アルゴリズム 10－2
（1989. 9. 22）

三次元におけるｃ－ｏｒｉｅｎｔｅｄ ｏｂｊｅｃｔｓの平行移動問題について

譚学厚， 平田富夫， 稲垣康善

名古屋大学 工学部

近年，幾何物体を与えられた方向に一個ずつ衝突が起こらないように平行移動する問題についての研究が盛んに行われている．本稿では三次元におけるｃ－ｏｒｉｅｎｔｅｄ ｏｂｊｅｃｔｓの平行移動問題を調べる．物体は辺の方向が定数種類の時にｃ－ｏｒｉｅｎｔｅｄ ｏｂｊｅｃｔｓという．計算幾何学からの手法を用い，この問題をより良いｗｏｒｓｔ－ｃａｓｅ ｂｏｕｎｄｓで解決することができた． このアルゴリズムは計算機による図形処理に応用を持つ．

On Translating a Set of C-oriented Objects in Three Dimensions

Xue-Hou TAN, Tomio HIRATA and Yasuyoshi INAGAKI

Faculty of Engineering, Nagoya University,

Chikusa-ku, Nagoya 464, Japan

Recently much attention has been devoted to the problem of translating a set of geometrical objects in a given direction, one at a time, without allowing collisions between the objects. This paper studies the translation problems in three dimensions on sets of "c-oriented objects", that is, objects whose bounding edges has a constant number of orientations. Applying some methods from computational geometry, these problems can be solved with better worst-case bounds than those for the corresponding general problems. The algorithms find uses in computer graphic systems.

1 Introduction

Motivated by applications in computer graphics and VLSI layout, much attention has recently been devoted to the problem of translating sets of objects, such as line segments and polygons in the plane or polyhedra in three dimensions, without allowing collisions between the objects. In [5] Guibas and Yao have presented an algorithm for computing the translation order of rectangles in the plane, and then generalized it to include line segments and convex polygons. However, the algorithm for line segments and convex polygons needs two passes. A one-pass algorithm is later discovered by Ottmann and Widmayer [13]. On the other hand, Nurmi [12] has modified Guibas and Yao's algorithm for line segments to work with arbitrary polygons, and also developed the algorithms for computing the translation orders of line segments, planar polygonal faces and polyhedra in three dimensions.

The translation algorithms for sets of objects in three dimensions have applications in the *hidden surface elimination problem.* If we use the *orthogonal projection* which moves the view point to infinity and thus takes a direction of view (in our case, a direction of translation) as input for a hidden surface elimination algorithm, a "painter's" algorithm that paints the objects onto the image buffer in reverse order of translation can produce the realistic image of given objects. Real time applications, such as flight simulation and computer animation, require efficient algorithms for the translation problems.

A successful experience in computational geometry is that orthogonal objects can be handled much easily than arbitrary objects. This suggests that the problem of translating a set of parallelepipeds might be solved much efficiently than arbitrary polyhedra. In this paper we consider the translation problems in three dimensions on sets of "c-oriented objects", a more general class of objects than orthogonal ones. The notion of "c-oriented objects" was introduced by Güting [6]. Polyhedra in three dimensions are called c-oriented if the bounding edges of all polyhedra have at most a constant number of orientations. Many practical scenes, such as architecture and furniture, have this property. So apart from theoretical interest of different algorithmic complexity classes, the results of this work may find applications in computer graphic systems.

The organization of the paper is as follows: Section 2 gives a brief review of the general approach to solving translation problems, and defines three "c-oriented" variants of the translation problem in three dimensions. Section 3 presents efficient algorithms for these problems, and Section 4 elaborates on an algorithm presented in Section 3. Section 5 discusses some properties of c-oriented objects.

2 The Translation Problem

The general translation problem can be described as follows: Given a set of disjoint objects, find an order in which the objects can be translated to infinity one after the other in a given direction, such that during the translation no object collides with another. Two immediate problems are how to *detect* the existence of a translation order and *compute* the translation order if it exists.

Let us recall the approach to solving translation problems (also see [12]). First, a binary relation *dom* (dominates) on the set of objects is constructed. X *dom* Y means that X must be translated before Y. Before giving the exact definition of *dom*, we first define other two relations on the set of objects. One is the relation *edom* (dominates eventually); X *edom* Y if and only if there exist points x on X and y on Y such that the line segment from y to x is parallel to the translation direction. The other is the relation *idom* (dominates immediately); X *idom* Y if and only if X *edom* Y and all the line segments from Y to X parallel to the translation direction do not intersect any other objects. Examples for *idom* and *edom* in the plane are shown in Figure 1. Clearly $idom^+ = edom^+$ where $idom^+$ ($edom^+$) denotes the transitive closure of *idom* (*edom*). Thus *dom* can be defined as $idom^+$. It suffices to know either *idom* or *edom*. Generally the *idom* relation is sparse but difficult to compute, while the *edom* relation is easy to compute but dense. This makes it difficult to compute either *idom* or *edom* exactly. Often we take a compromise solution, that is, we find a subrelation *edom'* of *edom* such that $(edom')^+ = idom^+$, and *edom'* has a suitable size and can easily be computed.

Second, *dom* $(= (edom')^+)$ is examined for asymmetry, namely, A *dom* B implies NOT (B *dom* A). If *dom* is asymmetric, it defines a partial order on the set of objects, and any linear extension of *dom* gives us the total order of objects. Otherwise, the translation order does not exist. To test *dom* for asymmetry, we transform the *edom'* relation into a directed graph $G(V, E)$, where V is the set of objects and E has a directed edge (X, Y) if and only if X *edom'* Y. Hence *dom* is asymmetric if and only if G is acyclic. By a depth-first search [16] we can determine the existence of cycles of G in $O(|V| + |E|)$ time or, in our terms, in $O(n + |edom'|)$ time where n is the number of objects and $|edom'|$ is the size of the *edom'* relation.

Finally, if *dom* is asymmetric we can topologically sort [9] the *edom'* relation in $O(n + |edom'|)$ time to obtain a linear extension of *dom*.

From the discussion above, we conclude that the kernel of a translation algorithm is to compute the *edom'* relation, and this computation dominates the time complexity of the algorithm. In the remainder of this paper, we will focus on computing the *edom'* relation.

We study three c-oriented versions of the translation problem in three dimensions. Before defining these restricted problems, we first define the notion of "orientations". An orientation in three dimensions is a vector with the positive z component which passes through the origin. We describe an orientation α as two (counterclockwise) angles from the x and y axes. A left-handed coordinate system is used as shown in Figure 2. This definition extends in the obvious way to higher dimensions. (The definition of "orientations" in the plane can be found in [15].)

Line segments in three dimensions are said to be c-oriented if there exist c orientations $\alpha_1, \alpha_2, \cdots, \alpha_c$ such that all line segments are restricted to be parallel to any one of them. Planar polygonal faces (and polyhedra) are said to be c-oriented if the bounding edges of all faces (and polyhedra) are c-oriented. It is obvious that a set of c-oriented faces has no more than c normal vectors. A face, whose normal vector is parallel to α, is said to have the orientation α. The three problems we consider are completely characterized by their input sets:

Problem A: A set of c-oriented line segments in three dimensions.
Problem B: A set of c-oriented polygonal faces in three dimensions.
Problem C: A set of c-oriented polyhedra in three dimensions.

3 C-oriented Objects in Three Dimensions

We assume that a polyhedron is represented by its bounding polygonal faces; a face is given by its normal vector, pointing away from the polyhedron, and the set of its edges; finally, an edge is given by two endpoints such that the order of endpoints tells us the interior of the face. We also assume without loss of generality that the translation direction is the positive z direction.

Problem A (C-oriented Line Segments)

When input is a set of c-oriented line segments, the n given line segments are first projected onto the (x, y) plane. An *edom'* relationship between two segments is then constructed if their images intersect. To compute the intersections of the images, Chazelle and Edelsbrunner's algorithm [2] might be used. Their algorithm solves the line segment intersection problem in $O(n \log n + k)$ time and $O(n + k)$ space, where k is the number of the intersections. In [15] a simpler algorithm for the c-oriented line segment intersection problem is given, which is optimal and runs in $O(n \log n + k)$ time and $O(n)$ space.

Problem B (C-oriented Polygonal Faces)

Now we consider the problem of translating a set of c-oriented polygonal faces. If two faces are related by *edom'*, their projected figures must intersect. (However, the reverse may not be true. Two projected faces intersect if and only if they are related by *edom*.) Two faces intersect in the projection either if their edges intersect or if the images of them intersect although their edges do not.

Our algorithm for computing the *edom'* relation then consists of two parts. The first part computes the *edom'* relationships due to the edge intersections in the projection. In order to simplify the algorithm, we report an *edom'* relationship at every edge intersection in the projection, namely, all the *edom* relationships between the faces with projected edge intersections. The problem is again reduced to that of finding edge intersections in the projection, which can be solved in $O(N \ log \ N + k)$ time and $O(N)$ space, where N is the number of edges of the faces and k the number of edge intersections in the projection plane.

The second part of our algorithm finds the *edom'* relationships which are determined in the case where two faces intersect in the projection but have no edge intersections. Having found all the *edom* relationships due to the edge intersections, we can assume that the faces have no edge intersections in the projection. To compute the *edom'* relation in a set of such faces, again one might find all the *edom* relationships. The result will depend on the number s of these face pairs. Observe that s can be $O(n^2)$, where n is the number of the faces.

The s bound can be reduced to $2n$. That is, there exists a subrelation *edom'* of *edom* on the set of the faces without edge intersections in the projections, such that $(edom')^+ = idom^+$ and $|edom'| \leq 2n$. We accomplish this by associating with each face F the point f with the maximal x-value as its representative point. An important property about the representative points is that, A *idom* B either if face A is the face nearest to point b (the representative of face B) in the $+z$ direction (Figure 3a) or, in turn, face B to point a (the representative of face A) in the $-z$ direction (Figure 3b). Note that if all of the given faces are simple, i.e., they have no holes, the representative point of a face can be any one in its interior. The problem now is reduced to finding for each representative point two faces nearest to it in the $+z$ and $-z$ direction respectively. Thus at most two *edom'* relationships are output at every representative point.

In the next section we will establish an algorithm of $O(N \ log \ ^2N)$ time and $O(N \ log \ N)$ space for finding the nearest faces for the n representative points. We assume in the following section that no faces are parallel to the z axis. (If face A parallel to the z axis is the face nearest to point b, face A and face B must have edge intersections in the projection).

Combining these two results above, we obtain an algorithm of $O(N \ log \ ^2N + k)$ time and $O(N \ log \ N)$ space for computing the *edom'* relation in a set of c-oriented faces. For comparison, let us consider the complexities of Nurmi's algorithm [12] that computes the *edom'* relation in a set of arbitrary faces. His algorithm runs in $O((k + N) \ log \ N)$ time and space. Note that there is no better bound on the size of k than $O(N^2)$, hence the terms in k may dominate the complexities.

Problem C (C-oriented Polyhedra)

The algorithm for translating a set of c-oriented polygonal faces can simply be modified to work with c-oriented polyhedra (for details see [12]). When the original algorithm outputs an *edom'* relationship between two faces of distinct polyhedra, the modified one outputs the *edom'* relationship between the polyhedra. When the original algorithm reports an *edom'* relationship between the faces belong to the same polyhedron, the modified one outputs nothing. Thus, the asymptotic complexities of the modified algorithm for a set of c-oriented polyhedra are the same as those for the set of bounding faces of all polyhedra.

4 The Algorithm for the nearest faces

In this section we present an algorithm that reports the nearest faces for the n representative points in the $+z$ direction, which works with the $-z$ direction in the obvious way. For conventions, let $\alpha \times \beta$ denote the orientation which is the vector product of α and β in the left-handed coordinate system.

To find the nearest faces, a plane, perpendicular to the y axis, is swept through the space in the positive y direction. At any position the intersection of the sweep plane with the set of faces is a set of line segments. A single face may produces several collinear segments if it is concave or has holes. The sweep plane coordinate system is chosen such that the positive z direction is vertical (Figure 4). From the assumption that no faces are parallel to the z axis, the segments in the sweep plane can not be vertical.

To support the plane sweep, a data structure is maintained representing the dynamic set of line segments in the sweep plane. The structure must be able to answer queries about the face immediately above any point of the sweep plane in the z direction. When a representative point p is encountered, a nearest face is asked for p. Clearly the answer is one whose segment in the sweep plane first intersects the vertical half-line emanating from p to $+\infty$, namely, the half-line (x_p, z_p)—$(x_p, +\infty)$. See Figure 4.

To report this segment in the sweep plane, we first find the segments whose x-projections contain p, then locate on these segments the segment immediately above p in the z direction, and hence determine the face nearest to p. How can these two searches be efficiently performed?

When the sweep plane is swept, two endpoints of a segment in the sweep plane move at different speeds to the left or right. It is difficult to answer point enclosure queries with respect to the varying segments in the sweep plane. At this point, our restriction to c-oriented faces becomes crucial. Observe that the movement of a segment is characterized by the orientations of the edges (of a face) along which two endpoints of that segment move. The segments in the sweep plane can thus be split into at most c^2 disjoint sets according to the combination of edge orientations. Of course some sets may be empty. Let $S_{\alpha,\beta}$ denote the set of the segments whose left and right endpoints are respectively characterized by edge orientations α and β, and θ' the orientation in the $(x,\ z)$ plane which is the projection of orientation θ in three dimensions. When the sweep plane is moved, the segments in $S_{\alpha,\beta}$ are fixed with respect to the $(\overline{\alpha'},\overline{\beta'})$-coordinate system in the $(x,\ z)$ plane, where $\overline{\alpha'}$ and $\overline{\beta'}$ are the orientations perpendicular to orientations α' and β' in the $(x,\ z)$ plane respectively. Then it suffices to answer point enclosure queries with respect to the set $S_{\alpha,\beta}$.

The second search locates the segment immediately above p in the sweep plane. Note that all the faces represented by $S_{\alpha,\beta}$ have the same orientation, namely, the $\alpha \times \beta$ orientation. In fact, $S_{\alpha,\beta}$ denotes a set of parallel segments in the sweep plane (Figure 5). On the segments of $S_{\alpha,\beta}$ whose x-projections contain p in the sweep plane, we can locate the segment immediately above p by finding one nearest to p in the direction perpendicular to the segments. That is, in three dimensions, of $S_{\alpha,\beta}$ the face nearest to p in the z direction is the same as that in the $\alpha \times \beta$ direction. So the task is a simple binary search on the $\alpha \times \beta$ coordinate.

Combining the discussions above, the segments in the sweep plane are first partitioned into at most c^2 disjoint sets $S_{\alpha,\beta}$. This means that we will use c^2 different data structures, one for each set. The cardinality of the union of all sets are $O(N)$. Within each set $S_{\alpha,\beta}$, a segment is represented by a triple $(\overline{\alpha'_0}, \overline{\beta'_0}, (\alpha \times \beta)_0)$, where $\overline{\alpha'_0}$ and $\overline{\beta'_0}$ are the $\overline{\alpha'}$ and $\overline{\beta'}$ coordinates of the edges in the $(x,\ z)$ plane which form the left and right endpoints of the segment, and $(\alpha \times \beta)_0$ is the $\alpha \times \beta$ coordinate of the face in three dimensions which is represented by this segment. Suppose that a query point p is now met, and represented in the $(\overline{\alpha'}, \overline{\beta'}, \alpha \times \beta)$-coordinate system as the triple $(\overline{\alpha'_p}, \overline{\beta'_p}, (\alpha \times \beta)_p)$. Of the set $S_{\alpha,\beta}$, the segment immediately above p in the sweep plane is obtained by the following searching:

> Given $\overline{\alpha'_p}$, $\overline{\beta'_p}$ and $(\alpha \times \beta)_p$, find a triple $(\overline{\alpha'},\ \overline{\beta'},\ \alpha \times \beta)$ from $S_{\alpha,\beta}$ whose $\alpha \times \beta$ is minimal in the set $\{(\overline{\alpha'}, \overline{\beta'}, \alpha \times \beta) \mid \overline{\alpha'} \leq \overline{\alpha'_p}$ and $\overline{\beta'_p} \leq \overline{\beta'}$ and $(\alpha \times \beta)_p \leq \alpha \times \beta\}$.

To solve this searching problem, we regard $S_{\alpha,\beta}$ as a 3-dimensional point set (each point has three coordinates) and thus search for the point with the minimal $\alpha \times \beta$-coordinate in the 1/8 space $([-\infty, \overline{\alpha'_p}], [\overline{\beta'_p}, \infty], [(\alpha \times \beta)_p, \infty])$. The standard data structure supporting this kind of searchings is 3-dimensional *range trees* (see [11, 14]). Using the 3-dimensional range tree over the $\overline{\alpha'}$, $\overline{\beta'}$ and $\alpha \times \beta$ coordinates, the above searching can be implemented in $O(\log{}^3 N)$ time. The structure requires $O(N \log{}^2 N)$ space and can be updated in $O(\log{}^3 N)$ time.

In our special case (where all ranges are not intervals, but half-spaces) we can save a factor $\log N$ on the time and space bounds. To achieve this, *priority search trees* (see [10]) are used. A priority search tree stores a set of n points (x, y) in linear space. It supports the computation of the minimal x-coordinate of any point in a semi-strip $x_0 \leq x \leq x_1$ and $y \leq y_1$ with logarithmic running time. To solve our problem, we organize the last two levels of range trees into one level of priority search trees over the $(\alpha \times \beta, -\overline{\beta'})$

coordinates. More specifically, all points in $S_{\alpha,\beta}$ are stored in the top level range tree over the $\overline{\alpha}'$ coordinates and the set of points associated with each node is then organized into a priority search tree. It follows that the range-priority search tree permits the retrieval of the minimal $\alpha \times \beta$-coordinate of any point in the 1/8 space in $O(log\ ^2N)$ time taking $O(N\ log\ N)$ space. A point can be inserted into or deleted from the rang-priority search tree in $O(log\ ^2N)$ time. Since all faces are given beforehand, the semi-dynamic versions of the c^2 range-priority search trees suit to our purpose.

The sweep plane also stops at the vertices of all faces. Whenever a vertex is encountered, certain segments disappear, and other segments occur in the sweep plane. The projection of the sweep plane in the (x, y) plane is a line, called the *sweep line*, which is parallel to the x axis and advances in the positive y direction. Figure 6 shows the change of a face while the sweep line is moved. On meeting a vertex of a face F, we have a difficulty in determining the segments of F to be deleted from and inserted into the structures. For example, a vertex v in Figure 6 does not give us any information about the edge pair $(b,\ e)$, which defines the segment to be deleted from the range-priority search trees. To overcome this difficulty we use, as a subsidiary data structure, a balanced binary search tree for each face currently cut by the sweep plane. The intersection of the sweep line and a projected face in the (x, y) plane is a set of collinear segments, which are totally ordered from $x = -\infty$ to $x = +\infty$ and thus maintained in F's binary search tree. When a vertex is scanned, we first determine the segments to be deleted and inserted in F's binary search tree, and then we can update the range-priority search trees correctly. The total cost of the balanced trees in the whole process of the sweep is $O(N\ log\ N)$ time and $O(N)$ space. More specific description can be found in the companion paper [15].

We have given an approach to finding the nearest faces. During the sweep, $O(N)$ segments are inserted into and deleted from the range-priority search trees. When a representative point is met, we query all c^2 range-priority search trees for the nearest faces. The real nearest face is selected from these c^2 candidates. Thus, we obtain an algorithm of $O(N\ log\ ^2N)$ time and $O(N\ log\ N)$ space for finding the nearest faces, where N is the total number of edges in the scene. Taking c into account the time complexity is $O(N\ log\ ^2N + c^2\ n\ log\ ^2N)$, since only n nearest face queries are dependent on c, where n is the number of faces. The space requirement is independent of c. In practical uses, c must be rather small to make our algorithm efficient.

5 Discussions

In an effort to bridge the gap in complexity between orthogonal objects and arbitrary objects, there have been a number of papers in "c-oriented" geometry [3, 6, 7, 8, 4, 15, 17]. In this paper it is once again shown that problems involving c-oriented objects can be solved with similar efficiency as the corresponding problems involving orthogonal objects. The general rule for processing c-oriented objects is to decompose a problem by orientations, which seems powerful to the *decomposable problems* (see [1]). Further topic of interest in this area is to look for other problems whose c-oriented variants can be solved efficiently, and furthermore, classify the subclass of the decomposable problems which can be decomposed by orientations.

References

[1] J.L.Bentley and J.B.Saxe, Decomposable searching problems I: Static-to-dynamic transformations, *J. of Algorithms* **1**, 1980, 571-577.

[2] B.Chazelle and H.Edelsbrunner, *An optimal algorithm for intersecting line segments in the plane*, Rep. 88-1419, Comput. Sci. Dept., Illinois Univ., 1988.

[3] J.Culberson and G.J.E.Rawlins, Turtlegons: Generating simple polygons from sequences of angles, in *Proceedings, ACM Symp. Comput. Geometry*, 1985, pp. 305-310.

[4] G.J.E.Gregory and D.Wood, Optimal computation of finitely oriented convex hulls, *Inform. Computation* **72**, 1987, 150-166.

[5] J.L.Guibas and F.F.Yao, On translating a set of rectangles, in *Proceedings, 12 Annu. ACM Symp. Theory of Computing*, 1980, pp. 154-160.

[6] R.H.Güting, Stabbing *c*-oriented polygons, *Inform. Process. Lett.* **16**, 1983, 35-40.

[7] R.H.Güting, Dynamic *c*-oriented polygonal intersection searching, *Inform. Control* **63**, 1984, 143-163.

[8] R.H.Güting and Th. Ottmann, New algorithms for special cases of hidden line elimination problem, *Comput. Vision Graphics Image Process.* **40**, 1987, 188-204.

[9] D.Knuth, *The Art of Computer Programming, Vol 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

[10] E.M.McCreight, Priority search trees, *SIAM J. Comput.* **14**, 1985, 257-276.

[11] K.Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.

[12] O.Nurmi, On translating a set of objects in 2- and 3-dimensional space, *Comput. Vision Graphics Image Process.* **36**, 1986, 42-52.

[13] Th.Ottmann and P.Widmayer, On translating a set of line segments, *Comput. Vision Graphics Image Process.* **24**, 1983, 382-389.

[14] F.P.Preparata and M.I.Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, Berlin, 1985.

[15] X-H.Tan, T.Hirata and Y.Inagaki, *An optimal algorithm for reporting intersections of c-oriented polygons*, Tech. Rep., Engineer. Dept., Nagoya University, 1989.

[16] R.E.Tarjan, Depth-first search and linear graph algorithms, *SIAM J.Comput.* **1**, 1972, 146-160.

[17] P.Widmayer, Y.F.Wu and C.K.Wang, Distance problems in computational geometry for fixed orientations, in *Proceedings, ACM Symp. Comput. Geometry*, 1985, pp. 186-195.

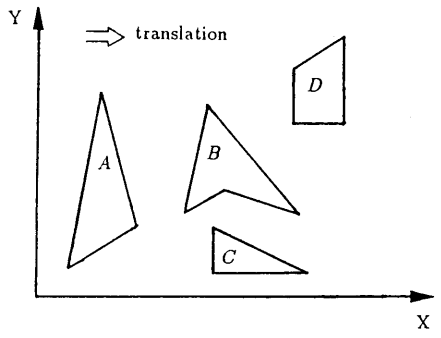

Figure 1

B idom A	*B edom A*
C idom A	*C edom A*
D idom B	*D edom B*
	D edom A

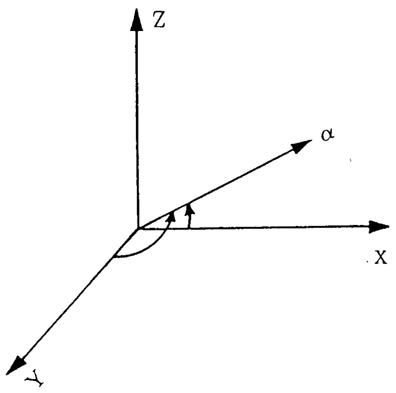

Figure 2

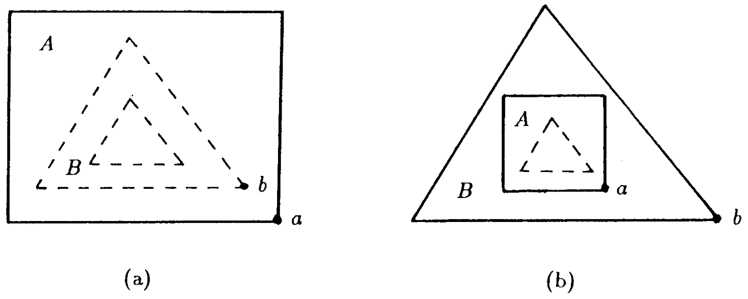

Figure 3

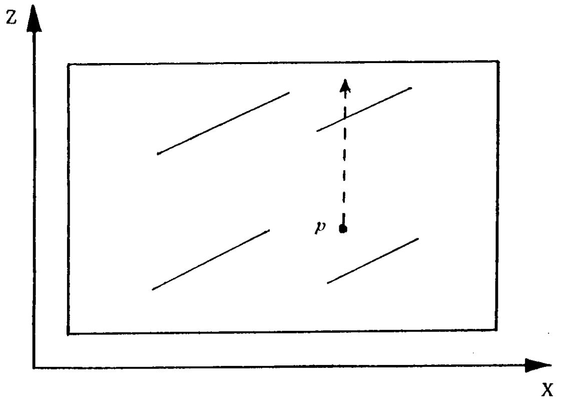

Figure 4

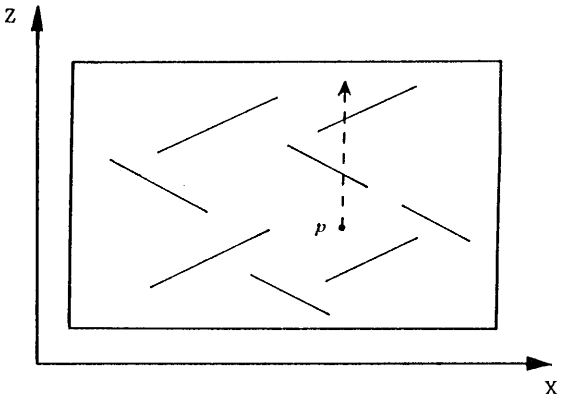

Figure 5

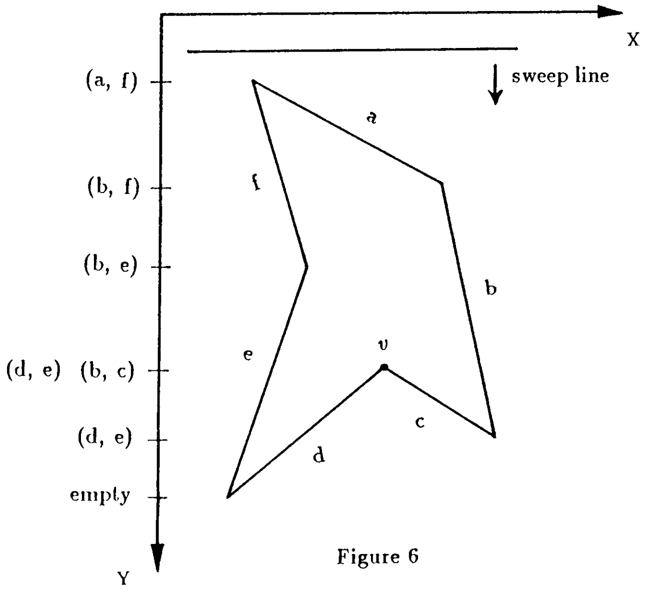

Figure 6