Shattering a Set of Objects in 2D

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 over the $O(n^2 \log n)$ time algorithm proposed in [7]. 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 on 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 pair of 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 contains 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 [6, 7], an $O(n^2 \log n)$ time algorithm is proposed for reporting the existence of a shatter. Of course, the problem of finding a minimum cardinality shatter for S is NP-complete [7]. The same problem in higher dimension is studied in [5]. 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 [4]. In this paper, we consider the general case of the problem as in [7], and propose an algorithm which decides the existence of a set of lines shattering S. In case of an 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 the members in S. The time complexity of our algorithm is $O(n^2)$, which is an improvement over the $O(n^2 \log n)$ algorithm of [7] in the general case. The space complexity of our algorithm is $O(n)$. The general version of the shattering problem for a set of disjoint line segments is shown to belong to the class of so called *three-sum hard* problem [8] by an $O(n \log n)$ time reduction from GEOMBASE problem [6] which is stated as follows: *given n points on three horizontal lines $y = 0$, $y = 1$ and $y = 2$ in R^2, does there exist a non-horizontal line containing three of the points ?* Thus the time complexity of our proposed algorithm is optimum in the sense that the existence of an algorithm for this problem with time complexity better than $\Omega(n^2)$ seems to be impossible [8]. We also show that our proposed algorithm can easily be extended to shattering of disjoint polygons keeping the time and space complexities invariant. Possible applications of the shattering problem are mentioned in [4, 6, 7].

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.

Without loss of generality, we may assume that all the line segments in S are non-vertical, and we shall 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 ℓ: $y = mx - c$ to the point $\ell^* = (m, c)$ in the dual plane. The incidence relation of the primal plane is preserved in the dual plane also. In other words, p is below, on or above ℓ if and only if p^* is above, on or below ℓ^* respectively. The dual of a non-vertical line segment $s \in S$ is a double wedge s^* formed by the union of 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^*. Obviously, a non-vertical line ℓ stabs s if and only if the corresponding dual point ℓ^* lies in the active zone of s^*.

Let us 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 $\{s_i^* \mid s_i \in S\}$. The line ℓ corresponding to 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 on one side of ℓ. Thus, the set of all possible separators can be obtained as follows: construct the arrangement of the duals of the members in S; then for each face of the arrangement, test whether it is in the complement of the union of active zones of all the double wedges. In [1], it is shown that for a set of n fat wedges (i.e., if the acute interior angle of the wedge is bounded from below by a constant α) the combinatorial complexity of their union is linear in n. But, for a set of arbitrary line segments, we cannot assure such a property. In fact, there may exist a set of line segments such that the combinatorial complexity of the union of the active zones of their corresponding double wedges in the dual plane is $\Omega(n^2)$. Hence, the complexity of the complement regions in the arrangement will be $\Omega(n^2)$, and the same will be the cardinality of the set of all possible separators.

It is easy to observe that, for a given set S of line segments, there may not exist a set of separators which can shatter S. For example, see Figure 1b where not even a single separator for the given set of line segments exists. An easy way to check whether the set of all possible separators L, obtained above, shatter S or not, is as follows:

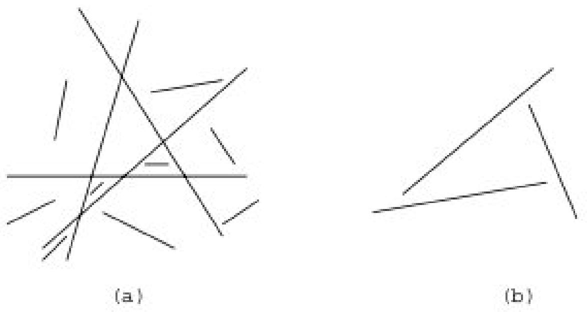


Figure 1: (a) Demonstration of shattering for a set of line segments S, (b) An example where a set of lines shattering S does not exist

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 said arrangement of L (see Figure 1a). So, we consider a set of points P that contains one end point of each of the line segments in S. If any of the cells in the arrangement contains more than one point of P, it indicates the non-existence of a shatter for S. The time required to locate the cell containing a given point is $O(|L|\log^2(|L|))$ [11] which is $O(n^2\log^2 n)$ as $|L|$ may be $O(n^2)$ in the worst case. Thus the overall time complexity is $O(n^3\log^2 n)$ since we need to check for all the elements in P. 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 [3]. In our case, $m_1 = |L|$, and $m_2 = n$. So, the expected time complexity of this method is also $O(n^2\log^2 n)$.

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

3 Outline of the algorithm

Let S be a set of n non-intersecting line segments on a plane. Initially, the members in S are not separated by any separator. During the execution of the algorithm, as soon as a separator ℓ is detected, S is split into two disjoint subsets S_1 and S_2. 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 empty, then S_1 is split into at most two disjoint subsets, namely $S_1 \bigcap S_3$ and $S_1 \bigcap S_4$, and S_2 is split into at most two disjoint subsets, namely $S_2 \bigcap S_3$ and $S_2 \bigcap S_4$. The process continues till a shatter is found for S if it at all exists, or the non-existence of the shatter is detected.

Lemma 1 *[7] If S is shatterable, then at most $n-1$ lines is sufficient to shatter S.*

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^*. We shall denote the arrangement of the members in S^* by $\mathcal{A}(S^*)$. The number of vertices, edges and faces in $\mathcal{A}(S^*)$ are all $O(n^2)$. From now onwards, a face in the arrangement $\mathcal{A}(S^*)$ will be referred as a *cell*.

Definition 1 A cell in $\mathcal{A}(S^*)$ is said to have *degree* δ if and only if the active zones of δ double wedges of S^* overlap on it. A cell of degree zero will be referred as a *zero-degree* cell.

We use topological line sweep technique [2] for identifying the zero-degree cells in $\mathcal{A}(S^*)$. A topological sweep line $\mathcal{L}$ is y-monotone; when $\mathcal{L}$ encounters a zero-degree cell C, any point ℓ^* inside C separates S^* into two subsets S_1^* and S_2^* of double wedges which lie above and below ℓ^* respectively. For each element $s_i \in S$, the dual lines of both the end points of s_i will either belong to S_1^* or S_2^*. Here, the line ℓ in the primal plane corresponding to ℓ^* separates the set of line segments S into two subsets corresponding to S_1^* and S_2^* respectively, of the dual plane.

3.1 Data structure

The input to our algorithm is an array containing the set of non-intersecting line segments S. We use the standard data structures for sweeping a topological line through the arrangement of a set of lines as described in [2]. In addition, 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 $\mathcal{A}(S^*)$, intersected by the sweep line $\mathcal{L}$ in its current position, and ordered from top to bottom. To ignore the cells above the upper envelope and below the lower envelope of S^*, the first and the last elements of this list are the two members in S^* that intersect $\mathcal{L}$ 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, called *self_ptr*, indicating its own occurrence in the *cluster* data structure, which is 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. Initially, it contains only one cluster having the entire set S^*, and its identifier is 1. As soon as an old cluster splits into two new clusters, one of them will carry the identifier of the previous cluster and the other one is assigned a new identifier. Finally, after considering all the cells in the arrangement, it contains at most n clusters. An element representing a cluster S_i^* contains a *member_list* and a *header* as described below.

member_list: A bidirectional link list containing the lines representing the double wedges of the cluster S_i^*. The lines in S_i^* are stored in this list in a top to bottom order with respect to their appearance on the topological line $\mathcal{L}$. In order to reach the cluster header from any element in this list 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 have been generated so far.

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

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

3.2 Algorithm

We shall follow the algorithm of sweeping a topological line $\mathcal{L}$ through the arrangement $\mathcal{A}(S^*)$ as described in [2]. During the execution, let v be the new vertex (generated by the intersection of two consecutive lines, say ℓ_1 and ℓ_2, in *list_1*) encountered by $\mathcal{L}$.

We now need to take the following actions:

Step 1: ℓ_1 and ℓ_2 need to be swapped in *list_1*. The sweep line leaves the cell to the left of v and enters the cell to the right of v. 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 S. In the former case, the degree of the new cell will remain the same as that of the previous cell (see Figure 2a). In the latter 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). In Figure 2, the expression within paranthesis in a cell indicates the degree of that cell.

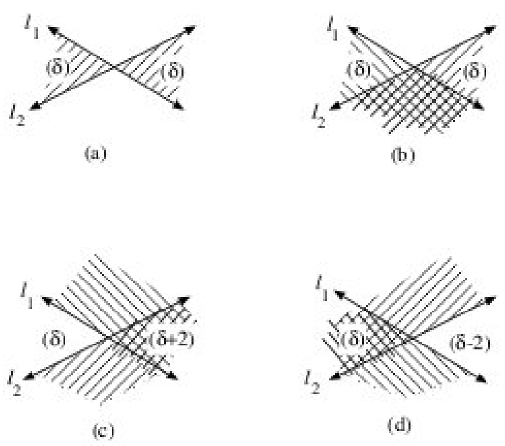


Figure 2: Degree computation for a new cell

Step 2: We use *self_ptr* attached to ℓ_1 and ℓ_2 in *list_1* to reach their own occurrences in *cluster* data structure.

If ℓ_1 and ℓ_2 belong to different clusters, no action needs to be taken in this step.

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 of that cluster needs 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.

Step 3: 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 splits at least one of the existing clusters, we need to execute the following sub-steps:

3.1: We traverse the *cluster* list to inspect all the clusters obtained so far. If the generated cell is within the lines indicated by the t and b fields of a cluster, that cluster needs to be partitioned by the separator corresponding to that cell.

3.2: 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 in the *member_list* of that cluster. A new cluster is formed whose *member_list* contains all the lines below and including ℓ_j. If k clusters are present prior to the split of the current cluster, then the identifier of the new cluster will be $k+1$. The *head_ptr* fields of all the members in the newly formed cluster will now point to the header of that cluster. Finally, the t and b fields of both the clusters are appropriately set.

Step 4: If at least one of the existing clusters split, we insert a new separator (i.e., a representative point of the current cell) in the *separator_list*. Otherwise, we do not introduce any separator for the current cell.

Finally at the end of entire sweep, if the number of clusters is observed to be n, the shatter exists for S (by Lemma 1), and the set of separators shattering S is obtained from the *separator_list*. The proof of correctness of our algorithm follows from the following theorem.

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

Proof: Suppose there exists a shatter, but at the end of the execution of our algorithm at least one cluster exists which has two or more members in S^*. Since all the separators in that shatter must correspond to some zero-degree cell of $\mathcal{A}(S^*)$, and our algorithm visits all the cells of $\mathcal{A}(S^*)$, the aforesaid cluster must split when our algorithm encountered that cell during the topological sweep. Hence a contradiction. $\Box$

Lemma 2 *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 [2]. The *list_1* and *separator_list* data structures require $O(n)$ space. As the clusters are disjoint, the space required to store the *member_lists* for all the clusters is also $O(n)$. $\Box$

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 [2], and it traverses all the $O(n^2)$ cells in $\mathcal{A}(S^*)$. Now, we need to consider the time complexity of processing each cell. As the topological sweep line crosses a vertex and enters a new cell, Step 1 consumes $O(1)$ time for swapping two lines in *list_1* and adjusting the degree of the newly encountered cell; in Step 2, updating a constant number of links in *list_1* and *cluster* data structure also require $O(1)$ time. The time complexity of the algorithm depends on the total execution time of Step 3 for all the zero-degree cells in $\mathcal{A}(S^*)$. For each zero-degree cell, Step 3.1 requires $O(k)$ time to check the t and b fields of all the k clusters present in the cluster data structure to explore the possibility of their split. If Step 3.1 returns at least one splittable cluster, Step 3.2 takes $O(n)$ time in the worst case for splitting all those clusters. By Lemma 1, Step 3.2 is invoked $n-1$ times; so the total time required for the splitting of clusters during the entire execution is $O(n^2)$ in the worst case. The lemma follows from the fact that the number of clusters (k) may be $O(n)$ at any instant of time, and the number of zero-degree cells may be $O(n^2)$ in the worst case. $\Box$

4 A further refinement

In the earlier section we observed that, during the processing of a zero-degree cell, $O(n)$ time may be required in the worst case to locate the splittable clusters, irrespective of whether such a cluster is detected. A better time complexity can be achieved if we can avoid the checking of the existence of a splittable cluster for all the zero-degree cells.

As mentioned earlier, a *cluster* is represented on a sweep line by its top-most and bottom-most lines (indicated by t and b fields). A data structure maintaining the overlapping information among the clusters will be helpful in avoiding the above-mentioned checking. Below, we introduce the data structure *list_2*, and a few modifications in the existing data structures for the said purpose.

list_2: It is a linear link list similar to *list_1*; the lines stored in this list are only the top-most and bottom-most lines of each of the clusters recognized so far. Two consecutive lines stored in *list_2* define a cell. The degree of a cell in *list_2* implies the number of clusters overlapped on that cell, and is denoted by Δ. As the top-most and/or the bottom-most lines of a cluster may change after encountering a vertex of $\mathcal{A}(S^*)$, the members in this list sometimes change during the execution as described in Step 2.2 of the algorithm of Section 3.2. When a new cluster is generated, two new lines are added in this list.

In addition, we need the following modifications in the existing data structures:

(i) With each element of *list_1* we attach a single character field which may contain t or b if that line is a top line or a bottom line of any cluster recognized so far, otherwise it contains 0. The role of this field is as follows: we can prepare *list_2* data structure at any position of the sweep line $\mathcal{L}$ by traversing *list_1* and considering only those lines which are marked as t and b in $O(n)$ time.

(ii) Each line in the *member_lists* of the *cluster* data structure will have a pointer, called *list1_ptr*, which points to its own occurrence in *list_1*.

(iii) A pair of pointers (t_{ptr}, b_{ptr}) is attached with the *header* of each cluster. These two pointers indicate the lines corresponding to t and b in the *list_2* data structure.

The following lemma describes the role of *list_2* data structure in deciding whether or not a newly encountered zero-degree cell splits at least one of the existing clusters.

Lemma 4 *If the topological sweep line enters 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 be split.*

(b) *the intersection of a pair of lines of different clusters, then the existence of cluster(s) which will be split depends on the value of the Δ parameter of the corresponding cell on list_2 to be non-zero or zero.*

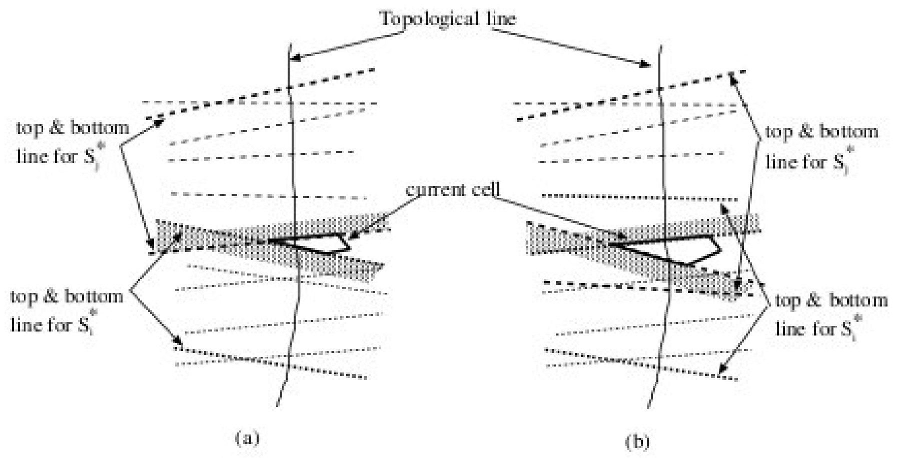
Proof: The proof of the part (a) is obvious. In order to prove part (b), we need to consider the following cases which arise when the vertex is obtained by the intersection of a pair of lines of different clusters:

Case 1: If the lines incident to the vertex corresponding to a zero-degree cell are top most and the bottom most lines of their corresponding clusters, both of them are present in *list_2*. In such a case, an old cell in *list_2* is replaced by a new cell.

Case 1.1: Now, if the Δ parameter of the new cell in *list_2* is 0 (as shown in Figure 3a) then no cluster splits.

Case 1.2: But if the Δ parameter of the new cell in *list_2* is non-zero, the number of clusters overlapping in the current cell is Δ, and all of them will split by a representative point inside the new cell.

Case 2: If either one or none of the participating lines is present in *list_2*, then at least one of the existing clusters overlap on the observed zero-degree cell on *list_1*. These clusters are sure to be split.


Topological line
top & bottom line for S_j^*
current cell
top & bottom line for S_j^*
top & bottom line for S_i^*
top & bottom line for S_i^*
(a)
(b)

list_2 data structure - Lines in the two clusters are differently shaded

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

Lemma 4 tells that, while processing a zero-degree cell of *list_1*, two cases may arise - (i) either one or none of the participating lines are present in *list_2*, and (ii) both the participating lines are present in *list_2*. In the former case at least one of the existing clusters is sure to be split. In the latter case, we must reach the corresponding cell in *list_2* as described in the proof of Lemma 5. The Δ parameter of that cell determines the splitting criterion of any cluster. It is already mentioned in the proof of Lemma 3 that in $O(1)$ time a non-zero degree cell of $\mathcal{A}(S^*)$ can be processed. The following two lemmas prove that if a zero-degree cell does not split any existing cluster, its processing time is $O(1)$; but if it splits at least one of the existing clusters, then its processing may require $O(n)$ time in the worst case.

Lemma 5 *While processing a zero-degree cell in the arrangement of S^*, an $O(1)$ time is enough to check whether any splittable cluster exists.*

Proof: We note the lines participating in a zero-degree from the *list_1* data structure. The same lines in *cluster* data structure are reached by using the *self_ptr* pointers attached to them in *list_1*. Using the *head_ptr* of those two lines in *cluster* data structure, we can reach the *header* of the corresponding clusters in $O(1)$ time.

If the *id* field of both of them are same, i.e., both the lines belong to the same cluster, then by Lemma 4(a) at least one cluster splits.

If the *id* field of these two clusters are different, then in $O(1)$ time we can check whether those lines are the top most line or the bottom-most line of their corresponding clusters by observing the t and b fields stored in the respective cluster headers.

(i) if both of them are the top-most and the bottom-most lines of their corresponding clusters, they are present in the *list_2* data structure, and they can be reached in *list_2* using the pointers t_{ptr} and b_{ptr} stored in the header of those clusters. We swap those two lines in *list_2* and adjust the Δ field of the new cell in *list_2*. This requires $O(1)$ time.

(ii) If the Δ parameter of the new cell in *list_2* is observed to be zero, no cluster will be split by Case 1.1 of Lemma 4(b).

(iii) Otherwise, at least one cluster is sure to be split (see Case 1.2 and Case 2 of Lemma 4(b)).

Lemma 6 *If a zero degree cell causes a split of at least one cluster, then the processing of that cell can be done in $O(n)$ time.*

Proof: The proof follows from the following four points:

- If k clusters are present in the *cluster* data structure, we spend $O(k)$ time to check their t and b fields to recognize the splittable clusters.
- In order to split those clusters, we traverse the *member_list* of all the splittable clusters as described in the step 3.2 of the algorithm of Section 3.2. This requires $O(n)$ time in the worst case.
- The newly introduced top- and bottom-lines are marked in the *list_1* data structure using the *list1_ptr* attached to those lines in *member_list* data structure.
- Finally, we rebuild *list_2* for the new position of the topological sweep line by traversing *list_1* in $O(n)$ time.

Lemma 7 *If the search in the cluster data structure is performed only when there exists at least one splittable cluster, the total time complexity is $O(n^2)$.*

Proof: The proof follows from the fact that (i) at most $n-1$ separators may exist in a shatter and for each of them the number of clusters has increased by at least one (Lemma 1), and (ii) $O(n)$ time search is required for a separator if splittable cluster(s) exists for that separator (Lemma 6).

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: The time complexity result follows from Lemma 7. The space complexity result follows from Lemma 2 and the fact that the size of the newly introduced *list_2* data structure may be $O(n)$ in the worst case. $\square$

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 takes $O(n)$ time [9], if n be the total number of vertices of 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 on, S will denote the set of convex hulls obtained above. In $O(n\log n)$ time we can check whether any pair of S overlap by sweeping a vertical line from left to right [10]. 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-overlapping 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, we refer the dual region of s_i as its *active region*, and it is bounded by two piecewise linear *curves* obtained respectively 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 dual of a convex polygon with k vertices can be computed in $O(k)$ time as follows:

Let $\{a_1, a_2, \ldots, a_k\}$ be the sequence of vertices of the upper chain of a convex polygon in a left to right (clockwise) order. By the property of the duality transform, the dual of these points, say $\{a_1^*, a_2^*, \ldots, a_k^*\}$, will appear in the lower envelope of the dual of the vertices of this polygon in a right to left order. The dual of the lower chain of a convex polygon can be obtained in a similar manner.

Thus, if n be the total number of vertices in all the polygons in S then the duals of all the members in S can be obtained in $O(n)$ time.

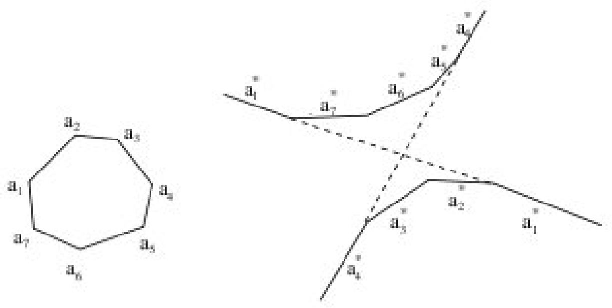


Figure 4: A convex polygon and its dual

Note that the dual line of a vertex of a convex polygon $s_i \in S$ can appear at most twice on the boundary of the *active zone* of s_i. Again, since the members of S are disjoint, the lines participating in the dual of one polygon are different from that of any other polygon in S. Thus, the number of line segments participating in the arrangement of the duals of m polygons may be $O(n)$ in the worst case, and the complexity of the union of *active zones* in the dual plane of all the members in S is also $O(n^2)$ in the worst case.

As in the earlier problem, we sweep a topological line in the arrangement of the duals of the members in S. Here we need to consider two types of event points: (i) the vertices on the boundaries of the active zones, (ii) vertices generated by the intersection of the duals of a pair of members in S. When the sweep line encounters a vertex of type (i), the line segment preceding that vertex will be replaced by the line segment following that vertex in each of *list_1*, *list_2* and *cluster* data structures, and the intersection of it with its two neighboring members in *list_1* (if exists) is computed to find a vertex of type (ii). When a vertex of type (ii) is faced by the sweep line, the actions are exactly same as described in the earlier problem. As the total number of vertices of both type (i) and type (ii) is $O(n^2)$ in the worst case, the worst case time complexity of our algorithm for polygonal objects remains $O(n^2)$.

Acknowledgment: We greatly acknowledge the two anonymous referees and Mr. Arijit Bishnu for their comments which have improved the presentation of the paper.

References

[1] A. Efrat, G. Rote and M. Sharir. *On the union of fat wedges and separating a collection of segments by a line*, Computational Geometry: Theory and Applications, vol. 3, pp. 277-288, 1993.

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

[3] 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.

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

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

[6] R. Freimer, *Investigations in geometric subdivisions: linear shattering and cartographic map coloring*. Tech Report No. TR2000-1784, Dept. of Computer Science, Cornell University, February 2000.

[7] 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.

[8] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Computational Geometry: Theory and Applications, vol. 5, pp. 165-185, 1995.

[9] D. T. Lee, *On finding the convex hull of a simple polygon*, Int'l Journal on Computer and Information Science, vol. 12(2), pp. 87-98, 1983.

[10] F. P. Preparata and M. I. Shamos, *Computational Geometry - an Introduction*, Springer Verlag, 1985.

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