INFORMATICA 1/90

DETECTION OF THE INTERSECTION OF TWO SIMPLE POLYHEDRA

Dušan Šurla, Zoran Budimac
Institute of Mathematics,
dr I. Djuričića 4, 21000 Novi Sad

Keywords: detection algorithm, polyhedron intersection, modula-2

ABSTRACT. An algorithm is described for detecting the intersection of two simple polyhedra. The corresponding programme, implemented in Modula-2, is essentially based on a procedure developed to test the intersection of the given segment and simple polygon. The basis for this procedure is the relations between a point, a straight line and a plane, expressed in the vector form.

1. INTRODUCTION

One of the fundamental problems in computational geometry is detection of the intersection of two polyhedra. The problem is directly related to linear programming, hidden surface elimination, computer vision, motion planning and robotics.

Of the numerous publications devoted to this subject we shall mention only those dealing with the problem of intersection [5,6,9] and detection of the intersection [1-4] of two polyhedra. Some of the authors have considered the computational complexity of the algorithms used for solving these problems [3-6,9,10].

In [7] and [8] we have described an algorithm and the corresponding programme for determination whether a given point belongs to the interior domain of a simple polyhedron, as well as for determination of the intersection of a straight line and a simple polyhedron. The basic procedures were formed on the basis of the relations (given in the vector form), between a point, a straight line and a plane.

The present article is a continuation of the above studies in which our considerations are being extended onto the problem of detection of the intersection of two simple polyhedra.

2. THE ALGORITHM

Let be given two simple polyhedra P and Q. Their possible relations may be as follows:

$$P \cap Q = C, \quad C \neq \emptyset, \ C \neq P, \ C \neq Q \tag{1}$$
$$P \cap Q = P, \quad (P \subseteq Q) \tag{2}$$
$$P \cap Q = Q, \quad (Q \subseteq P) \tag{3}$$
$$P \cap Q = \emptyset \tag{4}$$

If the intersection of at least one edge of P (resp. Q) and at least one facet of Q (resp. P) is not an empty set, then condition (1) is fulfilled. If condition (1) is not fulfilled, then P and Q intersect provided that one arbitrary vertex of P (resp. Q) belongs to the interior of the polyhedron Q (resp. P), i.e., condition 2 (resp. 3) is fulfilled. Obviously, if conditions (1)-(3) are not fulfilled, then case (4) holds, i.e., P and Q do not intersect.

Thus, testing condition (1) is reduced to the multiple use of the function for detecting the intersection of the segment (polyhedron edge) and the simple polygon (polyhedron facet). This function can be formed on the basis of the relations given in their vector form.

2.1. BASIC RELATIONS

An arbitrary point $Z \in \mathbb{R}^3$, considered as a vector of the same coordinates, we shall denote by $\vec{Z}$.

Let be given the points $A \in \mathbb{R}^3$ and $B \in \mathbb{R}^3$, and the plane α in $\mathbb{R}^3$. Let us form the following expression:

$$D_1 = (\vec{A} - \vec{X}) \circ \vec{N} \tag{5}$$
$$D_2 = (\vec{B} - \vec{X}) \circ \vec{N} \tag{6}$$
$$D = D_1 \cdot D_2 \tag{7}$$

where $\vec{N}$ is a vector perpendicular to α and $X \in \alpha$. The mark "$\circ$" denotes the scalar product of vectors. If $D < 0$ (resp. $D>0$), then the points A and B are on different (resp. on the same) side of the plane α. For $D_1=0$ and $D_2 \neq 0$ point A belongs to the plane α, and for $D_1 \neq 0$ and $D_2=0$ point B belongs to the plane α. If $D_1=0$ and $D_2=0$ then the segment AB belongs to the plane α.

Let us form the expressions:

$$\vec{E}_1 = (\vec{G} - \vec{U}) \times (\vec{V} - \vec{G}) \tag{8}$$
$$\vec{E}_2 = (\vec{H} - \vec{U}) \times (\vec{V} - \vec{H}) \tag{9}$$
$$E = \vec{E}_1 \circ \vec{E}_2 \tag{10}$$

where the points G, H, U and V belong to the same plane. The mark "x" stands for the vector product of vectors. If $E > 0$ (resp. $E < 0$), then the points G and H are on the same (resp. on different) side of the straight line determined by the points U and V. For $E_1 = 0$ (resp. $E_2 = 0$) point G (resp. H) lies on the straight line determined by the points U and V.

On the basis of relations (8)-(10) it can be determined whether the segments GH and UV intersect. Namely, if the points G and H are on the different sides of the straight line UV and the points U and V are on the different sides of the straight line GH, the two segments intersect, otherwise not.

2.2. DETECTION OF THE INTERSECTION OF A SEGMENT AND A SIMPLE POLYGON

Let us denote the vertices of an edge of the one polyhedron by A and B, and by S a facet of the other polyhedron. Facet S is a simple polygon. Let the plane α be determined by the polygon S. Let us suppose the values in expressions (5)-(7) are as follows: $D<0$; $D_1=0$ and $D_2\neq0$; $D_1\neq0$ and $D_2=0$. In these cases the intersection of the segment AB and the plane α is a point. Let us denote this point by R. If R belongs to the interior region or of the hull of the polygon S, then the intersection of the segment and polygon S is not an empty set. In the case when the segment AB belongs to the plane α, then detection of the intersection of the segment AB and polygon S consists in the following. The intersection of the segment AB and all the edges of S is tested on the basis of relations (8)-(10). If this intersection is an empty set, then it is necessary to test additionally if at least one of the points A and B belongs to the interior region of S. If it does, the intersection of the segment AB and polygon S is not an empty set.

Therefore, detection of the intersection of the segment AB and polygon S is reduced further to solving the following task.

Given a simple polygon S in α and the point $R\in\alpha$, determine if the point R belongs to the interior region of S.

Let r be an arbitrary straight line lying in the plane α and passing through the point R. Let us introduce the following definitions.

Definition 1. The intersection point between r and the hull of P is a piercing point if at this point r passes from the interior into the exterior domain of P, or vice versa.

Definition 2. The edge of the simple polygon S, lying on the straight line r is a piercing edge if one vertex of this edge borders upon the internal and the other on the external region of S.

Then, the following theorem holds.

Theorem. Point R belongs to the interior region of S if on the same side of the point R lying on the straight line r, the sum of piercing points and piercing edges is an odd number.

Proof. Let us suppose the point Y moves along the straight line r from infinity to the point R. Then Y belongs to the exterior domain of S until it reaches the first piercing point, or piercing edge. After passing through the first piercing point / piercing edge, the point Y enters the interior domain of S and remains in it until reaching the second piercing point / piercing edge. Afterwards, the point Y comes again to the exterior domain, and so on. Therefore, if point Y coincides with R after passing through an odd number of the sum of piercing points and piercing edges, then R belongs to the interior domain of polygon S.■

Let us denote an edge of S by $V_{i-1}V_i$. The straight line r and the edge $V_{i-1}V_i$ may have one of the following relations:

(*i*) $r \cap V_{i-1}V_i = T$;
T is different from V_{i-1} and V_i,

(*ii*) $r \cap V_{i-1}V_i = V_i$ or $r \cap V_{i-1}V_i = V_{i-1}$,

(*iii*) $r \cap V_{i-1}V_i = V_{i-1}V_i$,

(*iv*) $r \cap V_{i-1}V_i = \emptyset$

In case (*i*) T is a piercing point. Let in case (*ii*) the intersection be the vertex V_i. Then V_i is a piercing point if the vertices V_{i-1} and V_{i+1} are on different sides of the straight line determined by the points V_i and R, i.e. if the following condition is fulfilled:

$$((\vec{V}_{i-1}-\vec{V}_i)\times(\vec{R}-\vec{V}_{i-1})) \circ ((\vec{V}_{i+1}-\vec{V}_i)\times(\vec{R}-\vec{V}_{i+1})) < 0 \quad (11)$$

In case (*iii*), the edge $V_{i-1}V_i$ is a piercing edge if the vertices V_{i-2} and V_{i+1} are on different sides of the straight line r, i.e. if the following condition is fulfilled:

$$((\vec{V}_{i-2}-\vec{V}_{i-1})\times(\vec{V}_i-\vec{V}_{i-1}))\circ((\vec{V}_{i+1}-\vec{V}_{i-1})\times(\vec{V}_i-\vec{V}_{i+1}))<0 \quad (12)$$

Obviously, in case (*iv*), the edge $V_{i-1}V_i$ is not a piercing edge and there are no piercing points on it.

Figure 1 shows an illustrative example of the intersection of the straight line r and polygon S. The corresponding intersection points are K_1, K_2, K_3, K_4 and V_{16}, and the piercing edge is $V_{10}V_{11}$. Additionally, the edges V_2V_3 and V_6V_7 and the vertex V_{13} are lying on the straight line r. On the basis of these data, it is possible to determine if a point $R\in r$ is on the hull of S, or it belongs to the interior / exterior region of S. First, if the point R coincides with one of the piercing points K_1, K_2, K_3, K_4 and V_{16}, or with the vertex V_{13}, or it belongs to one of the edges V_2V_3, V_6V_7 and $V_{10}V_{11}$, then R is on the hull of S. Second, let us suppose that the point R is between K_3 and V_{10}. Then, on the one side of this point are found the piercing points K_1, K_2 i K_3, and on the other side, the piercing edge $V_{10}V_{11}$ and the piercing points K_4 and V_{16}. Obviously, on the basis of the given theorem, in both cases point R belongs to the interior region of S.

Let us consider now the segment RR_∞, where the point R_∞ is chosen to belong to the exterior region of S, which is easily achieved by taking that absolute values of the coordinates of the point R_∞ are large. The algorithm for determining if R belongs to the interior region of S, can be formed as a Modula-2 function procedure Internal in the following way:

```
PROCEDURE Internal(R, S): BOOLEAN;
(* Procedure Internal returns TRUE if point R
belongs to the interior region or to the hull of
the simple polygon S, whose vertices are denoted
by V_i, i=1, 2, ..., n, where it is assumed that
V_n=V_0, V_n+1=V_1, V_n+2=V_2.
K is the sum of piercing points and piercing
edges. *)
```

```
BEGIN
  K := 0;
  Determination of point R∞;
  FOR i := 1 TO n DO
    IF (R∈V_i V_i+1 ) THEN
       RETURN TRUE
    ELSIF (V_i V_i+1 ⊂RR∞) AND
    (V_i-1 ,V_i+2  are on different sides of the
     straight line RR∞) THEN
       INC(K)
    ELSIF (V_i ∈RR∞) AND
    (V_i-1 ,V_i+1  are on different sides of the
     straight line RR∞) THEN
       INC(K)
    ELSIF (V_i V_i+1 ∩ RR∞ ≠ ∅) THEN
       INC(K)
    END
  END
  RETURN K<>0 AND ODD(K)
END Internal;
```

In the given algorithm, the relations between two segments, and between a point and a segment are determined on the basis of relations (8)-(10).

The algorithm for determining if an edge and a facet intersect is the auxiliary one, and will be used in the final step. It is formed as the Modula-2 function procedure Intersect.

```
PROCEDURE Intersect(E, F): BOOLEAN;
{ Procedure Intersect returns TRUE if the edge E
and the facet F intersect, otherwise it returns
FALSE }

BEGIN
  IF (E ∩ plane(F) ≠ ∅) THEN
    IF (E ⊂ plane(F)) THEN
      IF (E ∩ hull(F) ≠ ∅) THEN
         RETURN TRUE
      ELSE
         R is one of the vertices of E
         IF Internal(R, F) THEN
            RETURN TRUE
         END
      END
    ELSE
         R := E ∩ plane(F)
         IF Internal(R, F) THEN
            RETURN TRUE
         END
    END
  END;
  RETURN FALSE
END Intersect;
```

2.3. PROCEDURE FOR DETECTING THE INTERSECTION OF TWO SIMPLE POLYHEDRA

Let us denote by EP_i, $i = 1,2,\ldots,|EP|$ and FP_i $i = 1, 2, \ldots, |FP|$ the edges and facets of the simple polyhedron P, and by EQ_i, $i = 1,2,\ldots,|EQ|$ and FQ_i $i = 1, 2, \ldots, |FQ|$ the corresponding edges and facets of the polyhedron Q. Then, the algorithm for detecting the intersection of P and Q may be presented in the form of Modula-2 function procedure PolyhedraIntersection.

```
PROCEDURE PolyhedraIntersection(P, Q): BOOLEAN;
{ Procedure PolyhedraIntersection returns TRUE
if Polyhedra P and Q intersect, otherwise
returns FALSE }

BEGIN
  FOR i := 1 TO |EP| DO
    FOR j := 1 TO |FQ| DO
      IF Intersect(EP_i , FQ_j ) THEN
         RETURN TRUE
      END
    END
  END
  FOR i := 1 TO |EQ| DO
    FOR j := 1 TO |FP| DO
      IF Intersect(EQ_i , FP_j ) THEN
         RETURN TRUE
      END
    END
  END
  IF (any vertex(P) ∈ Q) OR { P⊆Q, cond. (2) }
     (any vertex(Q) ∈ P)    { Q⊆P, cond. (3) }
  THEN
     RETURN TRUE
  ELSE
     RETURN FALSE
  END;
END PolyhedraIntersection;
```

The modules for testing whether any vertex of one polyhedron belongs to the interior domain of the other polyhedron has been given in [8].

3. TEST EXAMPLE

Data structure of the simple polyhedra P, Q and R is given in Tables 1.-3.

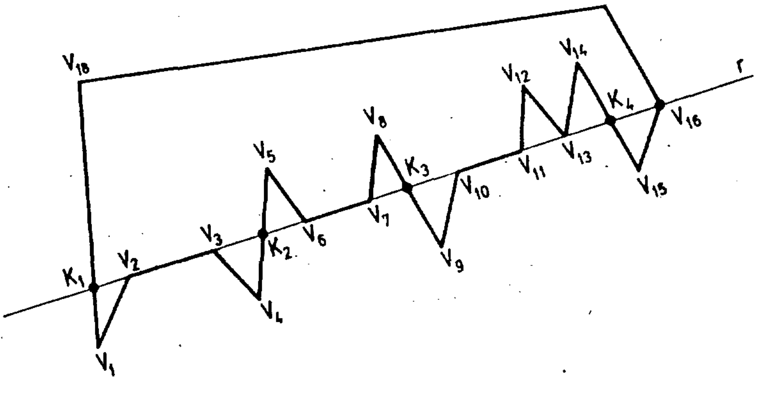

Figure 1

Table 1

Ordinal no. of vertex	Polyhedra P	Q	R
1	(0,0,0)	(1,1,1)	(0,0,6)
2	(5,0,0)	(6,1,1)	(5,0,6)
3	(3,2,0)	(4,3,1)	(3,2,6)
4	(4,4,0)	(5,5,1)	(4,4,6)
5	(2,2,5)	(3,3,6)	(2,2,11)

Table 2

Ordinal no. of edge	Edge determined by vertices
1	1, 2
2	2, 3
3	3, 4
4	3, 5
5	4, 5
6	4, 1
7	1, 5

Table 3

Ordinal no. of facet	Facet determined by ordered vertices
1	1, 2, 5
2	2, 3, 5
3	3, 5, 4
4	4, 1, 5
5	1, 2, 3, 4

$P \cap Q \neq \emptyset$ and $P \cap R = \emptyset$.

4. CONCLUSION

On the basis of the relations derived in vector form, a function can be easily formed for testing of the intersection of a given segment and a simple polygon. The multiple use of this function can serve for detecting the intersection of two simple polyhedra P and Q for the cases when $P \cap Q = C$, $C \neq \emptyset$, $C \neq P$ and $C \neq Q$. The introduced vector relations may be suited for solving other problems in computational geometry.

REFERENCES:

1. J. W. Boyse, Interference Detection Among Solids and Surfaces, *Communications of the ACM* 22(1), 3-9(1979).

2. G. Davis, Computing Separating Planes for a Pair of Disjoint Polytopes, *Proc. ACM Symposium on Computational Geometry*, 8-14(1985).

3. D. P. Dobkin, D. G. Kirkpatrick, A Linear Algorithm for Determining the Separation of Convex Polyhedra, *Journal of algorithms* 6, 381-392(1985).

4. D. P. Dobkin, D. G. Kirkpatrick, Fast Detection of Polyhedral Intersection, *Theoretical Computer Science* 27, 241-253(1983).

5. S. Hertel, M. Mäntylä, K. Mehlhorn, J. Nievergelt, Space Sweep Solves Intersection of Convex Polyhedra, *Acta Informatica* 21, 501-519(1984).

6. K. Mehlhorn, K. Simon, Intersecting Two Polyhedra One of which is Convex, *FCT*, 535-542(1985).

7. D. Surla, The Relation Between a Point and a Simple Polyhedron, *Informatica* 12(2), 65-68(1988).

8. D. Surla, Lj. Jerinic, An Algorithm for Determining the Relation Between a Straight Line (Point) and a Simple Polyhedron, *The Third International Conference on Computer Graphics*, Dubrovnik, Yugoslavia, 1988, in press.

9. M. Szilvási-Nagy, An Algorithm for Determining the Intersection of Two Simple Polyhedra, *Computer Graphics Forum* 3, 219-225(1984).

10. A. C. Yao, R. L. Rivest, On the Polyhedral Decision Problem, *SIAM J. Computing* 9(2), 343-347(1980).

APPENDIX

IMPLEMENTATION OF THE ALGORITHM

A.1. PRELIMINARIES

For representing a simple polyhedron, the following data structure has been adopted:

```
Point      = ARRAY [1..3] OF REAL;
Edge       = RECORD
               F, S: Point
             END;
Polygon    = RECORD
               V  : ARRAY [1..100] OF Point;
               No : [1..100]
             END;
Aux        = ARRAY [1..20] OF INTEGER;
Polyhedron = RECORD
                NoP      : CARDINAL;
                Vertices: ARRAY [1..100] OF Point;
                NoE      : CARDINAL;
                Edges    : ARRAY [1..50] OF Aux;
                NoF      : CARDINAL;
                Facets   : ARRAY [1..50] OF Aux;
                NoOfV    : ARRAY [1..50] OF INTEGER;
              END;
```

A vertex is represented by the array *Point* of three real numbers, i.e. coordinates of the point. An edge is represented by the record *Edge* of two points, and a polygon is represented by the record *Polygon* i.e. by the array *V* of *No* points.

A polyhedron is represented by the record *Polyhedron* i.e. by its vertices (array *Vertices* of *NoP* points), edges (array *Edges* of *NoE* vertex indices - pointers to array *Vertices*) and facets (array *Facet* of *NoF* vertex indices - pointers to array *Vertices*). The i-th element of the array *NoOfV* contain the information on the number of vertices of the i-th polyhedron facet.

There are two operations on data structures representing polyhedron. The first one (implemented as the function procedure *Edg(P: Polyhedron; i: CARDINAL): Edge* selects the i-th edge of the polyhedron P. The other one (implemented as the function procedure *Fac(P: Polyhedron; i: CARDINAL): Polygon* selects the i-th facet of the polyhedron P. Both procedures return their values in appropriate data structures, i.e. *Edge* and *Polygon*, respectively.

```
PROCEDURE Edg(P: Polyhedron; i: CARDINAL): Edge;
VAR E: Edge;
BEGIN
  WITH P DO
    E.F := Vertices[Edges[i,1]];
    E.S := Vertices[Edges[i,2]];
  END;
  RETURN E
END Edg;
PROCEDURE Fac(P: Polyhedron; i: CARDINAL):
                                        Polygon;
VAR S: Polygon;
    j: CARDINAL;
BEGIN
  WITH P DO
    S.No := NoOfV[i];
    FOR j := 1 TO S.No DO
      S.V[j] := Vertices[Facets[i,j]]
    END;
  END;
  RETURN S
END Fac;
```

We will cite without a source code some procedures for basic vector operations, which we need for implementation of the algorithm:

```
PROCEDURE ScalarMul(V1,V2:Point): REAL;
                              [ ScalarMul=V⃗1∘V⃗2 ]
PROCEDURE VecEqual(V1,V2:Point): BOOLEAN;
                              [ VecEqual=(V⃗1=V⃗2) ]
PROCEDURE VecAdd(V1,V2:Point): Point;
                              [ VecAdd=V⃗1+V⃗2 ]
PROCEDURE VecScMul(A:REAL; V1:Point): Point;
                              [ VecScMul=A*V⃗1 ]
PROCEDURE VecSub(V1,V2: Point): Point;
                              [ VecSub=V⃗1-V⃗2 ]
PROCEDURE VecMul(V1, V2: Point): Point;
                              [ VecMul=V⃗1xV⃗2 ]
```

A.2. PROCEDURE Internal

The procedure determines if the given point belongs to the interior domain of the simple polygon. It uses additional procedures *OppSides* and *Between* , which are based on relations (8)-(10).

If points A and B are on different sides of the straight line determined by C and D, then function procedure *OppSides* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE OppSides(A, B, C, D: Point): BOOLEAN;
VAR E1, E2: Point;
BEGIN
  E1 := VecMul(VecSub(A,C), VecSub(D,A));
  E2 := VecMul(VecSub(B,C), VecSub(D,B));
  RETURN ScalarMul(E1,E2) < 0.0
END OppSides;
```

If the point R is on the segment V_1V_2, then the function procedure *Between* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE Between(R, V1, V2: Point): BOOLEAN;
  PROCEDURE Opposite(): BOOLEAN;
  BEGIN
    RETURN ScalarMul(VecSub(R,V1),
                     VecSub(R,V2)) <= 0.0
  END Opposite;
  PROCEDURE SameLine(): BOOLEAN;
  VAR ZeroVec, E: Point;
  BEGIN
    ZeroVec[1] := 0.0;
    ZeroVec[2] := 0.0;
    ZeroVec[3] := 0.0;
    E := VecMul(VecSub(R,V1), VecSub(R,V2));
    RETURN VecEqual(E, ZeroVec)
  END SameLine;
BEGIN
  RETURN SameLine() AND Opposite()
END Between;
```

If the point R belongs to the interior domain of the simple polyhedron S, then function procedure *Internal* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE Internal(R: Point; S: Polygon):
                                        BOOLEAN;
CONST Inf = 200.0;
VAR i, K: CARDINAL;
    RInf: Point;
    PROCEDURE NextV(i: CARDINAL): Point;
    BEGIN
      RETURN S.V[(i MOD S.No) + 1]
    END NextV;
    PROCEDURE Next2V(i: CARDINAL): Point;
    VAR Ind: CARDINAL;
    BEGIN
      Ind := (i MOD S.No) + 1;
      RETURN S.V[(Ind MOD S.No) + 1]
    END Next2V;
    PROCEDURE PrevV(i: CARDINAL): Point;
    BEGIN
      IF i=0 THEN
        RETURN S.V[S.No]
      ELSE
        RETURN S.V[i-1]
      END
    END PrevV;
BEGIN
  K := 0;
  WITH S DO
    RInf := VecAdd(VecScMul(Inf, VecSub(R,V[1])),
                   V[1]);
    FOR i := 1 TO No DO
      IF Between(R, V[i], NextV(i)) THEN
        RETURN TRUE
      ELSIF Between(V[i], R, RInf) AND
            Between(NextV(i), R, RInf) AND
            OppSides(PrevV(i),Next2V(i),R,RInf) THEN
              INC(K)
      ELSIF Between(V[i], R, RInf) AND
            OppSides(PrevV(i),NextV(i),R,RInf) THEN
              INC(K)
      ELSIF OppSides(V[i], NextV(i), R, RInf) AND
            OppSides(R, RInf, V[i], NextV(i)) THEN
              INC(K)
      END
    END;
  END;
  RETURN (K<>0) AND ODD(K)
END Internal;
```

A.3. PROCEDURE PolyhedraIntersection

The procedure determines if two simple polyhedra intersect. Additional procedures *InterExists* and *Sameplane* are based on relations (5)-(7).

If the intersection between the segment E and the plane determined by polygon S is not an empty set, then the function procedure *InterExists* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE InterExists(E: Edge; S: Polygon):
                                        BOOLEAN;
VAR N: Point;
    E1, E2: REAL;
BEGIN
  WITH S DO
    N := VecMul(VecSub(V[1], V[2]),
                VecSub(V[2], V[3]));
    E1 := ScalarMul(VecSub(E.F, V[1]), N);
    E2 := ScalarMul(VecSub(E.S, V[1]), N);
  END;
  RETURN E1*E2 <= 0.0
END InterExists;
```

If the segment E belongs to the plane determined by polygon S, then the function procedure *SamePlane* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE SamePlane(E: Edge; S: Polygon):
                                       BOOLEAN;
VAR N: Point;
    E1, E2: REAL;
BEGIN
  WITH S DO
    N := VecMul(VecSub(V[1], V[2]),
                VecSub(V[2], V[3]));
    E1 := ScalarMul(VecSub(E.F, V[1]), N);
    E2 := ScalarMul(VecSub(E.S, V[1]), N);
  END;
  RETURN (E1 = 0.0) AND (E2 = 0.0)
END SamePlane;
```

If the intersection between the segment E and the hull of S is not an empty set, then the function procedure *HullIntersect* returns TRUE, otherwise it returns FALSE. Procedure is based on mutual application of procedure *OppSides*.

```
PROCEDURE HullIntersect(E: Edge; S: Polygon):
                                       BOOLEAN;
VAR i: CARDINAL;
   NV: Point;
BEGIN
  WITH S DO
    FOR i := 1 TO No DO
     NV := V[(i MOD No)+1];
     IF OppSides(V[i], NV, E.F, E.S) AND
        OppSides(E.F, E.S, V[i], NV) THEN
          RETURN TRUE
      END;
    END;
  END;
  RETURN FALSE
END HullIntersect;
```

Function procedure *CrossingPoint* returns the piercing point between the segment E and the plane determined by polygon S. The procedure is called only when E is piercing the plane.

```
PROCEDURE CrossingPoint(E: Edge; S: Polygon):
                                        Point;
VAR R: Point;
   Aa, Bb, Cc, Dd, L: REAL;
BEGIN
  WITH S DO
    Aa:= (V[2,2]-V[1,2])*(V[3,3]-V[1,3])
       - (V[3,2]-V[1,2])*(V[2,3]-V[1,3]);
    Bb:= (V[2,3]-V[1,3])*(V[3,1]-V[1,1])
       - (V[2,1]-V[1,1])*(V[3,3]-V[1,3]);
    Cc:= (V[2,1]-V[1,1])*(V[3,2]-V[1,2])
       - (V[3,1]-V[1,1])*(V[2,2]-V[1,2]);
    Dd:= -V[1,1]*(V[2,2]-V[1,2])*(V[3,3]-V[1,3])
       -V[1,3]*(V[2,1]-V[1,1])*(V[3,2]-V[1,2])
       -V[1,2]*(V[2,3]-V[1,3])*(V[3,1]-V[1,1])
       +V[1,3]*(V[3,1]-V[1,1])*(V[2,2]-V[1,2])
       +V[1,1]*(V[3,2]-V[1,2])*(V[2,3]-V[1,3])
       +V[1,2]*(V[2,1]-V[1,1])*(V[3,3]-V[1,3]);
    L:= (Aa*E.F[1]+Bb*E.F[2]+Cc*E.F[3]+Dd) /
       (Aa*(E.S[1]-E.F[1])+Bb*(E.S[2]-E.F[2])+
         Cc*(E.S[3]-E.F[3]));
  END;
  R[1]:=E.F[1]-L*(E.S[1]-E.F[1]);
  R[2]:=E.F[2]-L*(E.S[2]-E.F[2]);
  R[3]:=E.F[3]-L*(E.S[3]-E.F[3]);
  RETURN R
END CrossingPoint;
```

If the intersection between the segment E and the polygon S is not an empty set, then the function procedure Intersect returns TRUE, otherwise it returns FALSE. The procedure is based on afore-mentioned procedures and the algorithm described in section 2.2 of the paper.

```
PROCEDURE Intersect(E: Edge; S: Polygon):
                                       BOOLEAN;
BEGIN
  IF InterExists(E, S) THEN
     IF SamePlane(E, S) THEN
        IF HullIntersect(E, S) THEN
           RETURN TRUE
        ELSE
           RETURN Internal(E.F, S) OR
                  Internal(E.S, S)
        END
     ELSE
        RETURN Internal(CrossingPoint(E, S), S)
     END;
  END;
  RETURN FALSE;
END Intersect;
```

If the intersection of simple polyhedra P and Q is not an empty set, then the function procedure *PolyhedraIntersection* returns TRUE, otherwise it returns FALSE.

```
PROCEDURE PolyhedraIntersection(P, Q:
                          Polyhedron): BOOLEAN;
VAR i, j: CARDINAL;
BEGIN
  FOR i:=1 TO P.NoE DO
    FOR j:=1 TO Q.NoF DO
      IF Intersect(Edg(P,i), Fac(Q,j)) THEN
         RETURN TRUE
      END
    END
  END;
  FOR i:=1 TO Q.NoE DO
    FOR j:=1 TO P.NoF DO
      IF Intersect(Edg(Q,i), Fac(P,j)) THEN
         RETURN TRUE
      END
    END
  END;
  RETURN FALSE
END PolyhedraIntersection;
```