Shattering a Set of Objects in 2D

Subhas C. Nandy*
Indian Statistical Institute,
Calcutta 700 035, India,

Abstract : In this paper, we propose an algorithm for shattering a set of disjoint line segments of arbitrary length and orientation placed arbitrarily on a 2D plane. The time and space complexities of our algorithm are $O(n^2)$ and $O(n)$ respectively. It is an improvement of the $O(n^2 \log n)$ time algorithm proposed in [5]. A minor modification of this algorithm applies when objects are simple polygons, keeping the time and space complexities invariant.

Keywords : Duality, topological line sweep, separator, shattering.

1 Introduction

Given a set S of n non-intersecting line segments of arbitrary length and orientation in the plane, we say that a line ℓ is a *separator* of S if it does not intersect any member in S and partitions S into two non-empty subsets lying in both sides of ℓ. A set of separators L is said to *shatter* S if each line in L is a separator of S and every two line segments in S are separated by at least one line in L. In other words, each cell of the arrangement of the lines in L contain at most one member of S (see Figure 1a for illustration). For a given set S, a set of separators may not always exist which can shatter S (see Figure 1b). In [5], it was proved that the problem of finding a minimum cardinality shatter for S is NP-complete. They also proposed an $O(n^2 \log n)$ time algorithm for getting a feasible solution of the problem, if it exists at all. The same problem in higher dimension is studied in [4]. In 2D, for each member in S, if the ratio of its length and the diameter of the set S is larger than a predefined constant δ, the set of shattering lines for S can be obtained in $O(n \log n)$ time [3]. In this paper, we consider the general case of the problem as in [5], and propose an algorithm which decides the existence of a set of lines shattering S. In case of affirmative answer, it outputs a set of lines shattering S. Our algorithm is based on sweeping a topological line through the arrangement of the duals of S. The time complexity of our algorithm is $O(n^2)$, which improves the $O(n^2 \log n)$ algorithm of [5] in the general case. The space complexity is $O(n)$. We also show that our proposed algorithm can easily be extended if the objects are disjoint polygons keeping the time and space complexities invariant. Possible applications of the shattering problem are mentioned in [3, 5].

*Currently in Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan.

2 Preliminaries

As an initial step, we find whether there exists a set of vertical separators which can shatter S, by sweeping a vertical line on the plane in $O(n \log n)$ time. If such an attempt fails, we need to check whether a set of non-vertical lines exist which can shatter S following the method discussed below.

We use geometric duality for solving this problem. It maps (i) a point $p = (a, b)$ to the line $p^* : y = ax - b$ in the dual plane, and (ii) a non-vertical line $\ell : y = mx - c$ to the point $\ell^* = (m, c)$ in the dual plane. Needless to say, the incidence relation of the primal plane remains preserved in the dual plane also. In other words, p is below (resp., on, above) ℓ if and only if p^* is above (resp., on, below) ℓ^*. The dual of a line segment $s \in S$ is a double wedge s^* formed by the duals of all the points on s. All these lines pass through the dual (point) of the line containing s, and s^* is bounded by a pair of lines which are duals of the end points of s. The area inside the double wedge s^* will be referred to as the *active zone* of s^*. Easy to check, a non-vertical line ℓ stabs s, if and only if the corresponding dual point ℓ^* lies in the active zone of s^*.

Now, if we consider the arrangement of the duals of the members in S, and choose a point ℓ^* in the complementary region of the union of active zones of all the double wedges $\{s_i^* \mid s \in S\}$, its corresponding line ℓ in the primal plane will not stab any of the members in S. Again, if such a point ℓ^* is chosen above the upper envelope or below the lower envelope of $\bigcup_{i=1}^{n} s_i^*$, then all

the members in S will lie in one side of ℓ. Thus in order to get a separator of S, one needs to choose a point in the complement of the union of active zones of these double wedges, but not above or below all of them. The set of all the separators can easily be obtained by constructing the arrangement of the duals of the members in S and then determining for every face of the arrangement whether it is inside the active zone of any one of the double wedges. This can easily be done in $O(n^2)$ time by sweeping a topological line [1] through the arrangement. It is very easy to observe that there are sets of line segments for which the union of active zones of the double wedges has complexity $\Omega(n^2)$. Hence the complexity of the complement regions in the arrangement, and hence the cardinality of the set of all possible separators, may also be $O(n^2)$.

It is easy to observe that, for a given set S of line segments, there may not exist any set of separators which can shatter S (see Figure 1b). An easiest way to check whether the set of all possible separators L, obtained above, shatter S or not, is as follows :

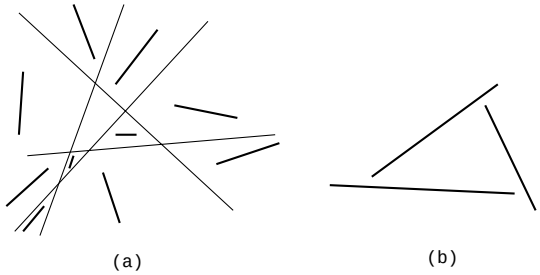

Figure 1: (a) Demonstration of shattering for a set of line segments S, (b) An example where a set of lines shattering S does not exist

Consider the arrangement of the lines in L. As the members of L are the separators of S, each member in S completely lies in one cell of the aforesaid arrangement (see Figure 1a). So, we consider a set of points P that contains one end point of each line segments in S. Now, if any of the cells in the arrangement contains more than one point of P, it indicates that shatter does not exist for S. The time required to locate the cell containing a given point is $O(|L|\log^2(|L|))$ [7]. A randomized algorithm of expected time complexity $O(m_1^{2/3}m_2^{2/3}\log(m_1) + m_1\log(m_1)\log(m_2))$ exists which outputs the cells of an arrangement of m_1 lines that contain a specified set of m_2 points [2]. In our case, $m_1 = |L|$, and $m_2 = n$. As $|L|$ may be $O(n^2)$ in the worst case, the time complexity of the former algorithm may be $O(n^3\log^2 n)$ and that of the later may be $O(n^2\log n)$.

In the following section we propose a simple algorithm using topological plane sweep through the arrangement of the lines defining the wedges corresponding to the line segments in S in the dual plane.

3 Outline of the algorithm

Let G be a n vertex graph whose nodes correspond to the members in S. Initially, the graph is complete, i.e., among each pair of vertices, there is an edge. The edges of the graph will be deleted during the execution of the algorithm. At any instant of time, an edge between a pair of nodes implies that the corresponding line segments are not yet separated by any separator. During the execution of the algorithm, as soon as a separator ℓ is detected which separates S into two disjoint sets S_1 and S_2 in its two sides, we remove all the edges among the members in S_1 and S_2. We refer S_1 and S_2 as two *clusters* separated by ℓ. Subsequently, if another separator ℓ' is located which partitions S into S_3 and S_4 such that the subsets $S_1 \bigcap S_3$, $S_1 \bigcap S_4$, $S_2 \bigcap S_3$ and $S_2 \bigcap S_4$ are not all equal to ϕ, then we don't eliminate edges in G among $S_1 \bigcap S_3$ and $S_2 \bigcap S_4$, and $S_1 \bigcap S_4$ and $S_2 \bigcap S_3$ as they are already deleted in an earlier step. Here, S_1 will split into two disjoint clusters $S_1 \bigcap S_3$ and $S_1 \bigcap S_4$ by eliminating the edges among them in G. Similarly, S_2 will also split into two disjoint clusters $S_2 \bigcap S_3$ and $S_2 \bigcap S_4$. During the entire swap, if all the edges are eliminated from G, the output set of separators shatter S. Next, we need to prove a very important lemma.

Lemma 1 *[5] If S is shatter-able by an arrangement of lines, then S is shatter-able by an arrangement of $n-1$ or fewer lines.*

Proof : If each member of the separators L, when it is detected, separates at least one existing subset of S which is not yet separated, then by the introduction of that separator, the number of (disjoint) clusters will be increased by one. If S is shatter-able, at the end of sweep the number of clusters will be n, and they are formed with at most $n-1$ separator lines. □

We consider the set S^* of double wedges in the dual plane corresponding to the set S of line segments in the primal plane. From now on, the set of lines in S^* will also be referred to as S^*. The members of S^* partition the dual plane into $O(n^2)$ disjoint cells.

Definition 1 A cell is said to have *degree* k if and only if the active zone of k double wedges of S^* overlap on it. A cell of degree zero will be referred to as *zero-degree* cell.

As mentioned earlier, a point ℓ^* inside a zero-degree cell corresponds to a line ℓ in the primal plane which does not stab any member of S. In addition, if that cell does not lie above the upper envelope or below the lower envelope of S^*,

it corresponds to a separator of S. We use topological plane sweep to identify the zero-degree cells in the arrangement of S^*. Needless to mention, as a topological sweep line $\mathcal{L}$ is y-monotone, when such a line encounters the leftmost vertex v of a zero-degree cell C, any point ℓ^* inside that cell separates two subsets S_1^* and S_2^* of double wedges which cross over $\mathcal{L}$ above and below v respectively. In other words, the line ℓ in the primal plane corresponding to ℓ^* separates the sticks corresponding to S_1^* and S_2^*. In such a situation, we shall say that the point ℓ^* (or the cell C) separates the set of double wedges S^* into two disjoint *clusters* S_1^* and S_2^*.

3.1 Data structure

In addition to the standard data structures of sweeping a topological line through the arrangements of a set of lines, as is used in [1], we need to maintain the following data structures during the execution of our algorithm.

list_1 : It is a linear link list whose elements alternately contain the lines in S^* and the cells in the arrangement, intersected by the sweep line $\mathcal{L}$ in its current position, and ordered from top to bottom. As we like to ignore the cells above the upper and below the lower envelopes of S^*, the first and the last element of this list are the two members in S^* that has intersected $\mathcal{L}$ at maximum and minimum y-coordinates.

An element representing a line contains (i) an identifier indicating the corresponding member in S, and (ii) a pointer to the corresponding line in the *cluster* data structure, described below.

An element representing a cell contains its degree.

cluster : It is a list of subsets of S^* partitioned by the zero-degree cells obtained so far. Needless to mention, initially, it contains only one cluster having the entire set S^*, and its identifier is 1. Finally, after considering all the vertices in the arrangement, it contains $n-1$ clusters. Each element of this list contains a *member_list* and a *header* as described below.

member_list : A bidirectional link list containing a cluster, say S_i^*. This indicates that the line segments in the primal plane, corresponding to S_i^*, are not yet separated by any separator, and hence the corresponding nodes in G are still connected to each other. This list is also maintained in a top to bottom order with respect to their appearance on the topological line $\mathcal{L}$. Note that, the upward (resp. downward) link of the top-most (resp. bottom-most) element is set to NULL. In order to reach the cluster header from any element in the cluster in $O(1)$ time, each node is attached with a *head_ptr* which points to the *header* of the corresponding cluster.

header : This contains the following information regarding the cluster.

id : A cluster identifier which is a natural number from $1 \ldots k$, if k clusters are generated up to the current instant of time.

t, b : The top-most and bottom-most members in *member_list*.

t_{ptr}, b_{ptr} : A pair of pointers indicating the lines t and b in the *list_2* data structure as described below.

separator_list : A list of points in the dual plane. Each point corresponds to a separator of S in the primal plane.

We also maintain the following data structure during the execution of the algorithm. We defer to state its usefulness up to the next section. But its management will be discussed in this section along with the management of other data structures.

list_2 : It is also a linear link list similar to *list_1*, but the lines stored in this list are only the top-most and bottom-most lines (indicated by t and b fields) of all the clusters which are recognized till the current instant of time. As the top-most and/or the bottom-most line of a cluster may change after encountering a vertex of the arrangement, the members in this list sometimes change during the execution. When a new cluster is generated, two new lines are added in this list.

3.2 Algorithm

We shall follow the algorithm of sweeping a topological line $\mathcal{L}$ through the arrangement of lines in S^* as described in [1]. During the execution, let v be the new vertex encountered by $\mathcal{L}$, which is generated due to the intersection of two consecutive lines, say ℓ_1 and ℓ_2, in *list_1*. We now need to take the following actions :

Step 1 : ℓ_1 and ℓ_2 need to be swapped in *list_1*. A cell will die out and a new cell will be generated. Note that, the vertex v may be of two types depending on whether it is generated due to the intersection of two lines corresponding to the end points of the same line segment in S or of two different line segments. In the former case, the degree of the new cell will remain same as that of the dying cell (see Figure 2a). In the later case, the degree of the new cell needs to be determined observing the sides of ℓ_1 and ℓ_2 containing the active zones (see Figure 2b, 2c and 2d).

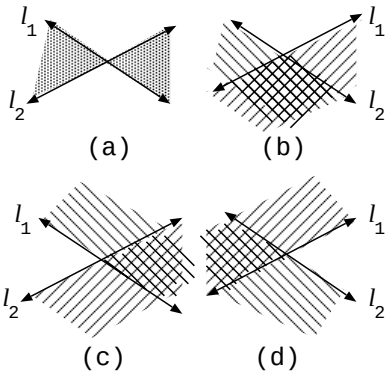

Figure 2: Degree computation for a new cell

Step 2 : If ℓ_1 and ℓ_2 belong to the same cluster, they must be consecutive in the *member_list* of that cluster. Here, the following actions need to be taken :

- **2.1** They need to be swapped in the *member_list* of *cluster* data structure.
- **2.2** If one of ℓ_1 or ℓ_2 is either the top line or the bottom line of that cluster, t or b field (along with the pointers t_{ptr} or b_{ptr}, as the case may be) of that cluster need to be changed. It can easily be checked by comparing ℓ_1 and ℓ_2 with the existing t and b fields of the cluster.
- **2.3** Easy to understand, if the t or b field of a cluster changes, the corresponding change in *list_2* is also necessary.

Step 3 : If ℓ_1 and ℓ_2 belong to different clusters, then apart from the line swap no other change in the *cluster* data structure is necessary. But if any one of them is either the top or the bottom line of the respective cluster, then a line swap may be required in *list_2* also. This can be checked consulting the t and b of the corresponding clusters. The corresponding lines in *list_2* can be reached in $O(1)$ time using the pointers t_{ptr} and b_{ptr}.

Step 4 : If the degree of the new cell, observed in Step 1, is zero, any point inside this cell is a separator for S. In order to check whether this separator separates at least one of the existing clusters, we need to execute the following sub-steps.

- **4.1** We traverse the *cluster* list to inspect all the clusters so far obtained. If the generated cell is within the lines indicated by the t and b fields of a cluster, it needs to be partitioned by that separator.
- **4.2** If a cluster is observed to be split, we visit the *member_list* from top to bottom to find a pair of lines ℓ_i and ℓ_j within which the currently generated zero-degree cell lies. The former cluster is shortened by deleting the link between ℓ_i and ℓ_j. A new cluster is formed with the lines below and including ℓ_j. The t and b fields of both the clusters are appropriately set.
- **4.3** Both the lists of the newly formed pair of clusters are visited to remove edges in the graph G.
- **4.4** The top line (represented by t field) of one of the newly generated clusters and the bottom line (represented by b field) of the other one are inserted in *list_2*.

Step 5 : If at least one of the existing clusters split, we introduce a separator by adding a representative point of the current cell in the *separator_list*. Otherwise, we do not introduce any separator for the current cell.

Lemma 2 *Each edge of the graph G will either be accessed exactly four times or never during the entire execution of the algorithm.*

Proof : While processing a cell, if its degree is greater than zero, the question of accessing the graph G does not arise. If the degree of a cell is zero, *we investigate each cluster separately.* So, an already deleted edge between a pair of vertices which lie in different clusters, will not be checked. Let a cluster S_i^* splits into S_{i1}^* and S_{i2}^*. Note that each pair of nodes of S_i^* are still attached in G. During its split, only the edges whose one vertex lies in S_{i1}^* and the other vertex is in S_{i2}^*, will be accessed. So, an edge which is not a candidate for deletion while processing a cell, will not be accessed during the processing of that cell. This proves the second part of the lemma.

Now, if a pair of wedges, say s_α^* and $s_\beta^* \in S_i^*$ appear in different sides of the zero-degree cell, both the edges of s_α^* will be checked with both the edges of s_β^*. So, the edge between s_α and s_β will be accessed exactly four times. Hence the proof of the first part. □

Theorem 1 *The algorithm stated above decides the decision problem — whether S is shatter-able or not.*

Proof : Suppose there exists a shatter, but our algorithm could not remove all the edges of the graph G. In other words, at least one cluster of size greater than or equal to 2 remains. As all the separators in that shatter must correspond to some zero-degree cell of the arrangement, and our algorithm visits all the cells of the arrangement, the aforesaid clusters must split when our algorithm encountered those cells during the sweep. Hence a contradiction. □

Lemma 3 *The time complexity of the above algorithm is $O(n^3)$ in the worst case.*

Proof : The topological line sweep requires $O(n^2)$ time [1], and it gives birth to $O(n^2)$ cells.

Now, we need to consider the time complexity of processing each cell. As soon as a new cell is reached after encountering a vertex by the topological sweep line, the algorithm consumes a constant amount of time for updating a constant number of links in *list_1*, *cluster*, and *list_2*, and for adjusting the degree of the newly generated cell. If the newly encountered cell is of non-zero degree, no additional task needs to be performed. However for a zero-degree cell, it needs to check all the existing clusters to explore the possibility of their split. If there exists some favorable cluster(s), the total time required to split all of them is proportional to the total size of the split-able clusters, which may be $O(n)$ in the worst case. Again, as the splitting takes place at most $n-1$ times (by Lemma 1), the total time required for the splitting of clusters during the entire execution is $O(n^2)$ in the worst case. But, we need to spend $O(k)$ time for visiting all the k clusters present in *cluster* data structure in order to find the split-able clusters as soon as we encounter a zero-degree cell. The lemma follows from the fact that k may be $O(n)$, and the number of zero-degree cells may be $O(n^2)$ in the worst case. □

Lemma 4 *The space complexity of our algorithm is $O(n)$.*

Proof : The space required for maintaining the required data structure for topological sweep is $O(n)$ in the worst case [1]. It is also easy to understand, *list_1*, *list_2* and *separator_list* require $O(n)$ space. As the clusters are disjoint, the space required to store the link lists of all the clusters is also $O(n)$. Only $O(n^2)$ space is required to store the graph G. But, by Lemma 2, an edge of G is accessed four times if it is deleted, otherwise, it is not accessed at all. So, in place of G, we may keep a counter *edge_count*, initially set to 0; as soon as an edge of G needs to be accessed, we increment *edge_count* by 1. At the end of the execution if the value of *edge_count* is equal to $2n(n-1)$, it implies all the edges of G is deleted, and the set S of line segments is shatter-able. □

4 A further refinement

In the earlier section we observed that, during the processing of a zero-degree cell, an $O(n)$ time may be required in the worst case to locate the split-able clusters, irrespective of whether it detects such a cluster or not. We may hope a better time complexity if we can avoid the checking when no split-able cluster exists for a zero-degree cell.

Now, we explain the use of *list_2* in order avoid the above mentioned checking. It is already mentioned that *list_2* maintains the overlapping information among the clusters. The lines stored in it correspond to the upper and lower lines of all the clusters, explored so far, and degree of an cell on *list_2* implies the number of clusters overlapped on that cell. We use Δ to refer the degree of an cell generated on *list_2*.

Lemma 5 *If the topological sweep line enters in a zero-degree cell after encountering a vertex generated by*

(a) *the intersection of a pair of lines of the same cluster, then at least one cluster is sure to split.*

(b) *the intersection of a pair of lines of different clusters, then there may or may not exist cluster(s) which split. It depends on whether the Δ parameter of the corresponding cell on list_2 is non-zero or zero.*

Proof : The proof of the part (a) is obvious. The proof of part (b) follows from that fact that, as the vertex is obtained by the intersection of a pair of lines of different clusters, those two clusters may not split if the participating lines are top most or the bottom most line of the corresponding clusters. In that case, the Δ parameter of the current cell will be 1 less than its neighboring cells as shown in Figure 3a. Otherwise these clusters must split, and the Δ parameter of the current cell will be same as that of its neighboring cells (see Figure 3b). In the former case also, if Δ of the current cell is non-zero, some other cluster must overlap on the current cell, and that gets split by a representative point inside that cell. But if Δ of the newly generated cell becomes zero due to the current swap, no cluster splits. □

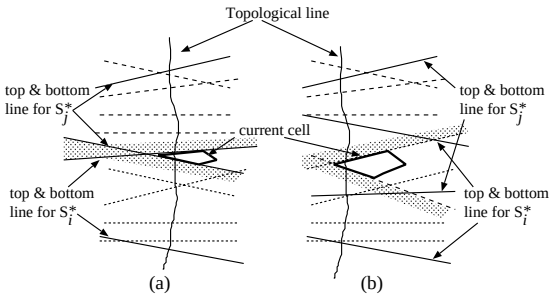

Figure 3: Proof of the (b) part of Lemma 5

Lemma 5 tells that, while processing a zero-degree cell, we can understand whether any of the clusters splits or not, by observing the Δ parameter of that cell in *list_2*. It needs to mention that, the cells stored on the *list_2* are not the same as that of the *list_1*, as the former list stores only $2k$ lines if k clusters are formed till now. But, if a cell with $\Delta = 0$ is reached, the corresponding intersecting lines are top-most and bottom-most lines of two clusters. So they are present in the *list_2*. These two lines need to be swapped and the Δ parameter of the new cell on *list_2* is to be appropriately adjusted. So, it only

remains to explain how we can reach the lines corresponding to the *list_2*, (if they are at all present) when a pair of lines on the *list_1* swap.

When a pair of lines in *list_1* swaps, the corresponding lines in *cluster* data structure are reached by using the pointer maintained with it in *list_1*. It can be checked very easily whether those lines are the top most line or the bottom most line of the corresponding clusters by observing the t and b fields stored in the cluster headers, and which is reachable using *head_ptr*. Now, if any of them is an extreme line of a cluster, it is also present in *list_2*. In order to update *list_2* we reach those lines in *list_2* using t_{ptr} or b_{ptr}, stored in the cluster header.

Lemma 6 *If the search in the cluster data structure is performed only when there exists at least one split-able cluster, the total time complexity reduces to* $O(n^2)$.

Proof : The proof follows from the following three points :

- The search for a split-table cluster requires $O(k)$ time when there are k clusters in the *cluster* data structure.
- Introduction of a new separator increments the number of clusters by at least one.
- At most $n-1$ separators may exist in a shatter (by Lemma 1). □

We are now in a position to state the complexity results of our algorithm.

Theorem 2 *The time and space complexities of our proposed algorithm are* $O(n^2)$ *and* $O(n)$ *respectively.*

Proof : Follows from Lemma 4 and 6. □

5 Shattering of arbitrary polygons

In this section, we describe how our algorithm can be tailored if the set S of objects are arbitrary simple polygons. A pair of polygons can be separated by a line if and only if their convex hulls are non-overlapping. So, as a first step of this problem, we need to compute the convex hulls of all the polygons, which need $O(n)$ time [6] if n be the total number of vertices on all the m polygons placed on the floor. Now our problem boils down to deciding whether shatter exists for the convex hulls of those polygons. From now onwards, S will denote the set of convex polygons obtained above. In $O(n\log n)$ time we can check whether any pair of S overlaps or not by sweeping a vertical line from left to right. This also finds whether a set of vertical lines exists which can shatter S. Below we explain the method of checking the existence of a set of non-vertical lines shattering S. This method will be invoked if and only if the members in S are non-intersecting and a set of vertical lines shattering S do not exist.

Here also we shall work with the duals of the convex polygons in S. The dual of a convex polygon $s_i \in S$ is a set of points whose corresponding lines in the primal plane stabs s_i. As in the case of line segments, here also we denote the dual region of s_i as its *active region*, and it is bounded by a piecewise linear *curve* obtained by the lower and upper envelopes of the dual lines corresponding to the vertices of s_i. In Figure 4, we demonstrate the dual of a convex polygon. The duals of all the m convex polygons can be obtained in $O(n\log n)$ time [7].

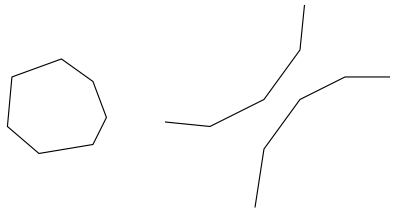

Figure 4: A convex polygon and its dual

Except for the degenerate cases for a convex polygon with empty interior, i.e., for a line segment, the boundaries of the *active region* of a member in s_i do not intersect. The boundary of the dual of two different objects s_i and s_j may intersect in at most two points. So, the complexity of the arrangement of the boundaries of the duals of all the members in S is also $O(n^2)$ in the worst case [7].

As in the earlier problem, here also we sweep a topological line in the arrangement of the duals of the members in S. But after processing a vertex of the arrangement, when we generate the next vertex, we may need to follow the chain of line segments along that *curve*. Moreover, we may have to consider the end points of the line segments on the boundaries of the *active regions* as the event points in addition to the vertices of the arrangement. If an event point of the former type is reached, we need to change the corresponding line description in each of *list_1*, *cluster* and *list_2* data structures, where the line preceding to that point appears. The other steps of the algorithm remain same. As the topological sweep can be executed in $O(m^2+n)$ step, the time complexity of the algorithm for polygonal objects is $O(m^2 + n\log n)$ in the worst case.

References

[1] H. Edelsbrunner and L. Guibas, *Topological sweeping an arrangement*, Journal of Computer and Systems Sciences, 38 (1989), 165-194.

[2] H. Edelsbrunner, L. Guibas and M. Sharir, *The complexity and construction of many faces in arrangements of lines and of segments*, Discrete Computational Geometry, 5 (1990), 161-196.

[3] A. Efrat and O. Schwarzkopf, *Separating and shattering long line segments*, Information Processing Letters, 64 (1997), 309-314.

[4] R. Freimer, *Shattering configurations of points with hyper-plane*, Canadian Conference on Computational Geometry, 1991, pp. 220-223.

[5] R. Freimer, J. S. B. Mitchell and C. D. Piatko, *On the complexity of shattering using arrangements*, Canadian Conference on Computational Geometry, 1990, pp. 218-222.

[6] D. T. Lee, *On finding the convex hull of a simple polygon*,
, vol. 12(2), pp. 87-
98, 1983.

[7] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequence and Their Geometric Applications*, Cambridge University Press, 1995.