Robust Polygons Clipping to Model Complex Objects

Rafael J. Segura, Francisco R. Feito

Departamento de Informática. Universidad de Jaén. Spain

Abstract

One of the most important problems to solve in Solids Modelling is computing the boolean operations for solids (union, intersection and difference). In order to implement these three operations an algorithm to compute the intersection between faces is needed (polygons clipping). In the case of solids with planar faces there are many solutions, although most of them are valid only when the faces are convex. In this paper we present an algorithm to determine the intersection between polygons of any nature (concave or convex, manifol or non-manifold, with or without holes, etc.) based on the idea of simplicial coverings proposed by Feito [11]. One of the advantages of this approach is the robustness of algorithms, since decompositions or complex operations that may alter the results obtained are not avoided. A very interesting application of this algorithm is the modelling of complex solids with this type of polygons as faces.

Keywords: *Solid Modelling, Geometric Modelling, Intersection, Clipping, Simplicial Coverings.*

1. Introduction.

Nowadays, the obtaining of robust and efficient algorithms to solve the problems of Geometric Modelling is one of the main fields of investigation in Computer Graphics. In this sense, at present there are many papers in journals and congresses on which new algorithms or representation methods to reach this objective appear. However, most of the proposed solutions limit the problem to some type of solids (convex[1, 2, 3], triangulated[4], ...). Skala[4] proposes an algorithm for polygon clipping in 3D, concave or convex ones, but it requires the faces to be triangulated, excluding in the cost of the algorithm the cost of this triangulation. Other solutions are only theorical, and some authors consider that the aim of Computational Geometry (and others fields) must be to find useful algorithms that can be implemented in practice[5, 6].

Using the initial idea proposed by Torres[7], and followed by Feito[10], we have developed a system for Solid Modelling, valid for any type of solids with planar faces (concave or convex, with or without holes, manifold or non-manifold). We have developing in a satisfactory way robust and efficient algorithms to solve the inclusion of points in a solid[8], and to study the intersection of a segment (or ray) and a polygon[9].

As result of these investigations, we have designed an algorithm to compute the intersection between polygons of any nature in 3D. This algorithm is the basis of the problem of computing the intersection between solids, which makes possible to compute the boolean operations between solids (union, intersection and difference). The basis of the algorithm is the simplicial covering of the polygons[10, 11], that will be presented later. The main advantage of this approach, opposite to others that use triangles to process the polygons, is that our approach computes the representation of the polygon in linear time (trivially), and it is not necessary to do any -complex- preprocessing; however, computing the triangulation of a polygon can be carried out in linear time theoretically[12], but there is not a practical algorithm to implement it.

Computing the boolean operations between solids is a very studied formal problem, specially for solids represented using a B-rep scheme. For other representation schemes, in practice the problem is finally reduced to the B-rep problem. However, due to the complexity of computing, most of the approaches tend to do some simplification, for example, they reduce the problem to 2D [13]. Other authors propose algorithms that work only with convex faces[2, 1]. But usually most of the solids used in practice, such as mechanical pieces, have not simple faces, but complex ones, with holes or other kind of faces. In this paper we face the problem of any polygon in 3D, and not in 2D, which is solved by Rivero[14] using a similar approach to ours. So, when we

need to solve the 2D problems, we will use the algorithms proposed by Rivero and Feito[14]. We will begin presenting a sumary of the theorical basis of the Solid Modelling by Simplicial Coverings[11]. Then, we will present an algorithm[9] to determine the intersection between segments and triangles (and, by extension, polygons), that is the base to compute the intersection between two polygons. in order to carry out it, we will use the idea proposed by Karasick[15]. Finally, we will present some results, and we will leave some problems opened.

2. Solid Modelling by Simplicial Coverings[11].

Definition 1. Let $x \in \mathbb{R}$. The function *sign(x)* is defined as:

$$sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \quad (1)$$

Definition 2. Let A,B,C and D four points in $\mathbb{R}^3$. The *signed volume*[16] of the tetrahedron of vertices D,A,B,C, denoted as [DABC], is defined as follows:

$$[DABC] = \frac{1}{6} * \begin{vmatrix} x_a - x_d & y_a - y_d & z_a - z_d \\ x_b - x_d & y_b - y_d & z_b - z_d \\ x_c - x_d & y_c - y_d & z_c - z_d \end{vmatrix} = \frac{1}{6} * \begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_b & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{vmatrix} \quad (2)$$

where D=(x_d,y_d,z_d), A=(x_a,y_a,z_a), B=(x_b,y_b,z_b) and C=(x_c,y_c,z_c). We can prove easily that the tetrahedron has a positive orientation (that is, the rest of vertices are seen anti-counterclockwise from the opposite side of the point) if the signed volume is positive (fig.1).

Definition 3. A pyramid, with extreme Q, and base $E_1E_2...E_n$ is said to be an *original pyramid* if Q is the origin of co-ordinates. The pyramid will be denoted as O$E_1E_2...E_n$.

Definition 4. Let an original pyramid P=O$E_1E_2...E_n$, which base is on the plane $\Pi \equiv$Ax + By + Cz + D = 0. The *sign* of the pyramid, denoted as *sign(P)*, is computed as it is shown in equation 3.

$$sign(P) = \begin{cases} 1 & \text{if } D > 0 \\ 0 & \text{if } D = 0 \\ -1 & \text{if } D < 0 \end{cases} \quad (3)$$

Theorem 1. [10] Let a solid S with faces $F_1F_2F_3..F_m$, F_i=$E_1^iE_2^i...E_n^i$, oriented positively. Then

$$S = \bigcup_{i \in I^+}^{*} (P_i -^{*} (\bigcup_{j \in I_i}^{*} P_j)) \quad (4)$$

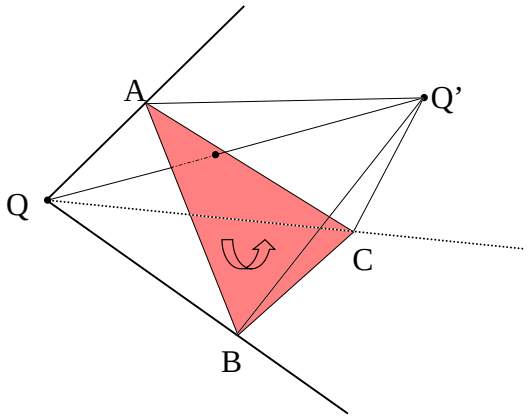

Figure 1: *Study of point Q respect to the the tetrahedron QABC.*

where P_i represents the original pyramid OF_i determinated by the origin of co-ordinates and the face F_i, and the union and difference operations are regularized.

Instead of using pyramids, we can use tetrahedra; this will allow us a simplification in the computing[10]. As it can be seen, the pyramids do not have to be disjoint. This will allow us to work with coverings of the solids, instead of disjoint partitions of them. The main advantage of this approach is that the covering can be obtained in a very simple way with an O(n) algorithm, keeping the initial representation of the solid (a vertex.edge-face graph). Logically, we could do a preprocessing to accelerate the computing, precomputing the simplices (triangles in 2D, tetrahedra in 3D) when we define the solids. Another advantage of this representation of the solids is that most of the algorithms presented are easily converted into parallel algorithms, since we are studying non disjoint triangles, and the results, as we will see later, are obtained by adding integer values.

Theorem 2. [8]. Let a point Q, and a solid S. Then Q is inside the solid S if

$$\sum_i sign(Q, T_i) \times [T_i] = 1 \quad (5)$$

where T_i is, in any case, one of the simplices in which the solid has been covered, $[T_i]$ is the signed volume (or signed area in 2D) of the simplex, and the function *sign(Q,T_i)* returns the signed volume of the simplex formed by the point Q and the triangle T_i (an edge in 2D or a face in 3D).

Corollary. Let a point Q inside the solid S. Then there is at least one T_i of the covering of S, $[T_i] \geq 0$, with Q included in T_i.

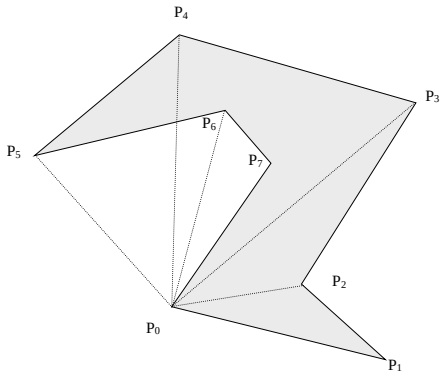

Figure 2: *Covering a polygon with triangles.*

PROOF. Trivially, it can be seen that, when the inclusion of a point in a solid is computed, we use only algebraic adding operations. So, at any moment it must be come true that the sign of T_i is positive to obtain a positive result. Also, it is trivial to prove that the points of the solids included in negative T_i are also included in, at least, two positive T_j, because the result must be positive.

The importance of this corollary is that it can be used to solve the problem of intersecting polygons in 3D using operations between *signed segments*.

3. Arithmetic of Segments.

Next, we are going to introduce the concept of *signed segment*. A signed segment is caracterizied by having associated a function, (the *presence function*, taking an integer value along all the points of the segment.

Definition 6. We define a *signed segment* as a pair (S,μ), with

$$(S,\mu) = \{x \in \mathbb{R} \mid x \in [P,Q]\} \tag{6}$$

being μ:$\mathbb{R} \longrightarrow \mathbb{Z}$ the *presence function*[10].

Definition 7. Let two signed segments (S_1,μ_1), (S_2,μ_2) aligned in the same line, with S_1=$\overline{PP'}$, and S_2=$\overline{QQ'}$. It is defined the adding of the signed segment (S,μ), as follows (see fig.3):

$$\begin{aligned} S_1+S_2 \;=\; & \{x \in \mathbb{R} \mid x \in [min\{P,P'\},\, max\{Q,Q'\}, \\ & \mu(x) \;=\; \mu_1(x) \;+\; \mu_2(x)\} \end{aligned} \tag{7}$$

In figure 3 some cases of adding signed segments are shown. The resultant signed segment appears in the figure below the two segments to be added; it only appears the part

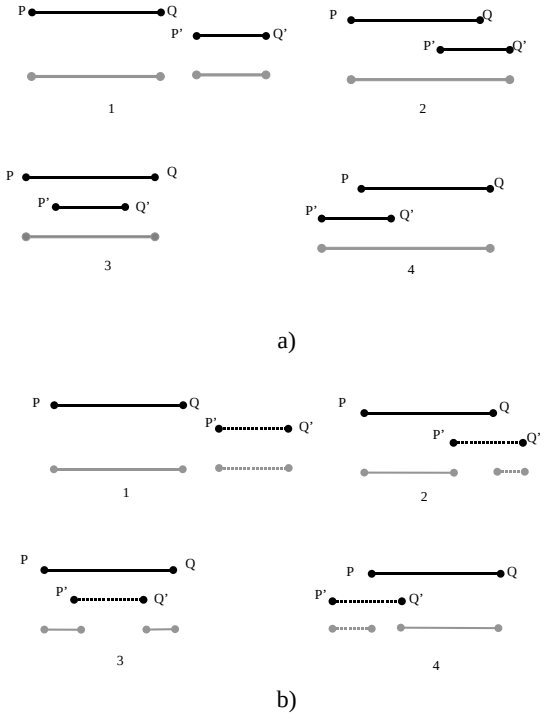

Figure 3: *Different cases of segments adding, with a)* $\mu([PQ]) = \mu([P'Q']) = 1$*; b)*$\mu([PQ]) = 1$*;* $\mu([P'Q']) = -1$.

of the resultant segment with value of presence function different from zero. We have used dotted lines to indicate that the value of the function μ in the signed segment is -1. It can be noted that a signed segment can have one or more components, depending on the value of the presence function. In figure 3, in case a), the result of the adding the segments $\overline{PQ}$ and $\overline{P'Q'}$ is a new signed segment in which presence function in the interval [QP'] is zero. The rest of values of the presence function in each interval of the examples is explained in table 1. Of course, we suppose that the operations are regularized.

4. Intersecting two triangles in 3D.

Lemma 1. [9].

Let a triangle T=ABC in $\mathbb{R}^3$, and a segment S=$\overline{QQ'}$ in $\mathbb{R}^3$, with Q and Q' placed at opposite sides of the plane Π defined by T, and ordered in such a way that the tetrahedron [QABC] has a positive orientation (fig.1). Then the segment S cuts the triangle T if

$$\begin{aligned} sign(Q'AQB) \geq 0 \;\wedge\; sign(Q'CBQ) \geq 0 \wedge \\ sign(Q'ACQ) \geq 0 \end{aligned} \tag{8}$$

	result
a.1	$\mu([P''Q'']) = \mu([P'''Q''']) = 1$
a.2	$\mu([PP']) = 1;\ \mu([P'Q]) = 2;\ \mu([QQ']) = 1$
a.3	$\mu([PP']) = 1;\ \mu([P'Q']) = 2;\ \mu([Q'Q]) = 1$
a.4	$\mu([P'P]) = 1;\ \mu([PQ']) = 2;\ \mu([Q'Q]) = 1$
b.1	$\mu([PQ]) = 1;\ \mu([P'Q']) = -1$
b.2	$\mu([PP']) = 1;\ \mu([P'Q]) = 0;\ \mu([QQ']) = -1$
b.3	$\mu([PP']) = 1;\ \mu([P'Q']) = 0;\ \mu([Q'Q]) = 1$
b.4	$\mu([P'P]) = -1;\ \mu([PQ']) = 0;\ \mu([Q'Q]) = 1$

Table 1: *Value of the presence funtion in the intervals appearing in figure 3. In cases a.1), a.2), a.3) and a.4):* $\mu([PQ]) = \mu([P'Q']) = 1$*; in cases b.1), b.2), b.3) and b.4):* $\mu([PQ]) = 1;\ \mu([P'Q']) = -1$.

As a previous step of the proposed test, it is necessary to prove that points Q and Q' present different signs with regard to plane Π, because if it does not occur, there is not intersection between the segment and the triangle. In order to carry out, that it is only necessary to substitute the points in the equation of plane Π and to study the sign (see definition 2). In the case that the two signs are zero, a 2D problem is found, and then we can use the solution proposed by Rivero[14].

To determine the intersection between two triangles, we will use the just above lemma as a previous step to compute the intersection between the two triangles edges: only when an edge of a triangle intersects the another triangle, the intersection point is computed. With this result, a simple algorithm to compute the intersection between two triangles in 3D can be formulated (fig.4). This algorithm is based on the computing of the intersection between all the edges of a triangle with the another one, and the process repetition interchanging the triangles. We suppose that the orientation of the triangles is positive.

The algorithm is valid for all the degenerated cases appeared when the intersection between an edge and a triangle is not an only point (see fig.5). In case *a)*, the intersection returns two points, and the result is a segment; in case *b)* there is only one point in the intersection, although this point is obtained twice; in cases *c)* and *d)* it could be necessary to eliminate those points appearing more than once in the list of points. These points appear duplicated because the intersection occurs in edges, and so they will appear once per edge.

When the triangles do not lay in the same plane, the result of the algorithm will be a segment (that can be degenerated in the same point). The sign of this segment (+ or -) will be calculated as it appears in table 2.

```
int triangle::intersection (*triangle T2, *listPunt3D result) {
// The orientation of the triangles is possitive.
// Returns the number of intersection between T1 and T2
// result contains the intersection points.
  result = new listPunt3D;
  p1=T1.plane();
  p2=T2.plane();
  For any edge of T1 A=Vi, Vj
    if (sign(Vi, p2)!=sign(Vj, p2))
      if (T1.testIntersectSegment(A) )
        result.insert (intersection (A,T1));
    For any edge of T2 A=Wi, Wj
      if (sign(Wi, p1)!=sign(Wj, p1))
        if (T2.testIntersectSegment(A)
          result.insert (intersection (A,T2));
    return (result.size);
  }
```

Figure 4: *Algorithm for determining intersection triangle/triangle in 3D.*

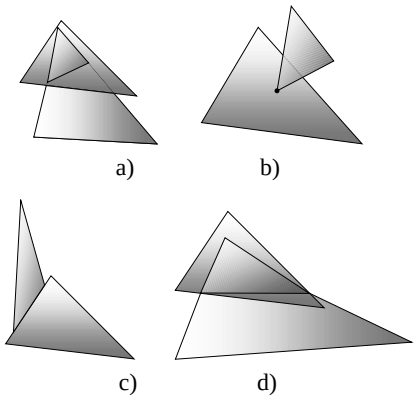

Figure 5: *Some cases of triangles intersection: a) general case; b) intersection in a vertex; c) part of an edge is shared by the triangles; and d) intersection in two edges.*

5. Clipping polygons in 3D.

Once we have solved the intersection between two triangles, we will extend such solution to another kind of polygons. In the case of convex polygons, the problem can be reduced to determining the intersection between two planes. It is trivial to demonstrate that the intersection between two convex polygons can be one point (when the intersection happens in a vertex or an edge) or a segment totally included in both polygons. For other type of polygons, computing the intersection is not so easy, although we can say that the intersection will be a set of segments.

We will use the idea proposed by Karasick[15] in order to

compute the intersection between two polygons P and Q. So, firstly we will compute the intersection of P with Q, obtaining the part of P included in Q. Then, we will repeat the process interchanging the polygons, that is, determining the intersection of Q with P, and combining the results.

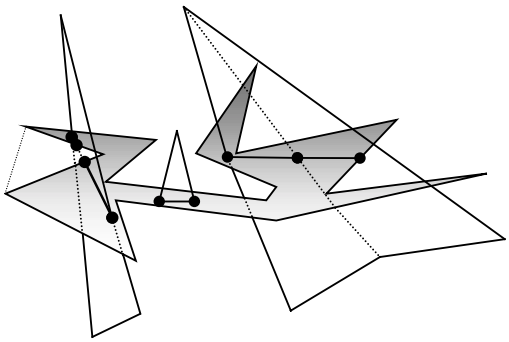

Figure 6: *Intersecting triangle/polygon.*

5.1. The algorithm.

First of all, we are going to reduce the problem to computing the intersection of a polygon P and a triangle T. For this, it is enough to cover the polygon P with *n-2* triangles (being *n* the number of vertices of P), and compute the intersection of any triangle of the covering with T. The result will be a set of segments aligned in the same line; the sign of the segments will be calculated as it appears in table 2. We will use the arithmetic of segments proposed in section 3 to obtain the final set of segments. As it can be seen, it is not necessary to order the segments because they will be added in such way that the final segments will be those with a presence function strictly higher than zero, that is, the segments are included in positives triangles of the polygons. Once the final signed segment has been obtained, it will be simplified, deleting all the segments with $\mu(x) < 1$. A summarized version of the algorithm is shown in figure 7.

Some special cases appear when the intersection happens between two edges, as shown in figure 6. But in this case, and making use of the covering of the polygon, each edge of the polygon will be treated twice, and so the intersection point appears twice in the list. Thus, the only problem will be that in the final result we will obtain a degenerated segment being the origin and the final the same point.

5.2. Results of the algorithm.

One of the advantages of the algorithm we have just presented is that it is easy to do a parallel version. To do it,

```
listSegment *polygon::intersection (polygon *p2) {
  listSegment SegmAux,SegmExit;
  If the polygons are not coplanary{
    Create the coverings of P1 and P2
    Foreach triangle T_i^1 and T_j^2
      SegmAux ← SegmAux ⋃ (T_i^1 ⋂ T_j^2)
    SegmExit ← ∑ SegmAux_i
    return (SegmExit)
  }
  else return (p1.intersection2D (p2))
}
```

Figure 7: *Algorithm for computing intersection Polygon-Polygon.*

Sign(T_1)	Sign(T_2)	Sign(S)
+	+	+
+	-	-
-	+	-
-	-	-

Table 2: *Sign of the segment $S=T_1 \bigcap T_2$ depending on the sign of the triangles.*

you have only to compute the intersection of a polygon with another one separately. In order to determine it, the intersection of any simplices of the polygons can be used separately. The only shared variables in this case is the counter of intersections, and the list of segments. In figure 8 the result of intersecting two complex polygons is showed. The result obtained in the first processing is a set of ten segments, that is reduced to only two segments.

The cost of the algorithm is difficult to be established. It depends on the complexity of the polygons and the size of the list of segments. The first part of the algorithm is $O(n^2)$, being n the number of vertices of the polygon. The second part of the algorithm is more difficult to estimate, and could be reduced using another more efficient data structure, as for example a tree. Anyway, the problem is similar to the one of ordering a list, so the cost could be O(m lg m), being m the size of the list of segments.

6. Conclussions and future work.

We have just presented a robust algorithm for determining the intersection between two polygons of any nature in 3D. The algorithm is based on the study of signs (integer arithmetic), and it only uses floating numbers to compute the intersection point, if it exists. When the algorithm is implemented, the problem appears when determining the sign of a

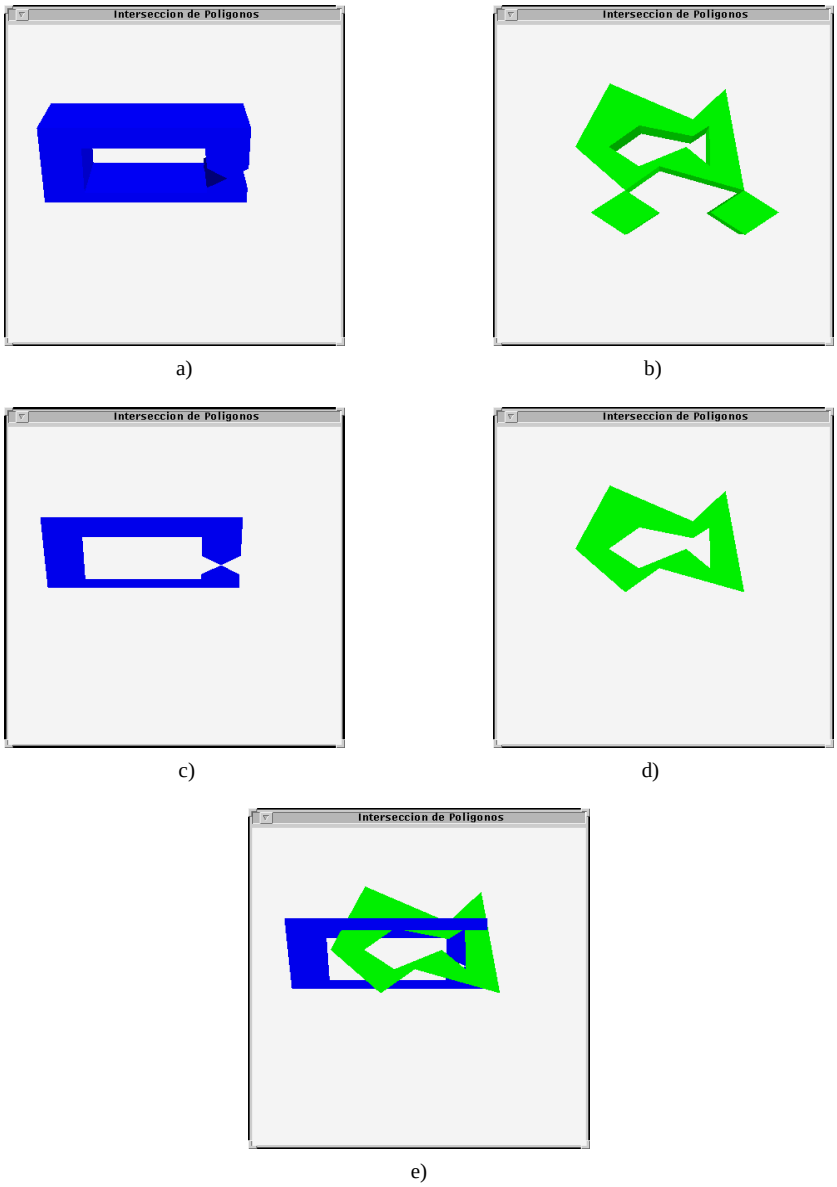

Figure 8: *Intersecting two faces of two complex solids: a) and b) are the original solids; c) and d) are two faces of the solids; e) the intersection.*

float number that can be done in several ways. The theoretical complexity of the algorithm is not higher than other algorithms for non-convex polygons. Also, the algorithm can be written using parallel sentences, computing the intersections between differents simplices separately.

Nowadays, we are developing algorithms and data structures to compute the intersection between two solids with planar faces of any nature. To do it, one of the main obstacles we are dealing with is the way of representing the degenerated solids. Another aspect to solve is to carry out a study of the cost (in time) of the proposed algorithms. The main problem is the impossibility of comparing the cost with other algorithms, because most of them are dependent on a concrete representation, so we would have to consider the cost of converting one representation into another one, and also the cost of storage.

Acknowledgements

The authors wish to thank to the anonymous reviewers for their constructive comments which led to several improvements in the presentation of this paper.

References

1. K.A. Sugihara. A Robust and Consistent Algorithm for Intersecting Convex Polyhedra. *Eurographics'94, Computer Graphics Forum*, **13**(3):C.45–C.54, 1994.
2. F.P. Preparata, M.I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, New York, 1988.
3. B. Chazelle. An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra. *SIAM Journal on Computing*. **21**(4):671–696,1992.
4. V. Skala. An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedra in E^3. *Computer Graphics Forum*, **15**(1):61–68, 1996.
5. B. Chazelle et. a. Application Challenges to Computational Geometry. CG Impact Task Force Reports. *www.cs.duke.edu/~jeffe/compgeom/taskforce.htm*.
6. D.T. Lee. Computational Geometry. *ACM Computing Surveys*, **28**(1):27–31, 1996.
7. J.C. Torres, B. Clares. Graphics Object: A Mathematical Abstract Model for Computer Graphics. *Computer Graphics Forum*, **12**(5):311-328, 1993.
8. F.R. Feito, J.C. Torres. Inclusion test in general polyhedra. *Computer & Graphics*, **21**(1):23–30, 1997.
9. R.J. Segura, F.R. Feito. An Algorithm for Determining Intersection Segment-Polygon in 3D. *Computer & Graphics*, **22**(5):587–592, 1998.
10. F.R. Feito, J.C. Torres. Boundary Representation of Polyhedral Heterogeneous in the context of a Graphic Object Algebra. *Visual Computer*, **13**:64–77, 1997.
11. F.R. Feito, R.J. Segura, J.C. Torres. Representing Polyhedral Solids by Simplicial Coverings. *Set-Theoretic Solid Modelling, Techniques and Applications, CSG'98*, pp: 203–219, 1998.
12. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, **6**:485–524, 1991.
13. Y. Gardan, E. Perrin. An Algorithm reducing 3D boolean operations to a 2D problem: concepts and results. *Computer-Aided Design*, **28**(4):277–287, 1996.
14. M.L. Rivero, F.R. Feito. Algoritmos para las operaciones del modelado sobre sólidos poliédricos a partir de sistemas formales. Una solución en 2D. Actas del VIII Congreso Español de Informática Gráfica, CEIG'98, Orense, 1998.
15. M. Karasick. On the representation and Manipulation of Rigid Solids, Ph.D. *Department of Computer Science, Cornell University*, 1989.
16. J. O'Rourque. Computational Geometry in C. Cambridge University Press, 1994.