Region Extraction and Verification for Spatial and Spatio-Temporal Databases

Mark McKenney

Texas State University, Department of Computer Science
mckenney@txstate.edu

Abstract. Newer spatial technologies, such as spatio-temporal databases, geo-sensor networks, and other remote sensing methods, require mechanisms to efficiently process spatial data and identify (and in some cases fix) data items that do not conform to rigorously defined spatial data type definitions. In this paper, we propose an $O(n \lg n)$ time complexity algorithm that examines a spatial configuration, eliminates any portions of the configuration that violate the definition of spatial regions, and constructs a valid region out of the remaining configuration.

1 Introduction

Spatial databases have become rather mature and many commercial products are based upon them; however, newer technologies have introduced new problems in spatial data management, and highlighted problems in existing technologies that have yet to be fully addressed. We illustrate this with an example from moving objects databases. Current spatial technologies are built upon a rigid definition of spatial data types. For example, a *complex region* [9] (which we will refer to simply as a region) can consist of multiple faces, each containing zero or more holes (e.g., Italy has multiple islands as faces and a hole where Vatican City lies). A *moving region* is a complex region that moves and changes over time (Figure 1a). A well known operation over moving regions is to extract a region at a specific time t [7]. Figure 1b depicts the region in Figure 1a at time t; note that the region contains some lines that do not form a face, but are one-dimensional components. Such components violate the definition of regions; yet, in the case of moving regions they are typically required to indicate that a face of a region is about to come into existence, or has just ceased to exist. Because the configuration at time t contains such anomalies, we cannot simply extract the spatial configuration at that time and apply other spatial operations to it since spatial operations must have valid input to preserve type closure properties. Therefore, a mechanism is required to differentiate the invalid and valid portions of the configuration. In the case of the region in Figure 1b, we must be able to generate the region in Figure 1c. Additional occurrences of this type of problem arise with respect to data quality issues in fields such as geo-sensor networks and remote sensing, in which large amounts of data are generated.

More formally, we can characterize the problem highlighted above as follows: given a set of straight line segments S in two dimensional space, does there exist

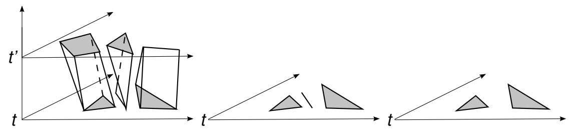

Fig. 1. A moving region (a), the scene at time t (b), and the region at time t (c).

some subset of segments $R \subseteq S$ that forms a valid spatial region. We consider a valid spatial region to fit the definition of complex regions as provided by [9]. We term this the *region extraction and verification problem* (REVP).

This paper provides a solution to the REVP that: i) identifies the parts of a scene that form a valid region, ii) identifies the parts of the scene that are invalid according to the definition of complex regions, iii) is efficient in terms of both computation and space requirements, and iv) is suited for implementation in spatial and spatio-temporal databases. Our algorithm takes an arbitrary set of segments and returns two sets of segments: those that form a valid region, and the remaining segments. Furthermore, the segments that form a valid region will be properly annotated for input to existing algorithms for spatial operations such as intersection, topological predicate operations, etc.

2 Related Work

Although much work has been completed in the literature with respect to geometric algorithms for spatial data applications, the authors are unaware of any work which directly addresses the problem stated in Section 1. Specifically, no algorithm exists that is applicable to complex regions. Algorithms, such as those presented in [1, 2, 5, 6], detect polygon structures in arrangements of line segments, or create planar maps from arrangements of line segments; however, the REVP is fundamentally different than the problems solved in these papers because (i) there are no restrictions on the polygons detected (i.e., they do not have to be convex, they can contain holes, etc.) (ii) holes and outer cycles of regions must be handled correctly, and (iii) all polygons formed must collectively fit the constraints provided by the definition of complex regions.

We employ the well known plane-sweep algorithmic paradigm in our algorithm. This type of algorithm is original to Shamos and Hoey [10], and a popular version is introduced by Bentley and Ottmann [4], and much additional work has been proposed on the technique [8, 3].

3 Data Model

A Halfsegment Representation of Regions: In this section, we provide an informal type definition. For a formal definition, see [9]. Spatial operation implementations between regions based on the plane sweep algorithm require input to be a region encoded not as a sequence of line segments, but as a sequence of halfsegments. We define the type $\mathit{halfsegment} = \{(s, d, l, r) | s \in \mathit{segment}, d \in$

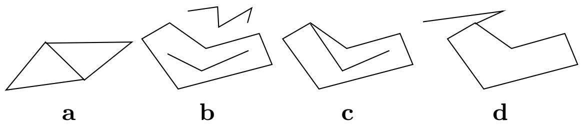

Fig. 2. Invalid halfsegment configurations.

bool$, l, r \in \mathbb{Z}\}$ where a segment is a straight line segment between two endpoints and l and r are *labels* corresponding to the portion of the embedding space that lies above or to the left of the line that the halfsegment lies on, and the portion that lies below and to the right, respectively. Thus, a halfsegment is said to have two *sides*, a left and right side corresponding to each label. For a halfsegment $h = (s, d, l, r)$, 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 $(s, false)$. Given one halfsegment h, we denote the halfsegment with identical endpoints and an opposite boolean flag as h's *brother*. We require an order relation on halfsegments. Informally, a complete ordering over halfsegments exists based on their dominating points. If two halfsegments have the same dominating point, then the smaller halfsegment is the one encountered first when rotating a vertical line extending above the dominating point counter-clockwise around the dominating point. A simple polygon is a connected sequence of segments that forms a single cycle. 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.

Classifying Invalid Cases: We make the assumption that halfsegments used as input intersect at endpoints only. This can be enforced with geometric algorithms without affecting the worst case time complexity of our algorithm (Section 4.3).

Based on our data model, we make the observation that all invalid configurations fall into one of two categories: (i) adjacent cycles, when two cycles are not edge-disjoint, and (ii) stick configurations, in which halfsegments do not form a cycle. Figure 2 depicts an example of each case. In Figure 2a, either the right or left cycle may be discarded and a valid region remains. In Figure 2b, Figure 2c, and Figure 2d, the halfsegments forming the sticks must be discarded to form a valid region. These figures respectively illustrate the various forms of the stick configuration that must be handled correctly: (i) disconnected stick configurations (when sticks do not connect to any cycle), (ii) internal stick configurations (when sticks connect to the interior of a cycle), and (iii) external stick configurations (where sticks connect to the exterior of a cycle). Lemma 1 shows that adjacent cycle and stick configurations are the only invalid configurations.

Lemma 1. *The only invalid configurations that arise in the given data model for complex regions are stick configurations and adjacent cycle configurations.*

Proof Sketch. Consider a region r of straight line segments. Clearly, the

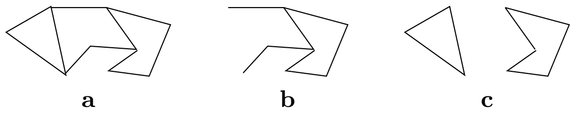
Fig. 3. An input to our algorithm containing adjacent cycle configurations (a), the scene after identifying the leftmost cycle as valid (b), and the final valid region (c).

removal of any of the segments in r cannot cause segments to intersect at points other than endpoints, but may cause stick or adjacent cycle configurations. Second, consider the addition of new line segments to an already valid region. Because of our definition of regions, and the requirement that line segments only intersect at endpoints, the addition of new segments can only result in the addition of new stick features or new cycles. Therefore, stick and adjacent cycle configurations are the only invalid configurations that must be handled. □

4 Algorithm

Our algorithm removes segments involved in adjacent cycle and stick configurations in an input halfsegment sequence until no such configurations remain. The result is a valid region with properly annotated halfsegments.

Our algorithm operates by removing halfsegments from consideration once it is able to identify them as being part of a valid cycle or stick configuration. Removing halfsegments is achieved by marking their labels to indicate that a halfsegment is part of a valid cycle (in which case one label indicates the interior of a region, and the other indicates the exterior), or that it is part of an invalid configuration (in which case both labels are marked as being invalid). Thus, any adjacent cycles will have some of their halfsegments removed; the result is that adjacent cycles are converted into stick configurations (Figure 3). Therefore, the algorithm is complete when all halfsegments are marked as valid or invalid. The algorithm proceeds in the following steps: (i) find the first halfsegment that is not yet removed from consideration, (ii) determine if the halfsegment lies on the interior of a known face, (iii) discover all other halfsegments that form a cycle or stick configuration containing the current halfsegment and label them, and (iv) repeat until all halfsegments are properly labeled.

4.1 Finding and Labeling an Unprocessed Halfsegment

The algorithm begins by finding the least halfsegment h in halfsegment order that is not yet removed from consideration. We say that such a halfsegment is *unprocessed*. We must then determine if h lies on the interior of a known face of a region, or if it lies in the exterior of a region (i.e., h is potentially part of an outer cycle of a face of a region). Therefore, we frame our algorithm in terms of a *plane sweep* algorithm. Recall that a plane sweep algorithm uses an imaginary line that traverses the input halfsegment sequence and each time it encounters a left halfsegment, that halfsegment is placed in the *active list*. The active list is ordered according to the point at which each segment in the active list intersects the sweep line. Each time a right halfsegment is encountered, its corresponding left halfsegment is removed from the active list.

When a new halfsegment h is encountered, we can always determine whether or not it is part of an outer cycle or hole cycle by looking at the labels of halfsegment b that will be immediately below h in the active list. This information allows us to move on to the next step (the *cycle walk* in Section 4.2) and identify all halfsegments that form a cycle with h. b will always have been previously processed due to the halfsegment ordering.

Once a cycle walk has been performed, the plane sweep portion of the algorithm resumes and traverses the halfsegment sequence until a partially processed (Section 4.2) or unprocessed halfsegment is found. If a halfsegment is found that is unprocessed, then the cycle walk portion of the algorithm is executed. If the halfsegment is partially processed, then it is marked as being part of an invalid configuration, and the plane sweep portion of the algorithm continues. The reasoning is given in the following lemma:

Lemma 2. *A partially processed halfsegment encountered in the plane sweep portion of the algorithm is always part of a stick configuration.*

Proof Sketch. According to Lemma 4, the cycle walk portion of the algorithm will convert any adjacent cycle configurations to stick configurations. Furthermore, the only time a halfsegment is partially processed is during the cycle walk portion of the algorithm. Therefore, since a partially processed halfsegment has already been visited by a cycle walk, it must be part of a stick configuration since the cycle walk will correctly identify halfsegments that belong to a valid cycle. □

4.2 Walking the Cycle

Once an unprocessed halfsegment h is found and it is known whether h lies in the interior of a face of a region or not, the next step is to identify all halfsegments that form the cycle or stick configuration that h is part of. In order to identify the segments that form a cycle containing h, we make the observation that according to the definition of complex regions, each cycle that forms part of the boundary of a region separates the embedding space into three, disjoint point sets: the interior, exterior, and boundary. Therefore, given a halfsegment h and the knowledge of upon which side of the halfsegment the interior of the cycle lies, the adjacent halfsegment in the cycle can be found by rotating h around its submissive point through the interior of the cycle until a new halfsegment h_{next} is found. Because h was rotated through the interior of a cycle, the next halfsegment encountered must also bound the interior of that cycle and we can trivially determine on which side of the halfsegment the interior lies for labeling. This is repeated until a stopping condition (see below) occurs. Due to halfsegment ordering, a clockwise rotation around submissive points is always used. We denote the procedure of visiting halfsegments in a cycle in cyclic order and assigning their appropriate labels as a *cycle walk*.

Let the notation $1_l, 1_r$ to refer to the left and right halfsegment of segment 1, etc. A cycle walk beginning at halfsegment h proceeds until one of three stopping

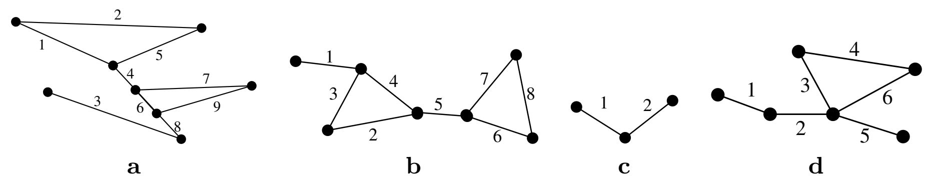

Fig. 4. Instances of stick configurations.

conditions occurs: (i) h is encountered a second time, (ii) the brother of h, h_b is encountered, or (iii) a halfsegment j is encountered that is already correctly labeled. In the first case, encountering h a second time during a cycle walk indicates that the complete and valid cycle has been walked (e.g., walking of the leftmost cycle in Figure 4a by visiting 1_l then 5_l then 2_r). The second case deals with a *leading stick configuration*. In this case, the exterior of a cycle (or cycles) is walked (e.g., a cycle walk on Figure 4b will visit 1_l, then 4_l then continue until it reaches 1_r). In this case, any segment that has had both corresponding halfsegments visited is a stick. Any other halfsegments cannot be marked as valid or invalid, so they are labeled as being *partially processed* (we know upon which side of the halfsegment the exterior lies, but cannot ensure that the segments are part of a valid cycle). The final case indicates that a stick configuration was formed due to the previous identification of a valid cycle. We denote this case a *trailing stick configuration*, and it arises in Figure 4a. Internal stick configurations are identified when the first stopping condition occurs, and both corresponding halfsegments forming internal stick configurations are visited.

Lemma 3. *The cycle walk portion of the algorithm correctly identifies segments belonging to stick configurations and segments that form valid cycles.*

Proof Sketch. A halfsegment rotation through a cycle's interior guarantees that halfsegments that bound the cycle will be visited. It is clear that segments forming internal stick configurations will have both corresponding halfsegments visited, and will therefore be marked as invalid. In order to classify the boundary halfsegments correctly, we must be able to identify if a cycle walk or an external walk occurred. In a cycle walk, the halfsegment h that began the cycle walk will be encountered before its brother h_b. Otherwise, an external walk must have been performed and h is the first halfsegment in a leading stick configuration. This follows directly from the halfsegment ordering and a clockwise rotation around endpoints. Therefore, we can discern the cycle walk scenarios. □

Lemma 4. *The cycle walk portion of the algorithm must only be able to identify stick configurations and valid cycles, not adjacent cycle configurations.*

Proof Sketch. Once a halfsegment has been identified as being part of a valid cycle or is identified as being invalid, then it is removed from consideration from the algorithm. In the case of adjacent cycles, a cycle walk will identify one cycle, and remove its halfsegments from consideration. This effectively converts adjacent cycles to a valid cycle and a stick configuration (Figure 3). Therefore, the cycle walk algorithm must only be able to identify valid cycles and stick configurations. □

4.3 Discussion

A plane sweep algorithm for finding line segment intersections can be implemented in $O(n \lg n + k)$ time complexity and $O(n + k)$ space complexity for n segments and k segment intersections. We begin with the assumption that line segments do not intersect; therefore, we have $O(n \lg n)$ and $O(n)$ time and space complexity, respectively. Each halfsegment is visited at most twice in the algorithm, once to partially process it, and once to fully process it. Whenever a halfsegment is marked as fully processed, we mark its brother identically. Thus, a logarithmic search technique is used to locate the brother. The cycle walk portion of the algorithm requires us to find a halfsegment in cyclic order from a given halfsegment. This can be done in logarithmic time using a binary search technique that takes advantage of halfsegment ordering. Therefore, the complete algorithm is bounded by $O(n \lg n)$ in time complexity and $O(n)$ space complexity for an input configuration of n segments. Running a line segment intersection algorithm to ensure that segments intersect only at endpoints takes $O(n \lg n + k)$ time complexity, and in practice does not affect running time in typical cases.

Algorithm 1 shows the final algorithm as it has been described. The final step is to ensure correctness. Our definition of correctness is that our algorithm can effectively handle all invalid spatial configurations, and that it returns a valid region. Lemmas 1-4 describe the possible invalid cases that must be handled by the algorithm, and show that the algorithm handles the cases correctly. Therefore, our algorithm will always return a valid region given a valid input:

Theorem 1. *Given valid input, the proposed algorithm will identify, correctly label, and return halfsegments forming a valid region.*

Proof Sketch. Lemmas 1-4 indicate the cases that the algorithm must handle and shows how the algorithm correctly handles each case. □

5 Conclusion

In this paper, we have identified the REVP as an important problem in the growing fields of spatio-temporal and moving objects databases, and provided examples of the problem in traditional spatial applications such as remote sensing and geo-sensor networks. We have developed, implemented, and presented an efficient $O(n \lg n)$ time complexity algorithm that can be used to solve this problem that can be incorporated into spatial systems. Furthermore, our algorithm uses an input format that is common in spatial algorithms and spatial data representations so that it can be easily incorporated into existing systems.

References

1. N. M. Amato, M. T. Goodrich, and E. Ramos. Computing Faces in Segment and Simplex Arrangements. *ACM Symp. on Theory of Computing*, pp. 672–682, 1995.

Algorithm 1: A pseudo-code implementation of the proposed algorithm.

```
    Input: Sequence of non-labeled halfsegments $H$
    Output: Sequence of labeled halfsegments $J$
1   while not end of plane sweep do
2       Advance sweep line to $h$, the left-most unprocessed or partially processed halfsegment.;
3       Find halfsegment $j$ below $h$ in the sweep line active list;
4       var $isOutercycle \leftarrow True$;
5       var $interiorIsAboveHalfsegment \leftarrow True$;
6       var $currCycle \leftarrow 5$;  var $unvisitedLabel \leftarrow 0$;
7       var $exteriorLabel \leftarrow 3$;  var $interiorLabel \leftarrow 4$;
8       var $invalidAndInExterior \leftarrow 1$;  var $invalidAndInInterior \leftarrow 2$;
9       var $invalidLabel \leftarrow invalidAndInExterior$;
        // check to see if $h$ lies in the interior of the face bordered by $j$
10      if $j.labelAbove \neq exteriorLabel$ OR $j.labelAbove = invalidAndInInterior$ then
11          $isOutercycle \leftarrow False$;
12          $invalidLabel \leftarrow invalidAndInInterior$;
        // handle the case that $h$ is partially processed. If $h$ has an exterior on one
           side, and an unvisited label on the other, it is partially processed.
13      if ($h.labelAbove = unvisitedLabel$ AND $h.labelBelow = exteriorLabel$) OR
        ($h.labelAbove = exteriorLabel$ AND $h.labelBelow = unvisitedLabel$) then
            // Fully process $h$, setting the labels of both sides to the invalid label
14          $h.aboveLabel \leftarrow h.belowLabel \leftarrow invalidLabel$;
15          Find $h_b$, the brother of $h$;
16          $h_b.aboveLabel \leftarrow h_b.belowLabel \leftarrow invalidLabel$;
17      else
            // Walk the cycle
18          if $isOutercycle$ then
19              $h.labelAbove \leftarrow currCycle$;  $h.labelBelow \leftarrow exteriorLabel$;
20          else $h.labelAbove \leftarrow exteriorLabel$;  $h.labelBelow \leftarrow currCycle$;
21          $interiorIsAboveHalfsegment \leftarrow isOuterCycle$;
22          $prev \leftarrow h$;
23          $k \leftarrow findNextInCycle(h)$;
24          $visitedStack \leftarrow emptystackofhalfsegments$;
            // (continued)
70
71
```

2. T. Asano, L.J. Guibas, and T. Tokuyama. Walking on an Arrangement Topologically. *ACM Annual Symp. on Computational Geometry*, pp. 297–306, 1991.
3. I. J. Balaban. An Optimal Algorithm for Finding Segments Intersections. *ACM Annual Symp. on Computational Geometry*, pp. 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. *ACM Annual Symp. on Computational Geometry*, pp. 44–55. ACM Press, 1988.
6. A. Ferreira, M. J. Fonseca, and J. A. Jorge. Polygon Detection from a Set of Lines. *Encontro Portugues de Computacao Grafica*, pp. 159–162, 2003.
7. L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. *ACM SIGMOD Intl. Conf. on Management of Data*, pp. 319–330, 2000.
8. J. Nievergelt and F. P. Preparata. Plane-Sweep Algorithms for Intersecting Geometric Figures. *Communications of the ACM (CACM)*, 25:739–747, 1982.
9. M. Schneider and T. Behr. Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems (TODS)*, 31(1):39–81, 2006.
10. M. Shamos and D. Hoey. Geometric Intersection Problems. *IEEE Symp. on Foundations of Computer Science*, 1976.

Algorithm 1: *(continued).*

```
25 while $True$ do
       // find if the interior of the face is above this halfsegment
26     if $\neg sameInteriorAbove(prev, k)$ then
27         $interiorIsAboveHalfsegment \leftarrow \neg interiorIsAboveHalfsegment$;
       // Check to see if the cycle walk is complete
28     if $k = h$ then
           // Convert the partially processed labels to fully processed labels
29         foreach $halfsegmenti$ in $visitedStack$ do
30             if $i.labelAbove > interiorLabel$ then
31                 $i.labelAbove \leftarrow interiorLabel$;
32             else if $i.labelAbove = unvisitedLabel$ then
33                 $i.labelAbove \leftarrow exteriorLabel$;
34             if $i.labelBelow > interiorLabel$ then
35                 $i.labelBelow \leftarrow interiorLabel$;
36             else if $i.labelBelow = unvisitedLabel$ then
37                 $i.labelBelow \leftarrow exteriorLabel$;
38             Find $i_b$, the brother of $i$;
39             $i_b.aboveLabel \leftarrow i.aboveLabel$;
40             $i_b.belowLabel \leftarrow i.belowLabel$;
41         goto ENDOFCYCLEWALK;
       // Now check for the invalid cases
       // First invalid case: we encounter the same segment twice in a cycle walk
42     Find $k_b$, the brother of $k$;
43     if $k_b.labelAbove = currCycle$ OR $k_b.labelBelow = currCycle$ AND $k \neq h_b$ then
           // Mark the halfsegment as invalid
44         $k.labelAbove \leftarrow k.labelBelow \leftarrow invalidLabel$;
45         $k_b.labelAbove \leftarrow k_b.labelBelow \leftarrow invalidLabel$;
46     // Second invalid case: we started on a stick and performed an exterior walk
       // Third invalid case: we encounter a fully processed halfsegment, which means we
          performed an exterior cycle walk on some halfsegments
47     else if $(k = h_b)$ OR $(k.labelAbove \neq currCycle$ AND $k.labelAbove \neq unvisitedLabel$
       AND $k.labelBelow \neq currCycle$ AND $k.labelBelow \neq unvisitedLabel)$ then
           // Flip the labels since we have walked the exterior
48         foreach $halfsegment\ i$ in $visitedStack$ do
49             $tmp \leftarrow i.labelAbove$;
50             $i.labelAbove \leftarrow i.labelBelow$;
51             $i.labelBelow \leftarrow tmp$;
52             if $i.labelAbove = i.labelBelow$ then
                   // Visited the halfsegment twice in a cycle walk. It is a stick
53                 $i.labelAbove \leftarrow i.labelBelow \leftarrow invalidLabel$;
54             else
                   // Assign the label that is not on the exterior side to unknown label
55                 if $i.labelAbove > interiorLabel$ then $i.labelAbove \leftarrow unvisitedLabel$;
56                 if $i.labelBelow > interiorLabel$ then $i.labelBelow \leftarrow unvisitedLabel$;
           // Mark $h$ and $h_b$as being invalid
57         $h.labelAbove \leftarrow h.labelBelow \leftarrow invalidLabel$;
58         $h_b.labelAbove \leftarrow h_b.labelBelow \leftarrow invalidLabel$;
59         goto ENDOFCYCLEWALK;
60     else
           // If we get here, then there is nothing wrong. process this halfsegment.
61         if $interiorIsAboveHalfsegment$ then
62             $k.labelAbove \leftarrow currCycle$;
63             if $k.labelBelow = unvisitedLabel$ then $k.labelBelow \leftarrow exteriorLabel$;
64         else
65             $k.labelBelow \leftarrow currCycle$;
66             if $k.labelAbove = unvisitedLabel$ then $k.labelAbove \leftarrow exteriorLabel$;
       // Set up for the next iteration of the cycle walk loop
67     $prev \leftarrow k$;
68     $k \leftarrow findNextInCycle(prev)$;
69 ENDOFCYCLEWALK;
```