Polygon Clipping and Polygon Reconstruction

Leonardo Guerreiro Azevedo[1], Ralf Hartmut Güting[2]

[1] Computer Science Department, Graduate School of Engineering, Federal University of Rio de Janeiro, PO Box 68511, ZIP code: 21945-970, Rio de Janeiro, RJ, Brazil

[2]LG Datenbanksysteme für neue Anwendungen, FB Informatik, Fernuniversität Hagen, D-58084 Hagen, Germany

leogazevedo@gmail.com, rhg@fernuni-hagen.de

Abstract. *Polygon clipping is an important operation that computers execute all the time. An algorithm that clips a polygon is rather complex. Each edge of the polygon must be tested against each edge of the clipping window, usually a rectangle. As a result, new edges may be added, and existing edges may be discarded, retained, or divided. Multiple polygons may result from clipping a single polygon. After clipping, we may have a set of segments, which must be handled to generate the clipped polygon. This work proposes two new algorithms: clipping polygon against a rectangle window, and polygon reconstruction from a set of segments. The algorithms were implemented in Secondo, a platform for implementing and experimenting with various kinds of data models.*

1. Introduction

Polygon clipping is one of those humble tasks computers do all the time. It's a basic operation in creating graphic output of all kinds. Polygon clipping is defined by Liang and Barsky (1983) as the process of removing those parts of a polygon that lie outside a clipping window. A polygon clipping algorithm receives a polygon and a clipping window as input. The algorithm must evaluate each edge of the polygon against each edge of the clipping window, usually a rectangle. As a result, new edges may be added, and existing edges may be discarded, retained, or divided. Multiple polygons may result from clipping a single polygon. Two examples of a polygon clipping are presented in Figure 1.

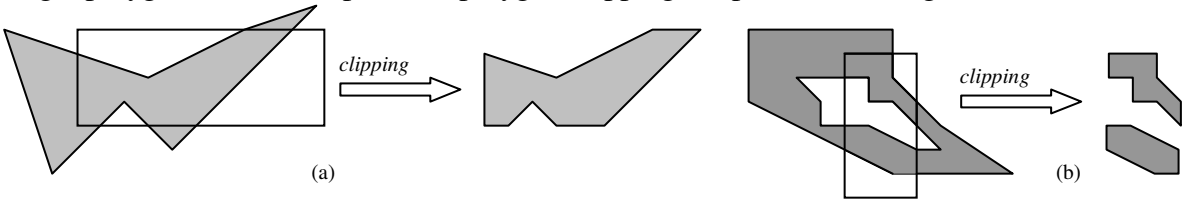


Figure 1 – Examples of polygon clipping by a rectangle window: (a) clipping a polygon that does not have hole; (b) clipping a polygon that has a hole.

There are several well-known polygon clipping algorithms, each having its strengths and weaknesses. The oldest one (from 1974) is called the Sutherland-Hodgman algorithm, as presented by Newman and Sproull (1979). In its basic form, it is relatively

simple. It is also very efficient in two important cases, one being when the polygon is completely inside the boundaries and the other when it's completely outside.

The Liang-Barsky algorithm (1983) is a good deal more complicated, but in certain cases fewer intersections need to be calculated than for Sutherland-Hodgman. Therefore, it may be somewhat faster when many polygon lines intersect with the clipping boundaries. The Weiler and Atherton (1977) algorithm is even more complicated. It allows clipping of a subject polygon by an arbitrarily shaped clip polygon. It is generally applicable only in 2D. Even more ways to clip a polygon exist. None of them is totally perfect. Often, it is possible to feed a weird polygon to an algorithm and retrieve an incorrect result. One of the vertices will disappear, or a ghost vertex will be created. Therefore, the hunt for the perfect clipping algorithm is still open.

Usually polygon's points are stored as an ordered list of points. This structure is employed by many applications, and it is simple to read the polygon and draw it on a Computer Graphics Interface. However, there are some cases where it is not possible to store segments as a simple connected list of points. For instance, when the polygon has holes, it is required an extra information to define which points belong to the external cycle and which ones belong to the internal cycle (or which ones belong to the hole). There are many approaches in the literature to store polygons. For example, Scholl and Voisard (1989) defined general polygons, and Voisard (1992) extended this to general types for points and lines, while Gargano *et al*. (1991) gave only a single type for all kinds of geometric objects; a value is essentially a set of sets of pixels. Güting and Scheneider (1995) proposed the introduction into the DBMS the concept of a *realm,* a finite, user-defined structure that is used as a basis for one or more system data types. Realms are somewhat similar to enumeration types in programming languages. A realm used as a basis for spatial data types is essentially a finite set of points and non-intersecting line segments. All points, lines, and polygons associated with objects *(*spatial attribute values*)* can be defined in terms of points and line segments present in the realm. In fact, spatial attribute values are created only by selecting some realm objects. The polygon structure employed in this work was proposed by Güting *et al*. (1995) and Güting and Schneider (1995), it is presented in section 2.1 (Definition 4).

Polygon reconstruction is the process of reconstructing a polygon from a set of segments those are not in any specific order. For instance, the segments may be stored in a way that a segment that follows another segment does not has a common point. One example of application where this algorithm may be used is polygon clipping. After clipping, the output segments may not be ordered, and the reconstruction algorithm could be used to compute the polygon.

In this work, we propose two new algorithms: an algorithm for polygon clipping by a rectangle window; and an algorithm for polygon reconstruction from a set of segments. The algorithms do not assume any specific orientation of polygon's segments, they do not rely on the computation of parity or wrap numbers of a reference point. Besides, each segment can be processed independent from the others, since all information needed to evaluate one segment is stored within it. The algorithms handle polygons that have multiple boundaries (a polygon that is composed by more than one part) as well as polygons with

holes. The algorithms were implemented in Secondo system (Dieker and Güting, 2000; Güting *et al.*, 2005), and according to the data structure described in the ROSE algebra (Güting, 1993; Güting *et al*, 1995; Güting and Schneider, 1995).

This work is divided in sections, as follows: Section 1 is this introduction; Section 2 presents important definitions for our proposals; Section 3 presents the polygon clipping algorithm; Section 4 presents the polygon reconstruction algorithm; Section 5 presents the implementations we have done in Secondo; finally, in Section 6, we present our conclusions.

2. Preliminary Definitions

In order to present the details of our algorithms proposals it is needed first to define some concepts.

Definition 1) Cycle Direction

The cycle direction defines where is located the enclosed part of a polygon related to its segments. A cycle having the enclosed part on the left is called counterclockwise. On the other hand, when the enclosed part is on the right the cycle is clockwise.

Definition 2) (x,y)-lexicographic Order of two Points

Let $p_1=(x_1,y_1)$ and $p_2=(x_2,y_2)$ be two points in 2-d, the (x,y)-lexicographic order is defined as $p_1 < p_2 \Leftrightarrow x_1 < x_2 \vee (x_1 = x_2 \wedge y_1 < y_2)$ (Güting *et al*, 1995).

Definition 3) Halfsegment

A crucial idea for the representation of the relatively complex polygons values is to regard them as ordered sequences of *halfsegments* (Güting *et al*., 1995; Güting and Schneider, 1995). Let $S = \{(p, q) \mid p, q \in X \times Y, p < q \}$ denote the set of line segments in the $X \times Y$ plane, where p and q are end points. The equality of two segments $s_1 = (p_1, q_1)$ and $s_2 = (p_2, q_2)$ is defined as $s_1 = s_2 \Leftrightarrow (p_1 = p_2 \wedge q_1 = q_2) \vee (p_1 = q_2 \wedge p_2 = q_1)$. Without loss of generality, we normalize S by the assumption that $\forall s \in S : s = (p, q) \Rightarrow p < q$ which enables us to speak of a *left* and a *right end point* of a segment. Let further $H = \{(s, d) \mid s \in S, d \in \{left, right\}\}$ be the set of *halfsegments*. A halfsegment $h = (s, d)$ consists of an segment s and a flag d emphasizing one of the segment's end points which is called the *dominating point* of h. If $d = left$ then the left (smaller) end point of s is the dominating point of h, and h is called *left halfsegment*. Otherwise, the right end point of s is the dominating point of h, and h is called *right halfsegment*. Hence, each segment s is mapped to two halfsegments (s, *left*) and (s, *right*), as presented in Figure 2.

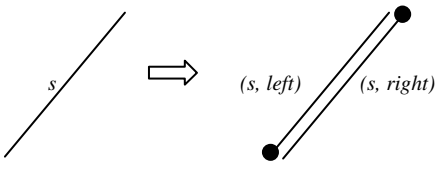


Figure 2 – The mapping of a segment in two halfsegments (*s*, *left*) and (*s*,*right*)

Let *dp* be the function which yields the dominating point of a halfsegment. For two distinct *halfsegments* h_1 and h_2 with a common end point *p*, let α be the enclosed angle such that $0 < \alpha \leq 180°$. Let a predicate *rot* be defined as follows: $rot(h_1, h_2)$ is true iff h_1 can be rotated around *p* through α to overlap h_2 in counter-clockwise direction. We can now define a complete order on *halfsegments* which is basically the (*x*, *y*)-lexicographic order by dominating points. For two *halfsegments* $h_1 = (s_1, d_1)$ and $h_2 = (s_2, d_2)$ it is:

$$h_1 < h_2 \Leftrightarrow dp(h_1) < dp(h_2) \vee (dp(h_1) = dp(h_2) \wedge ((d_1 = right \wedge d_2 = left) \vee (d_1 = d_2 \wedge rot(h_1, h_2)))) \quad (1)$$

Definition 4) Polygon

The polygon structure employed in this work was proposed by Güting *et al*. (1995) and Güting and Schneider (1995). In order to define a polygon, first it is need to define the concepts of cycle, hole and face. A cycle and a hole are sets of connected halfsegments. A face is a pair (*c*, *H*) where *c* is a cycle and $H = \{h_1, \ldots, h_m\}$, where each h_i is a hole (*H* can possibly be empty). Figure 3 presents an example of a polygon composed by three faces (*f* , *f'*, and *f''*). The face *f* is composed by the cycle *c* and the hole *h*. The face *f'* is composed by the cycle *c'* and the holes h_1' and h_2'. Finally, the face *f''* is composed by the cycle *c''* and it has no hole.

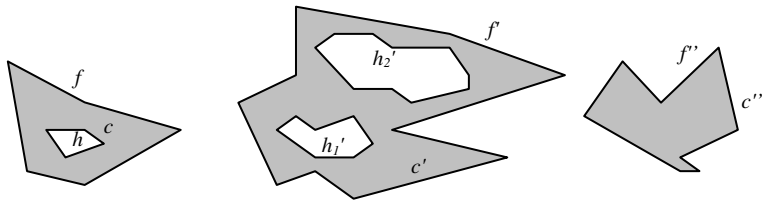
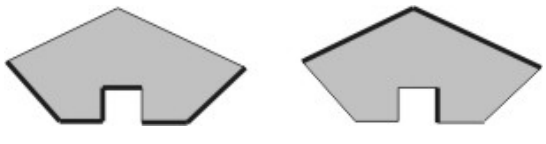


Figure 3 – Example of a polygon

In practice, a polygon is represented essentially as an ordered list (array) of halfsegments. The order used is the one suitable to support plane-sweep algorithms, basically lexicographic order on dominating points, presented in Definition 3. Each halfsegment has a set of attributes storing the cycle (or hole) and the face that it belongs. Besides, each halfsegment has an attribute named *edge number* that specifies the position of the halfsegment in the cycle that it belongs.

Definition 5) InsideAbove Flag of a Halfsegment



a) InsideAbove = true b) InsideAbove = false

Figure 4 – Examples of InsideAbove value.

The *InsideAbove* flag of a segment is true when the area inside the polygon lies above the segment; or, if the segment is a vertical line, it means that the area inside the polygon is on the left of its segment. Figure 4 presents examples of InsideAbove values.

Definition 6) PartnerNumber of a Halfsegment

As presented in Definition 4, a polygon is represented essentially as an ordered list (array) of halfsegments. The PartnerNumber attribute of a halfsegment stores the position of its partner in that array. In other words, the PartnerNumber attribute of a right dominating halfsegment is the position of its corresponding left dominating halfsegment in the array of halfsegments of the polygon, and vice-versa.

Definition 7) Turning Point

The turning point term has been introduced by Liang and Barsky (1983). A turning point is a point at the intersection of two clipping polygon edges that must be added to the clipped polygon in order to keep the connectivity of the original polygon. Figure 5.a presents an example of clipping a polygon by a window. In order to keep the connectivity of the original polygon it is needed to consider the edges corresponding to the turning points highlighted in Figure 5.b. In our work we extended the definition of turning point. We add a flag to the turning point, named *Direction* that stores the direction of the polygon's area on the edge that the turning point lies (*Left*, *Right*, *Up*, or *Down*), as presented in Figure 5.c. The resulting clipped polygon is presented in Figure 5.d.

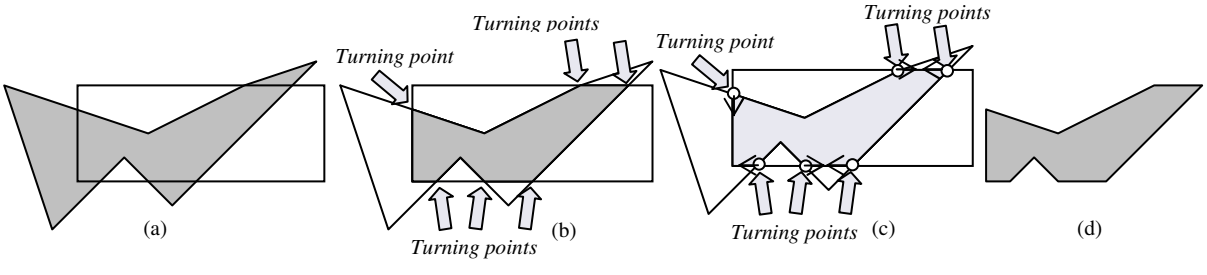


Figure 5 – Example of turning point.

Definition 8) Coverage Number of a Halfsegment

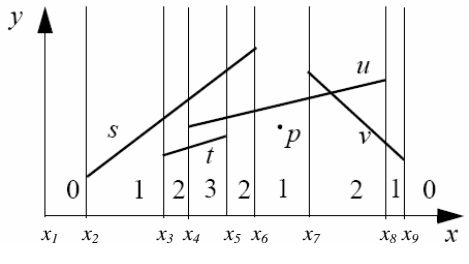


Figure 6 – Example of halfsegments including the coverage numbers of the vertical strips (Güting and Ding, 2004).

The coverage number of a halfsegment was defined by Güting and Ding (2004). Coverage number represents the number of segments that cross each vertical stripe of the plane between two x-coordinates. Figure 6 presents the coverage numbers for a set of halfsegments. In this example, two halfsegments cross the stripe between x_3 and x_4 coordinates. Güting and Ding (2004) present a simple algorithm to compute the coverage number of halfsegments in a single scan through an array of halfsegments.

3 Polygon Clipping

The polygon clipping algorithm has as input a set of halfsegments of a polygon, and produces a set of halfsegments corresponding to the portion of the polygon's halfsegments that are inside the window. In addition, new halfsegments corresponding to the turning points (Definition 7) are also returned. In other words, new halfsegments may be added, and existing halfsegments may be discarded, retained, or divided. Multiple polygons may also result from clipping a single polygon. It is important to emphasize that the polygon clipping algorithm, with few changes, can be used to return the portion of the polygon that is outside the window, instead of the portion that is inside the window. We have implemented both algorithms in Secondo; however, because of the space for this paper, we will present only the polygon clipping algorithm that returns the portion of the polygon inside the window.

In our proposal we use Sutherland-Cohen line clipping algorithm (Newman and Sproull, 1979) to clip the halfsegments against the window. We choose that algorithm because it is probably the most efficient method for trivial acceptance and rejection cases, which are both the most frequently encountered cases in window clipping. This algorithm can be implemented using either integer or floating point arithmetic; thus covering a wider set of applications (Maillot, 1992).

```
algorithm ClippingPolygonSegments
INPUT : HSA=<h1,h2,...hn> (Halfsegment Array)
        w = Rectangle
OUTPUT: cHSA = clipped halfsegments and the halfsegments
               resulting from the evaluation of turning points
cHSA = Ø;
turningPointSets = Ø;
FOR i=1 TO n DO
  IF (hi has left dominating point) THEN
    IF (SutherlandCohenLineClipping(hi, w, clippedhs, intersectionPoint,
        isIntersectionPoint)) THEN
      IF (isIntersectionPoint) THEN
        EvaluateTurningPoint(w, intersectionPoint, turningPointSets, hi);
      ELSE
        cHSA.Add(clippedhs);
        EvaluateTurningPoint(w, clippedH.leftPoint,  turningPointSets, hi);
        EvaluateTurningPoint(w, clippedH.rightPoint, turningPointSets, hi);
      END-IF;
    END-IF;
  END-IF;
END-FOR;
cHSA.Add(getTurningPointHalfSegments(turningPointSets));
```

Figure 7 – Algorithm for clipping polygon's segments by a window.

It is important to emphasize that the clipping of one halfsegment is completely independent of the clipping of any other halfsegment. Thus, it is possible to employ a parallel implementation. Besides, it is not needed to clip both left and right halfsegments. We can clip one type of halfsegment, and discard the other one. The only prerequisite is that the halfsegments must have the *InsideAbove* flag set. This flag is used to handle turning points. The fact that a fast algorithm for trivial rejection and trivial acceptance cases is used has oriented the method to spend most of the computing time evaluating the cases when a segment of the polygon boundaries is not completely rejected and not trivially accepted.

The algorithm for clipping the polygon's halfsegments by a window is presented in Figure 7. The *SutherlandCohenLineClipping* algorithm is used to compute the clipping. It has as input a halfsegment (h_i) and the window (w). The output may be a halfsegment (*clippedhs*) or a point (*intersectionPoint* is the point resulting from the intersection and *isIntersectionPoint* stores this information). If the intersection is a point, it is needed to evaluate this point as a turning points. In the case of a halfsegment intersection, the clipped halfsegment is added to the list (array) of the output halfsegments, and the evaluation of turning points is executed over the end points of the clipped halfsegment. Of course, if the halfsegment is completely inside the window and has no intersection point with the window (trivial acceptance), its end points do not need to be evaluated as turning points. This test is accomplished by the *EvaluateTurningPoint* sub-algorithm. This algorithm is presented in Sub-Section 3.1. The last step of the algorithm to clip halfsegments of a polygon is responsible for creating new halfsegments from the points that were considered as turning points. This algorithm is described in Sub-Section 3.2.

```
algorithm EvaluateTurningPoint
INPUT : w = a rectangle described by the coordinates (x_min, y_min) and (x_max, y_max)
        p = Point
        turningPointSets = for each window egde there is a set recording the
                           turning point of the edge
        h = halfsegment that the point p belongs to
OUTPUT: If the point is evaluated as a turning point it is added to the Turning
        Point Set of the edge
  tp = p;
  IF (p.x = w.x_min) THEN //left edge
    IF (h.insideAbove) THEN
      tp.direction = UP;
    ELSE
      Tp. direction = DOWN;
    END-IF;
    turningPointSets[LEFT].add(tp);
  ELSE //right edge
    IF (h.insideAbove) THEN
      tp. direction = UP;
    ELSE
      tp. direction = DOWN;
    END-IF;
    turningPointSets[RIGHT].add(tp);
  END-IF;
  IF (p.y = w.y_min) THEN //bottom edge
    IF (h.leftPoint > w.y_min) THEN
      tp.direction = GetDirection(p, h.leftPoint,  x_min, y_min, h.insideAbove);
    ELSE
      tp.direction = GetDirection(p, h.rightPoint, x_min, y_min, h.insideAbove);
    END-IF;
    turningPointSets[BOTTOM].add(tp);
  ELSE
    IF (p.y = w.y_max) THEN  //top edge
      IF (h.leftPoint > w.y_min) THEN
        tp.direction = GetDirection(p, h.leftPoint, x_min, y_min, h.insideAbove);
      ELSE
        tp.direction = GetDirection(p, h.rightPoint, x_min, y_min, h.insideAbove);
      END-IF;
      turningPointSets[BOTTOM].add(tp);
    END-IF;
  END-IF;
```

Figure 8 – Algorithm to evaluate turning points.

3.1 Evaluating Turning Points

The turning point evaluation algorithm uses the *InsideAbove* flag (Definition 5) to define how a point must be handled for creating new edges. Only the points that lie on window's

edges are considered as turning points. Figure 8 presents the algorithm used to evaluate turning points.

According to the turning point evaluation algorithm (Figure 8), it is easy to define the direction of the turning points that lie on the vertical window's edges (left and right edges). That is because when the *InsideAbove* flag of the halfsegment that the turning point belongs is true, then the polygon is above the turning point, and the direction is *UP*. Otherwise, if the *InsideAbove* flag has value equal to false, the polygon is under the halfsegment, and the direction of the turning point is *DOWN*. On the other hand, the same reasoning does not apply when handling turning points that lie on the horizontal edges (bottom and top edges). The *InsideAbove* flag's value is not enough to define the turning point direction. An additional test must be executed. This test has just to determine whether the area of the polygon is to the right or to the left of the turning point. Figure 9 presents the algorithm that returns the direction of turning points that lie on top/bottom window's edges.

```
algorithm GetDirection
INPUT: tp = turning point
        p = point of the same half segment that the turning point tp belongs
            and is above tp
       (x, y) = the left coordinate of the vertex of the window edge
        insideAbove = insideAbove flag's value
  IF (insideAbove) THEN
    IF (tp.x > p.x) THEN
      return RIGHT;
    ELSE
      return LEFT;
    END-IF;
  ELSE
    IF (tp.x > p.x) THEN
      return LEFT;
    ELSE
      return RIGHT;
    END-IF;
  END-IF;
```

Figure 9 – Algorithm to compute the direction of turning points that lie on the top/bottom window's edges.

3.2 Creating New Segments from Turning Points

The algorithm that creates new segments from turning points basically sort the turning points accordingly with the *x* and *y*-axis and point's direction. Afterwards, it connects properly the turning points to produce the new segments. The algorithm that creates new halfsegments from turning points is presented in Figure 10.

```
algorithm CreateNewSegments
INPUT: edge = indicates which edge is been handled(LEFT, RIGHT, TOP or
                                                   BOTTOM)
        bPoint,ePoint = end points of the edge
        turningPointSet = a set of the turning points of the edge
        cHSA = set of half segments in which the new half segments will be added

OUTPUT: cHSA with the new half segments
 IF edge == TOP or edge == LEFT THEN
   InsideAbove = false;
 ELSE /*RIGHT or BOTTOM edges*/
   InsideAbove = true;
 END-IF;
 begin = 0;
 end = turningPointSet.size();
 tp = turningPointSet[begin];
 IF (tp.Direction== LEFT or tp.Direction == DOWN) and not tp.Rejected THEN
     cHSA.addHalfSegments(tp,bPoint,InsideAbove);
     DiscardTurningPoints(turningPointSet, tp, ASCENDIN_GORDER, begin);
 END-IF;
 tp = turningPointSet[end];
 IF (tp.Direction== RIGHT or tp.Direction == UP) and not tp.Rejected
    and there is no rejected turning point equals to tp THEN
     cHSA.addHalfSegments(tp,ePoint,InsideAbove);
     DiscardTurningPoints(turningPointSet, DESCENDING_ORDER, end);
 END-IF;
 WHILE (begin<end) DO
  tp1 = GetNotRejectedTurningPoint(turningPointSet, ASCENDING_ORDER, begin);
  IF tp1 == NULL THEN
    return;
  END-IF;
  tp2 = GetNotRejectedTurningPoint(turningPointSet, DESCENDING_ORDER, end);
  IF tp2 == NULL THEN
    return;
  END-IF;
  cHSA.addHalfSegments(tp1,tp2,InsideAbove);
END-WHILE;
```

Figure 10 – Algorithm to create new halfsegments from turning points.

4 Polygon Reconstruction Algorithm

The polygon reconstruction algorithm has as input a set of halfsegments. The halfsegments do not have any information about which polygon's part they belong to (face, cycle or cycle's edge). The algorithm cross the halfsegments, and adjusts properly the face number, cycle number, and edge number (which we named as polygon attributes of a halfsegment), according to the definition of polygon presented in Definition 4. This algorithm can be used to reconstruct any kind of polygon from its halfsegments. For example, it can be used to reconstruct a polygon from a set of halfsegments resulting from clipping a polygon against another polygon. The algorithm is presented in Figure 11. Two sub-algorithms are called by the reconstruction polygon algorithm. They are *ComputeCycle* and *GetFaceNumber*. The algorithm *ComputeCycle* sets the face number, cycle number and edge number of halfsegments that belong to a particular cycle. The algorithm to get face numbers

(*GetFaceNumber*) returns the face number that a halfsegment belongs, or it returns -1, indicating that the halfsegment does not belong to any face that was processed yet.

```
algorithm PolygonReconstruction
INPUT: HSA=<h1,h2,...hn> (Halfsegment Array)
OUTPUT: HSA = each halfsegment has the face, cycle, and edge numbers set

VARIABLES: face = array that stores in position i the last cycle number of the
                  face i
           hsSet = array that stores in the position i if the half segment hi
                   had already the face number, the cycle number and the edge
                   number set. This array is initialized with values false.
  IF HSA is not sorted in halfsegment order THEN
    Sort HSA in halfsegment order;
  END-IF;
  IF the halfsegments of HSA do not have the partner number set THEN
    Set partner number of the halfsegments of HSA;
  END-IF;
  face[0] = 0; /*0 is assigned to the first cycle of the first face */
  lastFaceNumber = 0;
  isFirstHS = true;
  FOR i=1 TO n DO
    IF hi has left dominating point and not hsSet[i] THEN
      IF isFirstHS THEN
        isFirstHS = false;
        hi.faceNumber = 0;
        hi.cycleNumber = 0;
      ELSE
        existingFaceNumber = GetFaceNumber(HSA, hi, hsSet, i);
        IF existingFaceNumber is equal to -1 THEN
          lastFaceNumber++;
          hi.faceNumber = lastFaceNumber;
          hi.cycleNumber = 0;
          /*to store the first cycle number of the face lastFace*/
          face[faceNumber-1]=0;
        ELSE
          hi.faceNumber = existingFaceNumber;
          face[faceNumber]++;
          hi.cycleNumber = face[faceNumber];
        END-IF;
      END-IF;
      hi.edgeNumber = 0;
      ComputeCycle(HSA, hi, hsSet);
    END-IF;
  END-FOR;
```

Figure 11 – Algorithm for polygon reconstruction

5 Implementations in Secondo

Secondo (Diker and Güting, 2000; Güting *et al.*, 2005) is a new generic environment supporting the implementation of database systems for a wide range of data models and query languages. It is developed as a research prototype at the Fernuniversität in Hagen. The implementation of each algebra in Secondo is based on the concept of second-order signature (Güting, 1993) with the first signature describing type constructors and the second signature describing operations on these type constructors. An algebra can be plugged into Secondo with the central part of the Secondo code unchanged. After recompiling Secondo, we can use the newly added algebra.

In Secondo, the operators *windowclippingin* and *windowclippingout* were implemented in SpatialAlgebra (Figure 12). The operators can handle line and polygon (which are named as region), returning the portion inside or outside a rectangle window, respectively. If it receives as input a line and a rectangle window, it returns a clipped line. If it receives as input a region and a rectangle window, a clipped region is returned. The polygon reconstrucion algorithm was implemented as methods in the Region class of Spatial Algebra, and it is used by the clipping algorithms to compute the clipped polygons.

```
Name: windowclippingin
Signature: (line x rect) -> line, (region x rect) --> region
Syntax:   windowclippingin( _, _ )
Meaning:  computes the part of the object that is inside the window.
Example:  query Flaechen feed extend[InWindow: windowclippingin(
                  .geoData, bbox(thecenter))] project[InWindow]
                  filter[not(isempty(.InWindow))] consume
Name: windowclippingout
Signature: (line x rect) -> line, (region x rect) --> region
Syntax:   windowclippingout( _, _ )
Meaning:  computes the part of the object that is outside the window.
Example:  query windowclippingout(trajectory(train7), bbox(thecenter))
```

Figure 12 - Clipping operators implemented in Spatial Algebra in Secondo

6 Conclusion

Polygon clipping is an important operation that computers execute all the time. Often, it is possible to feed a weird polygon to an algorithm and retrieve an incorrect result. One of the vertices may disappear, or a ghost vertex may be created. There are many clipping polygon proposals in the literature. However, none of them is totally perfect. Therefore, the hunt for the perfect clipping algorithm is still open. In this work, we proposed two new algorithms: polygon clipping by a rectangle window; and polygon reconstruction.

Polygon reconstruction is a consequence of polygon clipping. In other words, clipping a polygon produces a set of unordered segments. These segments must be handled in order to adjust segments attributes to produce the polygon. So, we proposed a polygon reconstruction algorithm that may be used in any case where it is needed to compute a polygon from an unordered set of segments. For instance, compute a polygon from a set of segments produced by clipping a polygon by a rectangle window or clipping a polygon by another polygon.

In the case of the polygon clipping algorithm, we proposed and implemented two possibilities, an algorithm to return the portion of the polygon that are inside a rectangle window, and an algorithm to return the portion of the polygon that are outside the window. The algorithms were implemented in Secondo system (Diker and Güting, 2000; Güting *et al.*, 2005), a platform for implementing and experimenting with various kinds of data models. Hence we employed the data structures described in the ROSE algebra (Güting, 1993; Güting *et al*, 1995; Güting and Schneider, 1995). Althoug our algorithm

implementations in Secondo has a good performance, we do not execute an experimental evaluation against other similar algorithms, which we plan to do as future work.

7 References

Dieker, S. and Güting, R. H. (2000) "Plug and Play with Query Algebras: SECONDO A Generic DBMS Development Environment", In: Proceedings of the International Database Engineering and Applications Symposyum (IDEAS), Japan, September.

Gargano, M., Nardelli, E., and Talamo, M. (1991) "Abstract data types for the logical modeling of complex data", Information Systems, 16(5):565-584.

Güting, R. H. (1993) "Second-order signature: A tool for specifying data models, query processing, and optimization", In: ACM SIGMOD Record, vol. 22 , issue 2 (June), pp. 277 - 286.

Güting, R.H. and Schneider, M. (1995) "Realm-Based Spatial Data Types: The ROSE Algebra", VLDB Journal 4, 100-143.

Güting, R.H., and Ding, Z. (2004) "A Simple But Effective Improvement to the Plumbline Algorithm", Information Processing Letters 91 (2004), 251-257.

Güting, R.H., Almeida, V., Ansorge, D., Behr, T., Ding, Z., Höse, T., Hoffmann, F., Spiekermann, M. (2005) "SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching", In: 21st Intl. Conf. on Data Engineering (ICDE, Tokyo, Japan), 2005, 1115-1116.

Güting, R.H., de Ridder, Th., Schneider, M. (1995) "Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types", In: Proceedings of the 4th International Symposium on Large Spatial Databases, Portland, August.

Liang, Y. and Barsky, B. (1983) "An analysis and algorithm for polygon clipping", Commun. ACM 26, 11 (Nov), 868-877.

Mayllot, P.-G. (1992) "A new, fast method for 2D polygon clipping: analysis and software implementation", In: ACM Transactions on Graphics (TOG), v.11,issue 3, p.276-290, July.

Newman, W. M., and Sproull, R. F. (1979) "Principles of Intemctiue Computer Graphics", McGraw-Hill Book Company.

Scholl, M. and Voisard, A. (1989) "Thematic map modeling", In: Proceedings of the First International Symposium on Large Spatial Databases, Santa Barbara, CA, 1989.

Voisard, A. (1992) "Bases de donn6es g6ographiques: du module de donn6es fi l'interface utilisateur". Ph.D. Thesis, University of Paris-Sud (Centre d'Orsay).

Weiler, K. and Atherton, P. (1977) "Hidden surface removal using polygon area sorting", In: Proceedings of the 4th annual conference on Computer graphics and interactive techniques, San Jose, California, pp. 214 - 222.