STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

SIGNEDINTERSECTION - A NEW ALGORITHM FOR FINDING THE INTERSECTION OF TWO SIMPLE POLYGONS

ANDREEA SABAU

Abstract. The operation of determining the intersection of two polygons is one of the most important operations of computational geometry. This paper presents a new algorithm called `SignedIntersection` which founds the intersection of two simple polygons, convex or concave, without holes, in two steps. The first step is using a sweep line in order to find the intersection points and segments that form the polygons' intersection. This data is enriched with additional values that indicate the way the sweep line traverse the segments, if the vertexes are given in counter-clockwise order. Such an additional value is a sign, positive or negative. The final step consists in building the result using the data determined by the previous step. The result of the intersection of two (possible) non-convex polygons may be empty or may consist of one or more polygons, convex or concave.

Key words: Computer graphics, Polygons intersection, Convex polygons

1. Introduction

Determining the intersection of two polygons (in the 2D space) is one of the basic operations of the computational geometry. This operation is usually used within spatial data systems (like GIS), CAD and computer graphics.

The algorithms that determine the intersection of two polygons usually receive as input data two convex polygons [4, 6], or two polygons such as at least one of them is not convex (it is concave) [2, 3]. More, there are also algorithms that can handle polygons which may have holes [8]. The algorithms from the first category are easier to implement, based on simpler computations. The one presented in [6] is based on computing the convex hull of the two input polygons. The convex hull of the two polygons contains two segments that do not belong to any of the two polygons; starting with one of

Received by the editors: May 1, 2008.

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.3.5. [**Computing Methodologies**]: Computer Graphics – *Computational Geometry and Object Modeling*.

them, the vertexes (and the edges) of the polygons are traversed toward the other convex hull's considered segment in order to determine the intersection (which is also a convex polygon). Some algorithms belonging to the second category are splitting the input polygons into convex parts and apply a method for convex polygons [1, 7] (usually - trapezoids), followed by the reunion of the intermediate results. The paper [8] describes an algorithm that runs in three steps in order to find the intersection of two polygons, possibly with holes. First, the intersection points of the two input polygons are found using a sweep line. Some navigational data (numerical data) is associated with each intersection point during the second step. The last part of the algorithm determines the intersection by traversing the polygon edges and intersection points.

This paper presents another algorithm that determines the intersection of two simple (non-self-intersecting) polygons, convex or concave. The next section presents the data structures used by the algorithm and the algorithm itself.

2. The `SignedIntersection` Algorithm

An algorithm called `SignedIntersection` that builds the intersection of two simple, convex or concave, polygons is presented in this section. This algorithm is original, according to the author's knowledge, in the way it processes and analyzes data, the used data structures, and the manner in which the result is determined. This algorithm was implemented in order to be used within the 3SST relational data model [5]. Therefore this algorithm works with data stored within relations on a Microsoft SQL-Server [5] and it is written in the Transact-SQL language.

The algorithm's input data is given by two simple polygons, convex or concave. According to the 3SST relational data model, a polygon P is represented by the list of its vertexes, given in the counter-clockwise order.

Let R and Q be the two polygons considered as input data, where R = (R_1, R_2, ..., R_n) and Q = (Q_1, Q_2, ..., Q_m). R_i, i:=1..n, n≥3, are the R's vertexes, and Q_j, j:=1..m, m≥3, are the Q polygon's m vertexes. The two coordinates of a vertex V will be noted as V.x, and V.y respectively.

The main step of the `SignedIntersection` algorithm consists in sweeping the plane with a line (the sweep line) parallel with the Oy axis, beginning with the R's or Q's vertex with the minimum x coordinate. If there are (at least) two such points the whole ensemble can be rotated so that only one vertex has the minimum x coordinate. As the sweep line is moving toward the vertex with the maximum x coordinate, the segments that are forming the intersection's final result are determined. The last step of the presented algorithm consists in analyzing the segments previously found and building the intersection polygons (none, one or more such polygons).

2.1. The `SignedIntersection` Algorithm's Data Structures.

The data structures used during the execution of the `SignedIntersection` algorithm are arrays of which elements contain the following elements:

`Points [PID, x, y, PgID]`

- `PID` - one polygon's vertex identifier (`PID` is the unique identifier in `Points` table of the 3SST relational data model [5]),
- `x, y` - the coordinates of the point identified by `PID`,
- `PgID` - the identifier of the polygon of which one of the vertexes is `PID`.

`Segments [SgID, PID1, PID2]`

- `SgID` - segment identifier,
- `PID1, PID2` - the end-points of the segment given by `SgID`. `PID1` is the start end-point and `PID2` is the final end-point, in accordance with the counter-clockwise order of the polygons' vertexes.

`Intersections [PID, x, y, SgID1, SgID2]`

- `PID` - the identifier of an intersection point between two segments of polygons R, and Q respectively,
- `x, y` - the coordinates of the point identified by `PID`,
- `SgID1, SgID2` - the identifiers of the segments from whose intersection resulted `PID`.

`Overlapping [PID1, PID2]`

- `PID1, PID2` - the identifiers of the end-points of the overlapping between two segments or a vertex and a segment of the two polygons; if the overlapping is given by a single point, then `PID1` = `PID2`; `PID1` and `PID2` always represent polygon vertexes.

`OrderPoints [PID]`

- `PID` - the identifier of a point from `Points` or `Intersections` data structures.

`Stack [position, SgID, sgn, sw_y]`

- `SgID` - a segment identifier, from R or Q,
- `sgn` - the way the segment is swept according to the sweep line; if the sweep line goes first through the start end-point of the segment, then `sgn` = +1, else `sgn` = -1,

- `sw_y` - the y coordinate of the intersection between the segment and the sweep line (the x coordinate of the intersection is given by the position of the sweep line).

`Results [SgID, sgn, PID1, PID2, checked]`

- `SgID` - a segment identifier, from R or Q,
- `sgn` - the way the segment is swept according to the sweep line (this value is taken from one `Stack`'s entry),
- `PID1, PID2` - the end-points of the segment or of the part of the segment which is included in the final result of the polygons' intersection,
- `checked` - a Boolean value, used in order to build up the final result; indicates if the segment has already been analyzed or not.

In this point the following observation has to be made. The `PID` values from the `Points` and `Intersections` lists are unique (such a value uniquely identifies an intersection point or one of the polygons' vertexes).

The `OrderPoints` and `Stack` lists are sorted. Without describing the operations, any insertion or deletion in / from these lists is maintaining the sorting order. The items of the `OrderPoints` structure are sorted according to the x coordinate of the point identified by `PID`. The list `OrderPoints` contains all the polygons' vertexes and intersection points after the sweep line completely swept both polygons. If there are two (or more) points having the same value of their x coordinate, the sorting is made according to the y coordinate. Specifically, the order in which the points of the `OrderPoints` list are stored denotes the order in which the sweep line encounters the two polygons' vertexes. The segments of the `Stack` list are sorted in the ascending order of the `sw_y` values. The `Stack` list contains at a specific moment all the two polygons' segments which are currently intersected by the sweep line. These segments' order is given by the y coordinate of their intersection points with the sweep line.

2.2. **The SignedIntersection Algorithm.**

The algorithms that determine intersection points or intersection surfaces usually use the sweep line technique. The novelty and the name of the presented algorithm come from the importance of the way the polygon's edges are traversed by the sweep line. Also, there are identified four types of vertexes of a polygon and the segments in the `Stack` list are managed according to these types. The types of vertexes and the manner in which the segments are inserted, updated, or deleted in / from `Stack` are outlined next.

Let R be a simple concave polygon given by the list of vertexes R = (R_1, R_2, R_3, R_4, R_5, R_6, R_7) (see figure 1) and a sweep line parallel with the Oy axis. The order in which the vertexes are traversed by the sweep line is (R_1, R_7, R_2, R_6, R_4, R_3, R_5). The manner in which the edges of this polygon are managed in `Stack` is exemplified next. R_1 is called **extreme left vertex** of the R polygon, in which case both segments that leave from R_1 are inserted in `Stack`: R_1R_2 is inserted with `sgn`=+1 because the way the sweep line traverses it is from the initial end-point (R_1) through the final end-point (R_2); R_1R_7 is inserted with `sgn`=-1 because the seep line traverses the segment as the vertexes would be taken in clockwise order (the segment R_1R_2 is inserted in `Stack` under the segment R_1R_7 because, even if R_1 belongs to both segments, R_2 has a smaller y coordinate than R_7). Next, R_7 is called **transition point with negative sign**, in which case the segment finished by R_7 is replaced with R_7R_6 in `Stack` (also with `sgn`=-1). R_2 is called **transition point with positive sign**, in which case the segment finished by R_2 is replaced with the segment that starts at R_2 (the segment R_1R_2 is replaced with the segment R_2R_3 with `sgn`=+1). R_7R_6 is replaced with R_6R_5 in `Stack` when the vertex R_6 is encountered. R_4 is an other extreme left vertex of R, therefore the segments R_4R_3 and R_4R_5 are inserted in `Stack` between R_2R_3 and R_6R_5. The vertex R_3 is called **extreme right vertex**. This vertex indicates the moment when the sweep line has just finished traversing the segments R_2R_3 and R_4R_3, therefore they are deleted from `Stack`. R_4R_5 and R_6R_5 are deleted from `Stack` when R_5 is swept finally by the sweep line. Having two polygons, their individual segments are managed within the list `Stack` in the same manner as presented above.

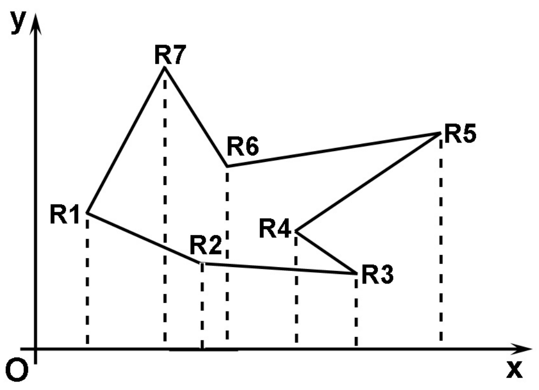

FIGURE 1. The simple concave polygon R given by R = (R_1, R_2, R_3, R_4, R_5, R_6, R_7), where the vertex R_1 has the minimum x coordinate.

The operation of determining the intersection's result is described in the algorithm presented below. The algorithm also handles the case when two segments of the two considered polygons are overlapping.

The following functions are considered to exist, without specifying their implementation details:

- `no_elem(L)` - returns the number of elements of one of the list structures (L) used by the `SignedIntersection` algorithm,
- `y_int(SgID, SWL)` - returns the y coordinate of the intersection point between the segment identified by `SgID` and the sweep line `SWL`,
- `coord_x(PID)` - determines the x coordinate of the point identified by `PID` (from the list `Points` or `Intersections`),
- `sgn(SgID)` - returns the sign of the segment `SgID` during the sweep line's movement through the polygons' vertexes.

Three routines are presented next: `InitializeStructures` initializes the lists `Points`, `Segments`, `Intersections`, `Stack`, and `Results`; `SignedIntersection` determines the intersection of the two given polygons; `ShowResults` analyzes and prints the intersection's result.

```
 InitializeStructures(R, Q)
// Input:
//    R, Q - polygons given by their vertexes lists,
//    R = (R1, R2, ..., Rn) and Q = (Q1, Q2, ..., Qm),
//    and identified by IdR, and IdQ respectively

   // Initialize the lists Points, Segments, Intersections,
   // Stack, Results
   For each point Ri, i:=1..n, do
     Insert the entry (Ri.PID, Ri.x, Ri.y, IdR) in Points
     Insert the entry (Ri.PID) in OrderPoints
     Let SgID be a new segment identifier
     Insert the entry (SgId, Ri.PID, R(i+1) MOD n.PID) in
       Segments
   endfor
   // The points Qi, i:=1..n, are handled in the same manner
   // as the vertexes of R
end InitializeStructures

 SignedIntersection(R, Q)
// Input:
//    R, Q - polygons given by their vertexes lists,
//    R = (R1, R2, ..., Rn) and Q = (Q1, Q2, ..., Qm),
//    and identified by IdR, and IdQ respectively
```

```
InitializeStructures(R, Q)

crt_pos:=1
// the position of the current point within OrderPoints

While crt_pos < no_elem(OrderPoints) do
  SWL:=coord_x(OrderPoints[crt_pos].PID)
  // SWL is the sweep line and its current position is
  // given by the x coordinate of the current point
  // (polygon vertex or intersection point)
  For each i:=1..no_elem(Stack) do
    // Update the y coordinate of the intersection point
    // between SWL and each segment within Stack
    Stack[i].sw_y:=y_int(Stack[i].SgID, SWL)
  endfor

  If OrderPoints[crt_pos].PID is in Points then
    // This point is a **polygon vertex**
    Let s1 be the identifier of the segment from Segments
      for which OrderPoints[crt_pos].PID is the *initial*
      end-point
    Let s2 be the identifier of the segment from Segments
      for which OrderPoints[crt_pos].PID is the *final*
      end-point
    Let Pg_complem be the identifier of the polygon such as
      OrderPoints[crt_pos].PID is not its vertex

    If (∄ k such as Stack[k].SgID=s1) and
      (∄ l such as Stack[l].SgID=s2) then
      // If neither the segment s1 nor s2 are in Stack
      // then OrderPoints[crt_pos].PID is **extreme left**
      // **vertex** of the polygon
      HandleExtremeLeftVertex(s1, s2, Pg_complem)
    else
    If (∃ k such as Stack[k].SgID=s1) and
      (∃ l such as Stack[l].SgID=s2) then
      // If both segments s1, s2 are in Stack then
      // OrderPoints[crt_pos].PID is **extreme right vertex** of
      // the polygon
      Delete from Stack the entries where SgID in s1, s2
      // The segments that have been "terminated" by the
      // current point are deleted from Stack
    else
    If ($\nexists$ k such as Stack[k].SgID=s1) and
      ($\exists$ l such as Stack[l].SgID=s2) and
      ($\nexists$ m such as
      Overlapings[k].PID1=OrderPoints[crt_pos].PID) then
      // The current point is **transition point with positive**
      // **sign** and it is not an overlapping point
      HandleTransitionPointPositiveSign(s1, s2)
    else
    If ($\exists$ k such as Stack[k].SgID=s1) and
      ($\nexists$ l such as Stack[l].SgID=s2) and
      ($\nexists$ m such as
      Overlapings[m].PID1=OrderPoints[crt_pos].PID) then
      // The current point is **transition point with negative**
      // **sign** and it is not an overlapping point
      HandleTransitionPointNegativeSign(s1, s2)
    else
      // The current point is a **transition point** and an
      // **overlapping point**
      If $\exists$ k such as
        Overlapping[k].PID1=OrderPoints[crt_pos].PID then
        Let s1 and s2 be the segments of the two polygons
          for which the current point is a right end-point
          (maximum x coordinate) or an overlapping point
        HandleInitialOverlappingPoint(s1, s2)
      else
      If $\exists$ k such as
        Overlapping[k].PID2=OrderPoints[crt_pos].PID
        and Overlapping[k].PID1<>Overlapping[k].PID2 then
        Let s1 and s2 the segments of the two polygons for
          which the current point is a left end-point
          (minimum x coordinate) or an overlapping point.
          Consider s1.PID2.x<s2.PID2.x
        HandleFinalOverlappingPoint(s1, s2)
      endif; endif
    endif; endif; endif; endif
    CheckIntersections(Stack)
  else // The current point is an **intersection point**
        // and it is not an overlapping point
    Let s1 and s2 be the segments that determined the
```

```
        current intersection point
       Invert the segments s1 and s2 in Stack
       // The sweep line reached the intersection point of the
       // two segments
       HandleIntersectionInResults(s1)
       HandleIntersectionInResults(s2)
       CheckIntersections(Stack)
     endif
     crt_pos:=crt_pos+1
   endwhile

   If no_elem(Results)>0 then
     ShowResults(Results)
   endif
End SignedIntersection

 HandleExtremeLeftVertex(s1, s2, Pg_complem)
// Input:
//    s1, s2 - two segments that have as left end-point the
//    current point
//    Pg_complem - the polygon that does not contain s1 and s2

   Insert the entry (s1, +1, y_int(s1, SWL)) in Stack
   Insert the entry (s2, -1, y_int(s2, SWL)) in Stack
   If the point OrderPoints[crt_pos].PID is inside the
     Pg_complem polygon then
     // (inside the polygon) or (on the frontier and the
     // other end-points of s1 and s2 are inside the
     // polygon)
     Let PID1, PID2 be the initial and final end-points of s1
     Insert the entry (s1, +1, PID1, PID2) in Results
     Let PID1, PID2 be the initial and final end-points of s2
     Insert the entry (s2, -1, PID2, PID1) in Results
   endif
end HandleExtremeLeftVertex

 HandleTransitionPointPositiveSign(s1, s2)
// Input:
//    s1, s2 - the segments for which the current point makes
//    a transition with positive sign; the current point
//    finishes the sweeping of s2 and starts the sweeping of
//    s1

   // Replace s2 with s1 in Stack
   Stack[k].SgID:=s1, where k such as Stack[k].SgID:=s2
   If ∃ k such as Results[k].SgID=s2 and Results[k].PID2 is
     the initial end-point of the s1 segment then
     Let PID1, PID2 be the initial and final end-points of s1
     // Insert s1 in Results
     Insert the entry (s1, Stack[k].sgn, PID1, PID2) in
       Results
   endif
end HandleTransitionPointPositiveSign

 HandleTransitionPointNegativeSign(s1, s2)
// Input:
//    s1, s2 - the segments for which the current point makes
//    a transition with negative sign; the current point
//    finishes the sweeping of s1 and starts the sweeping of
//    s2

   // Replace s1 with s2 in Stack
   Stack[k].SgID:=s2, where k such as Stack[k].SgID:=s1
   If ∃ k such as Results[k].SgID=s1 and Results[k].PID2
     is the final end-point of the s1 segment then
     Let PID1, PID2 be the initial and final end-points of s2
     // Insert s2 in Results
     Insert the entry (s2, Stack[k].sgn, PID2, PID1) in
       Results
   endif
end HandleTransitionPointNegativeSign

 HandleInitialOverlappingPoint(s1, s2)
// Input:
//    s1, s2 - two segments that are overlapping; the current
//    point represents the initial end-point of the
//    overlapping segment

   // Handle s1 and s2 as in the case when the point current
   // is an intersection point
   If ∃ k such as Results[k].SgID=s1 and
     Results[k].PID2 is a polygon vertex then
```

```
    Results[k].PID2:=OrderPoints[crt_pos].PID
   endif
   If ∃ k such as Results[k].SgID=s2 and
    Results[k].PID2 is a polygon vertex then
    Results[k].PID2:=OrderPoints[crt_pos].PID
   endif
   If Overlapping[k].PID1<>Overlapping[k].PID2 then
    Delete from Stack the entries where SgID in s1, s2
    Insert in Stack the segment determined by
     Overlapping[k].PID1 and Overlapping[k].PID2
    Insert in Results two entries corresponding to the two
     segments that overlap, given by Overlapping[k].PID1
     and Overlapping[k].PID2
   endif
end HandleInitialOverlappingPoint
```

```
 HandleFinalOverlappingPoint(s1, s2)
// Input:
//    s1, s2 - two segments that are overlapping; the current
//    point represents the final end-point of the overlapping
//    segment

   Insert in Stack two entries corresponding to the
    segments s1 and s2
   If sgn(s1)=+1 then
    If sgn(s2)=+1 then
     Insert in Results entries corresponding to s1 and s2
    else
     Insert in Results entry corresponding to s2
    endif
   else
    If sgn(s2)=+1 then
     // Insert no entry in Results
    else
     Insert in Results entry corresponding to s1
    endif
   endif
end HandleFinalOverlappingPoint
```

```
 HandleIntersectionInResults(s)
// Input:
```

```
//    s - one of the two segments involved in an intersection;
//    the current point is the corresponding intersection
//    point

   // Handle segment s in Results
   If ∃ k such as Results[k].SgID=s and
     Results[k].PID2 is a polygon vertex then
     // The part of the segment s that belongs to the
     // intersection is finished by the current point
     Results[k].PID2:=OrderPoints[crt_pos].PID
   else
     If (∄ k such as Results[k].SgID=s) or
       (∃ k such as Results[k].SgID=s1 and Results[k].PID2 is
       an intersection point) then
       Insert the entry
         (s, Stack[l].sgn, OrderPoints[crt_pos].PID, P) in
         Results, where l such as Stack[l].SgID=s, and P is
         the final end-point of s (if Stack[l].sgn=+1) or P
         is the initial end-point of s1 (if Stack[l].sgn=-1)
     endif
   endif
end HandleIntersectionInResults
```

The CheckIntersections routine checks whether two neighbor segments in Stack intersect when they belong to different polygons. If a new intersection point Pi is found, such as Pi is not an end-point, then it is inserted into list Intersections. If Pi is a polygon vertex such as it is the final end-point of a segment s1 and the initial end-point of a segment s2, and belongs to a segment's interior of the other polygon (s) then Pi is inserted in Overlapping as PID1. If s2 and s are overlapping then the pair of the two points that determine the overlapping segment is inserted in Overlapping.

```
 ShowResults(Results)
// Input:
//    Results - the list of segments that determine the result
//    of the two polygons' intersection

   While ∃ k such as Results[k].checked=false do
     // Initiate the building of a new intersection polygon
     Let P1 be the point PID1 or PID2 from Results which has
       the minimum x coordinate and is an end-point of a
```

```
      segment s′ with sgn=+1.  Let l be the position of P1
      in Results
    Let P_init:=P1
    Let P2 be the other end-point of s′
    While P_init P2 do
      Print P1, P2
      Results[k].checked:=true
      P1:=P2
      Find l′ such as Results[l′].checked=false and
        Results[l′].PID1=P1 or Results[l′].PID2=P1
    endwhile
    Show P1, P2
  endwhile
end ShowResults
```

The SignedIntersection algorithm determines the intersection of two simple polygons, where the intersection can be empty or can consists of one or more simple polygons.

3. Conclusions and Future Work

In this paper a new approach to determine the intersection between two simple polygons has been proposed. The presented algorithm uses the well-known technique of the sweep line and assigns a special sign value to each of the polygons' edges in order to find the intersection result. Also, the sign of each segment that belongs to the intersection's frontier is used to build the polygons' intersection. The SignedIntersection algorithm and the corresponding data structures are easy to implement on top of the 3SST relational data model.

It is intended to extend the SignedIntersection algorithm in order to be able to determine the intersection of two polygons with or without holes.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[2] A. Margalit, G. D. Knott, *An Algorithm for Computing the Union, Intersection or Difference of Two Polygons*, Computers & Graphics, Vol. 13 (2) (1989), pp. 167-184.

[3] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1993.

[4] J. O'Rourke, C.B. Chien, T. Olson, D. Naddor, *A New Linear Algorithm for Intersecting Convex Polygons*, Computer Graphics and Image Processing, No. 19 (1982), pp. 384-391.

[5] A. Sabau, *The 3SST Relational Model*, Studia Universitatis "Babes-Bolyai", Informatica, Vol. LII (1) (2007), pp. 77-88.
[6] G. T. Toussaint, *A Simple Linear Algorithm for Intersecting Convex Polygons*, The Visual Computer, Vol. 1 (1985), pp. 118-123.
[7] B. Zalik, G. Clapworthy, *A Universal Trapezoidation Algorithm for Planar Polygons*, Computers & Graphics, Vol. 23(3) (1999), pp. 353-363.
[8] B. Zalik, M. Gombosi, D. Podgorelec, *A Quick Intersection Algorithm for Arbitrary Polygons*, SCCG98 Conf. on Comput. Graphics and it's Applicat. (1998), pp. 195-204.

Faculty of Mathematics and Computer Science

Babes-Bolyai University, Cluj-Napoca
E-mail address: `deiush@cs.ubbcluj.ro`