

Ensuring the Semantic Correctness of Complex Regions

Mark McKenney, Alejandro Pauly, Reasey Praing, & Markus Schneider*

University of Florida

Department of Computer and Information Sciences and Engineering
{mm7,apauly,rpraing,mschneid}@cise.ufl.edu

Abstract. Ensuring the semantic and topological correctness of spatial data is an important requirement in geographical information systems and spatial database systems in order to preserve spatial data quality and enable correct operation execution. Spatial objects like complex regions are usually represented as an ordered sequence of segments (sequence view) to support and ease the computation of spatial operations by means of plane sweep algorithms. The semantic correctness of such a sequence is usually simply assumed but is not easy to see. In this paper, we present a novel and efficient algorithm to discover the cyclic structure and check for the semantic correctness of the sequence representation of a complex region by determining its cyclic structures (component view) in terms of multiple faces possibly containing holes. The algorithm producing the component view is also interesting for object construction, manipulation, and visualization.

1 Introduction

The study of spatial objects and spatial operations has received widespread attention in varied fields such as computational geometry, spatial databases, geographic information science (GIS), computer-aided design, computer vision, and computer graphics. A large amount of the research these areas has focused on the representation and manipulation of spatial objects for use in spatial systems. Within this research, the type of complex regions has received significant attention due to their ability to model many aspects of geographic reality. A complex region is a two-dimensional spatial object consisting of multiple *faces*, each of which can contain a number of *holes*.

Because of the varied uses of complex regions in spatial systems (as well as other complex data types such as points and lines), two forms of representation have emerged to store and manipulate them. The first form of representation is the *sequential view* of a complex region, which treats a complex region as an *ordered sequence of segments*. This view has become popular because it is used as an input and output format for many spatial algorithms, specifically

* This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574.

plane sweep algorithms, used to implement spatial operations and topological predicates. The main drawback to this view is that the structural components of a complex region (i.e., the faces and holes) are not explicitly represented. The second form of representation for complex regions is the *component view*, in which the segments of the region are grouped by the structural components of the region; in other words, the segments that form a hole or face are grouped together. The component view is typically used to create, manipulate, or visualize a complex region, because its cyclic structure (i.e., the holes and faces) is known and can be utilized. For example, if a user wants to remove a face from a region, then the segments that make up that face, as well as the segments that make up any holes in that face, must be identified and removed. Such information is explicitly encoded in the component view.

The study of spatial data quality has received significant attention in the field of spatial data. An important problem in this field is to determine if a given spatial object is valid in the sense that it conforms to the spatial object's type definition. In the case of complex regions, we need to know if the object defined by a set of segments is semantically and topologically correct. If a complex region is defined based on the component view, then the cyclic structure of the region is known. Therefore, this can be validated, and the topological constraints that the faces and holes must satisfy can be checked. However, given a region represented in the sequential view, the cyclic structure must be computed explicitly. Currently, there are no known algorithms to compute this information in an efficient manner.

At this point it is important to note that the two views of region representation can be merged into a hybrid view in which segments are ordered sequentially and annotated with cyclic information. Thus, this view allows input to spatial operations and explicitly encodes cyclic information so that semantic correctness can be checked. However, such a view cannot be maintained through spatial operations. In other words, even if the cyclic structure of two regions is known, the cyclic structure of their intersection cannot be computed based on the known structure of the original regions. This means that even though the input to an intersection operation is two regions in the hybrid view, the output is a region in the sequential view; thus the cyclic structure must still be computed. This holds for the intersection, union, and difference operations between regions.

The main contribution of this paper is an efficient algorithm that takes a region represented in the sequential view as input and returns the region in a component view as output. Thus, given a region in the sequential view, we are able to discover its cyclic structure. As we mentioned before, the need to *validate* a region, or ensure that it is semantically and topologically correct, is an important concern in spatial data management. In addition to finding the cyclic structure of a region, our algorithm simultaneously checks that the region is semantically and topologically correct. Therefore, this algorithm eliminates the need to store a region in the component view because regions can quickly be validated based on our algorithm, and the component view can be quickly computed. This allows regions to be stored in the sequential view, which is more

compact than the hybrid view and is typically easier to manage on disk than the component view.

In Section 2, we present existing work related to our problem. Section 3 introduces the formal definition of regions upon which our problem solution is based. Our algorithm for deriving the component view of regions is presented in Section 4. Section 5 describes the time complexity of the algorithm. Finally, in Section 6 we provide conclusions and outline future work.

2 Related Work

To the authors' knowledge, no previous publication includes a plausible solution to the problem we have introduced in the previous section. Instead, plenty of literature exists that explores problems for which solutions can provide us with important insight for solving our problem.

The well known plane sweep algorithmic technique for geometric intersection problems, original to Shamos and Hoey [9] and also popularly employed by Bentley and Ottmann [4] as well as many others [7, 3], serves as a basis for our solution. The plane sweep model proves useful in identifying properties of the segments that are critical for efficiently solving our problem.

Related problems of arrangements of lines and segments are presented in [1, 2, 5]. Such an arrangement consists of a partitioning of the space such that the lines or segments do not cross the boundaries of the partitions. Computation of planar maps to detect polygons from sets of segments are studied in [6]. The authors provide a solution to the problem of detecting the polygons formed by a set of intersecting segments. That is, each polygon detected is defined by three or more pairwise intersections of segments. This problem is fundamentally different to our problem in that (1) there are no restrictions on the polygons that are detected, (2) in our problem we must consider holes and outer cycles separately, and (3) all holes and outer cycles must adhere to the definition of complex regions.

3 Complex Regions

In this section, we are interested in the implementation model of the region data type. To simplify our discussion, we define the implementation data type of complex regions based on segments. Then, we modify this definition such that it is based on halfsegments, which allows processing of region objects in spatial predicates and operations. In order to define a complex region, we first define the required concepts such as points, segments, polygons, and faces. Due to space limitations, we can only provide an informal definition of these concepts. For a formal definition, see [8].

The type *point* includes all single, two dimensional points. We assume that the equality “=” relation and the lexicographic order relation “<” are defined between any two points. The type *segment* incorporates all straight lines bounded by two endpoints p and q . We make use of the predicates *disjoint*, *collinear* :

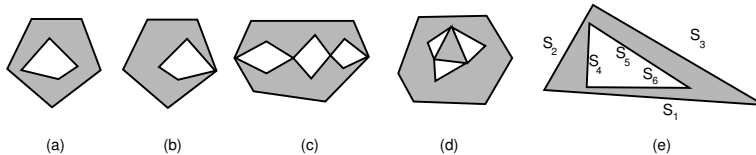


Fig. 1. Regions (a) and (b) consist of a single face. Regions (c) and (d) each have two faces. Region (e) is a single face and is annotated to identify its individual segments.

$segment \times segment \rightarrow bool$ to determine whether two segments are disjoint or are located on the same infinite lines, respectively. A function $len : segment \rightarrow real$ computes the length of a segment. These operations are required to define the order relation of halfsegments which are defined later. Using segments, a simple polygon can be implemented as a connected sequence of segments that forms a single cycle. We say that two simple polygons are *edge-disjoint* if their interiors are disjoint and they possibly share single boundary points but not boundary segments. A face is a simple polygon possibly containing a set of edge-disjoint holes, which are simple polygons, such that these holes do not collectively separate the interior of the face. A complex region is a set of edge-disjoint faces. Figure 1 shows some example regions.

Although the formal definition of *region* in [8] ensures uniqueness of representation for complex regions, in some cases, it is not obvious if a set of segments forms a hole cycle, or if the segments should be part of an outer cycle of a face. Furthermore, certain configurations of faces and holes form non-intuitive scenarios. For example, Figure 1(b) forms a face containing a hole, and not a single face that happens to meet itself at a point. Figure 1(c) depicts two faces that meet at four points, but contain no holes. Figure 1(d) shows two faces, the larger one containing a hole. The uniqueness of representation of complex regions is critical because given a sequence of segments, there is exactly one valid semantic interpretation of the cyclic structure of the region it represents. Our algorithm in Section 4 correctly interprets any valid region described as a sequence of segments.

Spatial operation implementations between regions based on the plane sweep algorithm require input to be a region encoded not as a sequence of segments, but as a sequence of halfsegments. We define the type $halfsegment = \{(s, d) | s \in segment, d \in bool\}$. A halfsegment is a hybrid between a point and a segment since it has features of both geometric structures. For a halfsegment $h = (s, d)$, if d is true (false), the smaller (greater) endpoint of s is the *dominating point* of h , and h is called a *left (right) halfsegment*. Hence, each segment s is mapped to two halfsegments $(s, true)$ and $(s, false)$. Furthermore, halfsegments are typically annotated with an *interior-above* flag, which indicates whether the interior of the region lies above or below the halfsegment. In addition to the use of halfsegments, the representation of a region object requires an order relation on halfsegments. Let dp be a function which yields the dominating point of a halfsegment. For two distinct halfsegments $h_1 = (s_1, d_1)$ and $h_2 = (s_2, d_2)$ with a common endpoint p ,

let α be the enclosed angle such that $0^\circ < \alpha \leq 180^\circ$. Let a predicate $rot(h_1, h_2)$ be true if, and only if, h_1 can be rotated around p through α to overlap h_2 in counterclockwise direction. We define a complete order on halfsegments as:

$$\begin{aligned}
 h_1 < h_2 &\Leftrightarrow & (1) \\
 dp(h_1) < dp(h_2) &\vee & (2a) \\
 (dp(h_1) = dp(h_2) \wedge (\neg d_1 \wedge d_2) \vee & & (2b) \\
 & (d_1 = d_2 \wedge rot(h_1, h_2)) \vee & \\
 & (d_1 = d_2 \wedge collinear(s_1, s_2) \wedge len(s_1) < len(s_2))) & (3)
 \end{aligned}$$

Since a segment can be substituted by two halfsegments, a region object can be implemented as an ordered sequence (array) of halfsegments. As an example, Figure 1(e) shows a complex region object (with a single face containing a hole) whose segments are labeled s_i . Let $h_i^l = (s_i, true)$ and $h_i^r = (s_i, false)$ denote the left and right halfsegments of a segment s_i respectively. The order sequence of halfsegments for this complex region is $\langle h_1^l, h_2^l, h_6^l, h_4^l, h_4^r, h_5^l, h_2^r, h_3^l, h_5^r, h_6^r, h_3^r, h_1^r \rangle$.

4 Computing the Cyclic Structure of Complex Regions

We assume that the input to our algorithm is a sequence of ordered halfsegments. If the input represents a region, the algorithm returns the region with its cyclic structure information; otherwise, the algorithm exits with an error message indicating the input sequence does not form a semantically correct region. We begin by providing a high level overview of the algorithm and then present the algorithm and provide a discussion of its details.

In general terms, the algorithm must identify all cycles present in the halfsegment sequence, and classify each cycle as either an outer cycle or a hole cycle of a particular face. To accomplish this, each halfsegment is *visited* once by the algorithm. Note that due to the definition of the type *region*, each segment belongs to exactly one cycle. When a halfsegment is visited, the algorithm marks the halfsegment indicating to which face and cycle it belongs, and whether that cycle is an outer cycle or a hole cycle. The algorithm does not alter the input when marking halfsegments, rather a parallel array to the input sequence is used to represent the cycle information. The algorithm visits halfsegments by stepping through the input list sequentially.

The first halfsegment in the input sequence will always be part of the outer cycle of a face, due to the definition of complex regions and the halfsegment ordering defined previously. Therefore, it can be visited and marked correctly. Once a halfsegment has been visited, it is possible to visit and correctly mark all other halfsegments in the cycle that it belongs to in a procedure which we denote as the *cycle walk*. Thus, all halfsegments that form the cycle to which the first halfsegment in the input sequence belongs are then visited. The algorithm then begins stepping through the remaining halfsegments. The next unvisited halfsegment encountered will be part of a new cycle. The algorithm then visits this new halfsegment. The algorithm can deduce whether this halfsegment is an

outer cycle of a new face or a hole in an existing face by examining where the halfsegment lies in relation to already known cycles. To determine this, we use a plane sweep algorithm to step through the halfsegments. Thus, we can take advantage of the plane sweep status structure to find whether or not the current halfsegment lies in the interior of a previously visited face. Once the new halfsegment is visited, we perform a cycle walk from it. Then, the algorithm continues stepping through the input list until it reaches another unvisited halfsegment, visits it, and repeats this procedure. The algorithm is shown in Algorithm 1.

To properly describe the algorithm outlined in Algorithm 1, we introduce several notations. The function $info(h)$ for a given halfsegment h returns its cyclic information, that is, its *owning* cycle, and if part of a hole, its owning face. A cycle *owns* a halfsegment if the halfsegment is part of the boundary of the cycle, and a face owns a hole if the hole is inside the face. We define the function $NewCycle(h)$ to annotate h with a unique identifier for a new cycle. Let f be a halfsegment belonging to an outer cycle of a face. The function $Owns(h, f)$ annotates the halfsegment h to indicate that it belongs to a hole in the face that owns f . Finally, we employ the function $Visit(p)$ to mark a point p as having been visited. The function $Visited(p)$ is used to verify if point p was marked as visited already. Points are only marked as visited when a halfsegment with dominating point p has been visited during the cycle walk. We mark points as being visited in order to identify the special case of a hole cycle that meets the outer cycle of a face at a point. The function $Visited(h)$ is used to verify if halfsegment h has been visited already. A halfsegment has been visited if it has been annotated with face/hole information. For a halfsegment h , we can directly compute its corresponding right (left) halfsegment h_b , which we call its *brother* by switching its boolean flag indicating which end point is dominant. We define the next halfsegment in the cycle to which h belongs as h_+ such that the dominating endpoint of h_b is equal to the dominating point $dp(h_+)$ and $h_+ \neq h_b$ and h_+ is the first halfsegment encountered when rotating h_b clockwise (in an outer cycle) or counter-clockwise (in a hole cycle) around its dominating point. The previous halfsegment in the cycle is similarly defined as h_- .

4.1 Classifying Outer and Hole Cycles

By using a sweep line, the algorithm steps through the halfsegment sequence to find the smallest unannotated halfsegment h , create a new cycle for this halfsegment, and mark its dominating point as visited (line 2-4). At this point, the algorithm needs to determine whether h belongs to a hole cycle (line 5) or an outer cycle (line 9). If a cycle is identified as a hole cycle, the outer cycle to which it belongs must also be identified (line 6-7), and the cycle must be walked using counter-clockwise adjacency of halfsegments (line 8). Recall that the plane sweep algorithm maintains the sweep line status structure, which is a ordered list of *active* segments, such that it provides a consistent view of all halfsegments that currently intersect the sweep line, up to the current *event* (the addition or removal of a halfsegment). By examining the halfsegment directly below a halfsegment h in the sweep line status, we can determine whether h is a part of

Algorithm 1: The algorithm for deriving the component view of a region.

```

Input: Sequence of unannotated halfsegments  $H$ 
Output: Sequence  $H$  with fully annotated halfsegments
1 while not end of sweep do
2   Advance sweep line to  $h$ .  $h$  is the left-most halfsegment yet to be annotated;
3   Using sweep line status, determine  $h$  as part of an outer cycle or a hole cycle;
4    $NewCycle(h)$ ;  $Visit(dp(h))$ ;
5   if  $h$  belongs to a hole then
6     Using sweep line status, retrieve halfsegment  $f$  from its owning outer cycle;
7      $Owns(h, f)$ ;
8     Set cycle walk mode to use counter-clockwise adjacency;
9   else
10    Set cycle walk mode to use clockwise adjacency;
11  end
12  /* Begin walking the cycle */
13   $c \leftarrow h_+$ ;
14  while  $c \neq h$  do
15    if  $Visited(dp(c))$  then
16       $q \leftarrow c$ ;  $c \leftarrow c_-$ ;  $NewCycle(c)$ ;  $Owns(c, h)$ ;
17      while  $dp(c) \neq dp(q)$  do
18        /* Trace back anchored hole */
19         $info(c_-) \leftarrow info(c)$ ;  $c \leftarrow c_-$ ;
20      end
21    else
22       $info(c) \leftarrow info(h)$ ;  $Visit(dp(c))$ ;  $c \leftarrow c_+$ ;
23    end
24  end

```

an outer cycle or a hole cycle of an existing face. In other words, if halfsegment p is directly below halfsegment h in the sweep line status structure and the interior-above flag of p is set to *true*, it follows that h is either in the interior of the cycle to which p belongs, or h is part of the cycle to which p belongs. Recall that as soon as a halfsegment is classified as being apart of a hole or face, the cycle to which it belongs is walked (Section 4.2) and all other halfsegments in that cycle are marked accordingly (lines 12-22). Therefore, if a halfsegment belongs to the same cycle as any halfsegment that has been previously encountered by the sweep line, it is already known to which face and/or hole cycle it belongs. Furthermore, all halfsegments that are less than a given halfsegment in halfsegment order have already been classified. Therefore, we can determine if an unmarked halfsegment belongs to a hole or outer cycle by examining the halfsegment immediately below it in the sweep line status structure.

From the definition of a face, the outer cycle of a face of a region always covers (encloses) all of its hole cycles. This means that the smallest halfsegment of this face is always a part of the outer cycle. This is also true for the entire region object where the smallest halfsegment in the ordered sequence is always a part of the first outer cycle of the first face. Furthermore, due to the order relation of halfsegments and the cyclic structure of a polygon, the smallest halfsegment of a face will always be a left halfsegment with the interior of the face situated above it. Thus, when we process this halfsegment, we set its interior-above flag to indicate this fact. Since we have classified this cycle as an outer cycle, we can walk the cycle and set the interior-above flag for all halfsegments of this cycle.

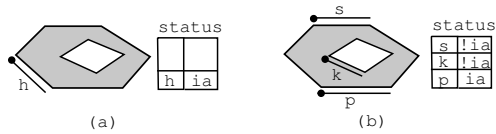


Fig. 2. Processing the smallest halfsegment h of the sequence (a) and k of a cycle (b).

For example, Figure 2(a) illustrates the case where the smallest halfsegment of the sequence is processed and the cycle is classified as an outer cycle.

Once the first outer cycle of a face in a region has been processed, we continue to process halfsegments that have not yet been classified based on the plane sweep status structure. Figure 2(b) shows an example. Here, we add/remove visited halfsegments into/from the sweep line status in sequence ordered up to the smallest unvisited halfsegment k . This halfsegment *must* be the start of a new cycle that we must now classify. We know k is the start of a new cycle because all halfsegments of an existing cycle that include a halfsegment j such that $j < k$ must have been marked as visited by the walking process. Once we reach this new cycle represented by its starting halfsegment k , we add this halfsegment into the sweep line status. We classify the type of cycle k belongs to by examining the interior-above flag of the halfsegment p (its predecessor) which was already visited and sits immediately below k in the sweep line status structure. If the predecessor indicates that the interior of the face is above it (the interior-above attribute of p is set to true), then k lies in the interior of the cycle to which p belongs; thus, k must be part of a hole cycle and the interior of the face to which k belongs must lie below k . If the interior-above flag of p indicates that the interior of the face to which p belongs is below p , then the current halfsegment k must be part of an outer cycle of a new face. In case that there is no predecessor, then the current halfsegment must be a part of an outer cycle of a new face, because it does not lie in the interior of any other face's outer cycle. Once the cycle is classified as either an outer cycle of a new face or a hole cycle of an existing face, the cycle walking procedure is carried out to determine all halfsegments that belong to the cycle.

4.2 Walking Cycles

In general terms, we use the phrase *walking a cycle* to indicate the traversal of a cycle such that each halfsegment that forms the cycle is visited. Furthermore, the halfsegments in such a traversal are visited in the order in which they appear in the cycle. In other words, given a halfsegment h , all halfsegments in the cycle to which h belongs are found by repeatedly finding h_+ until the the original halfsegment is encountered again. For example, when walking the outer cycle of the region in Figure 1e in clockwise order beginning from S_1 , the halfsegments would be encountered in the order $h_1^l, h_1^r, h_3^r, h_3^l, h_2^r, h_2^l$. The two main challenges to this portion of the algorithm are (i) to identify cycles correctly such that they correspond to the unique representation of a region as stated in the definition of

complex regions, and (ii) to achieve this efficiently. In this section we show how to satisfy the first challenge. Time complexity is discussed in the next section.

When a halfsegment h is encountered by the algorithm that has not yet been classified, it is classified as belonging to a hole or outer cycle in line 5. If h belongs to an outer cycle, then the cycle walk portion of the algorithm in lines 12-22 is executed. Due to the halfsegment ordering and the definition of regions, the smallest unvisited halfsegment in the input sequence that the plane sweep encounters is always a left halfsegment of an outer cycle of a face and the interior of that face always lies above the halfsegment. If we rotate h_b clockwise around its dominating point, it will intersect the interior of the face. Thus, the first halfsegment encountered when rotating h_b clockwise around its dominating point will be part of the outer cycle of the same faces (except for a special case discussed below) and will be h_+ . We know this to be true because if we find h_+ in this fashion and it turns out to be part of another face, then two faces would intersect, which is prohibited by the definition of complex regions. It follows that each successive halfsegment in the outer cycle can be found by rotating the brother of the current halfsegment clockwise around its dominating point because the location of the interior relative to the halfsegment can always be deduced based on the previous halfsegment encountered in the cycle walk.

One special case occurs when walking outer cycles: the existence of a hole in a face that meets the outer cycle at a point (see Figure 1b). When walking an outer cycle that contains such a hole, the halfsegments that form the hole will be classified as being part of the outer cycle using the procedure just described. In order to remedy this, we mark each point that is a dominating point of a halfsegment encountered during the cycle walk (line 20). Each time we find a new halfsegment that is part of an outer cycle, we first check if its dominating point has been visited yet (line 14). If it has been visited, then we know that we have encountered that point before, and a hole cycle that meets the outer cycle must have been discovered. When this happens, we loop backwards over the cycle until we find the halfsegment whose dominating point has been visited twice (lines 15-18). The halfsegments forming the hole are then marked as such. The remainder of the outer cycle is then walked.

Walking a hole is identical to walking an outer cycle, except that a counter-clockwise rotation from h_b is used to find h_+ . A counter-clockwise rotation is required because the interior of the face is intersected by h_b when rotating h_b around its dominating point. When walking holes, the special case exists that two holes may meet at a point. Thus, we employ the same strategy to detect this case as we did with the special case of a hole meeting a face (lines 15-18).

5 Complexity

The classification component of our algorithm requires the use of a plane sweep algorithm over n halfsegments; thus, a complexity of $O(n \log(n))$ is required. However, because a segment intersection indicates an invalid region, we do not need to compute or report intersections, and thus do not require an output

sensitive algorithm. The cycle walk algorithm requires that we locate a halfsegment based on its dominating point in a list of halfsegments. Because the list of halfsegments is ordered, it is possible to employ a searching technique to locate halfsegments quickly. If the list of halfsegments is implemented as an array, we can simply use a binary search. Because each halfsegment is searched for one time in this stage of the algorithm, the cycle walk can be computed in $O(n \log(n))$ time for n halfsegments. Furthermore, once a halfsegment has been classified, it will at most be visited one additional time by the sweep line portion of the algorithm. Finally, it is possible to keep track of the number of times a point has been visited in an ordered array. Thus, we can rely on a binary search to find points in order to mark them as visited. The number of points is bounded by the number of halfsegments, thus, this has complexity at most $O(n \log(n))$. Therefore, the algorithm has a worst case time complexity of $O(n \log(n))$.

6 Conclusions

In this paper we have introduced an $O(n \log(n))$ algorithm for computing the component view of a complex region originally represented by a sequence of halfsegments. Furthermore, the algorithm checks the semantic correctness of region objects since it will not be able to compute the component view of a halfsegment sequence that does not represent a region. The algorithm was successfully implemented as part of a spatial algebra that was embedded into an existing database management system for the purpose of managing spatial data.

References

1. N. M. Amato, M. T. Goodrich, and E. A. Ramos. Computing Faces in Segment and Simplex Arrangements. In *ACM Symposium on Theory of Computing*, pages 672–682. ACM Press, 1995.
2. T. Asano, L.J. Guibas, and T. Tokuyama. Walking on an Arrangement Topologically. In *ACM Annual Symp. on Computational Geometry*, pages 297–306. ACM Press, 1991.
3. I. J. Balaban. An Optimal Algorithm for Finding Segments Intersections. In *ACM Annual Symp. on Computational Geometry*, pages 211–219. ACM Press, 1995.
4. J.L. Bentley and T. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. on Computers*, C-28:643–647, 1979.
5. H. Edelsbrunner, L. J. Guibas, and M. Sharir. The Complexity of Many Faces in Arrangements of Lines of Segments. In *ACM Annual Symp. on Computational Geometry*, pages 44–55. ACM Press, 1988.
6. A. Ferreira, M. J. Fonseca, and J. A. Jorge. Polygon Detection from a Set of Lines. In *Encontro Portugues de Computacao Grafica*, pages 159–162, 2003.
7. J. Nievergelt and F. P. Preparata. Plane-Sweep Algorithms for Intersecting Geometric Figures. *Communications of the ACM (CACM)*, 25:739–747, 1982.
8. M. Schneider and T. Behr. Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems (TODS)*, 31:39–81, 2006.
9. M. Shamos and D. Hoey. Geometric Intersection Problems. In *IEEE Symp. on Foundations of Computer Science*, 1976.