the sweep line encounters an upper (bottom) endpoint of a line segment, the line segment becomes (ceases to be) *active*.

The data structure consists of a basic structure and an overlay structure. The basic structure is a doubly linked list with head and tail pointers. This list contains the reverse order of *ixdom*⁺|*below*. The overlay structure is a binary tree. Its leaves are the current active segments and the internal nodes containing the equation of the right-most child of the left subtree as a key.

Whenever the scan line stops at an upper endpoint, a new segment is inserted into the overall structure. The segment is inserted into the tree (the overlay structure) before its successor active segment (in an inorder traversal of the tree) and into the linked list (the basic structure) before its successor from left to right. Whenever a lower endpoint is

encountered the segment ceases to be active and it is deleted from the tree, but not from the linked list.

Algorithm 2.2.2: Translating a set of line segments in a single pass.

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

Input: A set of M line segments.

Output: A translation order of the set of line segments.

begin

1. Construct a priority queue E which consists of all the endpoints of the polygons in descending order of their y -values;
2. Initiate the overlay and the basic structures.
3. While ($E \neq \emptyset$)
 - begin
 - $p = \text{DeleteMax}(E)$
 - if (p is an upper endpoint of segment s) then
 - begin
 - find the successor s'' of s in the tree;
 - if (s'' exists) then
 - begin
 - insert s as the new predecessor of s'' in the tree;
 - insert s as the new predecessor of s'' in the linked list;
 - end; {if}
 - else
 - begin
 - insert s as the right most child (element) in the tree;
 - insert s as the new final element in the linked list;
 - end; {else}
 - end; {if}
 - else { p is a bottom endpoint of s }
 - begin
 - delete s from the tree (but not from the linked list);
 - end; {else}

```

 $p = \text{DeleteMax}(E);$ 
if ( $p$  is an upper endpoint of segment  $e_{ik}$ ) then
begin
  Insert  $e_{ik}$  into the tree;
  find the predecessor  $e_{jr}$  and the successor  $e_{uv}$  of  $e_{ik}$  in the tree;
  if ( $e_{jr}$  exists) then
  begin
    if ( $i \neq j$ ) then begin      { a partial order between  $P_i$  and  $P_j$  is output }
      Output an ixdom between  $P_i$  and  $P_j$ ; { as  $P_i$  ixdom  $P_j$  }
    end; {if}
  end; {if}
  if ( $e_{uv}$  exists) then
  begin
    if ( $i \neq u$ ) then begin      { a partial order between  $P_i$  and  $P_u$  is output }
      Output an ixdom between  $P_i$  and  $P_u$ ; { as  $P_u$  ixdom  $P_i$  }
    end; {if}
  end; {if}
end; {if}
else begin { $p$  is a bottom endpoint of  $e_{ik}$ }
  find the predecessor  $e_{jr}$  and the successor  $e_{uv}$  of  $e_{ik}$  in the tree;
  if ( $e_{jr}$  and  $e_{uv}$  exist) then begin
    if ( $j \neq u$ ) then begin      { a partial order between  $P_u$  and  $P_j$  is output }
      Output an ixdom between  $P_j$  and  $P_u$ ; { as  $P_u$  ixdom  $P_j$  }
    end; {if}
  end; {if}
  delete  $e_{ik}$  from the tree;
end; {else}
end; {end of first pass}
{Second pass}
5. Create a directed graph out of the partial order;
6. Execute a DFS algorithm on the partial order;
7. If no cycles occurs in step 6 then begin
  Do a topological sort on the partial order; { the second pass }
end; {if}
end;

```

Figure 2.2.2 Neither A nor B can be translated as $A \text{ dom } B$ and $A \text{ dom } A$.

Nurmi [N] has reduced the separability problem for a set of simple polygons to the separability problem of a set of line segments. Polygon P_i collides with polygon P_j during its translation if, and only if, their boundaries collide during the translation. The boundaries of the polygons are simply a set of edges (line segments). Hence, if all the edges of P_i can be translated before all the edges of P_j then polygon P_i can be translated before polygon P_j . Nurmi has represented the polygons by their sides (edges). Namely, each polygon P_i ($i = 1..M$) is presented by its edges $e_{i1}, e_{i2}, \dots, e_{in}$. Each of the edges is associated with the information on whether the interior of the polygon lies to the left or to the right of the edge with respect to the direction of translation.

Example 2.2.2: Consider the set O of Example 2.2.1. Polygon A is represented by e_{A1}, e_{A2}, e_{A3} and e_{A4} ($e_{A1} = v_1v_2$, $e_{A2} = v_2v_3$, $e_{A3} = v_3v_4$, $e_{A4} = v_4v_1$) and the information associated with the edges is left, right, left and right respectively (see Figure 2.2.3). ♦

Figure 2.2.3 Polygon A is represented by its edges

Nurmi has used Algorithm 2.2.1 with the following modifications: (i) whenever Algorithm 2.2.1 outputs a *dom* relation between two edges (as $e_{ik} \text{ dom } e_{jr}$) of two distinct polygons a *dom* relation is output between the two polygons P_i and P_j as $P_i \text{ dom } P_j$. (ii) whenever Algorithm 2.2.1 outputs a *dom* relation between two edges of the same polygons (e.g. $e_{ik} \text{ dom } e_{ij}$) the algorithm does not output anything. The *dom* relations between the edges is always asymmetric. However, this is not always the case between the polygons (see Example 2.2.1). Thus, an additional step is needed to determine whether a cycle

amongst the polygons has occurred. The *dom* relations produced by the algorithm can be represented as a directed graph, where the vertices are the polygons and the edges are the relation between the polygons. A directed edge in the graph between two polygons A and B , and directed from A to B , exists if and only if $B \text{ idom}^+ A$ (see Example 2.2.3). The detection of the cycles is done by a Depth First Search algorithm (DFS) [Ta] on the partial order. If no cycles occur then a topological sort can be done.

Example 2.2.3: Let A, B, C and D be four polygons (see Figure 2.2.4). The relations among the polygons are : $A \text{ idom} D, A \text{ idom} C, C \text{ idom} B, B \text{ idom} A$ and $C \text{ idom} D$. The directed graph that is constructed out of this partial order has four vertices A, B, C and D and five edges: (i) edge DA directed from D to A . (ii) edge CA directed from C to A . (iii) edge BC directed from B to C . (iv) edge AB directed from A to B and (v) edge DC which is directed from D to C (see Figure 2.2.5). ♦

Figure 2.2.4 Polygons A, B, C and D of Example 2.2.3

Figure 2.2.5 The digraph produced out of the partial order of the polygons in Figure 2.2.3.

Algorithm 2.2.3: Nurmi (Simple polygons in the plane)

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

Input: A set of M simple polygons.

Output: A translation order of the set of polygons if one exists.

begin

{ first pass }

1. For $i := 1$ to M do begin

Decompose polygons P_i to a set of edges $\{e_{i1}, e_{i2}, \dots, e_{in}\}$ and associate each edge with the information on whether the polygon is to the left or to the right of the edge with respect to the direction of translation (the positive x -axis).

end; { for }

2. Construct a priority queue E which consists of all the endpoints of the edges in a descending order of their y -values;

3. Initialize the binary balanced search tree;

4. While $(E \neq \emptyset)$ { first pass }

begin


```

 $p = \text{DeleteMax}(E);$ 
if ( $p$  is an upper endpoint of segment  $e_{ik}$ ) then
begin
  Insert  $e_{ik}$  into the tree;
  find the predecessor  $e_{jr}$  and the successor  $e_{uv}$  of  $e_{ik}$  in the tree;
  if ( $e_{jr}$  exists) then
  begin
    if ( $i \neq j$ ) then begin      { a partial order between  $P_i$  and  $P_j$  is output }
      Output an ixdom between  $P_i$  and  $P_j$ ; { as  $P_i$  ixdom  $P_j$  }
    end; {if}
  end; {if}
  if ( $e_{uv}$  exists) then
  begin
    if ( $i \neq u$ ) then begin      { a partial order between  $P_i$  and  $P_u$  is output }
      Output an ixdom between  $P_i$  and  $P_u$ ; { as  $P_u$  ixdom  $P_i$  }
    end; {if}
  end; {if}
end; {if}
else begin { $p$  is a bottom endpoint of  $e_{ik}$ }
  find the predecessor  $e_{jr}$  and the successor  $e_{uv}$  of  $e_{ik}$  in the tree;
  if ( $e_{jr}$  and  $e_{uv}$  exist) then begin
    if ( $j \neq u$ ) then begin      { a partial order between  $P_u$  and  $P_j$  is output }
      Output an ixdom between  $P_j$  and  $P_u$ ; { as  $P_u$  ixdom  $P_j$  }
    end; {if}
  end; {if}
  delete  $e_{ik}$  from the tree;
end; {else}
end; {end of first pass}
{Second pass}
5. Create a directed graph out of the partial order;
6. Execute a DFS algorithm on the partial order;
7. If no cycles occurs in step 6 then begin
  Do a topological sort on the partial order; { the second pass }
end; {if}
end;

```

The running time of Algorithm 2.2.3 is $O(Mn \log Mn)$ time and $O(Mn)$ space. The total number of edges in the polygons is Mn and, therefore, there are $2Mn$ points in the queue (Mn upper endpoints and Mn lower endpoints). The construction of the priority queue can be done in $O(Mn)$ time and $O(Mn)$ space and the DeleteMax operation on the priority queue costs $O(\log Mn)$ time [AHU], where M is the number of polygons and n is the number of edges in a polygon. In each iteration of the while loop in Line 3 (the first pass) the first element of the priority queue is extracted and a line segment (edge) is either deleted from the binary balanced search tree or inserted into the binary balanced search tree. Each of the insertions and the deletions into and from the balanced binary search tree costs, in the worst case, $O(\log Mn)$. Hence, the cost of an iteration of the while loop is $O(\log Mn)$ time. There are $2Mn$ iterations and, therefore, the total cost of the while loop is $O(Mn \log Mn)$ time and $O(Mn)$ space. In every iteration of the while loop body (Line 3) either an upper endpoint or a lower endpoint is extracted from the priority queue. Every time an upper endpoint is extracted from the queue, at most two *ixdom* relations are produced. There are Mn upper endpoints and, therefore, at most $2Mn$ *ixdom* relations are produced. Every time a lower endpoint is extracted, at most one *ixdom* relation is produced. There are Mn lower endpoints and, therefore, at most Mn *ixdom* relations are produced. Thus, the output from the first pass is at most $3Mn$. The time complexity of a topological sort algorithm [K] and of a DFS algorithm [Ta] are linear with respect to the number of relations between the elements. The input to the second pass is $3Mn$ (output of the first pass) and, therefore, the topological sort requires $O(Mn)$ time and $O(Mn)$ space.

2.3 ALL-DIRECTIONS SEPARABILITY OF TWO POLYTOPES

In this section the algorithm by Sack and Nurmi for the all-directions separability of two polytopes is described. The time complexity of this algorithm is $O(n)$, which is a

considerable improvement over the previous $O(n^2 \log n)$ algorithm by Toussaint [T], where n is the number of vertices of the two polytopes.

Let A and B be two disjoint polytopes and P a plane in three dimensional space.

Definition: P is *extreme* if P contains an edge of A (B) and a vertex of B (A), and A and B are contained in one of the closed halfspaces determined by P .

Definition: P *supports* A and B if $A \cap P \neq \emptyset$, $B \cap P \neq \emptyset$, and A and B are contained in the same halfspace determined by P . If P is also an extreme plane then P is an *extreme supporting plane* (see Figure 2.3.1(a)).

Definition: P *separates* A and B if A and B are contained in the different closed halfspaces determined by P . If P is, in addition, an extreme plane then P is an *extreme separating plane* (see Figure 2.3.1(b)).

Figure 2.3.1 (a) An extreme supporting plane of A and B . (b) An extreme separating plane of A and B .

The idea of the algorithm is first to compute all *invalid directions* γ in which A , when translated in direction γ , will collide with B . When all invalid directions γ have been found, then the *valid directions*, in which A can be translated with respect to B , are all directions d in the complement of the intersection of all invalid directions with a unit sphere.

This algorithm is based on the merging step of the three dimensional convex hull algorithm by Preparata and Hong [PH]. The polytopes are represented as a doubly-connected-edge-list. In the merge step of the algorithm by Preparata and Hong, the "cylinder faces" of the convex hull of the two polytopes are found (the two polytopes were generated as a result of the divide and conquer technique). The "cylinder faces" determine the extreme supporting planes of the polytopes. The extreme supporting planes are found as follows: first two vertices, one on each polytope, are found such that these vertices belong to an extreme supporting plane; second, by traveling on the DCEL, the extreme support vertices are found. A precondition of the merge step is the existence of a plane S such that the projections of the two polytopes on S do not intersect.

Sack and Nurmi's algorithm is based on the merge step of Preparata and Hong. Instead of computing the extreme supporting planes, as in Preparata and Hong's algorithm, the extreme separating planes are computed. First, a plane S that separates A and B is found by solving a 3-variable linear programming [PS,M]. The projections of A and B on a plane T , perpendicular to S , do not intersect because the line $S \cap T$ separates the projections. The existence of T fulfils the precondition of the merge step in Preparata and Hong's algorithm. Two vertices, one on A and one on B are then found such that the vertices belong to an extreme separating plane. The extreme separating planes are found in the same way as in Preparata and Hong's algorithm: i.e. by traveling on the DCEL. When all extreme separating planes are found they are translated to the origin and the intersection of the translated planes is found. The intersection of the translated planes forms a half open polyhedron denoted as a *cone*. The cone is all invalid directions γ .

Algorithm 2.3.1: All-directions separability**Input:** Two polytopes A and B with m and n vertices respectively.**Output:** All directions in which A can be separated with respect to B .

begin

1. Find a plane S that separates A and B ;
 2. Find the convex hull of the projections of A and B on a plane T perpendicular to S ;
 3. Find two vertices, one on A and one on B , such that the vertices belong to an extreme separating plane of A and B ;
 4. Compute all extreme separating planes P_i ($i = 1, 2, \dots, k$) of A and B . Let H_i denote the halfspace which is determined by P_i and which contains B ;
 5. For $i := 1$ to k do
 - begin
 - Translate P_i to the origin; denote the corresponding translated halfspace as H_i^0 ;
 - end;
 6. Compute the intersection of the planes H_i^0 ($i = 1, 2, \dots, k$). Denote this intersection as $C(A, B)$;
 7. Find the complement of the intersection of $C(A, B)$ with a unit sphere;
- end; {algorithm}

The running time of Algorithm 2.3.1 is $O(n + m)$. Finding a separating plane S of Line 1, between two non-intersecting polytopes, can be done in $O(n + m)$ time by solving a three variable linear programming of $n + m$ constraints [PS,M]. Computing the convex hull of the projection of A and B can be done on $O(n + m)$ time. In Line 3, the two vertices, one on A and one on B , can be found in $O(n + m)$ time [SN]. Computing all separating planes, Line 4, can be done in $O(n + m)$ time using the merge step of Preparata and Hong's algorithm. The translation of at most $n + m$ planes can be done in $O(n + m)$ time. The intersection of the translated planes can be found in $O(n + m)$ time because the planes are cyclically ordered [SN]. Thus, the total complexity is $O(n + m)$ time.

CHAPTER 3

TWO DIMENSIONAL SPACE

3.1 INTRODUCTION

As mentioned in Section 2.2 Guibas and Yao [GY] have shown that for any given direction a solution to the uni-directional separability problem for convex polygons, in two dimensional space, always exists. This property may not hold when the objects are arbitrarily simple polygons. It may be that two polygons can be separated (by a single translation) in a direction d_1 but cannot be separated (by a single translation) in some other direction d_2 (see Example 3.1.1).

Example 3.1.1: Let $P = \{(1,2), (3,5), (5,2), (3,6)\}$ and $Q = \{(2,1), (3,3), (4,1)\}$ be two simple non-intersecting polygons (see Figure 3.1.1). Let d_1 be the positive y -direction and let d_2 be the positive x -direction. If P is to be translated first in direction d_2 then P will collide with Q and vice versa. If P and Q are to be translated in direction d_1 then P can be translated first and, therefore, P and Q are separable in direction d_1 (see Figure 3.1.1). ♦

Figure 3.1.1 P will collide with Q if P is to be translated in direction d_1 .

Nurmi [N] has posed, as an open problem, the question as to whether an $O(Mn + M \log M)$ algorithm, that solves the uni-directional separability problem for a set of M simple non-intersecting n -vertex polygons, exists. This section will answer that question affirmatively.

In Section 3.2, preliminary definitions and results are presented. In Section 3.3, the *dom* (dominance) relation, which is used in the algorithms, is presented. The algorithms are presented in Section 3.4. In Section 3.5, a proof of correctness is given. Section 3.6 presents a tight lower bound for this uni-directional separability problem.

3.2 PRELIMINARY DEFINITIONS AND RESULTS

Definition: The *visibility hull* of a simple polygon P , with respect to a direction d , is the polygon obtained by taking the union of P and all line segments ab , for $a, b \in P$, which are parallel to the direction d . The *visibility hull* of P with respect to a direction d is denoted as $VH(P, d)$ (see Figure 3.2.1). The $VH(P, d)$ is a polygon monotone with respect to direction $d + \pi/2$.

Definition: The *pockets* of the $VH(P, d)$ are the polygons obtained by the difference between the $VH(P, d)$ and P . The edge of a pocket that does not belong to P is called a *lid*. The pockets are simple polygons [ST].

Figure 3.2.1 The visibility hull of a simple polygon P with respect to the positive x -axis direction. The $VH(P, 0)$ is monotone with respect to $\pi/2$.

Sack and Toussaint [ST] have proved the following lemma and theorem:

Lemma 3.2.1: Two non-intersecting polygons monotone with respect to the same direction d , are movably separable in direction $d+\pi/2$ and direction $d-\pi/2$.

Proof: Let P and Q be two non-intersecting polygons, monotone with respect to direction d . Without loss of generality assume that d is the x -axis. Let Q_{min} and Q_{max} be the vertices of Q with the minimum and maximum x -coordinate, respectively. Let L_1 and L_2 be two lines parallel to the y -axis and going through Q_{min} and Q_{max} , respectively.

Assume the contrary, i.e. that P and Q are not movably separable in direction $\pi/2$ ($-\pi/2$). If Q is to be translated in direction $\pi/2$ ($-\pi/2$) then there exists a point v_1 (v_2) in Q that will collide with P at u_1 (u_2). The point u_1 is above Q and the point u_2 is below Q . According to the Jordan Curve Theorem, u_1 and u_2 are connected. Thus, P has to intersect L_1 or L_2 twice contradicting the fact that P is monotone with respect to the x -axis (see Figure 3.2.2). ♦

Figure 3.2.2 Polygon P is not monotone with respect to the x -axis.

Theorem 3.2.1: Two simple non-intersecting polygons P and Q are movably separable with a single translation in direction d if, and only if, $VH(P, d) \cap VH(Q, d) = \emptyset$.

Proof: " $\Rightarrow$ " If $VH(P, d) \cap VH(Q, d) = \emptyset$ then P and Q are movably separable. $VH(P, d)$ and $VH(Q, d)$ are monotone in the same direction $d + \pi/2$ and therefore, according to Lemma 3.2.1, are movably separable in direction d .

" $\Leftarrow$ " If P and Q are movably separable then $VH(P, d) \cap VH(Q, d) = \emptyset$. Assume the contrary, i.e. that $VH(P, d) \cap VH(Q, d) \neq \emptyset$ and that P and Q are movably separable. Let u be a point in the intersection of the two visibility hulls. Consider the following four cases:

- (i) The point $u \in P$ and $u \in VH(Q, d)$. In this case u lies in a pocket of Q . A half line L_1 (L_2) parallel to d ($-d$) and through u will intersect the pocket of Q at v_1 (v_2). Hence, if P is to be translated in direction d or $-d$ then u will intersect Q at v_1 or v_2 . Contradicting the assumption that P and Q are movably separable.
- (ii) The point $u \notin P$ and $u \in VH(Q, d)$. In this case u lies in a pocket of P and in a pocket of Q . A half line parallel to d and originated at u will intersect either P or Q first. If L first intersects P at u' then this case is similar to case 1 ($u' \in P$ and $u' \in VH(Q, d)$). If L first intersects Q at v then $v \in VH(P, d)$ and this case is similar to case 3 ($v \in Q$ and $v \in VH(P, d)$).
- (iii) The point $v \in Q$ and $v \in VH(P, d)$. This case is identical to 1.
- (iv) The point $v \notin Q$ and $v \in VH(P, d)$. This case is identical to 2. ♦

Corollary 3.2.1: Let P, Q be two simple non-intersecting polygons with m and n vertices respectively, and let d be a direction of translation. Whether P and Q are separable by a single translation in direction d , can be determined in $O(n+m)$.

Proof: From Theorem 3.2.1 it is sufficient to examine whether the visibility hulls of P and Q , with respect to direction d , intersect. The visibility hulls of P and Q , with respect to direction d , are monotone in direction $d+\pi/2$ and can be found in $O(n+m)$ using any of the methods by El Gindy and Avis [E⁺], Lee [L] or Dean and Sack [DS]. Whether two monotone polygons with respect to the same direction intersect can be determined in $O(n+m)$ time using the polygons intersection algorithm by Shamos and Hoey [SH]. Thus, determining whether the visibility hulls of the polygons intersect can be done in $O(n+m)$ time. ♦

3.3 THE DOMINANCE RELATION

In this section it is shown that the relation $dom = idom^+|below$ defines a linear order when the visibility hulls, with respect to a direction d , of a set S of M simple polygons do not intersect. The relations $idom$, $idom^+$ and $below$ are those relations that are used by Ottmann and Widmayer [OW] and were introduced in Section 2.1.

Lemma 3.3.1: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple, non-intersecting, n -vertex polygons and let d be the direction of translation. The relation $idom$ is asymmetric if, and only if, the $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$.

Proof: " $\Rightarrow$ " Let d be a direction of translation. If the $VH(P_i, d) \cap VH(P_j, d) = \emptyset$ for all $1 \leq i, j \leq M$ and $i \neq j$ then $idom$ is asymmetric. Assume the contrary, i.e. that $idom$ is symmetric and that $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$. The relation $idom$ is symmetric and, therefore, there exist r and s ($1 \leq r, s \leq M$) such that $P_r idom P_s$ and $P_s idom P_r$. $P_r idom P_s$ implies that there exist points u_1 and v_1 on P_s and P_r , respectively such that a half line, emanating from u_1 and parallel to d , intersects P_r at point v_1 . For the same reasons there exist points v_2 and u_2 on P_r and P_s respectively such that a half line, emanating from v_2 and parallel to direction d , intersects P_s at point u_2 . If P_r is

translated first, in direction d , then P_r will intersect P_s (point v_2 on P_r will collide with point u_2 on P_s). If P_s is translated first, in direction d , then P_s will collide with P_r (point u_1 on P_s will collide with point v_1 of P_r). Hence, P_r and P_s are not movably separable and from Theorem 3.2.1 $VH(P_r, d) \cap VH(P_s, d) \neq \emptyset$, contradicting the assumption that $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$.

" $\Leftarrow$ " If *idom* is asymmetric then $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$. Assume the contrary, i.e. that there exist r and s such that $VH(P_r, d) \cap VH(P_s, d) \neq \emptyset$ and that *idom* is asymmetric. From Theorem 3.2.1 P_r and P_s are not movably separable in direction d . If P_r is translated first, in direction d , then there exist points v_1 and u_1 on P_r and P_s respectively such that P_r collides with P_s , for the first time, at v_1 and u_1 , respectively. Thus, $P_s \text{ idom } P_r$. If P_s is translated first in direction d then there exist points v_2 and u_2 on P_r and P_s respectively such that P_s collides with P_r , for the first time, at v_2 and u_2 , respectively. Thus, $P_r \text{ idom } P_s$. If $P_r \text{ idom } P_s$ and $P_s \text{ idom } P_r$ then *idom* is symmetric contradicting the assumption that *idom* is asymmetric. ♦

Lemma 3.3.2: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple, non-intersecting, n -vertex polygons and let d be the direction of translation. Let P_i, P_j and P_k be three polygons of S such that their visibility hulls in d are pairwise non-intersecting. $P_k \text{ idom } P_j$ and $P_j \text{ idom } P_i$ implies that either $P_k \text{ idom } P_i$, or P_k and P_i are incomparable in *idom*, when P_k and P_i are examined in isolation.

Proof: Without loss of generality assume that the direction of translation is the positive x -axis. If $P_j \text{ idom } P_i$ then there are four cases in which P_i and P_j can be related.

Case 1: A horizontal line L_1 , through the upper vertex (with the highest y -value) of P_i , intersects P_j and a horizontal line L_2 , through the lower vertex (with the lowest y -value), intersects P_j (see Figure 3.3.1). If there exists a point u in P_k such that u is between L_1

and L_2 then there exists a half line L_3 , emanating at u and parallel to $-d$, that intersects P_i at point v . The existence of L_3 implies $P_k \text{ idom } P_i$ (see Figure 3.3.1). If such a point u does not exist then P_k is either above L_1 or P_k is below L_2 and, therefore, P_k and P_i are incomparable in *idom* (see Figure 3.3.2).

Case 2: A horizontal line L_1 , through the upper vertex (with the highest y -value) of P_i , intersects P_j and a horizontal line L_2 , through the lower vertex (with the lowest y -value) of P_j , intersects P_i (see Figure 3.3.3). If there exists a point u in P_k such that u is between L_1 and L_2 then there exists a half line L_3 , emanating at u and parallel to $-d$, that intersects P_i at point v . The existence of L_3 implies $P_k \text{ idom } P_i$ (see Figure 3.3.3). If such a point u does not exist then P_k is above L_1 . In this case P_k and P_i are incomparable in *idom* contradiction (see Figure 3.3.4). P_k cannot be below L_2 because P_k and P_j are then incomparable in *idom* (see Figure 3.3.5).

Case 3: A horizontal line L_1 , through the upper vertex (with the highest y -value) of P_j , intersects P_i and a line L_2 , through the lower vertex (with the lowest y -value) of P_i , intersects P_j . This case is identical to Case 2 and is, therefore, omitted.

Case 4: A horizontal line L_1 , through the upper vertex (with the highest y -value) of P_j , intersects P_i and a horizontal line L_2 , through the lower vertex (with the lowest y -value) of P_j , intersects P_i (see Figure 3.3.6). P_j and P_k are comparable in *idom* and, therefore, there exists a point u in P_k such that u is between L_1 and L_2 . Thus, there exists a half line L_3 , emanating at u and parallel to $-d$, that intersects A at point v . The existence of L_3 implies $P_k \text{ idom } P_i$ (see Figure 3.3.6). If such a point u does not exist then P_j and P_k are incomparable in *idom* (because P_k is either above L_1 or below L_2) contradiction. ♦

Figure 3.3.1 $P_k \text{ idom } P_i$ Figure 3.3.2 P_i and P_k are incomparable in *idom* Figure 3.3.3 $P_k \text{ idom } P_i$

Figure 3.3.4 P_i and P_k are incomparable in *idom* Figure 3.3.5 P_j and P_k are incomparable in *idom* Figure 3.3.6 P_k *idom* P_i

Lemma 3.3.3: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple non-intersecting n -vertex polygons and let d be the direction of translation. The graph representation of the transitive closure of $idom$ (denote as $idom^+$) is acyclic if $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$.

Proof: Without loss of generality assume that the direction of translation is the positive x -axis ($idom^+$ is denoted as $ixdom^+$).

If $ixdom^+$ is not acyclic then either $ixdom^+$ is symmetric ($ixdom^+$ contains a cycle of length two) or there exists a cycle of at least three polygons.

There are three cases to examine:

- (i) The minimum cycle consists of two polygons P_r and P_s .
- (ii) The minimum cycle consists of three elements P_i, P_j and P_k .
- (iii) The minimum cycle consists of four or more elements $P_i, P_j, P_k, P_e, \dots, P_f, P_g$ and P_h (it may be that the sequence $P_e, \dots, P_f, P_g, P_h$ consists of one polygon).

(i) Assume that the minimum cycle consists of two polygons P_r and P_s . In this case $ixdom^+$ is symmetric ($P_r ixdom^+ P_s$ and $P_s ixdom^+ P_r$). $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$ and thus according to Lemma 3.3.1 $ixdom^+$ is asymmetric, contradiction.

(ii) Assume that the minimum cycle consists of three polygons P_i, P_j, P_k such that $P_k ixdom P_j, P_j ixdom P_i$ and $P_i ixdom P_k$ (a cycle $P_i P_j P_k P_i$). $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, $VH(P_i, d) \cap VH(P_k, d) = \emptyset$ and $VH(P_j, d) \cap VH(P_k, d) = \emptyset$ and thus by Lemma 3.3.2 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. If $P_k ixdom P_i$ then $P_i ixdom P_k$ and $P_k ixdom P_i$ implies that $ixdom^+$ is symmetric. $VH(P_i, d) \cap VH(P_k, d) = \emptyset$ and thus, the symmetry of $ixdom^+$ contradicts Lemma 3.3.1. If P_k and P_i are incomparable in $ixdom$ then P_i cannot $ixdom P_k$.

(iii) Assume that the minimum cycle consists of four or more polygons $P_i, P_j, P_k, P_e, \dots, P_f, P_g, P_h, P_i$ (it may be that the sequence $P_e, \dots, P_f, P_g, P_h$ consists of one polygon) such that $P_i \text{ ixdom } P_h, P_h \text{ ixdom } P_g, P_g \text{ ixdom } P_f, \dots, P_e \text{ ixdom } P_k, P_k \text{ ixdom } P_j$ and $P_j \text{ ixdom } P_i$ (a cycle $P_i P_j P_k P_e \dots P_f P_g P_h P_i$). $\text{VH}(P_i, d) \cap \text{VH}(P_j, d) = \emptyset$, $\text{VH}(P_i, d) \cap \text{VH}(P_k, d) = \emptyset$ and $\text{VH}(P_j, d) \cap \text{VH}(P_k, d) = \emptyset$ and thus, by Lemma 3.3.2 either $P_k \text{ ixdom } P_i$ or P_k and P_i are incomparable in *ixdom* when P_k and P_i are examined in isolation. If $P_k \text{ ixdom } P_i$ then a shorter cycle, consisting of $P_i, P_k, P_e, \dots, P_f, P_g, P_h, P_i$, would exist contradicting the assumption that the cycle $P_i P_j P_k P_e \dots P_f P_g P_h P_i$ is minimal. Hence, P_k and P_i are incomparable in *ixdom*, and a line L_1 (y -separator), that intersects P_j , exists such that P_k is above (below) L_1 and P_i is below (above) L_1 . Without loss of generality assume that P_k is above L_1 (see Figure 3.3.7). For similar reasons P_j and P_e are incomparable in *ixdom*. Therefore, a line L_2 , that intersects P_k , exists such that P_j is below L_2 and P_e is above L_2 (see Figure 3.3.8). If P_j was above L_2 and P_e below L_2 then P_i and P_j were incomparable in *ixdom* (P_j is above L_2 , L_2 is above L_1 and L_1 is above P_i , see Figure 3.3.9). For similar reasons P_f and P_h are incomparable in *ixdom* and a line L_k exists such that P_h is above L_k and P_f is below L_k . A sequence of lines $L_1, L_2, L_3, \dots, L_k$ is constructed such that L_1 is below L_2 , L_2 is below L_3 , $\dots$, L_{k-1} is below L_k . P_h is above L_k and, therefore, above L_1 . P_i is below L_1 and thus P_i and P_h are incomparable in *ixdom* contradicting $P_i \text{ ixdom } P_h$ (see Figure 3.3.10). ♦

Figure 3.3.7 P_i and P_k are incomparable in $ixdom^+$ (P_i is below P_k).

Figure 3.3.8 Lines L_1 and L_2 as y -separators.

Figure 3.3.9 P_i and P_j are incomparable in $ixdom^+$.

Figure 3.3.10 P_i and P_h are incomparable in $ixdom^+$

Theorem 3.3.1: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple n -vertex polygons and let d (the direction of translation) be the positive x -axis. If $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$ then the relation $dom = ixdom^+|_{below}$ defines a linear ordering.

Proof: The relation dom is a linear ordering if dom is total, asymmetric and transitive.

(i) The relation dom is total - For every i and j either $P_i dom P_j$ or $P_j dom P_i$. There are two cases to examine: (a) P_i and P_j are comparable in $ixdom^+$; P_i and P_j are incomparable in $ixdom^+$. (a) P_i and P_j are comparable in $ixdom^+$ - $VH(P_i, d) \cap VH(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$, and thus, according to Lemma 3.3.3 the graph representation of $ixdom^+$ is acyclic. P_i and P_j are comparable in $ixdom^+$ and, therefore, either $P_i ixdom^+ P_j$ or $P_j ixdom^+ P_i$. (b) P_i and P_j are incomparable in $ixdom^+$ - Thus, there exists a y -separator L such that $P_i(P_j)$ is below L and $P_j(P_i)$ is above L , which implies $P_i(P_j) dom P_j(P_i)$.

(ii) The relation *dom* is asymmetric - For every i and j , $P_i \text{ dom } P_j$ implies $\text{NOT}(P_j \text{ dom } P_i)$. Assume that $P_i \text{ dom } P_j$ then one of the following cases occurs:

(a) $P_i \text{ ixdom}^+ P_j$ then by asymmetry of ixdom^+ it implies $\text{NOT}(P_j \text{ ixdom}^+ P_i)$ (see Lemma 3.3.3). Thus, $\text{NOT}(P_j \text{ dom } P_i)$ holds because P_i and P_j are comparable in ixdom^+ .

(b) $(P_i \text{ below } P_j)$ and $(P_i \text{ and } P_j \text{ are incomparable in } \text{ixdom}^+)$ - This implies $\text{NOT}(P_j \text{ below } P_i)$ because of asymmetry of *below*. Therefore, $\text{NOT}(P_j \text{ dom } P_i)$ holds as P_j and P_i are incomparable in ixdom^+ .

(iii) The relation *dom* is transitive [$(P_i \text{ dom } P_j)$ and $(P_j \text{ dom } P_k)$ implies $(P_i \text{ dom } P_k)$] - Assume that $(P_i \text{ dom } P_j)$ and $(P_j \text{ dom } P_k)$ then there are four cases to consider:

(a) $(P_i \text{ ixdom}^+ P_j)$ and $(P_j \text{ ixdom}^+ P_k)$ - $\text{VH}(P_i, d) \cap \text{VH}(P_j, d) = \emptyset$, for all $1 \leq i, j \leq M$ and $i \neq j$, and thus, from Lemma 3.3.3 the transitivity of ixdom^+ implies $(P_i \text{ ixdom}^+ P_k)$, therefore $P_i \text{ dom } P_k$.

(b) $(P_i \text{ ixdom}^+ P_j)$ and $(P_j \text{ below } P_k)$ and $(P_j \text{ and } P_k \text{ are incomparable in } \text{ixdom}^+)$ - Assume the contrary, i.e. that $(P_k \text{ dom } P_i)$. Then either $(P_k \text{ ixdom}^+ P_i)$ or $(P_k \text{ below } P_i)$.

If $P_k \text{ ixdom}^+ P_i$ then by the transitivity of ixdom^+ $P_k \text{ ixdom}^+ P_j$, see Lemma 3.3.3, which contradicts the assumption that P_j and P_k are incomparable in ixdom^+ .

If $(P_k \text{ below } P_i)$ and $(P_i \text{ and } P_k \text{ are incomparable in } \text{ixdom}^+)$ then the transitivity of *below* yields $P_j \text{ below } P_i$. Hence, if $P_i \text{ ixdom}^+ P_j$ then there is a sequence $P_{e1}, P_{e2}, \dots, P_{er}$ such that $P_{et} \text{ ixdom } P_{e(t+1)}$, for all $1 \leq t \leq r-1$, $P_{er} \text{ ixdom } P_j$ and $P_i \text{ ixdom } P_{e1}$ (see Lemma 3.3.3). P_k lies completely between P_i and P_j and, therefore, there exists P_{eu} such that either $P_{eu} \text{ ixdom } P_k$ or $P_k \text{ ixdom } P_{eu}$. If $P_{eu} \text{ ixdom } P_k$ then the transitivity of ixdom^+ (see Lemma 3.3.3) yields $P_i \text{ ixdom}^+ P_k$, contradicting the assumption that P_i and P_k are incomparable in ixdom^+ (see Figure 3.3.11). If $P_k \text{ ixdom } P_{eu}$ then the transitivity of ixdom^+ yields $P_k \text{ ixdom}^+ P_j$ contradicting the assumption that P_j and P_k are incomparable in ixdom^+ (see Figure 3.3.12).

(c) (P_i below P_j) and (P_j $ixdom^+$ P_k) and (P_j and P_i are incomparable in $ixdom^+$) - Assuming the contrary that (P_k dom P_i) then either (P_k $ixdom^+$ P_i) or (P_k below P_i).

If P_k $ixdom^+$ P_i then by the transitivity of $ixdom^+$ (see Lemma 3.3.3) P_j $ixdom^+$ P_i contradicting the assumption that P_i and P_j are incomparable in $ixdom^+$.

If (P_k below P_i) and (P_i and P_k are incomparable in $ixdom^+$) then the transitivity of *below* yields P_k below P_j . Hence, if P_j $ixdom^+$ P_k then there is a sequence $P_{e1}, P_{e2}, \dots, P_{er}$ such that P_{et} $ixdom$ P_{et+1} , for all $1 \leq t \leq r-1$, P_{er} $ixdom$ P_k and P_j $ixdom$ P_{e1} (see Lemma 3.3.3). P_i lies completely between P_k and P_j therefore there exists P_{eu} such that either P_{eu} $ixdom$ P_i or P_i $ixdom$ P_{eu} . If P_{eu} $ixdom$ P_i then the transitivity of $ixdom^+$ (see Lemma 3.3.3) yields P_j $ixdom^+$ P_i , contradicting the assumption that P_i and P_j are incomparable in $ixdom^+$. If P_i $ixdom$ P_{eu} then the transitivity of $ixdom^+$ (see Lemma 3.3.3) yields P_i $ixdom^+$ P_k , contradicting the assumption that P_i and P_k are incomparable in $ixdom^+$.

(d) (P_i below P_j) and (P_j below P_k) and (P_j and P_i are incomparable in $ixdom^+$) and (P_j and P_k are incomparable in $ixdom^+$) - By the transitivity of *below* P_i below P_k and P_i and P_k are incomparable in $ixdom^+$ and thus (P_i dom P_k). ♦

Figure 3.3.11 P_{eu} $ixdom$ P_k and, therefore, P_i $ixdom^+$ P_k

Figure 3.3.12 $P_k \text{ idom } P_{eu}$ and, therefore, $P_k \text{ idom } P_j$.

3.4 UNI-DIRECTIONAL SEPARABILITY ALGORITHMS

In this section two algorithms are presented. The first algorithm (Algorithm 3.4.1 in Section 3.4.1) determines whether a set of M polygons, monotone with respect to the y -axis, contains a pair of intersecting polygons. If no intersection occurs then the algorithm produces a translation order of the polygons in the direction of the positive x -axis. The second algorithm (Algorithm 3.4.2 in Section 3.4.2) solves the uni-directional separability problem for a set M simple non-intersecting polygons in two dimensional space. In Section 3.4.3 the complexity analysis of the algorithms is given.

3.4.1 ALGORITHM FOR MONOTONE POLYGONS

This algorithm is based on the algorithm by Ottmann and Widmayer [OW], described in Section 2.2.1.2. Instead of using the endpoints of line segments the algorithm uses the upper and lower endpoints of the polygons. Whenever the scan line stops at an upper endpoint, for example vertex p of polygon P , P is inserted into the overall structure

Polygon P is inserted into the tree (the overlay structure) before its successor active polygon (in an inorder traversal of the tree). During the insertion in the balanced binary search tree p is examined with each of the nodes on the path (from the root to the position of P 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 in the polygon (stored at the node) then an intersection occurs and the algorithm terminates. If p is not contained in any of the polygons then the successor P'' of P and the predecessor P' of P in the tree are found. P is then tested for an intersection with P' and with P'' . If no intersection occurs between these polygons then P is inserted into the linked list (the basic structure) before P'' (its successor) left to right. Whenever a lower endpoint, for example q of Q , is encountered the polygon Q ceases to be active. The predecessor Q' and the successor Q'' of Q are found and Q' and Q'' are tested for an intersection. If no intersection occurs between them then Q is deleted from the tree, but 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 set is not uni-directional separable in the specified direction.

Algorithm 3.4.1: Translating a set of simple monotone polygons.

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

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

Output: A translation order of the polygons.

begin

1. Find the vertices of the polygons with the highest and the lowest y -value (denote these vertices as the endpoints of the polygons);
2. Construct a priority queue E which consists of all endpoints of the polygons in a descending order of their y -values;
3. Initiate the balanced binary search tree, the doubly linked list and the variable Intersect?;
4. While ($E \neq \emptyset$)

```
begin
5.    $q := \text{DeleteMax}(E)$ ;
    if ( $q$  is an upper endpoint of polygon  $P$ ) then
        begin
6.   Insert_Polygon_Into_Tree(Root,  $P$ , Intersect?);
        If (Intersect? = true) then
            begin
                Print (A TRANSLATION ORDER DOES NOT EXIST !);
                exit program;
            end; {if}
7.   find the predecessor  $P'$  and the successor  $P''$  of  $P$  in the tree;
        if ( $P'$  exists) then
            begin{ $P'$  and  $P$  are examined for intersection}
8.   Check_Intersection ( $P$ ,  $P'$ , Intersect?);
            end; {if}
        if ( $P''$  exists) then
            begin{ $P$  and  $P''$  are examined for intersection}
9.   Check_Intersection ( $P$ ,  $P''$ , Intersect?);
            end;
        If (Intersect? = true) then
            begin
                Print (A TRANSLATION ORDER DOES NOT EXIST !);
                exit program;
            end; {if}
        if ( $P''$  exists) then
            begin
10.   insert  $P$  as the new predecessor of  $P''$  in the linked list;
                end; {if}
            else {there is no successor to  $P$ }
                begin
11.   insert  $P$  as the new final element in the linked list;
                end; {else}
            end; {if}
        else { $q$  is a lower endpoint of  $P$ }
            begin
12.   Delete_Polygon_From_Tree( $P$ , Intersect?);
```

```

    If (Intersect? = true) then
    begin
        Print (A TRANSLATION ORDER DOES NOT EXIST ! );
        exit program;
    end;
    delete  $P$  from the tree (but not from the linked list);
end; {else}
13.end; {while}
end; {algorithm}

```

Procedure Insert_Polygon_Into_Tree (Root, P , Intersect?);

{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 - a polygon.

Intersect? - a boolean variable. If Intersect? = true then two visibility hulls intersect}

begin

While (P not inserted into the tree) do

begin

1. 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 (denoted as p) is less than the y-value of v_{Li} (v_{Rj}) and greater than the y-value of v_{Li-1} (v_{Rj+1}) (see Figure 3.4.1);
{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}

2. 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

begin

Intersect? := true; {the polygons P and Root intersect}

return();

end;

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

begin

Root := Left_Son_Of_Root;

Insert_Polygon_Into_Tree (Root, P , Intersect?);


```

    return();
end;
4. If  $p$  is to the right of edge  $(v_{R_j}, v_{R_{j+1}})$  then
    begin
        Root := Right_Son_Of_Root;
        Insert_Polygon_Into_Tree (Root,  $P$ , Intersect?);
        return();
    end;
end;

```

Figure 3.4.1 The vertices v_{Li} and v_{Rj} of Q are found when the vertex p of P is inserted.

Procedure Delete_Polygon_From_Tree (Root, P , Intersect?);

{ This procedure deletes the polygon P from a binary tree.

Input: Root - the root of the tree.

P - a polygon.

Intersect? - a boolean variable. If Intersect? = true then two visibility hull intersect. }

begin

1. Find the predecessor P' and the successor P'' of P in the tree;
 2. If (P' and P'' exist) then
 - begin {VH(P' , 0) and VH(P'' , 0) are examined for intersection}
 - Check_Intersection (P' , P'' , Intersect?);
 - end; {if}
 3. Delete P from the tree;
- end;

Procedure Check_Intersection (P , Q , Intersect?);

{This procedure checks whether two monotone polygons with respect to the y-axis intersect. If the polygons intersect then Intersect? = true.}

3.4.2 ALGORITHM FOR SIMPLE POLYGONS

Algorithm 3.4.1 determines whether any pair of M simple polygons, monotone with respect to the y-axis, intersect. If the polygons do not intersect then it determines a translation order of the polygons. Algorithm 3.4.1 accepts, as input data, only polygons monotone with respect to the y-axis.

When simple polygons are to be translated in the positive x-direction the visibility hulls of the polygons are found. The visibility hulls, which are monotone polygons, are then given as input to Algorithm 3.4.1. The output of Algorithm 3.4.1 determines the translation order of the simple polygons. If the direction of translation d is not the positive x-axis then the polygons are rotated until the direction of translation is the positive x-direction.

Algorithm 3.4.2: Translating a set of simple non-intersecting polygons.

Input: 1) A set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons.
 2) The direction of translation d .

Output: A translation order of the polygons.

begin

1. If d is not the positive x -axis then rotate S such that d is the positive x -axis
2. Find the visibility hulls of the polygons (with respect to the positive x -axis).
3. Let S' be the set of the visibility polygons found in Line 2. Execute Algorithm 3.4.1 on the set S' .

end;{algorithm}

3.4.3 COMPLEXITY ANALYSIS

3.4.3.1 TIME AND SPACE COMPLEXITY OF ALGORITHM 3.4.1

Procedure Check_Intersection

The procedure checks whether two polygons P and Q , monotone with respect to the y -axis, intersect. This can be done, in $O(m+n)$ time and space, by using the algorithm due Shamos and Hoey [SH], where m and n are the number of vertices in P and Q , respectively. In the sequel, in Section 3.6, it will be shown that this is optimal, even if it is assumed that the polygons are already stored in the main memory

Procedure Insert_Polygon_Into_Tree

The procedure inserts a polygon into its proper place in the binary balanced tree (the overlay structure).

- a) Line 1 - Each time the algorithm calls the procedure a search for the new vertices v_{Li} and v_{Rj} of the root is done. However, 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 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 $v_{Li+1}, \dots, v_{Lu}$ ($v_{Rj+1}, \dots, v_{Rw}$) that have been examined and discarded (as their y -coordinate is too

large). Let C_i denote the number of vertices that have been discarded at the i -th call to the procedure. Thus the total cost of this line is less than $C_i + O(1)$.

The search for the new v_{Li} and v_{Ri} can be improved by a binary search on the chains (this improvement depends on the size of M). This improvement, however, will not change the overall complexity of the algorithm.

b) Line 2 - checking if the upper vertex is to the right of v_{Li} and to the left of v_{Ri} is done in $O(1)$.

c) Line 3 - checking if the upper vertex is to the left of v_{Li} is done in $O(1)$.

d) Line 4 - checking if the upper vertex is to the right of v_{Ri} is done in $O(1)$.

The time complexity at the i -th call is $C_i + O(1)$, where $0 < C_i$. The procedure is called M times, once for each polygon, by Algorithm 4.3.1. Each time at most $\log M$ additional recursive calls are done and, thus, the maximum number of times that Line 1 is executed is $M \log M$. Hence, the cost of the procedure, during all the execution time of Algorithm 4.3.1 is $\sum_{i=1}^{M \log M} (C_i + O(1)) = \sum_{i=1}^{M \log M} C_i + \sum_{i=1}^{M \log M} O(1)$. $\sum_{i=1}^{M \log M} C_i$ is the total number of edges that have been discarded. Each edge is discarded at most once and, therefore, $\sum_{i=1}^{M \log M} C_i \leq Mn$. Thus, the cost of the procedure, during the execution time of Algorithm 4.3.1 is $\sum_{i=1}^{M \log M} C_i + \sum_{i=1}^{M \log M} O(1) \leq Mn + M \log M$.

Procedure Delete_Polygon_From_Tree

The procedure deletes a polygon from the balanced binary tree (the overlay structure).

a) Line 1 - finding the successor and the predecessor of an element in a binary balanced tree can be accomplished in $O(\log M)$.

b) Line 2 - the execution time of Check_Intersection is $O(n)$.

c) Line 3 - deleting an element from a binary balanced tree is done in $O(\log M)$.

The total cost of the procedure is $O(n + \log M)$.

Algorithm 3.4.1

- a) Line 1 - finding the vertices with the maximum (minimum) y-coordinate can be done in $O(n)$ for each polygon and thus in $O(Mn)$ for all polygons.
- b) Line 2 - constructing a priority queue can be done in $O(M)$. (see Section 1.5.3)
- c) Line 3 - initializing the overlay and the basic structure is done in $O(1)$.
- d) The cost of an iteration of the while loop (Lines 4 - 13):
 - 1) Line 5 - extracting the first element in the priority queue and updating the queue is done in $O(\log M)$.
 - 2) Line 6 - the cost of the Procedure Insert_Polygon_Into_Tree is $O(n + \log M)$.
 - 3) Line 7 - finding the successor and the predecessor of an element in a binary balanced tree can be accomplished in $O(\log M)$.
 - 4) Line 8 - the cost of the Procedure Check_Intersection is $O(n)$.
 - 5) Line 9 - the cost of the Procedure Check_Intersection is $O(n)$.
 - 5) Line 10 - inserting an element into the doubly connected linked list is done in $O(1)$ time. P'' (the successor of P) points to its position in the linked list.
 - 6) Line 11 - inserting an element into the doubly connected linked list as the top element is accomplished in $O(1)$ time.
 - 7) Line 12 - the cost of the Procedure Delete_Polygon_From_Tree is $O(n + \log M)$.

The while loop is executed $2M$ times (M upper endpoints and M lower endpoints in the queue). Line 5 is executed $2M$ times. Lines 6, 7, 8, 9, 10, 11 are executed when an upper vertex of a polygon is encountered. Line 12 is executed when a lower vertex of a polygon is encountered. Hence, the total cost of the while loop is $2M * \log M + M * (\log M + n + \log M + n + n + 1) + M * \log M$ which yields $O(Mn + M \log M)$. There are at most M polygons in the binary balanced tree and in the linked list. Hence, the space required is $O(M)$.

The execution time of Algorithm 3.4.1 is $O(Mn + M \log M)$ using $O(Mn)$ space

3.4.3.2 TIME AND SPACE COMPLEXITY OF ALGORITHM 3.4.2

- a) Line 1 - the rotation of the polygons (if needed) can be accomplished in $O(Mn)$. (Each point has been rotated exactly once).
- b) Line 2 - the visibility hulls can be found in $O(Mn)$ time and space using any of the methods by El Gindy and Avis [EA], Lee [L] and Dean and Sack [DS].
- c) Line 3 - The cost of Algorithm 3.4.1 is $O(Mn + M \log M)$ time and $O(Mn)$ space.

The cost of Algorithm 3.4.2 is $O(Mn + M \log M)$ time and $O(Mn)$ space.

3.5 CORRECTNESS OF ALGORITHMS 3.4.1 AND 3.4.2

The upper most intersection point w between two polygons A and B , monotone with respect to the positive y -axis, can be one of the following types:

- (i) Type 1- If the upper most vertex of A (B) is contained in B (A) then the point w is the upper most vertex of A (B) (see Figure 3.5.1);
- (ii) Type 2 - If the upper most vertex of A is not contained in B and the upper most vertex of B is not contained in A then the point w is an intersection between two edges, edge(A_i, A_{i+1}) of polygon A and edge(B_j, B_{j+1}) of polygon B (see Figure 3.5.2). A_i denote vertex i of polygon A .

Let L denote a sweep line that moves from top to bottom (from $+\infty$ to $-\infty$ on the y -axis). L divides the x - y plane into two half planes: (i) T_u the half plane above L . (ii) T_b the half plane below L . If L is above w (the highest intersection point) then the half plane T_u is free of intersections and from Theorem 3.3.1 and Lemma 3.3.3 all polygons that L intersects are sorted by *idom*.

Figure 3.5.1 The upper vertex of A is the highest intersection point w .

Figure 3.5.2 The intersection between two edges one on A and one on B forms w .

Theorem 3.5.1: Given a set $S = \{P_1, P_2, \dots, P_M\}$ of M n -vertex polygons, monotone with respect to the same direction, Algorithm 3.4.1 finds an intersection between a pair P_i, P_j if one exists.

Proof:

- (i) If there is no intersection between any pair of polygons then the algorithm does not report anything. Hence, the algorithm does not fail when an intersection does not exist.
- (ii) If one or more intersections exist then the algorithm finds the highest intersection point (the intersection point with the greatest y -coordinate). If there is more than one intersection point with the same (greatest) y -coordinate then the algorithm finds one of the intersection points.

Let u denote the highest intersection point and let P_i and P_j be the two polygons with u as their intersection point. Without loss of generality 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 . There are then two cases to examine: (a) the intersection point u is Type 1; (b) the intersection point u is Type 2.

- (a) The intersection point u is of Type 1 - The point u is the upper most vertex of P_i . When L stops at u (P_i has not been inserted yet) all polygons that are stored in the binary balanced search tree do not intersect. Hence, these polygons are sorted by *ixdom*. The algorithm inserts P_i into the proper place into the binary balanced search tree. The insertion is done by the procedure `Insert_Polygon_Into_Tree`. Whenever the algorithm tests whether u lies to the left or to the right of a polygon P_k , the algorithm also tests whether u lies inside P_k (at Line 2 of procedure `Insert_Polygon_Into_Tree`). Therefore, when u is examined with P_j (P_j is already in the tree) the algorithm will report that P_i intersects P_j . The vertex u must be examined with P_j , otherwise, there exists a polygon P_k to the right (left) of P_j such

that u is to the right (left) of P_k contradicting the assumption that u is the highest intersecting point.

(b) The intersection point u is of Type 2 - The point u is an intersection between two edges: $\text{edge}(p_{i,r}, p_{i,r+1})$ of P_i and $\text{edge}(p_{j,s}, p_{j,s+1})$ of P_j ($p_{i,j}$ denote vertex j of polygons P_i). Without loss of generality assume that when P_i is inserted into the tree the *ixdom* relation between P_j and P_i is $P_j \text{ ixdom}^+ P_i$. There are two cases to examine. (i) If, after the insertion of P_i , P_j is the successor of P_i in the tree then P_i and P_j are tested for intersection (Line 9 in Algorithm 3.4.1) and thus, the intersection is found. If, after the insertion of P_i , P_j is not the successor of P_i in the tree then case b applies. (ii) The two polygons P_i and P_j are already in the tree and u has not been found yet. Hence, there is at least one polygon P_k such that the *ixdom* relation among P_i , P_j and P_k is $P_j \text{ ixdom} P_k$ and $P_k \text{ ixdom} P_i$ (if there is more than one polygon then let P_k be the polygon with the lowest lower vertex such that $P_j \text{ ixdom} P_k$ and $P_k \text{ ixdom} P_i$ see Figure 3.5.3). The y-coordinate of the lower vertex of P_k is greater than y-coordinate of u and, therefore, P_k is deleted before the sweep line reaches u . When the sweep line reaches the lower vertex of P_k the predecessor and the successor of P_k are P_i and P_j , respectively. When P_k is deleted from the tree P_i and P_j are tested by the algorithm for intersection (Line 2 in the procedure `Delete_Polygon_From_Tree`). Hence, the intersection point u is found. ♦

Figure 3.5.3 $P_j \text{ ixdom } P_k$ and $P_k \text{ ixdom } P_i$

3.6 LOWER BOUND

In this section a lower bound of $\Omega(Mn + M \log M)$ for the uni-directional separability problem for a set of M simple non-intersecting n -vertex polygons is given. It will be shown that $\Omega(Mn + M \log M)$ time is needed to determine the translation order and that $\Omega(Mn + M \log M)$ time is needed to detect whether a set of polygons is separable (in a given direction). Although the $\Omega(Mn)$ lower bound can be derived easily by using the input/output lower bound argument, it will be shown that even if the polygons are stored in the main memory the uni-directional separability problem requires $\Omega(Mn + M \log M)$ time.

In Section 3.6.1, an $\Omega(Mn)$ lower bound is derived for detecting whether any pair of polygons in a given set of monotone polygons intersects. Section 3.6.2 examines this

lower bound in the case that the monotone polygons are visibility hulls of a set of non-intersecting simple polygons. In Section 3.6.3, an $\Omega(Mn + M \log M)$ lower bound is given for determining a translation order. Section 3.6.4 describes an $\Omega(Mn + M \log M)$ lower bound for detecting whether a set S of polygons is separable (in a given direction).

3.6.1 LOWER BOUND FOR MONOTONE POLYGONS

In this section a lower bound of $\Omega(Mn)$ for the following problem is examined: Given a set $S = \{P_1, P_2, \dots, P_M\}$ of M n -vertex polygons, monotone with respect to the same direction, detect whether any pair of the polygons in S intersect. The lower bound is proven by using the adversary argument [AK] (as explained in Section 1.5.4).

Although the trivial input/output lower bound of $\Omega(Mn)$, for reading the vertices, can be used, it may not hold for all cases. The input/output lower bound will not apply when: (i) the polygons are already stored in the main memory or (ii) when the polygons are convex (a sub linear $(M \log Mn)$ time algorithm is sufficient).

First a lower bound of $\Omega(m+n)$ will be given for the following problem (denoted as **Problem1**): 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 Problem1 will be called **Detector1**.

Definition: Let P and Q be two simple polygons monotone with respect to y -axis. A vertex p_i of polygons P , *faces* an edge (q_j, q_k) , of Q , if there exists a line L parallel to the x -axis through p_i that intersects edge (q_j, q_k) at point u and the segment $p_i u$ does not intersect P and Q (except at its endpoints).

Lemma 3.6.1: Let P and Q be two simple polygons, monotone with respect to the same direction d , with m and n vertices, respectively. Detector1 requires at least $m+n-4$ steps.

Proof: Let P and Q be two polygons monotone with respect to the y -axis. Let the vertices $q_2, q_3, \dots, q_{n-1}$ of Q face the edge (p_m, p_1) of P . Let the vertices $p_3, p_4, \dots, p_m$ 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 3.6.1). Assume the contrary of Lemma 3.6.1, i.e. that there exists a Detector1 that requires less than $m+n-4$ steps. To solve Problem1, Detector1 must be able to detect whether P and Q intersect. Whenever Detector1 accesses vertex p_i (q_j) the adversary will give Detector1, 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 Detector1. If Detector1 examines less than $m+n-4$ vertices then there exist at least one vertex among $p_3, p_4, \dots, p_m$ and $q_2, q_3, \dots, q_{n-1}$ that was not examined by Detector1. Assuming that vertex p_k , $2 < k \leq m$, (q_r , $2 \leq r < n$) was not examined. If Detector1 outputs that the polygons intersect then Detector1 fails because the polygons do not intersect. If Detector1 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 Detector1 fails to give a correct answer (see Example 3.6.1). The adversary can find in $O(m+n)$ steps which vertex should be moved by traversing all the vertices. ♦

Example 3.6.1: Let P and Q be two simple polygons, monotone with respect to the y -axis, with m and n vertices, respectively. Without loss of generality assume that $m > n$. Let the vertices of P be: $p_1 = (1, 1)$, $p_2 = (1, 2m)$ and $p_i = (2+(i \text{ modulus } 2), 2m-i)$, for all $3 \leq i \leq m$. Let the vertices of Q be: $q_1 = (6, 1)$, $q_n = (6, 2m)$ and $q_i = (4+(i \text{ modulus } 2), i)$, for all $2 \leq i \leq n-1$ (see Figure 3.6.1). Vertices $p_3, \dots, p_m$ of P face edge (q_{n-1}, q_n) and vertices $q_2, \dots, q_{n-1}$ face edge (p_1, p_m) . Assume that Detector1 did not examine vertex q_3 .

The adversary moves q_3 by decreasing the value of the x -coordinate of q_3 from 5 to $1+1/2m$ and force Detector1 to produce an incorrect answer (see Figure 3.6.2). ♦

Figure 3.6.1 Vertices $p_3, \dots, p_m$ of P face edge(q_{n-1}, q_n) and vertices $q_2, \dots, q_{n-1}$ face edge(p_1, p_m).

Figure 3.6.2 Polygons P and Q intersect.

The adversary argument, used in Lemma 3.6.1 can be modified to prove the lower bound of $\Omega(Mn)$ for the following problem (denoted as **Problem2**). Given a set $S = \{P_1, P_2, \dots, P_M\}$ of M n -vertex polygons, monotone with respect to the same direction, detect whether any two polygons in S intersect. A program that can solve Problem2 will be called **Detector2**. One way of achieving the $\Omega(Mn)$ lower bound is by arranging the M polygons in pairs in such a way that each pair has the properties of the simple polygons of Lemma 3.6.1. Another arrangement of the polygons for achieving the $\Omega(Mn)$ lower bound is described in Lemma 3.6.2.

Let $p_{i,j}$ denote vertex j of P_i .

Lemma 3.6.2: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M n -vertex polygons, monotone with respect to the same direction d . Detector2 requires at least $Mn-2M$ steps.

Proof: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M n -vertex polygons monotone with respect to the y -axis. The following construction of S is illustrated in Figure 3.6.3. Let all vertices $p_{i,(n/2+1)}$ be the upper vertex ($p_{i,l}$ be the lower vertex) of P_i for all $1 < i < M$. Let the vertex $p_{1,l}$ and the vertex $p_{M,l}$ ($p_{1,2}$ and $p_{M,n}$) be the low : (upper) vertex of P_1 and P_M respectively. Let the y -coordinate of $p_{i,(n/2+1)}$ ($p_{i,l}$) be equal to the y -coordinate of $p_{i-1,(n/2+1)}$ ($p_{i-1,l}$) and let the x -coordinate of $p_{i,(n/2+1)}$ ($p_{i,l}$) be greater than the x -coordinate of $p_{i-1,(n/2+1)}$ ($p_{i-1,l}$), for all $2 < i < M$. Let the x -coordinate of $p_{1,l}$ ($p_{1,2}$) be less than the x -coordinate of $p_{2,l}$ ($p_{2,(n/2+1)}$) and let the y -coordinate of $p_{1,l}$ ($p_{1,2}$) be equal to the y -coordinate of $p_{2,l}$ ($p_{2,(n/2+1)}$). Let the x -coordinate of $p_{M,l}$ ($p_{M,n}$) be greater than the x -coordinate of $p_{M-1,l}$ ($p_{M-1,(n/2+1)}$) and let the y -coordinate of $p_{M,l}$ ($p_{M,n}$) be equal to the y -coordinate of $p_{M-1,l}$ ($p_{M-1,(n/2+1)}$). Let the vertices $p_{i,2}, \dots, p_{i,n/2}$ face edge $(p_{i-1,l}, p_{i-1,n})$, for all $2 < i < M$ and the vertices $p_{i,(n/2+2)}, \dots, p_{i,n}$ face edge $(p_{i+1,(n/2)}, p_{i+1,(n/2+1)})$, for all $1 < i < M-1$. Let the vertices $p_{1,3}, \dots, p_{1,n}$ face edge $(p_{2,(n/2)}, p_{2,(n/2+1)})$ and let the vertices $p_{2,2}, \dots, p_{2,n/2}$ face edge $(p_{1,l}, p_{1,n})$. Let the vertices $p_{M,1}, \dots, p_{M,n-1}$ face edge $(p_{M-1,l}, p_{M-1,n})$ and let the vertices $p_{M-1,(n/2+2)}, \dots, p_{M-1,n}$ face edge $(p_{M,n-1}, p_{M,n})$.

Assume the contrary to Lemma 3.6.2, i.e. that there exists a Detector2 that requires less than $Mn-2M$ steps. This Detector2 must be able to detect whether any pair P_i and P_k intersect. Whenever Detector2 accesses vertex $p_{k,j}$ the adversary will give Detector2, for free, the information which edge the vertex $p_{k,j}$ is facing (and if $p_{k,j}$ intersects that edge) by using the following rules: a) edge $(p_{k-1,n}, p_{k-1,l})$ of P_{k-1} if $k \neq 1$ and $k \neq M$ and $2 \leq j \leq n/2$. b) edge $(p_{k+1,n/2}, p_{k+1,(n/2+1)})$ of P_{k+1} if $k \neq 1$ and $k \neq M, M-1$ and $n/2+2 \leq j \leq n$. c) edge $(p_{M-1,l}, p_{M-1,n})$ of P_{M-1} if $k = M$. d) edge $(p_{2,n/2}, p_{2,(n/2+1)})$ of P_2 if $k = 1$. e) edge $(p_{M,n-1}, p_{M,n})$ of P_M if $k = M-1$ and $n/2+2 \leq j \leq n$. The adversary maintains a list of all vertices that have not yet been checked by Detector2. If Detector2 has examined less than $Mn-2M$ vertices then there exist at least one vertex among $p_{1,3}, \dots, p_{1,n}, p_{M,2}, \dots, p_{M,n-2}$ and $p_{i,2}, \dots, p_{i,n/2}, p_{i,n/2+2}, \dots, p_{i,n}$, for all $1 < i < M$, that Detector2 has not examined. Without loss of generality assume that $p_{i,k}$ was not examined. If Detector2

outputs that the polygons intersect then Detector2 fails because the polygons do not intersect. If Detector2 outputs that the polygons do not intersect then just before Detector2 outputs the answer that the polygons do not intersect the adversary "moves" $p_{i,k}$ by increasing (decreasing) the x -coordinate of $p_{i,k}$. The x -coordinate of $p_{i,k}$ value is increased (decreased) if $i = 1$ or if $i \neq 1$ and $n/2+2 \leq k \leq n$ (if $i=M$ or if $i \neq M$ and $2 \leq k \leq n/2$) until $p_{i,k}$ intersect P_{i+1} (P_{i-1}). "Moving" $p_{i,k}$ does not change the property of monotonicity of P_i as the x -values of the vertices are independent (see Example 3.6.1). Hence, Detector2 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. ♦

Figure 3.6.3 The arrangement of polygons $P_1, \dots, P_M$.

Theorem 3.6.1: Determining whether any pair of M simple n -vertex polygons monotone with respect to the same direction intersect requires $\Omega(Mn)$ time.

Proof: The proof is a direct result of Lemma 3.6.1 and Lemma 3.6.2. ♦

3.6.2 LOWER BOUND FOR VISIBILITY HULLS

In the previous lower bound argument an intersection between two simple polygons P and Q , monotone with respect to a direction d , can occur between any edges pair (e_p, e_q) , e_p in P and e_q in Q , as long as the monotonicity of P and Q is not violated. However, this is not the case when the monotone polygons are visibility hulls of two simple non-intersecting polygons.

Observation 3.6.1: Let P and Q be two simple non-intersecting n -vertex polygons such that the P is not contained in the $VH(Q, d)$ and vice versa Q is not contained in the $VH(P, d)$. An intersection between the visibility hulls of P and Q , with respect to a direction d , occurs between a lid of $VH(P, d)$ ($VH(Q, d)$) and an edge of Q (P) (see Figure 3.6.4).

Figure 3.6.4 An intersection between the visibility hulls of P and Q (d is the x -axis).

Despite the limitation described in Observation 3.6.1, an $\Omega(Mn)$ lower bound can be achieved for testing whether a pair of visibility hulls of M non intersecting n -vertex simple polygons intersects. This lower bound is achieved by modifying the polygons defined in Lemmas 3.6.1 and 3.6.2. In this section only the concept of how to construct the polygons is given. The proofs are similar to those of Lemmas 3.6.1 and 3.6.2 and are, therefore, omitted. In Lemma 3.6.3 an arrangement is given of two polygons P and Q , with m and n vertices respectively, that requires at least $(n-3)/2$ tests in order to determine whether the visibility hulls of P and Q intersect (where n is less than m).

Lemma 3.6.3: Let P and Q be two simple non-intersecting polygons, with m and n vertices respectively. Let d be a direction of translation. Detecting whether the visibility hulls, in direction d , of P and Q intersect requires at least $(n-3)/2$ steps, where $n < m$.

Proof: Let P and Q be two non-intersecting simple polygons with m and n vertices, respectively. Let m and n be odd and let m be equal to $n+4$. Without loss of generality assume that d is the positive x -direction.

Let the vertices of P be:

$p_1 = (2.5, -1)$, $p_2 = (2.5, (m-3)/2 + 2)$, ..., $p_i = (3.5, (m-3)/2 - (i-4)/2)$ if $2 < i < m-1$ is even and $p_i = (5.5, (m-3)/2 - (i-7)/2) - \epsilon$ if $2 < i < m-1$ is odd, ..., $p_{m-1} = (3.5, 1)$, $p_m = (5.5, 2 - \epsilon)$, where $\epsilon \rightarrow 0$.

Let the vertices of Q be:

$q_1 = (5, 1)$, ..., $p_i = (7, i/2 + 2)$ if $1 < i < n-1$ is even and $p_i = (5, i/2 + 1)$ if $1 < i < n-1$ is odd, ..., $q_{n-1} = (8, n/2 + 3)$, $q_n = (8, 1)$. See Figure 3.6.5.

Figure 3.6.5 The polygons P and Q .

Direction d is the positive x -axis and, therefore, the visibility hulls of P and Q are as follows:

The vertices of the visibility hull of P be:

$p_1 = (2.5, -1)$, $p_2 = (2.5, (m-3)/2 + 2)$, ..., $p_i = (4.5, (m-3)/2 - (i-6)/2)$ if $2 < i < m-1$ is even and $p_i = (5.5, (m-3)/2 - (i-7)/2)$ if $2 < i < m-1$ is odd, ..., $p_{m-1} = (4.5, 2)$, $p_m = (5.5, 2)$.

The vertices visibility hull of Q be:

$q_1 = (5, 1)$, ..., $q_i = (6, i/2 + 2)$ if $1 < i < n-1$ is even and $q_i = (5, \lfloor i/2 \rfloor + 1)$ if $1 < i < n-1$ is odd, ..., $q_{n-1} = (8, n/2 + 3)$, $q_n = (8, 1)$. See Figure 3.6.6.

Figure 3.6.6 The visibility hulls of P and Q .

Whenever vertex $q_i(p_j)$ is accessed by the detecting algorithm the adversary will give, for free, the information which edge of P (Q) vertex $q_i(p_j)$ may intersect. If there exists an algorithm that examines less than $(n-3)/2$ vertices then there is at least one vertex q_j among $q_3, q_5, \dots, q_{n-2}$ that was not examined and that its corresponding vertex p_{m-j+3} was not examined. The adversary then "moves" q_j by decreasing the y -coordinate of q_j until it intersects edge (p_{m-j+3}, p_{m-j+2}) . Therefore, the visibility hulls of P and Q intersect and the algorithm fails to detect the intersection (see Figure 3.6.7). "Moving" q_i does not cause the simple polygons to intersect. The adversary can find which vertex should be moved by traversing all vertices of Q . ♦

Figure 3.6.7 The visibility hulls of P and Q intersect

The adversary argument, which was used in Lemma 3.6.3, can be modified and used to prove the lower bound of $\Omega(Mn)$ for the following problem: Given a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple n -vertex polygons detect whether the visibility hulls of any two polygons intersect. One way of achieving the $\Omega(Mn)$ lower bound is by arranging the M polygons in pairs in such a way that each pair has the properties of the simple polygons of Lemma 3.6.3.

Lemma 3.6.4: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple n -vertex polygons, and let d be a direction of translation. Detecting whether the visibility hulls of any pair of polygons intersect requires at least $M * (n-4)/8$ steps.

Proof: The proof is similar to the proof of Lemma 3.6.2 and, therefore, omitted. The construction of the polygons and the visibility hulls is illustrated in Figure 3.6.8 and Figure 3.6.9. ♦

Figure 3.6.8 Two simple polygons P_i and P_k that are used in Lemma 3.6.4

Figure 3.6.9 A construction of the visibility hulls

Theorem 3.6.2: Let $S = \{P_1, P_2, \dots, P_M\}$ be a set of M simple n -vertex polygons, and let d be a direction of translation. Detecting whether the visibility hulls of any pair of polygons intersect requires at least $\Omega(Mn)$ steps.

Proof: The proof is a direct result of Lemma 3.6.3 and Lemma 3.6.4. ♦

3.6.3 LOWER BOUND FOR THE UNI-DIRECTIONAL SEPARABILITY PROBLEM

In this section a lower bound to the uni-directional separability problem for a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons is given. In Section 3.6.3.1, a lower bound of $\Omega(M \log M + Mn)$ for determining a translation order is given. In Section 3.6.3.2, it is shown that the same lower bound, $\Omega(M \log M + Mn)$, holds when the problem of detecting whether a translation order exists is presented.

3.6.3.1 LOWER BOUND FOR DETERMINING A TRANSLATION ORDER

In this section an $\Omega(M \log M + Mn)$ lower bound to the problem of determining a translation order, of a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons in the direction d , is given. The $\Omega(M \log M)$ lower bound is obtained by using the Integer Sorting Problem. The Integer Sorting Problem can be expressed as follows: given M distinct integers N_i , $1 \leq i \leq M$, sort all points in a decreasing order.

Theorem 3.6.3: Determining a translation order of a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons in the direction d requires at least $\Omega(M \log M)$ time.

Proof: The transformation technique for lower bounds is used in this proof. The theorem is proven by transforming an instance of the Integer Sorting Problem to the problem of determining a translation order of a set of polygons, denoted as the *Translation Order Determination Problem* (see Figure 3.6.10).

The transformation from the Integer Sorting Problem to the Separability Detecting Problem is as follows. For each integer N_i , $1 \leq i \leq M$, of the Integer Sorting Problem a square-shaped simple 4-vertex polygon P_i , $1 \leq i \leq M$, is constructed. The vertices of P_i , $1 \leq i \leq M$, are the following: $P_{i,1} = (N_i, 0)$, $P_{i,2} = (N_i - 0.25, 0)$, $P_{i,3} = (N_i - 0.25, 0.25)$ and $P_{i,4} = (N_i, 0.25)$ see Figure 3.6.11. The total width of each polygon P_i , $1 \leq i \leq M$, is less than 1 (x -coordinate of $P_{i,4}$ - x -coordinate of $P_{i,2}$) and, therefore, there is no pair of polygons P_i and P_j , with $P_{i,1} \neq P_{j,1}$ that intersect. Transforming the Integer Sorting Problem to the Translation Order Determination Problem requires $O(M)$ time because each polygon consists of only four vertices. Let $f(M, n)$ denote the time required for determining the translation order of the M polygons. Polygon P_i is translated before polygon P_j if, and only if, $N_i > N_j$. Therefore, the output of the Translation Order Determination Problem determines the sorted order of the integers. Converting the output of the Separability

Detecting Algorithm into a correct output for the Integer Sorting Algorithm require $O(M)$ time. The Integer Sorting Problem requires $\Omega(M \log M)$ time [AK].

Hence, the Separability Detecting Problem requires at least $\Omega(M \log M) - O(M) - O(M)$, which yields $f(M, n) \geq \Omega(M \log M)$. ♦

Figure 3.6.10 The transformation of the Integer Sorting Problem to the Translation Order Determination Problem

Figure 3.6.11 The transformation of the points to rectangles

Theorem 3.6.4: Determining a translation order of a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons in the direction d requires at least $\Omega(M \log M + Mn)$ time.

Proof: (i) $\Omega(Mn)$ - A translation order exists if, and only if, the visibility hulls of the polygons do not intersect. Since determining a translation order for the set of polygons is at least as hard as detecting whether the visibility hulls intersect, the $\Omega(Mn)$ lower bound, from Theorem 3.6.2, follows.

(ii) $\Omega(M \log M)$ - This lower bound follows from Theorem 3.6.3. ♦

3.6.3.2 LOWER BOUND FOR DETECTING WHETHER A TRANSLATION ORDER EXISTS

In Section 3.6.3.1 an $\Omega(Mn + M \log M)$ lower bound for the problem of determining a translation order has been derived. Clearly, that problem is at least as hard as that of detecting whether such an order exists. However, as shown in this section $\Omega(Mn + M \log M)$ is also a lower bound to the seemingly simpler problem of detecting whether a given set of M simple non-intersecting n -vertex polygons is separable in a given direction. The problem of detecting whether a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons is separable in a given direction d is denoted as ***Separability Detecting Problem***. An algorithm that solves the Separability Detecting Problem is denoted as ***Separability Detecting Algorithm***. First an $\Omega(Mn)$ lower bound is given and then an $\Omega(M \log M)$ lower bound is given.

Theorem 3.6.5: Detecting whether a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons is separable in a given direction d requires at least $\Omega(Mn)$ time.

Proof: The transformation technique for lower bounds is used in this proof. The theorem is proven by transforming the problem of detecting whether there exists a pair of polygons whose visibility hulls intersect to the Separability Detecting problem (see Figure 3.6.12).

From Theorem 3.6.2 it is known that the problem of detecting whether there exists a pair of polygons whose visibility hulls intersect requires at least $\Omega(Mn)$ time. Transforming of the M visibility hulls to the Separability Detecting Algorithm can be done in $O(M)$ time by passing the M head pointers of the polygons. Let $f(M,n)$ denote the time require for testing the M polygons for separability. A set of polygons is separable in direction d if, and only if, each pair of polygons has non-intersecting visibility hulls in direction d . Thus, if the output of the separability test is "YES" then there is no pair of polygons whose visibility hulls intersect. If the answer is "NO" then there exists at least one pair of polygons whose visibility hulls intersect. Converting the output of the Separability Detecting Algorithm into a correct output for the visibility hulls algorithm require $O(1)$ time.

Hence, the Separability Detecting Problem requires at least $\Omega(Mn) - O(M) - O(1)$, which yields $f(M,n) \geq \Omega(Mn)$. ♦

Figure 3.6.12 The transformation of the intersecting visibility hulls problem to the Separability Detection problem

2 of/de 2

Micro-D

2 of/de 2

MicroD

Next an $\Omega(M \log M)$ lower bound to the Separability Detecting Problem is given using the Integer Element Uniqueness Problem. The Integer Element Uniqueness Problem can be expressed as follows: given M integers N_i , $1 \leq i \leq M$, decide if any two are equal.

Theorem 3.6.6: Detecting whether a set $S = \{P_1, P_2, \dots, P_M\}$ of M simple non-intersecting n -vertex polygons is separable in a given direction d requires at least $\Omega(M \log M)$ time.

Proof: The transformation technique for lower bounds is used in this proof. The theorem is proven by transforming an instance of the Integer Element Uniqueness Problem to the Separability Detecting problem (see Figure 3.6.13).

The transformation from the Integer Element Uniqueness to the Separability Detecting Problem is as follows. For each integer N_i , $1 \leq i \leq M$, of the Integer Element Uniqueness Problem a V-shaped simple 4-vertex polygon P_i , $1 \leq i \leq M$, is constructed. The vertices of P_i , $1 \leq i \leq M$, are the following: $P_{i,1} = (N_i, i)$, $P_{i,2} = (N_i - 0.25, i + M + 2)$, $P_{i,3} = (N_i, i + 0.5)$ and $P_{i,4} = (N_i + 0.25, i + M + 2)$ see Figure 3.6.14. The total width of each polygon P_i , $1 \leq i \leq M$, is less than 1 (x -coordinate of $P_{i,4}$ - x -coordinate of $P_{i,2}$) and, therefore, there is no pair of polygons P_i and P_j that intersect. 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. Two polygons P_i and P_j are nested inside each other if, and only if, they cannot be separated. Two polygons P_i and P_j are nested if, and only if, $N_i = N_j$. Therefore, if the output of the separability test is "YES" then no two integers are equal in the Integer Element Uniqueness Problem. If the output of the separability test 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. Converting the output of the Separability Detecting Algorithm into a correct output for the

Integer Element Uniqueness Algorithm requires $O(1)$ time. The Integer Element Uniqueness Problem requires $\Omega(M \log M)$ time [SP].

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

Figure 3.6.13 The transformation of the Integer Element Uniqueness Problem to the Separability Detection problem

Figure 3.6.14 The construction of the polygons during the transformation

Theorem 3.6.7: The uni-directional separability problem of a set of M n -vertex non-intersecting simple polygons can be solved in $O(M \log M + Mn)$ time, and this is optimal.

Proof: The proof is a direct result of Algorithm 3.4.2 and Theorems 3.6.3, 3.6.4 and 3.6.5. ♦

CHAPTER 4

THREE DIMENSIONAL SPACE

4.1 INTRODUCTION

In this chapter directional separability problems in three dimensional space are discussed. Separability problems in three dimensional space are more complex than those in two dimensional space. Guibas and Yao [GY] have shown that line segments and convex polygons in two dimensional space can be separated in any given direction. In three dimensional space the existence of such a translation order for a given direction is not always guaranteed, not even when the objects are line segments. An example of three line segments that cannot be separated in a given direction, in three dimensional space, is given by Nurmi [N] (see Example 4.1.1). Naturajan [Na] gives an example of eight convex polygons that cannot be separated in any given direction. Dawson [D] gives an example of 12 convex polyhedra that cannot be separated in any given direction.

Example 4.1.1: Let A , B and C denote three line segments each of length 60cm. A , B and C are intertwined such that A is above B and A touches B , B is above C and touches C , and C is above A and touches A (see Figure 4.1.1). Let d , the direction of translation, be perpendicular to the page (the plane that contains the page). None of the objects can be translated without colliding with one of the two remaining elements.

Figure 4.1.1 Line segments A , B and C , d is directed from the page up

In Section 4.2 the uni-directional separability problem and the multi-directional separability problems of line segments are presented. In Section 4.3 the uni-directional separability problem of polygonal faces in three dimensional space is discussed. In Section 4.4 the uni-directional separability problem of polyhedra in three dimensional space is presented. In Section 4.5 the all-directions separability problem of simple polyhedra and simple polygonal faces, in three dimensional space, is presented.

4.2 SEPARABILITY OF A SET OF LINE SEGMENTS

In this section the uni-directional and the multi-directional separability problems, introduced in Section 1.2, of a set of line segments are discussed. In Section 4.2.1, an example of six lines, that can be arranged, such that for any given direction no solution to the uni-directional separability problem exists. In Section 4.2.2 the multi-directional separability of a set of line segments is studied.

4.2.1 INTERLOCKING LINES

The uni-directional separability problem of a set of line segments can be solved, by finding the *idom* relations for pairs of line segments that intersect, in $O((M + K) \log M)$ using the algorithm by Bentley and Ottmann [BO], where K is the number of intersections between the line segments. A DFS (Depth First Search) algorithm is then used to examine whether a cycle exists in the graph representation of the *idom* relation. If no cycles occur then a linear extension of the relation *idom* is found using a topological sort algorithm. This algorithm can be improved by using the $O(M \log^2 M / \log \log M + K)$ algorithm by Chazelle [C] for finding all intersection points among a set of line segments, where K is the number of intersections between the line segments. Although these algorithms may report that no solution to the uni-directional separability problem exists, (see Example 4.1.1), it may be that a translation order exists for a different direction. In this section it is shown that as few as six line segments can be arranged in such a way that no solution for the uni-directional separability problem exists, i.e. for any given direction no translation order of the line segments exists.

Let A, B, C, D, E and F denote six line segments each of length 400cm. Let T denote a plane parallel to the x - y plane and let R be an equilateral triangle with vertices a, b and c with a side length equal to 60cm. Triangle R is placed on the x - y axis of plane T such that the edge bc is on the x -axis and a is on the positive y -axis (see Figure 4.2.1). Lay the line segments A, B and C on R such that A (B, C) is on the edge ab (edge bc , edge ca) and the center of A (B, C) is on the center of the edge ab (edge bc , edge ca). Intertwine A, B and C such that A is above B and A touches B at b , B is above C and B touches C at c , and C is above A and touches A at a (see Figure 4.2.2). Line segment D (E, F) is placed such that D (E, F) is perpendicular to T , touching A and C (A and B, B and C) at a (b, c), and is outside triangle R (see Figure 4.2.3).

Figure 4.2.1 The placement of triangle R on the plane T

Figure 4.2.2 Line A , B and C are interlaced on T

Figure 4.2.3 The final construction of the six line segments

Line segments A , B and C interlace with each other and thus, none of the line segments lies completely on T . Hence, each of the line segments intersects T at exactly one point. Line segment A intersects T at a , line segment B intersects T at b , and line segment C intersects T at c .

Let L denote the half line that originates at a and contains the line segment A . Let L' denote the half line, parallel to L , in the opposite direction (direction $-L$) and originating at a . Let ∂ denote the angle between L and the projections of L on T and let $-\partial$ denote the angle between L' and the projections of L' on T .

Let d_1 (d_2) denote a direction vector with the following properties. a) The angle between the projection of d_1 (d_2) on T and the x -axis, is $\pi/3$ ($4\pi/3$). The unit vector of the

projection of d_1 (d_2) on T is the vector $(1/2, \sqrt{3}/2)$ ($(-\sqrt{3}/2, -1/2)$). b) The angle between d_1 (d_2) and its projection on T is $-\partial$ (∂).

Lemma 4.2.1: Line segment A can be translated only in the direction vector d_1 or direction vector d_2 .

Proof: Assume the contrary, i.e. that there exists a direction vector d , that its unit vector differs from the vectors d_1 and d_2 , and that A can be translated in direction d . Let α denote the angle of the projection of d on T .

There are three cases to be considered:

1. The angle α differs from the angle of the projections of d_1 or d_2 on T .

a) $\pi/3 < \alpha < \pi$ - In this case A will collide with E .

b) $\pi \leq \alpha < 4\pi/3$ - In this case A will collide with D .

c) $4\pi/3 < \alpha < 7\pi/3$ - In this case A will collide with B or C .

2) $\alpha = \pi/3$ - and the angle of the projection of d on T is not equal to $-\partial$ otherwise, the unit vector of d is identical to the unit vector of d_1 . If A is to be translated in a direction vector d then A will collide with B or C .

3) $\alpha = 4\pi/3$ - and the angle of the projection of d on T is not equal to ∂ otherwise, the unit vector of d is identical to the unit vector of d_2 . If A is to be translated in a direction vector d then A will collide with B or C . ♦

In a similar way it is easy to prove that there are only two directions in which B can be translated and two directions that C can be translated in. Thus, there are only six possible directions of translation for A , B , C , D , E and F .

Lemma 4.2.2: Only the line segments A and D can be translated in direction d_1 .

Proof: Assume that A and D have already been translated in direction d_1 . When the rest of the line segments are to be translated in direction d_1 the following situation occurs: C is blocked by F , and F and E are blocked by B . Line segment B is above C , hence if B is translated in direction d_1 then B will collide with C (B is translated in direction $-\partial$). A cycle consisting of $CFBC$ exists in the digraph representing the translation order of B, C, E, F . Therefore, none of the remaining line segments B, C, E and F can be translated. ♦

4.2.2 TRANSLATION ORDER IN 3-DIMENSIONAL SPACE

In Section 4.2.1 an arrangement of six lines, such that no uni-directional translation order exists, was presented. This arrangement of the line segments can be separated when the multi-directional separability problem is used. In this section it will be shown that a translation order for any given set of line segments can always be found when the multi-directional separability problem is used.

First, it is shown that in any given set of line segments there is at least one line segment that can be translated to infinity without colliding with the rest of the line segments. This is proven by showing that a line segment can always be translated when one of its endpoints lies on the convex hull of the set of the line segments.

Lemma 4.2.3: Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of n line segments, let a_i and b_i denote the two endpoints of s_i and let $Q = \{a_i, b_i \mid \text{for } i = 1..n\}$ be the set of the endpoints of S . In this case $\text{CH}(S) = \text{CH}(Q)$.

Proof: a) $\text{CH}(S) \supseteq \text{CH}(Q)$ because $S \supseteq Q$.

b) $\text{CH}(Q) \supseteq \text{CH}(S)$. The convex hull of Q contains all line segments joining any two points in Q . Let S' denote all possible line segments connecting points of Q . Q is the set of endpoints of S , therefore, $S' \supseteq S \implies \text{CH}(Q) = \text{CH}(S') \supseteq \text{CH}(S)$. ♦

Lemma 4.2.3 guarantees the existence of at least one line segment such that one of its endpoints lies on the convex hull of the set of the line segments. Such a line segment can be translated to infinity using the following:

Lemma 4.2.4: Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of n line segments. Let a_i and b_i denote the endpoints of s_i and let $V = \{v_1, v_2, \dots, v_m\}$ be the set of vertices of $\text{CH}(S)$. Each of the line segments having one of its endpoints in V can be translated.

Proof: By Lemma 4.2.3 V consists of the endpoints of S . Assume, without loss of generality, that $v_1 = b_j$. Then s_j can be translated in direction $(a_j b_j)$. If s_j cannot be translated in direction $(a_j b_j)$ then there exists another line segment s_k which blocks s_j . The trajectory of s_j is outside of the $\text{CH}(S)$, therefore, s_k is outside $\text{CH}(S)$. This contradicts the convexity of $\text{CH}(S)$. ♦

Lemma 4.2.3 and Lemma 4.2.4 describe a brute force algorithm for the multi-directional separability problem of a set of line segments in three dimensional space.

Algorithm 4.2.1: Brute force algorithm

Input: A set $S = \{s_1, s_2, \dots, s_n\}$ of n line segments.

Output: A translation order of the set of line segments.

begin

while $(S \neq \emptyset)$ do

begin

1. Find the convex hull of S ;
2. Translate (output) all line segments having one of their endpoints on $CH(S)$;
3. Delete all line segments that were translated in Line 2 from S ;

end; {while}

end; {algorithm}

The complexity of Algorithm 4.2.1 is $O(n^2 \log n)$. Constructing the convex hull, of a set of n points in three dimensional space, can be done in $O(n \log n)$ using the algorithm by Preparata and Hong [PH]. Thus, Line 1 of the algorithm requires $O(n \log n)$. Deleting the k line segments found in Line 2 can be done in $O(k)$. Therefore, Line 3 requires at most $O(k)$ time. In the worst case, only two line segments are translated at each iteration of the while loop. Thus, the overall time complexity is $O(n^2 \log n)$ and $O(n)$ space.

Algorithm 4.2.1 is a brute force algorithm. It requires $O(n^2 \log n)$ time and $O(n)$ space. The lower bound for the problem is $\Omega(n)$ as a result of the size of the input/output of the problem. The big gap between the lower bound of $\Omega(n)$ and the upper bound of $O(n^2 \log n)$ has led to the development of another algorithm in order to obtain a better upper bound.

A different approach is used to achieve a complexity of $O(n \log n)$ time and $O(n)$ space. Here each of the line segments determines a sphere, where the center of each sphere is the origin. The radius of the sphere is defined as the maximum distance of the endpoints of the line segments from the origin. These spheres are sorted in decreasing value of their radii and this determines the translation order. Each of the line segments s_i is translated in the direction (a_i, b_i) (direction (b_i, a_i)) if b_i (a_i) was the endpoint with maximum distance.

Algorithm 4.2.2: Sphere algorithm:

Input: A set $S = \{s_1, s_2, \dots, s_n\}$ of n line segments.

Output: A translation order of the set of line segments.

begin

1. For $i := 1$ to n do begin

Find the maximum distance of the endpoints of line segment s_i from the origin;

end; {for}

2. Sort all line segments using the distance as a key;

3. Translate (output) the line segments in descending order using the sorted list;

end; {algorithm}

The correctness Algorithm 4.2.2 relies on the following lemma.

Lemma 4.2.5: Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of n line segments. Let r_i denote the maximum distance of s_i from the origin and let E be a priority queue of S sorted by decreasing value of r_i ($i = 1..n$). If r_k is the first element of E then s_k can be translated.

Proof: Without loss of generality, let a_k be the endpoint of s_k which corresponds to r_k . Let W denote a sphere that is centred at the origin and has radius r_k (a_k is on the boundary of W). If line segment s_k is translated in direction (b_k, a_k) then its trajectory is outside the sphere W . Assume that s_k cannot be translated, then it is blocked by another segment s_j . The line segment s_j lies partially (or totally) outside the sphere W . Therefore, $r_j > r_k$ contradicts the fact that r_k is the largest radius among $r_1, r_2, \dots, r_n$. ♦

The time complexity Algorithm 4.2.2 is $O(n \log n)$ and $O(n)$ space. Lines 1 and 3 each require $O(n)$ time and Line 2 requires $O(n \log n)$ time.

Theorem 4.2.1: The multi-directional separability problem can be solved in $O(n \log n)$ and $O(n)$ space.

Proof: The theorem is a direct result of Algorithm 4.2.2 and Lemma 4.2.5. ♦

4.3 POLYGONAL FACES IN THREE DIMENSIONAL SPACE

In this section an algorithm for the uni-directional separability problem of a set of polygonal faces in three dimensional space is presented. In Section 4.3.1 some preliminary definitions and operations are given. In Section 4.3.2, the idom relation is described. Section 4.3.2 presents an algorithm for finding a translation order of simple polygonal faces in three dimensional space. In Section 4.3.3 an algorithm for convex polygonal faces is given.

4.3.1 PRELIMINARY OPERATIONS AND DEFINITIONS

Let P and Q be two polygonal faces with m and n vertices respectively and let d be a direction in which P and Q are to be translated.

Notation:

Let T_P and T_Q denote the two planes that contain P and Q , respectively.

Let L_{PQ} denote the intersection, if one exists, between T_P and T_Q (see figure 4.3.1).

Let T_d denote a plane such that its normal is d .

Let P^d and Q^d and L_{PQ}^d denote the projections, in direction d , of P , Q and L_{PQ} on T_d respectively.

If T_P and T_Q are parallel then L_{PQ}^d is empty. If T_P and T_Q are perpendicular to T_d then L_{PQ}^d is a point. In any other case, L_{PQ}^d is a line. L_{PQ}^d is *non-degenerate* if it is a line, otherwise L_{PQ}^d is *degenerate*.

Figure 4.3.1 The intersection L_{PQ} between T_P and T_Q .

Definition: Let L^d_{PQ} be a line. Let L^d_{PQ} be the x -axis of T_d . A point p of T_P is *above* (*below*) L_{PQ} if the y -value of its projection on T_d is greater or equal to (less than) 0.

Observation 4.3.1: Objects in three dimensional space can collide only if their projections (images) on a plane perpendicular to the direction of translation intersect. (This has also been observed by Nurmi [N]). ♦

Observation 4.3.2: Let line L^d_{PQ} be non-degenerate. Without loss of generality assume that L^d_{PQ} is the x -axis of T_d . Assume that a ray R , in direction d , penetrates T_P before it penetrates T_Q . Furthermore, assume that the projection of R on T_d is above L^d_{PQ} . Then every ray R' whose projection on T_d is above L^d_{PQ} first penetrates T_P and then T_Q . ♦

Observation 4.3.3: Let line L^d_{PQ} be non-degenerate. Without loss of generality assume that L^d_{PQ} is the x -axis of T_d . Assume that a ray R , in direction d , penetrates first T_P and

then T_Q . Furthermore, assume that the projection of R on T_d is below L^d_{PQ} . Then every ray R'' whose projection on T_d is below L^d_{PQ} first penetrates T_Q and then T_P . ♦

An illustration of Observation 4.3.2 is: Consider the planes T_P and T_Q to be two consecutive pages in a book, for example, pages 50 and 52 respectively, and L^d_{PQ} to be the part of the binding that holds the pages together. When the book is open and pages 50 and 52 are held vertically then every ray passing from left to right will penetrate page 50 first and then page 52.

The illustration of Observation 4.3.3 is similar. When one opens a book at pages 50 and 52 and holds the book upside down so that pages 50 and 52 are hanging down then every ray passing from left to right will penetrate page 52 first and then page 50 (the book is held so that one can read the title).

Lemma 4.3.1: Let P and Q be two polygonal faces such that T_P and T_Q are not parallel to each other. Let d be a direction of translation such that neither T_P nor T_Q is perpendicular to T_d . If L^d_{PQ} does not intersect P^d or Q^d then P and Q are movably separable.

Proof: Let L^d_{PQ} be the x -axis of T_d . If L^d_{PQ} does not intersect P^d then P^d is either completely above or completely below L^d_{PQ} . Without loss of generality assume that P^d is above L^d_{PQ} . If P^d and Q^d do not intersect then P and Q are movably separable, according to Observation 4.3.1. Thus, assume that P^d and Q^d intersect. Let z be an intersection point between P^d and Q^d . Let u and v be the points on P and Q , respectively, such that the projections of u and v on T_d are u^d and v^d , respectively. Let R be a ray through v and u . The ray R which is parallel to d penetrates T_d above L^d_{PQ} . If R does not penetrate T_d above L^d_{PQ} then u^d and v^d are both above L^d_{PQ} contradicting the assumption that P^d and Q^d intersect only above L^d_{PQ} . Without loss of generality assume that R , through u and v , penetrates

first P at u and then Q at v . From Observation 4.3.2 every ray R' penetrates first T_P and T_Q and, therefore, Q can be translated before P . ♦

Lemma 4.3.2: Let P and Q be two polygonal faces such that T_P and T_Q are not parallel to each other. Let d be a direction of translation such that T_P and T_Q are not perpendicular to T_d . Let L^d_{PQ} be the x -axis of T_d . Let L^d_{PQ} intersect P^d and Q^d . P and Q are movably separable if P^d does not intersect Q^d or if P^d intersects Q^d either above or below L^d_{PQ} but not on both sides of L^d_{PQ} .

Proof: There are three cases to examine: (i) P^d does not intersect Q^d ; (ii) P^d intersects Q^d above and below L^d_{PQ} ; and (iii) P^d intersects Q^d either above or below L^d_{PQ} .

(i) P^d does not intersect Q^d - From Observation 4.3.1, P and Q are movably separable.

(ii) P^d intersects Q^d above and below L^d_{PQ} - In this case it is shown that P and Q are not movably separable.

Assume the contrary, i.e. that P^d intersects Q^d both above and below L^d_{PQ} and that P and Q are movably separable. If P^d intersects Q^d above (below) L^d_{PQ} then there exist points u^d_1 (u^d_2) on P^d and v^d_1 (v^d_2) on Q^d such that P^d intersects Q^d above (below) L^d_{PQ} at u^d_1 and v^d_1 (u^d_2 and v^d_2). Let u_1 (u_2) and v_1 (v_2) be the points on P and Q , respectively, such that the projections of u_1 , u_2 , v_1 and v_2 on T_d are the points u^d_1 , u^d_2 , v^d_1 and v^d_2 respectively. Let R and R' be two rays, parallel to d and oriented as d , through u_1 and v_1 and through u_2 and v_2 , respectively. The ray R penetrates T_d above L^d_{PQ} and the ray R' penetrates T_d below L^d_{PQ} . Without loss of generality assume that R penetrates first T_P and then T_Q . From Observation 4.3.3 R' will then penetrate T_Q before T_P . If P is to be translated first then point u_1 on P will collide with point v_1 on Q . If Q is to be translated first then point v_2 will collide with point u_2 . Hence, P cannot be translated before Q and Q cannot be translated before P contradicting the assumption that P and Q are movably separable.

(iii) P^d intersects Q^d either above or below L^d_{PQ} - In this case it is shown that P and Q are movably separable. Without loss of generality assume that P^d and Q^d intersect above L^d_{PQ} and that P^d and Q^d do not intersect below L^d_{PQ} . Let z be an intersection point between P^d and Q^d . Let u and v be the points, on P and Q respectively, such that the projections of u and v on T_d are u^d and v^d , respectively. Let R be a ray through v and u . The ray R , which is parallel to d , penetrates T_d above L^d_{PQ} . If R does not penetrate T_d above L^d_{PQ} then u^d and v^d are below L^d_{PQ} contradicting the assumption that P^d and Q^d intersect only above L^d_{PQ} . Without loss of generality assume that R , through u and v , penetrates first P at u and then Q at v . From Observation 4.3.2 every ray R' penetrates first T_P and T_Q and, therefore, Q can be translated before P . ♦

Lemma 4.3.3: Let P and Q be two polygonal faces such that T_P and T_Q are not parallel to each other. Let d be a direction of translation such that L^d_{PQ} is degenerated to a point. Let u_1 and u_2 (v_1 and v_2) be two points on P (Q) such that all intersection points of P (Q) with L_{PQ} are contained in the segment u_1u_2 (segment v_1v_2). P and Q are movably separable if segments u_1u_2 and v_1v_2 do not intersect.

Proof: L^d_{PQ} is degenerated to a point and therefore, T_P and T_Q are perpendicular to T_d . Thus, if P^d and Q^d intersect then an intersection can occur only on L^d_{PQ} (L^d_{PQ} is a point and P^d and Q^d are lines). From Observation 4.3.1, P and Q collide only if their projections intersect, therefore, the relation between P and Q should be examined only along L_{PQ} .

Assume the contrary, i.e that the segments u_1u_2 and v_1v_2 intersect and P and Q are movably separable. Without loss of generality assume that v_1 is contained in segment u_1u_2 . Let the vector u_1u_2 be parallel to d and in the same direction as d . If Q is to be translated first then v_1 will collide with u_2 and therefore Q cannot be translated before P . If P is to be translated before Q then u_1 will collide with v_1 and, therefore, P cannot be

translated before Q . Hence, P cannot be translated before Q and Q cannot be translated before P , contradicting the assumption that P and Q are movably separable. ♦

Lemma 4.3.4: Let P and Q be two polygonal faces on the same plane ($T_P = T_Q$) and let d be a direction of translation. P and Q are movably separable if one of the following conditions is fulfilled: a) T_P is not perpendicular to T_d . b) T_P is perpendicular to T_d and the visibility hulls of P and Q , with respect to d , do not intersect.

Proof: There are two cases to examine:

- (i) T_P is not perpendicular to T_d - in this case P^d and Q^d do not intersect and from Observation 4.3.1 P and Q are movably separable;
- (ii) T_P is perpendicular to T_d and the visibility hulls of P and Q , with respect to d , do not intersect - this case is the same as the uni-directional separability problem of simple polygons in two dimensional space. From Theorem 3.2.1 P and Q are movably separable if, and only if, the visibility hulls of P and Q , with respect to d , do not intersect. ♦

Lemma 4.3.5: Let P and Q be two polygonal faces such that T_P and T_Q are parallel to each other and $T_P \neq T_Q$. P and Q are movably separable in any given direction of translation d .

- Proof:** There are two cases to examine: (i) P^d does not intersect Q^d ; (ii) P^d intersects Q^d .
- (i) P^d does not intersect Q^d - From Observation 4.3.1 P and Q are movably separable.
 - (ii) P^d intersects Q^d - In this case it is shown that P and Q are likewise movably separable.

Let z be an intersection point between P^d and Q^d . Let u and v be the points, on P and Q respectively, such that the projections of u and v on T_d are u^d and v^d , respectively. Let R be a ray through v and u . The ray R , which is parallel to d , penetrates T_d above L^d_{PQ} . If R does not penetrate T_d above L^d_{PQ} then u^d and v^d are not be above L^d_{PQ} contradicting the

assumption that P^d and Q^d intersect only above L^d_{PQ} . Without loss of generality assume that R , through u and v , penetrates first P at u and then Q at v . From Observation 4.3.2 every ray R' penetrates first T_P and T_Q and, therefore, Q can be translated before P . ♦

Lemma 4.3.6: Let P and Q be two polygonal faces and let d be a direction of translation. P and Q are movably separable if, and only if, the *idom* relation between P and Q is asymmetric.

Proof: The proof is similar to that of Lemma 3.3.1 and is, therefore, omitted. ♦

4.3.2 THE DOMINANCE RELATION

The *dom* relation, used by the algorithms, is defined as $dom = idom^+$ (the transitive closure of *idom*), see Section 2.1. The relation *idom* between two polygonal faces P and Q is asymmetric if, and only if, P and Q are movably separable (see Lemma 4.3.6). However, even if the *idom* relation between the polygonal faces is asymmetric the transitive closure of *idom* (*idom*⁺) may not necessarily be asymmetric (as illustrated in Example 4.3.1). Thus, the *dom* relation should be tested for asymmetry.

Example 4.3.1: Let A , B and C be the three line segments of Example 4.1.1. The transitive closure, *dom*, of *idom* is not asymmetric because $A \text{ dom } B$ and $B \text{ dom } A$. In Figure 4.3.2 the digraph representing the *idom* relation among A , B and C , of Example 4.1.1, is given.

Figure 4.3.2 The digraph representing the *idom* relation of A , B and C

4.3.3 SIMPLE POLYGONAL FACES ALGORITHM

4.3.3.1 THE ALGORITHM

The algorithm uses three passes. In the first pass the algorithm finds all pairs P_i and P_j for which the images P_i^d of P_i and P_j^d of P_j intersect. For all such pairs (P_i, P_j) it produces an *idom* relation between P_i and P_j . In the second pass, the graph representation of *idom* is examined for cycles. In the third pass, a topological sort is performed, on the digraph produced the *idom* relation. The output of the later step is a linear ordering of the polygonal faces. The third pass is executed provided that the graph representation of *dom* is acyclic.

In the first pass, the algorithm finds all pairs P_i and P_j for which the images P_i^d of P_i and P_j^d of P_j intersect (this is done by examining all possible pairs P_i and P_j). Each pair of polygons P_i and P_j whose projections P_i^d and P_j^d intersect is then examined as to whether the relation *idom* on P_i and P_j is asymmetric. If *idom* is symmetric then no translation order exists and the algorithm stops, otherwise, (*idom* is asymmetric) the algorithm determines whether $P_i \text{ idom } P_j$, or $P_j \text{ idom } P_i$, or P_i and P_j are incomparable in *idom*. At the end of this pass a digraph representing the *idom* relation among the polygonal faces is produced. The vertices of the graph represent the polygonal faces and an edge of the digraph directed from P_i to P_j represents the *idom* relation between P_i and P_j as $P_j \text{ idom } P_i$.

In the second pass, the graph is tested for acyclicity. The search for a cycle is done by using a DFS (Depth First Search algorithm) [Ta]. If no cycles occurred then the polygonal faces can be translated in direction d . A topological sort is then performed, provided that the graph is acyclic, on the dom relation to determine the translation order.

Algorithm 4.3.1: Polygonal faces in three dimensional space

Input: A set $S = \{P_1, P_2, \dots, P_M\}$ of M non intersecting n -vertex polygonal faces in three dimensional space.

d - direction of translation

Output: A translation order of the set of polygonal faces.

begin

1. Project the polygonal faces on the plane T_d .
2. For each pair of polygons P_i and P_j ($1 \leq i < j \leq M$) do
 - begin
 - If P_i and P_j are on the same plane then
 - begin { Lemma 4.3.4 }
 - Separability_On_The_Same_Plane ($P, Q, d, \text{Separable?}$);
 - end; {if}
 - else begin { P_i and P_j are not on the same plane }
 - If T_{P_i} and T_{P_j} are parallel then
 - begin { Lemma 4.3.5 }
 - Separability_Of_Parallel_Planes (P_i, P_j, d);
 - end; {if}
 - else begin { T_{P_i} and T_{P_j} are not parallel }
 - Find the intersection of the planes T_{P_i} and T_{P_j} ;
 - If T_{P_i} and T_{P_j} are perpendicular to T_d then
 - begin { Lemma 4.3.3 }
 - Separability_Of_Perpendicular_Planes ($P_i, P_j, d, \text{Separable?}$);
 - end; {if}
 - else
 - begin { T_{P_i} and T_{P_j} are not perpendicular }

```

    If  $L^d_{PQ}$  intersects  $P^d_i$  and  $P^d_j$ , then
    begin { Lemma 4.3.2}
        Separability_Of_Polygonal_Faces_2 ( $P_i, P_j, d, \text{Separable?}$ );
    end;{if}
    else
    begin { Lemma 4.3.1}
        Separability_Of_Polygonal_Faces_1 ( $P_i, P_j, d, \text{Separable?}$ );
    end;{else}
    end;{if}
    end;{if}
    end;{if}
    If Separable? = false then
    begin
        print( NO TRANSLATION ORDER EXISTS);
        halt;
    end; {if}
end; {for}
3. Execute a DFS algorithm in the graph representation of the dom relation;
4. If no cycles have occurred in 3 then
    begin
        perform a topological sort on the graph representation of the idom relation;
    end;
    else
    begin
        print( NO TRANSLATION ORDER EXISTS);
        halt;
    end; {else}
end. {algorithm}

```

Next is a short description of the procedures used by Algorithm 4.3.1. A more detailed description and the complexity analysis of the procedures can be found in Appendix A.

Procedure Separability_On_The_Same_Plane ($P, Q, d, \text{Separable?}$);

{This procedure checks whether two polygonal faces on the same plane are movably separable (see Lemma 4.3.4).

Input: P - polygonal face
 Q - polygonal face
 d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

The total cost of the procedure is $O(n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Parallel_Planes (P, Q, d);

{This procedure checks whether $\hat{r} \text{ dom } Q$ or $Q \text{ dom } P$ or P and Q are incomparable in dom . The planes T_P and T_Q are parallel to each other (see Lemma 4.3.5).

Input: P - polygonal face
 Q - polygonal face
 d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Perpendicular_Planes ($P, Q, d, \text{Separable?}$);

{This procedure checks whether P and Q are movably separable when T_P and T_Q are perpendicular to T_d (see Lemma 4.3.3).

Input: P - polygonal face
 Q - polygonal face
 d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

The total cost of the procedure is $O(n)$ time and $O(1)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Polygonal_Faces_2 ($P, Q, d, \text{Separable?}$);

{This procedure checks whether two polygonal faces (P and Q) on different planes are movably separable (see Lemma 4.3.2).

Input: P - polygonal face
 Q - polygonal face
 d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Polygonal_Faces_1 ($P, Q, d, \text{Separable?}$);

{ This procedure determines which of the two polygonal faces (P and Q) in different planes should be moved first (see Lemma 4.3.1).

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Cut_Polygon ($P, P_{\text{above}}, P_{\text{below}}, \text{Line}$);

{ This procedure rotates and translates the polygon P until the line Line is on the x -axis. The procedure then creates two new polygons P_{above} and P_{below} by partitioning the polygon P at the x -axis. P_{above} contains that portion of P located above or on the x -axis and P_{below} contains that portion of P located below or on the x -axis (see Figure 4.3.3)

Input: P - polygon

Line - line;

Output: P_{above} - polygon

P_{below} - polygon}

The cost of this procedure is $O(n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Figure 4.3.3 (a) the polygon P (b) P_{above} and P_{below}

4.3.3.2 COMPLEXITY OF ALGORITHM 4.3.1

Line 1 - Projecting a polygon is done in $O(n)$ time and thus for all the polygons the cost of the projection is $O(Mn)$.

Line 2 - In each iteration of Line 6 one of the following procedures is been executed:

- a) Procedure Separability_On_The_Same_Plane - The cost of the procedure is $O(n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .
- b) Procedure Separability_Of_Parallel_Planes - The cost of the procedure is $O(n)$ space and $O(n \log n)$ time, where n is the number of vertices in P and Q .
- c) Procedure Separability_Of_Perpendicular_Planes - The cost of the procedure is $O(n)$ time, where n is the number of vertices in P and Q .
- d) Procedure Separability_Of_Polygonal_Faces_2 - The cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .
- e) Procedure Separability_Of_Polygonal_Faces_1 - The cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

The number of iterations of Line 2 is $M*(M-1)/2$ ($O(M^2)$). In the worst case, the cost of each iteration is $O(n \log n)$ (where M is the number of polygons and n is the number of vertices in each polygon). Hence, in the worst case, the complexity of Line 2 is $O(M^2 n \log n)$ time and $O(n)$ space.

Line 3 - The DFS algorithm is executed on the graph representation of the *dom* relation which was produced by Line 2. The execution time of a DFS algorithm is linear in the number of *dom* relation pairs [Ta]. In the worst case, the number of *dom* relation pairs is $O(M^2)$ and thus the execution time of Line 7 ranges from $O(1)$ to $O(M^2)$.

Line 4 - A topological sort is done on the *dom* relation produced by Line 2. The execution time of a topological sort is linear in the number of *dom* relations [K]. In the worst case,

the number of *dom* relation pairs is $O(M^2)$ and, thus, the execution time ranges from $O(M)$ to $O(M^2)$.

Theorem 4.3.1: The uni-directional separability problem for a set of M simple non-intersecting n -vertex polygonal faces can be solved in $O(M^2 n \log n)$ time and $O(Mn)$ space.

Theorem 4.3.1 states a worst case running time of $O(M^2 n \log n)$. In the first pass the algorithm finds all pairs P_i and P_j , for which the images P_i^d of P_i and P_j^d of P_j intersect and produces an *idom* relation between P_i and P_j (Line 2). Instead of comparing all possible pairs of polygonal faces, as in Line 2, two approximations can be used in order to find the pairs P_i and P_j for which the projections on T_d (P_i^d and P_j^d) might intersect. In the first approximation the algorithm finds all vertices of P_i^d , $1 \leq i \leq M$ with the minimum and the maximum x -coordinate, denoted as P_{imin}^d and P_{imax}^d respectively, $1 \leq i \leq M$. All vertices P_{imin}^d and P_{imax}^d , $1 \leq i \leq M$, are then sorted, in increasing x -value, and the intervals $[P_{imin}^d, P_{imax}^d]$, $1 \leq i \leq M$, are examined for an intersection. All pairs of polygons that have intersecting intervals are put on a linked list. In the second approximation, the convex hulls of the projection of each pair P_i and P_j on the linked list are examined for an intersection (namely the convex hulls of P_i^d and P_j^d). If the convex hulls of P_i^d and P_j^d intersect then the pair P_i and P_j is left on the linked list, otherwise the pair is deleted from the linked list. After the approximation steps the algorithm continues as before and examines each pair of polygons P_i and P_j on the linked list as to whether the relation *idom* on P_i and P_j is asymmetric.

Enhancement code:

begin

1. Find the vertices with the minimum and maximum x -coordinate of each P_j^d , $1 \leq j \leq M$.
2. Find the convex hulls of the projection of each polygonal face (P_j^d for $1 \leq j \leq M$).

3. Find all pairs of polygons P_i and P_j that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect, $1 \leq i, j \leq M$ and $i \neq j$. {Approximation 1}
 4. For each pair of polygons P_i and P_j found in 3, check whether the convex hulls of P_i^d and P_j^d intersect. {Approximation 2}
- end; {enhancement}

Line 1 - Finding the vertices with extremal x-coordinates is executed in $O(n)$ time and $O(1)$ space for each projected polygon and, thus, the total cost for all the projected polygons is $O(Mn)$ time and $O(M)$ space.

Line 2 - Finding the convex hull of a polygon can be done in $O(n)$ time and $O(n)$ space using the algorithm by El Gindy and Avis [EA], Dean and Sack [DS], and Lee [L]. Thus the total cost of finding the convex hulls of the M projected polygons can be done in $O(Mn)$ time and $O(Mn)$ space.

Line 3 - Finding all pairs of polygons P_i and P_j that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect with each other takes $O(K + M \log M)$ time using $O(K)$ space, where K is the number of pairs of intervals that intersect (output). In the worst case, all segments (intervals) will intersect with each other. Therefore, K ranges between 0 and M^2 .

Line 4 - Finding all pairs P_i^d and P_j^d whose convex hulls intersect can be done in $O(U + K \log n)$ time and $O(U)$ space, where K is the number of intervals that intersect (from Line 3) and U is the number of pairs whose convex hulls intersect (output). In the worst case, all convex polygons will intersect with each other. Therefore, U ranges between 0 and K .

All pairs of polygonal faces whose convex hulls of their projections intersect, are used as input to Line 2 of Algorithm 4.3.1. Thus, the total cost of Algorithm 4.3.1 is $O(Mn + M \log M + K \log n + Un \log n)$ and $O(Mn)$ space, where K is the output size of Line 3 of the enhancement code and U is the output of Line 4 of the enhancement code.

Theorem 4.3.2: The running time of Algorithm 4.3.1, with the enhancement code, ranges between $O(Mn + M \log M)$, in the best case, and $O(M^2 n \log n)$ in the worst case.

Proof: The total cost of Algorithm 4.3.1, when the enhancement code is used, is $O(Mn + M \log M + K \log n + Un \log n)$ and $O(Mn)$ space, where K is the output size of Line 3 and U is the output size of Line 4 of the enhancement code. K and U range between 0 and $O(M^2)$ and thus the execution time of Algorithm 4.3.1 is, in the best case, $O(Mn + M \log M)$, when K and U are equal to 0, and $O(M^2n \log n)$ in the worst case which arises when U is equal to $O(M^2)$. ♦

4.3.4 CONVEX POLYGONAL FACES ALGORITHM

4.3.4.1 INTRODUCTION

In the previous section (Section 4.3.3), a solution to the uni-directional separability problem of simple polygonal faces was given. A reduction in the time complexity can be achieved if the polygonal faces are known to be convex. In this case, in three dimensional space, the time complexity of Algorithm 4.3.1 can be reduced to $O(Mn + M \log M + K \log n)$, where M is the number of polygonal faces, n is the number of vertices in each polygonal face, and K is the number of pairs of faces that were examined for an intersection. In Section 4.3.4.2 some preliminary operations are given. In Section 4.3.4.3 the algorithm is described. In Section 4.3.4.5 the time complexity of the algorithm is analysed.

4.3.4.2 PRELIMINARY OPERATIONS

The following property of convex polygonal faces in three dimensional space enables a reduction in the time complexity over Algorithm 4.3.1.

Property 4.3.1: Let P and Q be two non-intersecting convex polygonal faces in three dimensional space. The relation *idom* for P and Q is asymmetric.

Proof: Assume the contrary, i.e. that there exist two non-intersecting convex polygonal faces P and Q , in three dimensional space, and a direction d such that $P \text{ idom } Q$ and $Q \text{ idom } P$.

P . $P \text{ idom } Q$ and, therefore, there exist points u_1 on P and v_1 on Q such that a half line (parallel to d , oriented as d and emanating at v_1) intersects P at u_1 . $Q \text{ idom } P$ and, therefore, there exist points v_2 on Q and u_2 on P such that a half line (parallel to d , oriented as d and emanating at u_2) intersects Q at v_2 . Let T' be a plane perpendicular to T_d that contains v_1 and u_1 . T' is perpendicular to T_d and, therefore, T' contains also the point v_2 and u_2 . The points v_1, u_2, v_2 and u_1 form a trapezoid on T' (the segment u_1v_1 is parallel to the segment u_2v_2). The segments u_1u_2 and v_1v_2 (the two diagonals of the trapezoid v_1, u_2, v_2 and u_1) intersect at point u . Since segment u_1u_2 (one diagonal of the trapezoid) contains u and P is convex u belongs to P . Similarly, u belongs to Q . The point u is contained in P and in Q contradicting the assumption that P and Q do not intersect. ♦

Corollary 4.3.1: Let P and Q be two simple non-intersecting convex polygonal faces in three dimensional space. To determine whether P should be translated before Q , or vice versa, it is sufficient to find one intersection point between the projections of P and Q on T_d .

4.3.4.3 THE ALGORITHM

The algorithm, for the uni-directional separability problem of convex polygons, is based on Algorithm 4.3.1. Line 2 of Algorithm 4.3.1 is omitted as a result of Property 4.3.1 (as there is no need to check whether *idom* is asymmetric). From Corollary 4.3.1 testing the projections of two polygons for an intersection determines the *idom* relation between the two polygons. The rest of the algorithm is similar to Algorithm 4.3.1.

Algorithm 4.3.2: Convex polygonal faces in three dimensional space

Input: A set $S = \{P_1, P_2, \dots, P_M\}$ of M non intersecting n -vertex convex polygonal faces in three dimensional space.

d - direction of translation

Output: A translation order of the set of polygonal faces.

begin

1. Project the polygonal faces to the plane T_d .
 2. For each pair of polygons P_i and P_j ($1 \leq i < j \leq M$) do
 - begin
 - If P_i and P_j intersect then
 - begin
 - Find an intersection point u between P_i and P_j ;
 - If a ray R (parallel to d , oriented as d and whose projection is u) intersects T_{P_i} first then
 - begin
 - output $P_j \text{ dom } P_i$;
 - end; {if}
 - else
 - begin
 - output $P_i \text{ dom } P_j$;
 - end; {else}
 - end; {if}
 - end; {for}
 3. Execute a DFS algorithm in the graph representation of the *dom* relation;
 4. If no cycles have occurred in 3 then
 - begin
 - perform a topological sort on the graph representation of the *idom* relation;
 - end;
 - else
 - begin
 - print(NO TRANSLATION ORDER EXISTS);
 - halt;
 - end; {else}
- end. {algorithm}

4.3.4.4 COMPLEXITY OF ALGORITHM 4.3.2

Line 1 - Projecting a polygon is done in $O(n)$ time and, thus, the total cost of projecting all the polygonal faces is $O(Mn)$.

Line 2 - In each iteration of Line 2 a pair of convex polygons, with n vertices each, is tested for an intersection. Each test can be done in $O(\log n)$ time. Determining whether $P_i \text{ dom } P_j$ or $P_j \text{ dom } P_i$ is done in $O(1)$. The number of times Line 2 is executed is $O(M^2)$, as all possible pairs are tested. Thus, the cost of Line 2 is $O(M^2 \log n)$.

Line 3 - The DFS algorithm is executed on the graph representation of the *dom* relations produced by Line 2. The execution time of a DFS algorithm is linear in the number of *dom* relations [Ta]. In the worst case, the number of *dom* relation pairs is $O(M^2)$ and, thus, the execution time of Line 3 ranges between $O(1)$ and $O(M^2)$.

Line 4 - A topological sort is performed on the *dom* relations produced by Line 2. The execution time of a topological sort is linear in the number of *dom* relation pairs [K]. The number of *dom* relation pairs is $O(M^2)$ and, thus, the execution time of Line 4 ranges between $O(M)$ and $O(M^2)$.

Theorem 4.3.3: The uni-directional separability problem for a set of M convex non-intersecting n -vertex polygonal faces can be solved in $O(M^2 \log n)$ time and $O(Mn)$ space.

An enhancement code, that is similar to the enhancement code for simple polygonal faces in Section 4.3.3.2, can be used for improving the running time of Algorithm 4.3.2. Due to the convexity of the polygonal faces, only the first approximation of the two used in Algorithm 4.3.1, is used in this algorithm.

Enhancement code:

begin

1. Find the vertices with the minimum and maximum x -coordinate of each P_i^d , $1 \leq i \leq M$.
2. Find all pairs of polygons P_i and P_j that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect, $1 \leq i, j \leq M$ and $i \neq j$. {Approximation 1}

end; {enhancement}

Line 1 - Finding the vertices with extremal x -coordinates is executed in $O(\log n)$ time and $O(1)$ space for each projected polygon. Thus, the total cost for M polygons is $O(M \log n)$ time and $O(M)$ space.

Line 2 - Finding all pairs of polygons P_i and P_j , that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect with each other, takes $O(K + M \log M)$ time using $O(K)$ space, where K is the number of pairs of intervals that intersect (output). In the worst case, all segments (intervals) will intersect with each other. Therefore, K ranges between 0 and M^2 .

All pairs of polygonal faces, whose intervals intersect, are used as input to Line 2 of Algorithm 4.3.2. Thus, the total cost of Algorithm 4.3.2 is $O(Mn + M \log M + K \log n)$ and $O(Mn)$ space, where K is the output size of Line 2 of the enhancement code.

Theorem 4.3.4: The running time of Algorithm 4.3.2, with the enhancement code, ranges between $O(Mn + M \log M)$ in the best case and $O(M^2 \log n + Mn)$ in the worst case.

Proof: The total cost of Algorithm 4.3.3 is $O(Mn + M \log M + K \log n)$ and $O(Mn)$ space, where K is the output size of Line 2 of the enhancement code. K ranges between 0 and $O(M^2)$ and, thus, the execution time of Algorithm 4.3.2 is in the best case $O(Mn + M \log M)$, when K is equal to 0, and $O(M^2 \log n + Mn)$ in the worst case, which arises when K is equal to $O(M^2)$. ♦

4.4 UNI-DIRECTIONAL SEPARABILITY FOR POLYHEDRA

In this section, a solution to the uni-directional separability problem for a set of polyhedra is given. In Section 4.4.1 the uni-directional separability problem for simple polyhedra is discussed. In Section 4.4.2 the uni-directional separability problem of convex polyhedra (polytopes) is discussed.

4.4.1 SIMPLE POLYHEDRA

4.4.1.1 PRELIMINARY OPERATIONS

A polyhedron is isomorphic to a planar graph. Thus, the number of vertices ($|V|$), edges ($|E|$) and faces ($|F|$) obeys Euler's formula $|V| - |E| + |F| = 2$. The degree of each vertex is at least 3 and, therefore, the following inequalities hold:

1. $ V \leq 2 E /3$	4. $ E \leq 3 V - 6$
2. $ E \leq 3 F - 6$	5. $ F \leq 2 E /3$
3. $ V \leq 2 F - 4$	6. $ F \leq 2 V - 4$

The solution to the uni-directional separability problem relies on the following observation and is based on the fact that a polyhedron is isomorphic to a planar graph.

Observation 4.4.1: Let A, B be two simple polyhedra in three dimensional space and let d be a direction of translation. If all faces of A can be translated, in direction d and over an arbitrarily large distance, without colliding with any of the faces of B then A will not collide with B when it is translated in direction d .

As a result of Observation 4.4.1 it is now possible to describe an algorithm for determining a translation order of a set of polyhedra. Instead of examining the polyhedra for separability, the polygonal faces are examined. For this purpose, Algorithm 4.3.1 is used with the following modifications: Whenever Algorithm 4.3.1 outputs an *idom*

relation between two faces of two distinct polyhedra, the new algorithm will output an *idom* relation between the two polyhedra (that the faces belong to). If the faces belong to the same polyhedron no relation is output. When all faces have been examined, the *idom* relation between the polyhedra is obtained. A DFS algorithm is then executed on the digraph representing the *idom* relation in order to detect whether there exists a cycle. If there are no cycles then the polyhedra can be translated. A topological sort on the *idom* relation is then performed to determine the translation order.

4.4.1.2 COMPLEXITY

Let K_i denote the number of polygonal faces that have i vertices. Each edge of the polyhedra is shared by at most two faces. The number of vertices of a face is equal to the number of edges of that face. Thus, the total number of vertices in all the faces is equal to, at most, twice the number of edges of all the polyhedra. Since the total number of edges of all the polyhedra is linear in the number of vertices $\sum_{i=1..n} K_i i = O(Mn)$. The total number of faces of all polyhedra is also linear in the number of vertices and, therefore using inequality 3, $\sum_{i=1..n} K_i \geq (Mn - 4)/2 > Mn/4$. Let T_{ij} denote the time complexity to determine how two faces A and B , with i and j vertices respectively, are related by the *dom* relation (Line 2 of Algorithm 4.3.1). When the polygonal faces are simple polygons $T_{ij} = i \log j$ and when the polygonal faces are convex polygons $T_{ij} = \log i + \log j$ (see Algorithm 4.3.1 and 4.3.2 respectively).

In the worst case each of the faces is examined with all other faces (this section, of Algorithm 4.3.1 and Algorithm 4.3.2, dominates the complexity). Thus, in the worst case the cost of comparing all the faces is:

$$\sum_{i=1..n} \sum_{j=1..n} K_i K_j T_{ij} \leq \sum_{i=1..n} \sum_{j=1..n} K_i K_j i j = \sum_{i=1..n} K_i i \sum_{j=1..n} K_j j = O(Mn) * O(Mn) = O(M^2 n^2).$$

On the other hand, an improvement, when this method is used, on the time complexity T_{ij} of comparing two faces, will not improve the overall complexity of the algorithm.

$$\sum_{i=1..n} \sum_{j=1..n} K_i K_j T_{ij} \geq \sum_{i=1..n} \sum_{j=1..n} K_i K_j = \sum_{i=1..n} K_i \sum_{j=1..n} K_j > Mn/4 * Mn/4 = M^2 n^2 / 16 = O(M^2 n^2).$$

Hence, in the worst case the time complexity of the algorithm is $O(M^2 n^2)$, where M is the number of polyhedra and n is the number of vertices in each polyhedra. In the best case (none of the intervals, in Line 2 of the enhancement code of Algorithm 4.3.1, intersect) the time complexity is $O(Mn \log Mn)$.

Theorem 4.4.1: The uni-directional separability problem can be solved in the worst case in $O(M^2 n^2)$ time using $O(Mn)$ space.

4.4.2 CONVEX POLYHEDRA (POLYTOPES)

4.4.2.1 PRELIMINARY OPERATIONS

In Section 4.3.4 it was shown that a reduction in the time complexity can be achieved when the polygonal faces in three dimensional space are known to be convex. A similar reduction can be achieved when the polyhedra are convex. The reduction in the time complexity based on the following property.

Property 4.4.1: Let P and Q be two polytopes in three dimensional space. The relation *idom* on P and Q is asymmetric.

Proof: The proof is identical to that of Property 4.3.1 and is, therefore, omitted. ♦

Corollary 4.4.2: Let P and Q be two simple non-intersecting polytopes in three dimensional space. To determine whether $P(Q)$ should be translated before $Q(P)$ it is sufficient to find one intersection point between the projections of P and Q on T_d .

Let P and Q be two polytopes, let d be a direction of translation and let L be a line parallel to the x -axis of T_d and that intersects P^d and Q^d .

Notation:

p_L^d and p_R^d - denote the left and right intersection points, respectively, of P^d with a line L parallel to the x -axis (see Figure 4.4.1).

q_L^d and q_R^d - denote the left and right intersection points of Q^d with a line parallel to the x -axis.

p_L and p_R - denote the points of P whose projections on T_d are p_L^d and p_R^d , respectively.

q_L and q_R - denote the points of Q whose projections on T_d are q_L^d and q_R^d , respectively.

Figure 4.4.1 The intersection points p_L^d and p_R^d .

4.4.2.2 THE ALGORITHM

The algorithm is based on Algorithm 4.3.2. The only difference between the two algorithms is the computation of the *idom* relation between two polytopes. After the polytopes have been projected on T_d the convex hulls of the projections are found. From each vertex on the convex hull of the projection a pointer is kept to the original vertex on the polytope. Let u be an intersection point between two images, for example P^d and Q^d . Let L be a line parallel to the x -axis and through u . If a ray R parallel to d , whose projection is u , intersects segment $p_L p_R$ before it intersects segment $q_L q_R$ then $Q \text{ idom } P$.

Algorithm 4.4.1: Convex polyhedra in three dimensional space

Input: A set $S = \{P_1, P_2, \dots, P_M\}$ of M non intersecting polytopes.

d - direction of translation

Output: A translation order of the polytopes.

begin

1. Project the polytopes on the plane T_d and find the convex hulls of each projection.

2. For each pair of polygons P_i and P_j ($1 \leq i < j \leq M$) do

begin

 If P_i and P_j intersect then

 begin

 Find an intersection point u between P_i and P_j ;

 Compare_Polytopes(P_i, P_j, u);

 end; {if}

end; {for}

3. Execute a DFS algorithm in the graph representation of the *idom* relation;

4. If no cycles have occurred in 3 then

begin

 perform a topological sort on the graph representation of the *idom* relation;

```

    end; {if}
  else
  begin
    print( NO TRANSLATION ORDER EXISTS);
    halt;
  end; {else}
end. {algorithm}

```

Procedure Compare_Polytopes(P, Q, u);

{This procedure determine the *idom* relation between two polytopes P and Q .

Input: P, Q - two polytopes with n vertices each.

u - an intersection point between P^d and Q^d (the projection of P and Q).}

begin

1. Find the two intersection points between P^d and a line L parallel to the x -axis through u (denoted as points p_L^d and p_R^d);
2. Find the two intersection points between Q^d and a line L parallel to the x -axis through u (denoted as points q_L^d and q_R^d);
3. Find points p_L and p_R on P (q_L and q_R on Q) such that the projection of the segment $p_L p_R$ (segment $q_L q_R$) on T_d is the segment $p_L^d p_R^d$ (segment $q_L^d q_R^d$).
4. If a ray R (parallel to d , oriented as d and whose projection is u) intersects segment $p_L p_R$ first then

begin

output Q *idom* P ;

end; {if}

else

begin

output P *idom* Q ;

end; {else}

end; {procedure}

4.4.2.3 COMPLEXITY OF ALGORITHM 4.4.1

Procedure Compare_Polytopes(P, Q, u);

Line 1 and Line 2 - Finding an intersection between a line and a convex polygon, in two dimensional space, can be done in $O(\log n)$ time using the algorithm by Chazelle and Dobkin [CD], where n is the number of vertices in the polygon.

Line 3 - Finding the points on the polytopes can be done in $O(1)$ because each vertex of the convex hull of the projection points to its original vertex on the polytope.

Line 4 - Determining which of two given segments is intersected first by a ray R can be done in $O(1)$.

The total cost of the procedure is $O(\log n)$, where n is the number of vertices in P and Q .

Algorithm 4.4.1

Line 1 - Projecting a polytope and finding its convex hull can be done in $O(n)$ time and, thus, finding all the convex hulls of the projected polytopes is $O(Mn)$.

Line 2 - In each iteration of Line 2, a pair of convex polygons is tested for an intersection. Each test can be done in $O(\log n)$ time where n is the number of the vertices in the pair of polygons using the algorithm by Chazelle and Dobkin [CD]. Determining whether $P_i \text{ idom } P_j$ or $P_j \text{ idom } P_i$ is done in $O(\log n)$ by calling the procedure Compare_Polytopes. The number of times Line 2 is executed is $O(M^2)$, as all possible pairs are tested. Thus, the cost of Line 2 is $O(M^2 \log n)$, where n is the number of vertices in a polygon.

Line 3 - The DFS algorithm is executed on the graph representation of the *dom* relations produced by Line 2. The execution time of a DFS algorithm is linear in the number of *idom* relations [Ta]. In the worst case the number of *dom* relations is $O(M^2)$ and, thus, the execution time of Line 3 ranges between $O(1)$ and $O(M^2)$.

Line 4 - A topological sort is done on the *idom* relations produced by Line 2. The execution time of a topological sort is linear in the number of *dom* relations [K]. In the

worst case the number of the *idom* relations is $O(M^2)$ and, thus, the execution time of Line 8 ranges between $O(M)$ and $O(M^2)$.

Theorem 4.4.2: The uni-directional separability problem for a set of M non-intersecting n -vertex polytopes can be solved in the worst case in $O(M^2 \log n)$ time and $O(Mn)$ space.

An enhancement code, that is similar to the enhancement code for convex polygonal faces in Section 4.3.4.4, can be used for improving the running time of Algorithm 4.4.1.

Enhancement code:

1. Find the vertices with the minimum and maximum x -coordinates of each P_j^d , $1 \leq j \leq M$.
 2. Find all pairs of polygons P_i^d and P_j^d that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect, $1 \leq i, j \leq M$ and $i \neq j$. {Approximation 1}
- end; {enhancement}

Line 1 - Finding the vertices with extremal x -coordinates is executed in $O(\log n)$ time and $O(1)$ space for each projected polygon. Thus, the total cost for M polygons is $O(M \log n)$ time and $O(M)$ space.

Line 2 - Finding all pairs of polygons P_i^d and P_j^d , that their intervals $[P_{imin}^d, P_{imax}^d]$ and $[P_{jmin}^d, P_{jmax}^d]$ intersect with each other, takes $O(K + M \log M)$ time using $O(K)$ space, where K is the number of pairs of intervals that intersect (output). In the worst case, all segments (intervals) will intersect with each other. Therefore, K ranges between 0 and M^2 .

All pairs of polytopes, whose intervals intersect, are used as input to Line 2 of Algorithm 4.4.1. Thus, the total cost of Algorithm 4.4.1 is $O(Mn + M \log M + K \log n)$ and $O(Mn)$ space, where K is the output size of Line 2 of the enhancement code.

Theorem 4.4.3: The running time of Algorithm 4.4.1, with the enhancement code, ranges between $O(Mn + M \log M)$, in the best case, and $O(M^2 \log n + Mn)$ in the worst case.

Proof: The total cost of Algorithm 4.4.1 is $O(Mn + M \log M + K \log n)$ and $O(Mn)$ space, where K is the output size of Line 2 of the enhancement code. K ranges between 0 and $O(M^2)$ and thus the execution time of Algorithm 4.4.1 is in the best case $O(Mn + M \log M)$, when K is equal to 0, and $O(M^2 \log n + Mn)$ in the worst case, which arises when K is equal to $O(M^2)$. ♦

4.5 ALL-DIRECTIONS SEPARABILITY PROBLEM

4.5.1 INTRODUCTION

In this section the all-directions separability problem in three dimensional space for simple polyhedra is discussed. In two dimensional space the problem has been well studied. Sack and Toussaint [ST], and Toussaint [T1] have given an $O(n \log n)$ algorithm for solving the all-directions separability problem of two simple n -gons. Recently, Sack [S], and Bhattacharya and Toussaint [BT] have developed an $O(n \log \log n)$ [$O(n + \text{time of triangulation})$] algorithm for this problem. When the convex hull of the two n -gons do not intersect the all-directions separability problem can be solved in $O(n + \log n)$ time [ST]. In three dimensional space the problem is more complex. Toussaint [T] has given an $O(n^2 \log n)$ algorithm for the all-directions separability problem of two convex polyhedra with n vertices each. Sack and Nurmi [SN] have given an optimal $O(n)$ solution to the all-directions separability problem for two convex polyhedra with n vertices each. Their method is based on the merge step of the three dimensional convex polyhedra algorithm by Preparata and Hoey [PH] (see Section 2.3).

In Section 4.5.2, an $O(m^2n^2 \log mn)$ algorithm all-directions separability problem of two simple polyhedra with n and m vertices is given. Section 4.5.3 gives an $\Omega(m^2n^2)$ lower bound to the all-directions separability problem of two simple polyhedra with n and m vertices. Section 4.5.4 shows that the $\Omega(m^2n^2)$ lower bound still holds when two simple polygonal faces are considered, and an $O(m^2n^2 \log mn)$ algorithm will be given.

4.5.2 ALL-DIRECTIONS SEPARABILITY PROBLEM OF TWO SIMPLE POLYHEDRA

4.5.2.1 THE ALGORITHM

In the previous section (Section 4.4) the faces of the polyhedra have been examined instead of the polyhedra as a whole. The same technique is used in this section. Instead of examining the polyhedra with respect to each other, each face of one polyhedron is examined with respect to the faces of the second polyhedron. Without loss of generality, assume that each face of the polyhedra is convex. In the sequel, the case of simple polygonal faces will be presented. Using the algorithm by Sack and Nurmi [SN] one can determine in $O(n)$ time all directions of separability in which a convex n -vertex polygonal face can be translated with respect to another convex n -vertex polygonal face. When all faces have been compared the union of all direction-cones is found.

Algorithm 4.5.1: All-directions separability of simple polyhedra

Input: Two non-intersecting simple polyhedra P and Q with n and m vertices respectively.

The polygonal faces of P and Q are convex.

Output: All the directions of separability of P with respect to Q .

begin

1. Decompose polyhedron P to a set of faces $(f_{P1}, f_{P2}, \dots, f_{Pr})$;
2. Decompose polyhedron Q to a set of faces $(f_{Q1}, f_{Q2}, \dots, f_{Qs})$;
3. For $i := 1$ to r do
begin
4. For $j := 1$ to s do
begin
5. Find all directions of separability of f_{Pi} with respect to f_{Qj} ;
end; {for}
- end; {for}
6. Find the union of all the $O(nm)$ cones produced in Line 5. (Use the sphere technique by Sack and Nurmi [SN].
- end;{algorithm}

4.5.2.2 COMPLEXITY

P and Q are isomorphic to a planar graph and, therefore, decomposing P and Q into faces requires $O(n+m)$ time. In Lines 3, 4 and 5 the construction of the cones is done. Let K_i denote the number of polygonal faces that have i vertices in polygon P and let K'_j denote the number of polygonal faces that have j vertices in polygon Q . Analogous to Section 4.4.1.1 $\sum_{i=1..n} K_i i = O(n)$ and $\sum_{j=1..m} K'_j j = O(m)$ and the number of faces of P and Q is $\sum_{i=1..n} K_i = O(n)$ and $\sum_{j=1..m} K'_j = O(m)$, respectively. Each face of polyhedron P with i vertices is examined against each face of polyhedron Q with j vertices and a new direction cone is constricted at cost $i+j$ [SN]. The total cost of Lines 3, 4 and 5 follows from Equation 4.5.1.

Equation 4.5.1:

$$\begin{aligned} \sum_{i=1..n} \sum_{j=1..m} K_i K'_j (i+j) &= \sum_{i=1..n} \sum_{j=1..m} K_i K'_j i + \sum_{i=1..n} \sum_{j=1..m} K_i K'_j j = \\ \sum_{j=1..m} K'_j \sum_{i=1..n} K_i i + \sum_{i=1..n} K_i \sum_{j=1..m} K'_j j &\leq O(mn) + O(mn) = O(mn). \end{aligned}$$

The direction cone of an n -vertex polygonal face with respect to an m -vertex polygonal face may have at most $n+m$ vertices [SN]. Thus, the total number of vertices in all cones is $O(mn)$ which follows from Equation 4.5.1. The union of the $O(mn)$ cones with $O(mn)$

vertices altogether can be done in $O((mn + k) \log mn)$ time and $O(mn + k)$ space, where k is the number of intersections of the walls of the cones ($0 \leq k \leq m^2 n^2$). The union of the cones can be computed by using the sphere sweep by Sack and Nurmi [SN] (see Section 2.3). Thus, in the worst case the cost of Algorithm 4.5.1 is $O(m^2 n^2 \log mn)$ time and $O(m^2 n^2)$ space.

When the faces of the polyhedra are simple polygonal faces a triangulation is done on each polygonal face. The cost of triangulating the polygonal faces of a polyhedron with n vertices is $\sum_{i=1..n} K_i i \log i \leq \log n \sum_{i=1..n} K_i i \leq O(n \log n)$. Therefore, in $O(n \log n + m \log m)$ time a triangulation of the P and Q is obtained. Each of the triangles is considered to be a face. The number of triangles is linear in the number of vertices $\sum_{i=1..n} K_i i \leq O(n)$. Hence, using the triangles as faces does not change the complexity of the algorithm of $O((mn + k) \log mn)$ time and $O(mn + k)$ space.

Theorem 4.5.1: The all-directions separability problem for two simple polyhedra, with m and n vertices, can be solved in the worst case in $O(m^2 n^2 \log mn)$ time.

4.5.3 LOWER BOUND

In this section an $\Omega(m^2 n^2)$ lower bound to the all-directions separability problem is presented. Let P and Q be two simple polyhedra. P consists of two combs arranged perpendicularly to each other and having m teeth each. The teeth point into the right angle (see Figure 4.5.1). Q consists of two combs, perpendicular to each other and with a single tooth on each comb, such that the teeth point away from the right angle (see Figure 4.5.2).

There are $\Omega(m^2)$ regions in which Q can be translated with respect to P such that the teeth of Q pass between the teeth of P . Each such region, of the $\Omega(m^2)$ regions, is rectangles. The construction of such a rectangular region is given in Example 4.5.1.

Figure 4.5.1 Polyhedron P Figure 4.5.2 Polyhedron Q

Example 4.5.1: Q is translated such that tooth q_1 is between teeth p_1 and p_2 of P and tooth q_2 of Q is between teeth p_{n+1} and p_{n+2} of P (see Figure 4.5.3). The 'freedom' that Q

has when it is translated through teeth p_1, p_2, p_{n+1} and p_{n+2} is bounded by the distance between p_1 and p_2 with respect to q_1 and by the distance between p_{n+1} and p_{n+2} with respect to q_2 (see Figure 4.5.3). The shape of this region is rectangular. The gray rectangle in Figure 4.5.3 is the allowed translation region of point z with respect to P . ♦

Figure 4.5.3 The translation region of Q with respect to P when Q is translated through teeth p_1, p_2, p_{n+1} and p_{n+2} .

A rectangular region is constructed every time Q is translated in a direction such that the tooth q_1 is between p_i and p_{i+1} for $i = 1..n-1$ and tooth q_2 is between p_j and p_{j+1} for $j = n+1..2n-1$. The number of combinations to choose i and j is n^2 . Thus, the rectangular regions in which Q can be translated with respect to P consists of n^2 rectangles (see Figure 4.5.4).

Figure 4.5.4 all the n^2 translation regions of Q with respect to P (from example 4.5.1)

In order to achieve an $\Omega(m^2 n^2)$ lower bound tooth, q_2 is replaced by m teeth $q_{m+1}, q_{m+2}, \dots, q_{2m}$ and q_1 is replaced by m teeth $q_1, q_2, \dots, q_m$ (see Figure 4.5.5). If the teeth of P and Q are correctly spaced then each rectangle in Figure 4.5.4 will be divided into m^2 smaller rectangles (see Figure 4.5.6). Thus, the total number of such rectangular regions is $m^2 n^2$.

Figure 4.5.5 Teeth q_1 and q_2 are replaced each by m teeth

Figure 4.5.6 All m^2n^2 translation regions of Q with respect to P

Theorem 4.5.2: In the worst case $\Omega(m^2n^2)$ time is needed to solve the all-directions separability problem for two simple polyhedra with m and n vertices.

Recently, Polack, Sharir and Sifrony [PSS] have used a similar example to show that the configuration space of two simple polygons A and B consists of $\Omega(m^2n^2)$ connected components, where m and n are the number of vertices of A and B respectively.

4.5.4 SIMPLE POLYGONAL FACES IN THREE DIMENSIONAL SPACE

4.5.4.1 INTRODUCTION

In this section the all-directions separability problem of simple polygonal faces in three dimensional space is discussed. Even though simple polygonal faces are less complex objects than simple polyhedra, in three dimensional space, the solution to the all-directions separability of simple polygonal faces seems as hard as for simple polyhedra.

In Section 4.5.4.2 an algorithm for the all directions separability problem of simple polygonal faces in three dimensional space is given. In Section 4.5.4.3 the complexity of the algorithm is analysed. The lower bound of $\Omega(m^2n^2)$ for the all-directions separability problem for simple polygonal faces is identical to the lower bound for simple polyhedra and, therefore, it is not presented.

4.5.4.2 THE ALGORITHM

The algorithm is very similar to Algorithm 4.5.1. In Algorithm 4.5.1 each face of one polyhedron is examined with respect to the faces of the second polyhedron. A similar technique is used for polygonal faces. Instead of decomposing the polyhedra into polygonal faces the polygonal faces are partitioned into triangles. Each triangle of one polygonal face is examined with all triangles of the second polygonal face. Using the algorithm by Sack and Nurmi [SN] one can determine in $O(I)$ time, all the directions of

separability in which a polygonal face with 3 vertices can be translated with respect to another polygonal face with 3 vertices. When all the triangles have been compared the union of all the direction-cones is then found.

Algorithm 4.5.2: All-directions separability of simple polygonal face

Input: Two non-intersecting simple polygonal face P and Q with n and m vertices respectively.

Output: All the directions of separability of P with respect to Q .

begin

1. Triangulate P to a set of triangles $(f_{P1}, f_{P2}, \dots, f_{Pr}); \{r = n - 2\}$
2. Triangulate Q to a set of triangles $(f_{Q1}, f_{Q2}, \dots, f_{Qs}); \{s = m - 2\}$
3. For $i := 1$ to r do
 - begin
 - 4. For $j := 1$ to s do
 - begin
 - 5. Find all directions of separability of f_{Pi} with respect to f_{Qj} ;
 - end; {for}
 - end; {for}
 - 6. Find the union of all the $O(nm)$ cones produced in Line 5; {Use the sphere technique by Sack and Nurmi [SN]}
- end; {algorithm}

4.5.4.3 COMPLEXITY OF THE ALGORITHM

P and Q are isomorphic to a planar graph and, therefore, decomposing P and Q into triangles requires $O(n+m)$ time. In Lines 3, 4 and 5 the construction of the cones is done. There are $n - 2$ triangles of polygon P and $m - 3$ triangles of polygon Q . Each triangle of P is compared with each triangle of Q and a new cone is constructed. Thus, the total number of cones is $O(nm)$ with at most 6 vertices each. The total cost of Lines 3, 4 and 5 is $O(nm)$ time.

The total number of vertices in all the cones is $O(mn)$. The union of the $O(mn)$ cones with $O(mn)$ vertices, altogether, can be done in $O((mn + k) \log mn)$ time and $O(mn + k)$ space, where k is the number of intersections of the sides of the cones ($0 \leq k \leq m^2n^2$). The union of the cones can be computed by using the sphere sweep by Sack and Nurmi [SN] (see Section 2.3). Thus, in the worst case the cost of Algorithm 4.5.2 is $O(m^2n^2 \log mn)$ time and $O(m^2n^2)$ space.

Theorem 4.5.3: The all-directions separability problem for two simple polygonal faces, with m and n vertices, can be solved in the worst case in $O(m^2n^2 \log mn)$ time.

Proof: The proof is similar to that of Theorem 4.5.1 and is, therefore, omitted. ♦

The construction of a lower bound for the all-directions separability problem for two simple polygonal faces is identical to the construction of the lower bound of two simple polyhedra.

Theorem 4.5.4: In the worst case $\Omega(m^2n^2)$ time is needed to solve the all-directions separability problem for two simple polygonal faces with m and n vertices.

Proof: The proof is similar to that of Theorem 4.5.2 and is, therefore, omitted. ♦

CHAPTER 5

APPLICATIONS

In this section several applications are described in which directional separability problem arises and, thus, directional separability techniques might be used. All the applications, given in this chapter, are ideas only and further research is needed before conclusions can be made.

5.1 Ray tracing - This technique is used to generate computer images. Rays from the eye of the observer are traced (through the window-monitor) until they hit a source of light or are absorbed by an object (black hole). When the rays hit an object in the environment, a ray may bounce off the object (if the object is reflective), go through the object (if the object is transparent), or be absorbed by the object. After the ray has reached a source of light, it is traced back to the eye of the observer and the colour of the ray is calculated. Generating a picture by the computer is done as follows: a fixed point P is determined as a view point. For each pixel on the screen the ray emanating from P through that pixel is traced until the right colour is calculated for the pixel.

Ray tracing algorithms require a great deal of computer time. Simple pictures with two sources of light require up to 15 hours of computer time (Vax 11/750) [WHG] (the complexity analysis is not given).

There are two methods, both related to computational geometry, for speeding up the ray tracing algorithms:

(i) speeding up the intersection process itself (the speed of testing an intersection between a ray and an object);

(ii) reducing the number of ray-object intersection calculations by pre-processing the objects in the environment in such a way that, when a ray tracing algorithm is used, the information on which object(s) the ray intersects, is answered faster.

When a ray d bounces off an object A , a search for the next object B that ray d hits begins. One can look at this problem in a different way: find the first object B such that object A , when it is translated in the direction of the ray d , will intersect B . When B is found then an intersection between the ray and B is examined. Uni-directional separability, multi-directional separability and all-directions separability might be used in a preprocessing stage. During the execution of the algorithm queries of the following type would be posed for each ray.

Query 1: given a ray and an object, output a list of all objects that the ray might intersect.

Query 2: given a ray, an object and the origin of the ray on the object, determine which object the ray intersects first.

These queries will probably save time during the actual execution of a ray tracing algorithm. Only relevant candidate objects would be examined in order to determine the next object hit by the ray.

Another possible advantage of directional separability over other methods is the ability to update the entire picture without repeating the preprocessing stage. Hence, when a new object is inserted or deleted from the picture, the existing information (obtained from a previous preprocessing step) could be updated but need not be recomputed.

Figure 5.1 The principle of ray tracing

5.2 Intelligent Dishwasher - Consider a dining room table with a built-in dishwasher at one end (see Figure 5.2). The built-in dishwasher has a prismatic (telescopic) arm, at the height of the table, that can move in and out and sideways (see Figure 5.2). When a meal is finished, the intelligent built-in dishwasher starts to clear the table. In order to clear the table without breaking any of the dishes, the robot dishwasher must know which of the dishes should be cleared first. The information about the set-up of the table, namely the location of the dishes on the table, can be obtained from a camera placed either on the dishwasher or on the ceiling. When all the information is gathered it could be used as input to the uni-directional separability algorithm to determine the order in which the objects on the table should be cleared. Figure 5.2 illustrates a mistake done by the BDN dishwasher that did not use the uni-directional algorithm.

Figure 5.2 The intelligent dishwasher

Figure 5.3 The intelligent dishwasher BDN

5.3 Intelligent Assembly Line - More and more robots are being used in assembly lines. Many robots are primitive and simply repeat the task in which they are initially programmed. When the product is changed the assembly robots need to be reprogrammed. The ability to examine the parts to be assembled and to determine the order in which the parts should be assembled will enhance the use of robots in assembly lines. Uni-directional separability, multi-directional separability and all-directions separability techniques could be used when a product is assembled from a given direction or from several directions. For example, uni-directional separability can be used when a machine is to be assembled from above.

Designing new types of assembly lines represents another potential application for directional separability. Consider, for example, an assembly line for printed circuit boards (PCBs). Today most PCBs are assembled by multi-links robots (see Figure 5.4). This kind of assembly requires a great deal of computation. Instead a new kind of assembly line is suggested: a prismatic (telescopic) arm and a plate that could move left and right with respect to the arm (see Figure 5.5). The PCB is placed on the table and the assembly is done by moving the arm in and out and the table left and right. Uni-directional separability might be used in order to determine the order in which the electronic components are put on the PCB.

Figure 5.4 Multi-link assembly robot (PUMA arm)

Figure 5.5 Prismatic arm assembly robot.

5.4 Rescue Robot - There are many types of accidents which pose risks to rescue workers. Examples include: buildings that collapse, oil fields fires, mine explosions and collapses, nuclear reactor melt-downs et cetera. For these types of accidents, the rescuers often need to remove elements from the scene of the accident, for example, parts of the collapsed building or rocks and debris from a mine disaster. Using uni-directional separability, multi-directional separability and all-directions separability techniques one could obtain the order in which the elements should be removed and determine whether there is a need to break one of the elements into smaller pieces. This would be very useful when a rescue team must operate in a dangerous environment, i.e nuclear melt-downs.

All directional separability could be used when a straight emergency road had to be constructed or a special pipe (e.g. water pipe) had to be brought to a disaster area. All-direction separability could provide the information of whether there exists a direction that such a pipe or road could be built without hitting any obstacles (rocks, trees et cetera). If such a direction does not exist then a direction in which the minimum number of obstacles that must be eliminated, could be given.

5.5 Automatic Loader (Intelligent Crane) - Uni-directional separability could be used to solve the problem of how to unload a boat or a truck using an intelligent crane. The crane would obtain the loading plan of the boat by using sophisticated cameras and would be able to determine which part of the load should be removed next.

5.6 Computer Graphics - An important part of computer graphics is displaying the images of objects on a screen. For a given three dimensional picture, consisting of one or more objects and a viewing specification which defines the type of projection, the projection plane, et cetera, the problem is to determine which edges and surfaces of the object or objects are visible from the viewpoint (perspective or parallel view) and display

those edges and surfaces (see Figure 5.6). While this idea is simple, the implementation requires a large amount of computing time.

Two brute force algorithms for this problem are: (i) for each pixel on the screen find which objects should be contained in the pixel by examining all the objects and finding the top-most object that covers the pixel; and (ii) compare all pairs of objects and determine which part of the object is visible, and then display the parts of each object, that were not eliminated, on the screen. Numerous algorithms have been developed for this purpose (see Foley and Van Dam [FV] and Magnenat-Thalmann and Thalmann [MT] for a more detailed survey of this problem). These algorithms all solve the problem from one given viewpoint. When the viewpoint changes the image has to be recomputed.

All-directions separability and multi-directional separability could be used in a preprocessing stage to reduce the number of computations when the viewpoint is changed. Later, when a viewpoint is given, the objects would be displayed on the screen according to their translation order with respect to the direction of translation. Displaying the objects according to the translation order produces the correct picture. Each object, when displayed, may cover any of the objects that were already displayed. These parts are not visible by the observer and, therefore, should not be on the screen. When the viewpoint is changed the objects are redisplayed according to a new translation order and the picture is obtained without recomputing the visible parts of the objects.

Figure 5.6 The role of hidden lines and surfaces

5.7 Designing new product - Uni-directional separability and multi-directional separability could be used to facilitate the design of new machines. Almost every machine has parts that need to be tuned or replaced after a certain amount of time, either because the parts are destroyed during usage (e.g. electrical fuses) or because the parts deteriorate over time and have to be replaced for better performance (e.g. air and oil filters in a car). Using multi-directional separability or uni-directional separability or all-directions separability it might be possible to determine whether there is easy access to the part needing replacing. Another example, in which uni-directional separability technique might be useful, is designing a new electronics device. In many cases the designer would like to know whether sensitive elements with short life expectancy can be accessed easily (by a single move) and replaced without taking several pieces apart.

5.8 Medicine - All-directions separability techniques could be used to facilitate many modern medical techniques including, for example, eye operations and brain scans. All-direction separability could be applied in the following ways: (i) Eye operation - when a tumour is to be removed, the surgeon needs information on how to direct the laser beam so that no other parts of the eye will be damaged; (ii) Ultra sound or brain scan - when a scan is done on a patient, in order to obtain information on a specific organ, for example, a tumour, one would like to know whether there exists a direction in which the tumour can be seen as a whole with no other organs (obstacles) interfering.

5.9 Automatic Vacuum Cleaner - Cleaning the home and vacuuming are very routine jobs in almost every household. An automatic vacuum cleaner (AVC) is proposed as a device to do this work. The AVC would be able to move furniture, vacuum the carpets and then return the furniture to its original place. Uni-directional separability, multi-directional separability and all-directions separability could provide the AVC with the ability

to determine which piece of furniture should be moved first and whether there are areas that cannot be cleaned.

CHAPTER 6

CONCLUSIONS

This thesis focused on three problems of directional separability in two and three dimensional space.

- (i) The uni-directional separability problem.
- (ii) The multi-directional separability problem.
- (iii) The all-directions separability problem.

In two dimensional space an optimal algorithm was given for the uni-directional separability problem for a set S of M simple, non-intersecting, n -vertex polygons. $O(Mn + M \log M)$ algorithms were presented, as well as, matching lower bounds for detecting whether S can be separated and for determining a translation order for S .

In three dimensional space the following problems were discussed: (i) the uni-directional separability problem for line segments, polygonal faces and polyhedra; (ii) the multi-directional separability problem for line segments; and (iii) the all-directions separability problem for two simple polyhedra and for two simple polygonal faces.

A new approach for solving the uni-directional separability problem for polygonal faces and simple polyhedra was presented. This method improved on the time complexity of the existing algorithms. Furthermore, when the polygonal faces or the polyhedra are convex a further improvement on the time complexity was obtained. In addition, it was shown that as few as six line segments can be arranged so that for any given direction no solution to the uni-directional separability problem exists.

The algorithms for solving the all-directions separability problem, presented for simple polygonal faces and for simple polyhedra, were based on the algorithm by Sack and Nurm [SN]. It was shown that, in the worst case, $\Omega(m^2 n^2)$ time is required for solving this

problem compared to the $O(n)$ upper bound achieved for convex polytopes. Hence, the $O(m^2n^2 \log mn)$ algorithm presented here is close to optimal. The results obtained in this thesis are summarized in Table 6.1.

In addition, two lower bounds were established:

- (i) $\Omega(Mn)$ for detecting whether there exists a pair P_i, P_j among M simple, n -vertex polygons, monotone with respect to the same direction, that intersect;
- (ii) $\Omega(Mn)$ for detecting whether there exists a pair P_i, P_j among M simple, non-intersecting, n -vertex polygons whose visibility hulls, with respect to the same direction, intersect.

Several open problems, related to this thesis, arose while working these topics.

- (i) It was shown that the multi-directional separability problem for M line segments in three dimensional space can be solved in $O(M \log M)$ time. An open problem is whether this is optimal?
- (ii) Theorem 4.4.1 states an $O(M^2n^2)$ worst case algorithm for the uni-directional separability problem of M simple non-intersecting n -vertex polyhedra in three dimensional space. Can a lower bound better than $\Omega(Mn \log Mn)$, or a better algorithm be found?
- (iii) In Section 3.6 an $\Omega(M \log M + Mn)$ lower bound for the uni-directional separability problem for a set S of M simple, non-intersecting, n -vertex polygons in two dimensional space, which are stored in the main memory, was given. In Theorem 3.6.3 an $\Omega(M \log M)$ for determining a translation order for S was given. Can a better algorithm than Algorithm 3.4.2 be found or a lower bound of $\Omega(M \log M + Mn)$ be established for determining a translation order when it is known that S is separable?
- (iv) Can a one pass algorithm, or an $O(M \log M + Mn)$ algorithm be found for determining the translation order of a set S of M simple, non-intersecting, n -vertex polygonal faces or polyhedra, in three dimensional space, when it is known that the polygonal faces or polyhedra are separable?

Problem	Objects	Dimension	Time complexity	Best known time complexity
Uni-directional separability	Simple polygons	2-D	$O(Mn + M \log M)$	$O(Mn \log Mn)$
Multi-directional separability	Line segments	3-D	$O(M \log M)$	—————
Uni-directional separability	Simple polygonal faces	3-D	$O(Mn + M \log M + K \log n + U \log n)$ $0 \leq K \leq M^2, 0 \leq U \leq K$	$O(K \log Mn)$ $Mn \leq K \leq M^2$ if
Uni-directional separability	Convex polygonal faces	3-D	$O(Mn + M \log M + K \log n)$ $0 \leq K \leq M^2$	$O(K \log Mn)$ $Mn \leq K \leq M^2$ if
Uni-directional separability	Simple and convex polyhedra	3-D	$O(M^2 \log^2 M)$ worst case	$O(M^2 \log^2 Mn)$ worst case
All-directions separability	Simple polyhedra	3-D	$O(M^2 \log^2 Mn)$	—————
All-directions separability	Simple polygonal face	3-D	$O(M^2 n^2 \log Mn)$	—————

Table 6.1 Summary of results, where M is the number of objects, n or m is the number of vertices in an object.

APPENDIX A

PROCEDURES OF ALGORITHM 4.3.1

Procedure Separability_On_The_Same_Plane ($P, Q, d, \text{Separable?}$);

{This procedure checks whether two polygonal faces on the same plane are movably separable (see Lemma 4.3.4).

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

begin

1. If T_{P_i} is perpendicular to T_d then

begin

2. Find the visibility hulls of P and Q with respect to d ;

3. If the visibility hulls intersect then

begin

$\text{Separable?} := \text{false};$

end; {if}

else

begin

4. If P^d and Q^d intersect then

begin

5. If a ray R (parallel to d and oriented as d) intersects T_P first then

begin

Output Q idom P ;

end; {if}

else

begin

Output P idom Q ;

```

        end; {else}
    end; {if}
    end; {else}
    end; {if}
end; {procedure}

```

Procedure Separability_Of_Parallel_Planes (P, Q, d);

{This procedure checks whether $P \text{ dom } Q$ or $Q \text{ dom } P$ or P and Q are incomparable in dom . The planes T_P and T_Q are parallel to each other (see Lemma 4.3.5).

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

begin

1. If P^d and Q^d intersect then

begin

2. If a ray R (parallel to d and oriented as d) intersects T_P first then

begin

Output $Q \text{ idom } P$;

end; {if}

else

begin

Output $P \text{ idom } Q$;

end; {else}

end; {if}

end; {procedure}

Procedure Separability_Of_Perpendicular_Planes ($P, Q, d, \text{Separable?}$);

{This procedure checks whether P and Q are movably separable when T_P and T_Q are perpendicular to T_d (see Lemma 4.3.3).

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

begin

1. If P^d and Q^d intersect then

begin

2. Find the points u_1 and u_2 and v_1 and v_2 of P and Q respectively such that the intersection of P (Q) with L_{PQ} is contained in the segment u_1u_2 (segment v_1v_2);

3. If segment u_1u_2 intersects segment v_1v_2 then

begin

Separable? := false;

end; {if}

end; {if}

else

begin

4. If a ray R (parallel to d and oriented as d) intersects u_1 before v_1 then

begin

output Q idom P ;

end; {if}

else

begin

output P idom Q ;

end; {else}

end; {else}

end; {if}

end; {procedure}

Procedure Separability_Of_Polygonal_Faces_2 (P , Q , d , Separable?);

{This procedure checks whether two polygonal faces (P and Q) on different planes are movably separable (see Lemma 4.3.2).}

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

begin

1. Cut_Polygon (P^d , P_{above} , P_{below} , L_{PQ}^d);

2. Cut_Polygon (Q^d , Q_{above} , Q_{below} , L_{PQ});

3. If P_{above} intersects Q_{above} and P_{below} intersects Q_{below} then

begin

Separable? := false;

end; {if}

else begin

4. If P_{above} and Q_{above} intersect then

begin

5. Find an intersection point u between P_{above} and Q_{above} ;

6. If a ray R (parallel to d , oriented as d and whose projection is u) intersects T_P

first then

begin

output Q idom P ;

end; {if}

else

begin

output P idom Q ;

end; {else}

else

begin

7. If P_{below} and Q_{below} intersect then

begin

8. Find an intersection point u between P_{below} and Q_{below} ;

9. If a ray R (parallel to d , oriented as d and whose projection is u) intersects

T_P first then

begin

output Q idom P ;

end; {if}

else

begin

```

        output  $P \text{ idom } Q$ ;
    end; {else}
end; {if}
end; {else}
end; {procedure}

```

Procedure Separability_Of_Polygonal_Faces_1 ($P, Q, d, \text{Separable?}$);

{This procedure determines which of the two polygonal faces (P and Q) in different planes should be moved first (see Lemma 4.3.1).

Input: P - polygonal face

Q - polygonal face

d - direction of translation

Output: Separable? - boolean variable, if true then polygons are separable}

begin

1. If P^d and Q^d intersect then

begin

2. Find an intersection point u between P^d and Q^d ;

3. If a ray R (parallel to d , oriented as d and whose projection is u) intersects T_P first then

begin

output $Q \text{ dom } P$;

end; {if}

else

begin

output $P \text{ dom } Q$;

end; {else}

end; {if}

end; {procedure}

Procedure Cut_Polygon ($P, P_{above}, P_{below}, Line$);

{ This procedure rotates and translates the polygon P until the line $Line$ is on the x -axis. The procedure then creates two new polygons P_{above} and P_{below} by partitioning the polygon P at the x -axis. P_{above} contains that portion of P located above or on the x -axis and P_{below} contains that portion of P located below or on the x -axis (see Figure 4.3.3)

Input : P - polygon

Line - line;

Output: P_{above} - polygon
 P_{below} - polygon}

COMPLEXITY OF THE PROCEDURES USED BY ALGORITHM 4.3.1

Procedure Separability_On_The_Same_Plane

Line 1 - Checking whether T_P is perpendicular to T_d is performed in $O(1)$.

Line 2 - Finding the visibility hulls of P and Q is done in $O(n)$ time and $O(n)$ space using any of the algorithms by El Gindy and Avis [EA], Dean and Sack [DS], and Lee [L], where n is the number of vertices of P and Q .

Line 3 - Checking whether the visibility hulls intersect is done on $O(n)$ time using the algorithm by Shamos and Hoey [SH].

Line 4 - P^d and Q^d are lines and therefore, examining P^d and Q^d for an intersection is executed in $O(1)$.

Line 5 - Checking whether a ray R , parallel to d and oriented as d , intersects T_P first is executed in $O(1)$.

The total cost of the procedure is $O(n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Parallel_Planes

Line 1 - Testing whether two simple polygons (P^d and Q^d) intersect is executed in $O(n \log n)$ time and $O(n)$ space using the algorithm by Shamos and Hoey [SH].

Line 2 - Checking whether a ray R , parallel to d and oriented as d , intersects T_p first is executed in $O(1)$.

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Perpendicular_Planes

Line 1 - P^d and Q^d are lines and therefore Line 1 is executed in $O(1)$.

Line 2 - Finding the points u_1 and u_2 (v_1 and v_2) of P (Q) such that the segment u_1u_2 (segment v_1v_2) contains all intersection points of P (Q) with L_{PQ} is executed in $O(n)$ and $O(1)$ space.

Line 2 - Finding the points u_1 and u_2 (v_1 and v_2) of P (Q) such that all other intersection of P (Q) with L_{PQ} are contained in the segment u_1u_2 (segment v_1v_2) can be done in $O(n)$ time and $O(1)$ space.

Line 3 - Checking whether two line segment intersect is done in $O(1)$.

Line 4 - Checking whether a ray R , parallel to d , oriented as d and through u_1 and v_1 , intersects first u_1 is executed in $O(1)$.

The total cost of the procedure is $O(n)$ time and $O(1)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Polygonal_Faces_2

Line 1 - The cost of procedure Cut_Polygon is $O(n)$ time and $O(n)$ space.

Line 2 - The cost of procedure Cut_Polygon is $O(n)$ time and $O(n)$ space.

Line 3 - Testing whether two simple polygons (P^d and Q^d) intersect is executed in $O(n \log n)$ time and $O(n)$ space using the algorithm by Shamos and Hoey [SH].

Line 4 - P_{above} and Q_{above} were already examined in Line 3 and thus the cost of this predicate is $O(1)$.

Line 5 - An intersection point between P_{above} and Q_{above} was already found in Line 3 and therefore the cost is $O(1)$.

Line 6 - Checking whether a ray R parallel to d and oriented as d , intersects T_p first is executed in $O(1)$.

Line 7 - P_{below} and Q_{below} were already examined in Line 3 and thus the cost of this predicate is $O(1)$.

Line 8 - An intersection point between P_{below} and Q_{below} was already found in Line 3 and therefore the cost is $O(1)$.

Line 9 - Checking whether a ray R , parallel to d and oriented as d , intersects T_p first is executed in $O(1)$.

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Separability_Of_Polygonal_Faces_1

Line 1 - Testing whether two simple polygons (P^d and Q^d) intersect is executed in $O(n \log n)$ time and $O(n)$ space using the algorithm by Shamos and Hoey [SH].

Line 2 - P^d and Q^d were already examined in Line 3 and, thus, the cost is $O(1)$.

Line 3 - Checking whether a ray R , parallel to d , oriented as d and whose projection is the intersection point u , intersects first T_p is executed in $O(1)$.

The total cost of the procedure is $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

Procedure Cut_Polygon

The cost of this procedure is $O(n)$ time and $O(n)$ space, where n is the number of vertices in P and Q .

REFERENCES

- [AHU] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.
- [AHU1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [AK] M. J. Atallah and S. R. Kosaraju, "An Adversary-Based Lower Bound for Sorting", *Information Processing Letters*, Vol. 13, No. 2, November 1981, pp. 55-57.
- [BO] J. L. Bentley and T. A. Ottmann, "Algorithms for Reporting and Counting Geometric Intersections", *I.E.E.E. Transactions on Computers*, Vol. c-28, No. 9, September 1979, pp. 643-647.
- [BT] B. K. Bhattacharya and G. T. Toussaint, "An $O(n \log \log n)$ Time Algorithm for Determining Translation Separability of Two Simple Polygons", Technical Report no. SOCS 86.1, School of Computer Science, McGill University, Montreal, Quebec, January 1986.
- [C] B. Chazelle, "Intersection is Easier than Sorting", *Proceedings of the 16th Annual Symposium of Theory of Computing*, 1984, pp. 119-127.

-
- [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.
- [DS] J. A. Dean and J.-R. Sack, "Efficient Hidden-Line Elimination by Capturing Winding Information", *Proceedings 23rd Annual Allerton Conference On Communication, Control, and Computing*, Monticello, Illinois, October 1985, pp. 496-505.
- [DS1] F. Dehne and J.-R. Sack, "Translation Separability of Sets of Polygons", *The Visual Computer*, No. 3, 1981, pp. 227-235.
- [EA] H. El-Gindy and D. Avis, "A Linear Algorithm for Computing the Visibility Polygon from a Point", *Journal of Algorithms*, Vol. 2, No. 2, 1981, pp. 186-197.
- [FV] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

-
- [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, "Visibility of a Simple Polygon", *Computer Vision, Graphics, and Image Processing*, Vol. 22, 1983, pp. 207-221.
- [K] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading MA, 1968.
- [M] N. Megiddo, "Linear Time Algorithm for Linear Programming in R^3 and Related Problems", *SIAM Journal on Computing*, Vol. 12, No. 4, November 1983, pp. 759-776.
- [MP] D. E. Muller and F. P. Preparata, "Finding the Intersection of Two Convex Polyhedra", *Theoretical Computer Science* Vol. 7, No. 2, October 1978, pp. 217-236.
- [MT] N. Magnenat-Thalmann and D. Thalmann, *Image Synthesis Theory and Practice*, Ed. T. L. Kunii, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo 1987.
- [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.

-
- [Na] B. K. Natarajan, " On Planning Assemblies", Proceedings of the Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, Illinois, June 1988, pp. 299-308.
- [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.
- [PH] F. P. Preparata and S. J. Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", Communication of the ACM, Vol. 20, No. 2 February 1977, pp. 87-93.
- [PS] F. P. Preparata and M. I. Shamos, *Computational Geometry An Introduction*, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.
- [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.
- [RU] I. Rival, J. Urrutia, "Representing Orders on the Plane by Translating Convex Figures", Technical Report No. TR-87-05, School of Computer Science, University of Ottawa, Ottawa, Ontario, February 1987.
- [S] J.-R. Sack, Personal Communication.

-
- [SH] M. I. Shamos and D. Hoey, "Geometric Intersection Problems", Proceedings of the 17th Annual Conference on Foundations of Computer Science, 1976, pp. 208-215.
- [SN] J.-R. Sack and O. Nurmi, "Separating a Polyhedron by One Translation from a Set of Obstacles", Proceeding Workshop on Graph Theory, Amsterdam 1988, to appear in Lecture Notes in Computer Science. Springer-Verlag Berlin, Heidelberg, New York, Tokyo.
- [ST] J.-R. Sack and G. T. Toussaint, "Some New Results On Moving Polygons in The Plane", Proceedings Robotic Intelligence and Productivity Conference, Detroit, Michigan, November, 1983, pp. 158-163.
- [ST1] 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.
- [T1] G. T. Toussaint, "Shortest Path Solves Translation Separability of Polygons", Technical Report No. SOCS-85.27, School of Computer Science, McGill University, Montreal, Quebec, October 1985.
- [Ta] R. E. Tarjan, "Depth First Search and Linear Graph Algorithms", *SIAM Journal on Computing*, Vol. 1, 1972, pp. 146-160.

- [WHG] H. Weghorst, G. Hooper and D. P. Greenberg, "Improved Computational Methods for Ray Tracing", ACM Transactions on Graphics, Vol. 3, No. 1, January 1984, pp. 52-69.

END

1|7·0|2·90

FIN