アルゴリズム 6－3
(1989. 3. 22)

C-Oriented Polygon の交差列挙問題について

譚学厚, 平田富夫, 稲垣康善

名古屋大学 工学部

本報告では平面上のいくつかの c-oriented polygon に対し，交わる対をすべて報告する問題を調べる。多辺形の集合が c-oriented と呼ばれるのは，含まれる多辺形の方向が定数種類しかないときをいう。この問題は，VLSIの設計規則適合検査と建物／家具データベース等に応用を持つ。ここで与えるアルゴリズムは，O(n)の記憶領域と O(n log n + t)の計算時間を要する。ここで，nは多辺形の数，tは報告される対の数である。この時間と領域計算量はともに最適である。

The C-Oriented Polygon Intersection Problem

Xuehou TAN, Tomio HIRATA and Yasuyoshi INAGAKI

Faculty of Engineering, Nagaya University, Chikusaku-ku, Nagoya, Japan

We examine the problem of reporting all intersecting pairs in a set of c-oriented polygons in the plane. A set of polygons is called c-oriented if the edges of all polygons have only a constant number of orientations. The problem arises in many applications such as the VLSI design rule checking and architecture or furniture databases. We present an optimal algorithm that runs in O(n log n + t) time and O(n) space, where n is the number of polygons and t the number of intersecting pairs.

1. Introduction

Let P denote a set of simple polygons, each having at most a fixed number of edges in the Euclidean plane. A polygon intersection problem requires to report all intersecting pairs in P. The classical rectangle intersection problem in which P is a set of axis-parallel rectangles is already mentioned in [1, 9], and much work has been done on this problem [3, 5, 12, 15]. Whereas the arbitrary polygon intersection problem has received few attention.

We consider so-called c-oriented objects, a notion proposed by Guting in order to bridge the gap in complexity between orthogonal objects and arbitrary objects [6, 7, 8], and thus investigate the c-oriented polygon intersection problem. In the plane a set of polygons is called c-oriented if the edges of all polygons have only a constant number of orientations.

Apart from the theoretical interest, motivations for studying c-oriented polygon intersection problem also stem from some applications. For example, in VLSI artwork analysis two distinct components must be seperated by a certain distance. The detection of whether or not the seperation rule is obeyed (usually, called VLSI design rule checking) can be cast as such an instance, where fast algorithms are required to detect or report intersecting pairs among thousands of objects having a few orientations. The importance is that we can perform VLSI design rule checking on real boundaries of components. In contrast to this, a common approach is based on "bound boxes" -- each component is represented by the smallest rectangle enclosing it, and then two components are checked in detail if their respective rectangles intersect (see [1, 9]). The similar problem also arises in maintaining architecture or furniture databases.

Optimal algorithms for the rectangle intersection problem have been obtained independently by McCreight[12] and Edelsbrunner[5]. Their algorithms run in optimal time O(n log n + t) and space O(n), where n is the number of rectangles and t the number of intersecting pairs. In this paper we present an algorithm that solves the c-oriented polygon intersection problem in the same optimal time O(n log n + t) and space O(n). In [7] Guting gives an algorithm for this problem, as a by product of c-oriented polygon intersection search problem, which runs in $O(n \log^2 n + t)$ time and O(n log n) space, and remains it open whether his bounds could be improved to O(n log n + t) time and O(n) space. Our result here is an answer to his open problem.

The paper is organized as follows: In section 2 we examine the c-oriented polygon intersection problem and see how it can be reduced to two subproblems: the c-oriented version of line segment intersection problem and batched point enclosure search problem. In section 3 and 4 we give algorithms based on the line-sweep mechanism for these two subproblems respectively. The methods used in section 3 and 4 may report an intersection more than once (but at most a constant times, see section 2). Section 5 gives another algorithm to avoid multiple reports with slightly lesser performance than that of optimal algorithm. Directions for further work and conclusions are finally offered in section 6.

2. The C-Oriented Polygon Intersection Problem

For preciseness let us recapitulate the definition of the notion "c-orientedness" and the polygon intersection problem.

Definition 1 (c-orientedness): A set of line segments in the plane is c-oriented if there exist c orientations $\alpha_1, \alpha_2, \cdots, \alpha_c$ such that the orientations of all line segments are among $(\alpha_1, \alpha_2, \cdots, \alpha_c)$. A set of polygons is called c-oriented if the set of edges of all polygons is c-oriented.

Definition 2 (the c-oriented polygon intersection problem): Given a set P of c-oriented polygons, each having at most a fixed number of edges, report all pairs of polygons in P which have at least one point in common.

We assume that each polygon is simple (that is, no pair of nonconsecutive edges sharing a point) and given by the sequence of its vertices in clockwise or counterclockwise order. For conventions, we call a line segment α-segment if it is parallel to α.

It is obvious that the polygon intersection problem is reducd to two subproblems. Namely, two polygons intersect each other either if their edges intersect or if one entirely encloses the other. Hence the problem can be solved in two stages. The first stage is to find all pairs of polygons with edge intersections. For this purpose an algorithm for reporting line segment intersections is developed in the following section. The second stage is to find all pairwise polygon enclosures. In this stage we choose for each polygon X a representative vertex x from its vertices, and then for each representative vertex we find out all enclosing polygons. This is a special case of the batched point enclosure search problem, which we will discuss in section 4.

The two subproblems are respectively solved in the next two sections. Combining these results we obtain that, the polygon intersection problem for n polygons whose edges have a constant number of orientations can be solved in O(n log n + t), where t is the number of reported intersecting pairs.

Our algorithm reports each pair of intersecting polygons at least once, but no more than a constant times since the number of edges of a polygon is bounded by a constant. This is the only reason we posed a restriction on the number of edges of a polygon. In fact our algorithm works correctly without this restriction, but each intersecting pair might be reported too often.

3. The Line Segment Intersection Problem

In this section we consider the special case of line segment intersections in which all of the n given line segments have a constant number c of orientations $\alpha_1, \alpha_2, \cdots, \alpha_c$.

We decompose the problem by orientations; that is, we report the intersections of α_i-segments with non α_i-segments for each α_i $(1 \leqq i \leqq c)$.

To compute the intersections for orientation α, we sweep through the plane a scan-line which is parallel to α and stops at each endpoint of a non α-segment and at each α-segment. During the sweep we maintain a data structure to store the non α-segments currently cut by the scan-line (the "active" non α-segments). Initially the structure is empty. Whenever the scan-line stops at a left (right) endpoint of a non α-segment, this segment is inserted into (deleted from) the structure. When an α-segment is scanned, we check it for intersections with the active non α-segments in the structure.

The c-orientedness enables us to split the active non α-segments into c-1 disjoint sets by their orientations. Let S_β denote the set corresponding to orientation β. The cardinalities of all sets S_β are O(n). The segments in S_β is then ordered by their β^--coordinates where β^- is the orientation perpendicular to β, and implemented as a balanced binary leaf-search tree with leaves storing these β^--coordinates [2]. Thus our data structure consists of c-1 balanced binary trees. Each of the insertion, deletion and search operations is performed in O(log n) time. When the scan-line meets an α-segment (a, b), we find the intersections of the segment (a, b) with segments in S_β by projecting two endpoints a and b onto the β^- axis (see Figure 1). Reporting the intersecting pairs can be done in time proportional to their number if the leaves of the tree are kept in a doubly linked list.

As done in [2], some special cases must be handled carefully, such as the line segments overlapping and sharing endpoints. Besides, our method reports each intersection twice. To avoid this, we remove all α-segments from the given set of line segments as soon as the sweep for α is finished. Thus balanced binary trees are decreased one by one, and the last sweep is really unnecessary.

From the previous discussion, the intersection problem for c-oriented line segments can be solved in O(n log n + t) time and O(n) space, where t is the number of intersecting pairs. Taking c into account the time complexity is O(c n log n + t). The space requirements are independent of c. These time and space complexies are the same as those for the orthogonal case. Although the same time complexity has been obtained by Chazelle and Edelsbrunner [4] for the arbitrary line segment intersection problem, their algorithm requires O(n + t) space and our algorithm is simpler.

Using the method described by Bentley and Ottmann [2], we can modifiy the algorithm for counting the total number of intersecting pairs of c-oriented line segments (without reporting them), the modified algorithm will run in O(n log n) time and O(n) space.

4. The Batched Point Enclosure Search Problem

In this section we study the batched point enclosure search problem: Given a mixed set of query points and c-oriented polygons, report for each point all of the polygons enclosing it. It is assumed that the total number of points and edges is n. (We need not place the restriction on the number of edges of a polygon.)

As discussed in [3], one solution to the batched point enclosure search problem is to cast it as a search problem, that is, the set of polygons is first organized into a data structure and then for each point an enclosure query is asked with respect to the set of polygons. In [7] Guting presents an algorithm for the c-oriented version of the search problem, which leads to a result of O(n $\log^2 n$ + t) time and O(n log n) space, where t is the total number of reported polygons.

We now give a solution of the batched problem by using the scan-line paradigm, that is, we sweep a vertical line from left to right, and as it sweeps over a query point we report the enclosing polygons. During the sweep, a data structure is maintained to store the polygons currently cut by the scan-line, the "active" polygons. The structure must be able to answer queries about the polygons enclosing any point of the scan-line. The intersection of the sacn-line with the set of active polygons forms intervals of the scan-line. A single polygon may produce several intervals if it is concave. See Figure 2. When a query point is encountered, we report the polygons whose intervals contain the point.

Unlike the orthogonal case, the endpoints of intervals on the scan-line are not fixed when the scan-line is swept, but move at different speeds to the top or bottom. Obviously the segment tree of Bently and Wood [3] can not be used to answer point enclosure queries on a set of continuously varying intervals. We need a data structure to hold the intervals that vary with the scan-line.

Observe that an endpoint of an interval moves along an edge of an active polygon. Thus we can describe endpoints by the corresponding edges. An edge with orientation α is represented by $\bar{\alpha}$-coordinate where $\bar{\alpha}$ is the orientation perpendicular to α. It is assumed that the introduced $\bar{\alpha}$ axis points upwards from the origin of the (x, y)-coordinate system. Then an interval in the scan-line is represented by a pair ($\bar{\alpha}_0$, $\bar{\beta}_0$), where α and β are the orientations of it's top and bottom edges respectively. See Figure 3. Since α and β have no more than c different orientations, the intervals are partitioned into at most c^2 disjoint sets. Let $I_{\alpha,\beta}$ denote the set corresponding to orientations α and β. The cardinalities of all sets $I_{\alpha,\beta}$ are O(n). Therefore we can represent the intervals varying with the scan-line by c^2 different data structures, one for each set $I_{\alpha,\beta}$.

Suppose that a query point p is now encountered. If p is represented in ($\bar{\alpha}$, $\bar{\beta}$)-coordinate system as a pair ($\bar{\alpha}_p$, $\bar{\beta}_p$), an interval ($\bar{\alpha}_0$, $\bar{\beta}_0$) in $I_{\alpha,\beta}$ contains p iff

$$\bar{\alpha}_p \leqq \bar{\alpha}_0 \text{ and } \bar{\beta}_0 \leqq \bar{\beta}_p .$$

To answer this kind of range queries efficiently, a standard data structure of computational geometry can be used. We are able to store the intervals of $I_{\alpha,\beta}$ in a priority search tree (see [13] or [14]), and hence c^2 priority search trees are used. A priority search tree stores a dynamic set of points (x, y) in linear space. It supports 1.5-dimensional range searches with logarithmic query time. A 1.5-dimensional range search is given by a rectangle unbounded to the bottom side, i.e. it asks for all points (x, y) in a semi-infinite strip $x_0 \leqq x \leqq x_1$, $y \leqq y_1$. Moreover, a point can be inserted into (deleted from) a priority search tree in logarthmic running time. Since all polygons are given beforehand, radix priority search trees suit to our purpose.

The scan-line also stops at all vertices of the polygons. When no edge parallels to the scan-line, a vertex v is one of the kinds shown in Figures 4a to 4f. In these figures inside of a polygon is indicated by hatching. Whenever a vertex v is encountered, some intervals cease to be active, and some intervals become active. A vertex in Figure 4a or 4b terminates an interval and initiates an interval. In Figure 4c two intervals are terminated and an interval is initiated, while in Figure 4d an interval is terminated and two intervals are initiated. A vertex of Figure 4e terminates an interval, while that of Figure 4f initiates an interval.

On meeting a vertex we have a difficulty in determining which intervals to be inserted into/deleted from the priority search trees. For example, a vertex of Figure 4a gives us only one endpoint of the intervals to be deleted and inserted. But the intervals in the priority search trees are represented by a pair of endpoints. To overcome this difficulty we use, as a subsidiary data structure, a balanced binary search tree [11, 17], for each active polygon. It is known that the time bounds for access, insertion and deletion operations are O(log n) for balanced search trees, where n is the number of tree nodes. Initially these balanced search trees are empty. If an interval of a polygon P is stored in the priority search tree, it is also maintained in P's search tree. An interval in the search tree is represented by two equations $y = a_1x + b_1$ and $y = a_2x + b_2$ of the edges (supporting lines) forming it. Given x, we can determine the interval exactly. The intervals of a polygon in the scan-line are totally ordered in y coordinate (also see Figure 2).

The determination of the intervals to be inserted into/deleted from our data structures on meeting a vertex v is now straightforward. Let the vertex v belong to polygon P. In Figure 4a, we access P's search tree with the left neighboring edge l, delete the node representing the equations of l and l's partner, and insert a node representing the equation of the right neighboring edge r and that of l's partner. The inserted and deleted intervals in P's search tree are simultaneously processed in the priority search trees. The similar treatment is done when the vertices represented by Figures 4b to 4c are scanned. A vertex v in Figure 4d has no corresponding edges in P's search tree, but it is contained by the sole interval. Hence we first locate the interval containing v in P's search tree, and then combinate the top and bottom edges of the interval with the top neighboring edge t of v and the bottom neighboring edge b of v respectively. A vertex v in Figures 4e and 4f determines the interval in the obvious way.

When there exists an edge parallel to the scan-line, the edge will be of a kind shown in Figures 5a to 5d. (Symmetric cases are omitted.) On meeting an edges represented by Figures 5a to 5d the intervals can be determined in the similar way as above. For example, Figure 5a is treated in the same way as Figure 4e.

Note that in our original problem where each polygon has at most a fixed number of edges, a polygon has at most a constant number of intervals, which makes subsidiary search trees unnecessary, and that if the given polygons are monotonic in some direction, a polygon has only one varying interval during the sweep and subsidiary search trees are also unnecessary.

By now we obtain an algorithm that solves the batched problem. When a query point is met, we query all the priority search trees. The query time for a priority search tree is $O(\log n + t_i)$ time where t_i is the number of answers. Thus our algorithm runs in $O(n \log n + t)$ time and $O(n)$ space, where t is the total number of reported polygons. Taking c into account the time complexity is $O(c^2 n \log n + t)$. Again the space requirements are independent of c. In practical applications, c must be rather small to make our algorithm efficient, for instance, less than 5. These time and space complexities are also the same as those for the orthogonal case [3].

It is noted that Guting [6] has given an optimal algorithm of $O(n \log n)$ time and $O(n)$ space that solves the batched point enclosure counting problem: given a mixed set of query points and c-oriented polygons where the total number of points and edges is n, determine for each point the number of polygons enclosing it. But his method does not solve the batched point enclosure reporting problem.

5. Another Solution with a Single Report

In section 3 and 4 we have given an optimal algorithm for the c-oriented polygon intersection problem. It reports an intersecting pair as soon as it is found, and the same intersecting pair may be reported more than once (but at most a constant number of times). We present here another algorithm that stores each found pair in an vector instead of reporting it instantly, and avoids multiple reports. Unfortunately the algorithm is not space-optimal and requires O(n log n) space.

The new algorithm proceeds as follows: The polygons are first sorted into some order, and then deleted one by one from the sorted sequence. Furthmorte, when polygon p is deleted, we check for intersection of p with the rest of polygons. Obviously this is a one-pass algorithm. If we sort polygons according to their maximum x values in increasing order, and choose the rightmost vertex of a polygon as its representative vertex, the algorithm for the batched point enclosure problem in the previous section will works to report all pairwise polygon enclosures in the desired order.

To report edge intersections as described above, the method presented in section 3 can not be used. A new method is to delay computing edge intersections until a rightmost vertex of a polygon p is encountered. At that time p is deleted from the sequence, and edge intersections of p with the rest polygons are checked. This delay makes the solution of edge intersections independent of the scan-line. In fact we are faced with a special kind of dynamic line segment intersection search problem. As polygons are deleted one by one from the sequence, edge intersection queries are asked in the same order. Fortunately Imai and Asano[10] have studied this kind of problems for the orthogonal case. That is, if the underlying set is updated only by deletions or insertions, an intermixed sequence of O(n) queries and updates can be excuted on-line in O(n log n + t) time and O(n log n) space, where t is the total number of reported intersections. By reducing the c-oriented line segment intersection search problem to several instances of the orthogonal problem (for detials see[8]), it is not difficult to extend Imai and Asano's algorithm to c-oriented objects. We omit the details here.

To report each pair only once, we use a vector V(t) (with t ranging from 1 to n) to keep the intersecting information. Just before the ith polygon is deleted, V[j] ($j \geq i$) holds the maximum number of the polygon which is intersected with the jth polygon and whose number is less than i. That is,
V[j] =

$$\max_k \{\text{ the kth polygon intersets with the jth polygon and } k < i\}.$$

Hence the strategy for processing the ith polygon is as follows: First, if V[i] is not 0, the pair (V[i], i) is output and V[i] is reset to 0. Second, the intersections of the ith polygon with all of the jth (j > i) polygons are checked. After having detected the jth polygon intersecting with the ith polygon, we check the value of V[j], if V[j] is neither 0 nor i (that is, in the range of 1 to i-1) the pair (V[j], j) is output, and then set V[j] to i. In this method each intersecting pair is reported exactly once.

Let us now give a short description of the algorithm:

```
Sort all polygons according to their
  maximum x values in increasing
  order;
Initially the vector V is 0;
FOR  i = 1  To  n  DO
  Delete the ith polygon p from the
    sorted sequence;
  IF V[i] ≠ 0 THEN report the pair
    (V[i], i);  V[i] = 0  FI;
  Check the intersections of  p
    with the rest polygons;
  FOR EACH intersected polygon q DO
    (Let q be the kth polygon)
    IF 0 < V[j] < i-1 THEN
      report the pair (V[j], j) FI;
    V[j] = i;
  OD;
OD;
```

By now we have solved the problem posed in the beginning of this section. Due to the fact that edge intersections must be reported in some designated order, the method presented here fails to obtain optimal space bound. We leave open the question of the existence of a time- and space-optimal algorithm for the c-oriented polygon intersection problem which reports each intersecting pair only once.

6. Conclusions

In this paper we presents an optimal algorithm for the c-oriented polygon intersection problem. As intermediate steps of this algorithm, we also give algorithms for two related problems: the c-oriented version of line segment intersection problem and batched point enclosure problem. To aviod multiple reports occurred in the optimal algorithm, we present another algorithm, but which requires to increase the space bound by a factor of log n.

The solution to the c-oriented polygon intersection problem is particularly important in practice because of its relation to VLSI design rule checking. It may also have applications where a problem involves computing some property of a given set of c-oriented objects. For example, in the translation problem which requires to translate geometrical objects in a given direction, one at a time, without collisions occuring between the objects, we can give efficient algorithms for c-oriented objects in three dimensions based on the methods developed in this paper [16].

The result we have obtained partially answers the challenging open question of whether all t intersections among n polygons can be reported in O(n log n + t) time. To solve this problem, one might use Chazelle and Edelsbrunner's algorithm [4] to find all edge intersections in optimal time O(n log n + k), where k is the number of pairwise edge intersections. The remaining problem is how to find all polygon enclosures in the same time bound.

References

[1] H.S.Baird, Fast algorithms for LSI artwork analysis, J. Des. Autom. Fault-Tolerant comput. , 2(1978), 979-209.
[2] J.L.Bentley and Th.Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. on Comput. C-28(1979), 643-647.
[3] J.L.Bentley and D.Wood, An optimal worst-case algorithm for reporting intersections of rectangles. IEEE trans. on Comput. C-29(1980),571-577.
[4] B.Chazelle and H.Edelsbrunner, An Optimal Algorithm for Intersecting Line Segments in the Plane, Rep. 88-1419, Comput. Sci. Dept., Illinois Univ., 1988.
[5] H.Edelsbrunner, A new approch to rectangle intersections, Part II, Int. J. Comput. Math. 13(1983), 209-219.
[6] R.H.Guting, Stabbing c-oriented polygons, Inform. Process. Lett. 16(1983), 35-40.
[7] R.H.Guting, Dynamic c-oriented polygonal intersection searching, Inform. Control 63 (1984), 143-163.
[8] R.H.Guting and Th. Ottmann, New algorithms for special cases of hidden line elimination problem, Comput. Vision Graphics Image Process. 40(1987), 188-204.
[9] U.Lauther, 4-dimensional binary search trees as a means to speed up associative searches in design rule verification of integrated circuits, J. Des. Autom. Fault-Tolerant comput. , 2(1978) 241-247.
[10] H.Imai and T.Asano, Dynamic segment intersection search with application, in Proceeding 25th IEEE Sympos. Math. Found. Comput. Sci., 1984,393-402.
[11] D.E.Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 2d. ed., Addison-Wesley, Reading, Mass., 1973.
[12] E.M.McCreight, Efficient Algorithms for Enumrating Intersecting Intervals and Rectangles, Xero Palo Alto Res. Cen., Palo Alto, CA, Tech. Rep. PARC CSL-80-9, 1980.
[13] ----, Priority Search Trees, SIAM J. Comput. 14(1985),257-276.
[14] K.Mehlhorn, Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry, Springer-Verlag, Berlin, 1984.
[15] H.-W.Six and D.Wood, The rectangle intersection problem revisited, BIT 20(1980), 426-433.
[16] X.H.Tan, T.Hirata and Y.Inakaki, On translating a set of c-oriented objects in three dimensions, manuscript in preparation, 1989.
[17] R.E. Tarjan, Data Structures and Networks Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1983.

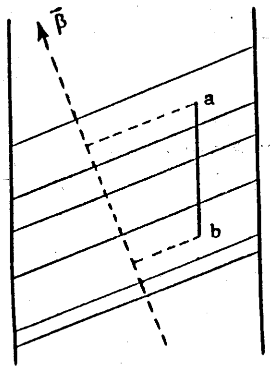

Figure 1

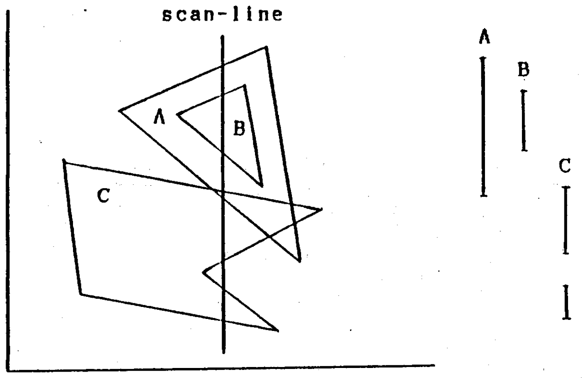

Figure 2

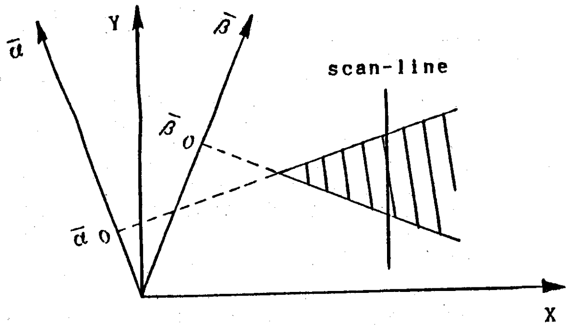

Figure 3

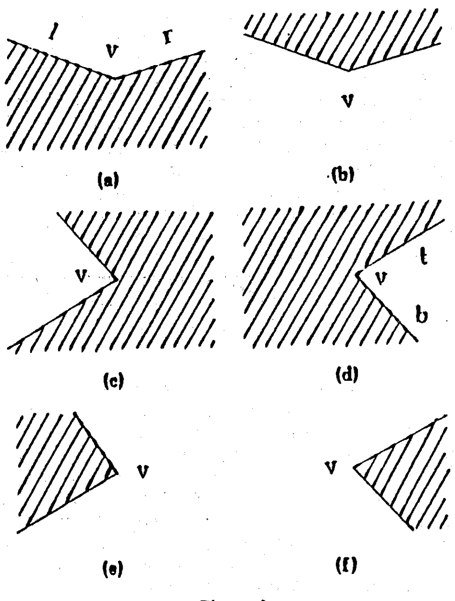

Figure 4

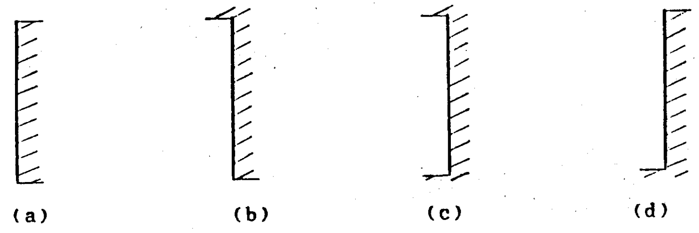

Figure 5