SOLVING POINT AND PLANE VS. ORTHOGONAL POLYHEDRA USING THE EXTREME VERTICES MODEL (EVM)

Antonio Aguilera.
Universidad de las Américas-Puebla.
Puebla, México.
antonio@mail.pue.udlap.mx
aguilera@lsi.upc.es

and

Dolors Ayala.
Universitat Politècnica de Catalunya.
Barcelona, España.
ayala@lsi.upc.es

ABSTRACT

In a previous work, Orthogonal Polyhedra (OP) were proposed as geometric bounds in CSG. Primitives in the CSG model were approximated by their respective bounding boxes. The polyhedrical bound for the CSG object was obtained by applying the corresponding Boolean Algebra to those boxes. Also in that paper, a specific and very concise model for representing and handling OP was presented: the Extreme Vertices Model (EVM). The EVM allows simple and robust algorithms for performing the most usual and demanding tasks. This paper deals with the classification of point, and plane vs. OP. These operations can be done on the EVM in linear time. Furthermore, a very important feature of EVM algorithms is that, even though their input data (i.e., vertices' coordinates) can be floating-point values, no time-consuming floating-point arithmetic is ever performed (except when explicitly noted), so there are absolutely no propagation errors due to partial results (which do not exist). All results are obtained by just classifying and selecting vertices' coordinates of the initial data.

1 INTRODUCTION

The Extreme Vertices Model (EVM), first published in [Aguil96], was introduced as a restricted model for *two-manifold Orthogonal Polyhedra* (OP). Also in that paper, a Boolean operations algorithm that works for that model was presented. Moreover, in more recent works [Aguil97, 98a], a natural domain extension for the EVM basics that now handles *Orthogonal Pseudo-Polyhedra* (see a definition below), is presented. All the theoretical foundations for the EVM in its full domain can be found in [Aguil98b].

This paper deals with three processes: (a) determining the set membership classification (IN, ON, or OUT) of a point against an OP; (b) testing whether a general plane intersects an OP; (c) splitting an OP with a plane when it is perpendicular to a specific coordinate axis. The cases when the splitting plane is perpendicular to another coordinate axis, as well as the classification of a line against an OP are problems that can be easily handled by the EVM (see [Aguil97, 98b]). However, they are not considered here because they require a deeper exposition of the EVM concepts and the space limitations for this paper do not allow that.

Splitting an OP with a general plane is not considered because we want a closed operation in the EVM, otherwise the results would not be OP and could not be represented in our model.

This work is organized as follows: The next section introduces some necessary concepts on orthogonal and pseudo-polyhedra, then presents a vertex classification for OP. Section 3 introduces the most basic EVM concepts and its interface. Section 4 deals with the Set Membership Classifications on the EVM, and section 5 analyzes and compares the performance of the proposed algorithms. Finally section 6 contains our conclusions.

2 BACKGROUND

2.1 Terminology.

A pseudo-polyhedron is a finite collection of planar faces such that (a) every edge has at least two adjacent faces, and (b) if any two faces meet, they meet at a common edge [Tang91]. A two-manifold edge is adjacent to exactly two faces, and a two-manifold vertex is the apex of only one cone of faces. Conversely, a non-manifold edge is adjacent to more than two faces, and a non-manifold vertex is the apex of more than one cone of faces [Rossi91].

Polyhedra are two-manifold r-sets. Pseudo-polyhedra (almost polyhedra) are pseudo-manifold r-sets, i.e., r-sets with non-manifold boundary (edges or vertices). Polyhedra are a subset of pseudo-polyhedra. Finally, a non-regular polyhedron is a non homogeneously three-dimensional object, i.e., it has "dangling" faces or edges. [Rossi91, Tang91].

2.2 Orthogonal Polyhedra.

Orthogonal polyhedra (OP) are polyhedra with all their edges and faces oriented in three orthogonal directions [Prepa85]. Orthogonal Pseudo-Polyhedra (OPP) is defined as regular and orthogonal polyhedra that may have non-manifold boundary. In an OPP, a non-manifold edge is adjacent to exactly for faces and a non-manifold vertex is the apex of two cones of faces (see Fig. 1).

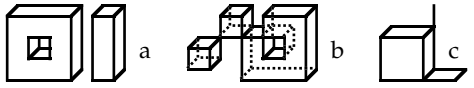


Figure 1: a) An OP. **b)** An OPP. **c)** A non-regular orthogonal polyhedron.

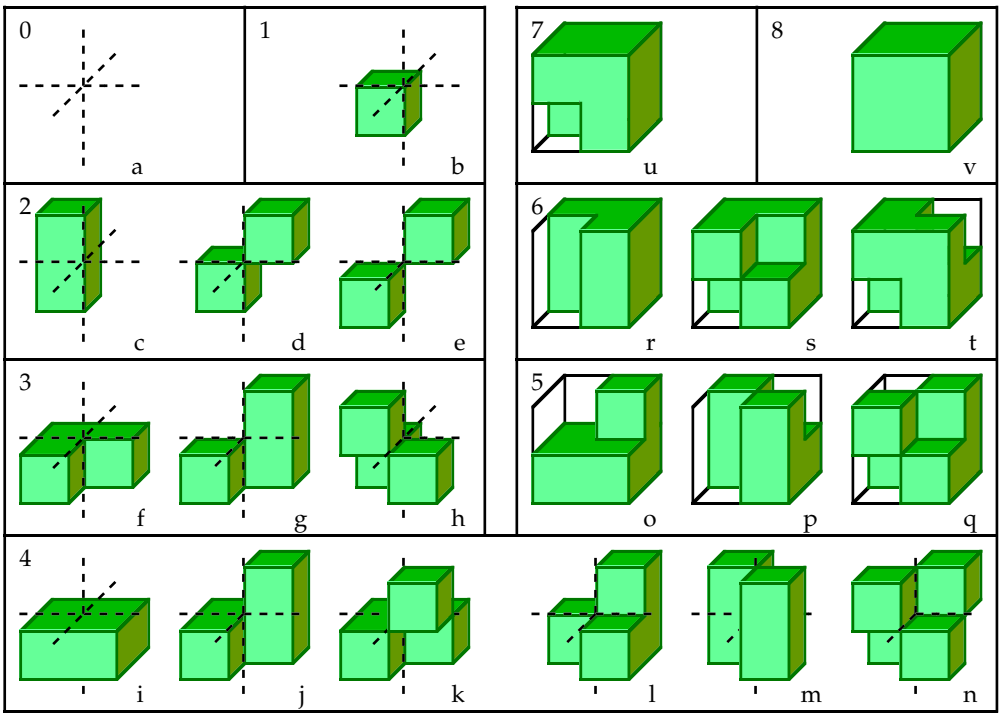


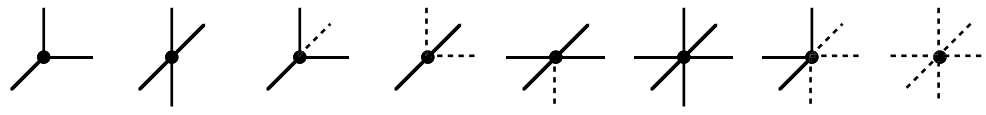
Figure 2: Possible configurations with 0 to 8 surrounding boxes.

2.3 Vertex Classification for OPP.

In an OP the number of incident edges for any vertex can be only three, four or six [Juan89]. In this sub-section we characterize vertices of OPP. In [Aguil97, 98b], it is shown that, as an OPP can be understood as the resulting B-Rep of an orthogonal spatial enumeration model as a voxelization, then vertices of an OPP can be characterized by studying vertices in a voxelization. Moreover, the classification of these vertices follows the same pattern as the classification of nodes in the marching cubes algorithm [Loren87]. Considering the common vertex of eight octants, which can be either full or empty, there are $2^8 = 256$ possible combinations which, by applying rotational symmetries, may be grouped into 22 equivalence classes (configurations) [Sriha81], shown on Fig. 2. Grouping complementaries leads to the 14 basic patterns [Loren87], configurations *a* to *n*, also called major cases [vGeld94].

Configurations *o* to *v* correspond to the complements of those from *a* to *n*. Finally, configurations *i* to *n* are self-complementaries. It is also shown that each configuration represents a vertex for the final OPP, except *a*, *c*, *i*, *m*, *r*, and *v*. These configurations are special cases which represent a point outside the OPP (*a*), on a two-manifold edge (*c*), on a face (*i*), on a non-manifold edge(*m*), on a two-manifold edge (*r*), and a point interior to the solid (*v*).

From the analysis of Fig. 2, vertices can be classified into eight types depending on the number of two-manifold and non-manifold edges incident to them. Fig. 3 shows these types which will be referred as V3, V4, V4N1, V4N2, V5N, V6, V6N1 and V6N2 (the first digit shows the number of incident edges, the "N" is present if at least one Non-manifold edge is incident to it, and the second digit is included to distinguish between two different types that otherwise would receive the same name).



	V3	V4	V4N1	V4N2	V5N	V6	V6N1	V6N2
Number of incident edges	3	4	4	4	5	6	6	6
Number of two-manifold edges	3	4	3	2	4	6	3	-
Number of non-manifold edges	-	-	1	2	1	-	3	6
Number of incident faces	3	4	5	6	6	6	9	12
Number of surrounding boxes	1,3,5,7	4	3, 5	4	2, 6	2, 4, 6	3, 5	4
Corresponding configurations	b,f,o,u	j	g, p	k	d, s	e, l, t	h, q	n
Extreme Vertex	Yes	no	Yes	no	no	no	Yes	no

Figure 3: Vertex classification according to its number of incident edges (dashed lines represent non-manifold edges).

3 THE EXTREME VERTICES MODEL (EVM) FOR OPP

3.1 Brinks, Extreme Vertices, and Planes (Lines) of Vertices.

A brink is the longest uninterrupted segment, built out of a sequence of collinear and contiguous two-manifold edges of an OPP, *P*.

Non-manifold edges do not belong to brinks. Every two-manifold edge belongs to a brink, whereas every brink consists of one or more edges and contains as many vertices as the number of edges plus one (see Fig. 4.a, and Fig. 4.b).

In a brink each ending vertex is V3, V4N1, or V6N1 and the remaining (interior) are V4, V4N2, V5N or V6. Vertices V6N2 do not belong to any brink. According to the above analysis, Vertices V3, V4N1, and V6N1 have in common that they are the only ones that have exactly three incident two-manifold and linearly independent edges, regardless of the number of incident non-manifold edges (if any), therefore those vertices mark the end of brinks in all three orthogonal directions. Any V4, V4N2, V5N or V6 is the sole common point of two collinear edges of a same brink, so they can not be ending vertices of a brink. Finally all six incident edges of a V6N2 are non-manifold edges, so none of them belongs to a brink, and therefore this vertex does not belong to any brink, either (see Fig. 3 and Fig. 4).

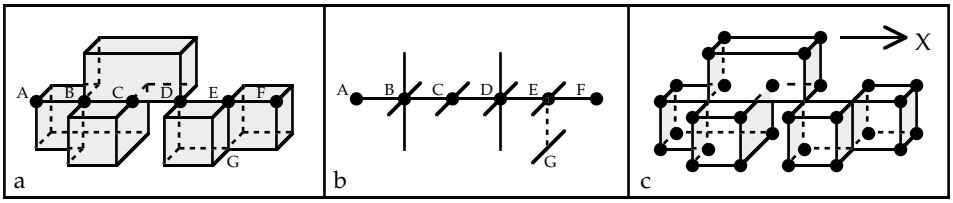


Figure 4: a) An OPP with a brink having five edges and six vertices. Vertices *A* and *F* are V3, *B* and *D* are V6 (configurations *l* & *e*), *C* is V4, and *E* & *G* are V5N. **b)** There are also seven other brinks of two edges each. The non-manifold elements are vertex *D* and edge $\overline{EG}$ (dashed). **c)** Its EV set viewed as six planes of vertices perpendicular to X (the shaded polygons).

We will call Extreme Vertices (EV) of an OPP to the ending (or extreme) vertices of all the OPP brinks, i.e., vertices V3, V4N1, and V6N1 of the OPP. We define the Extreme Vertices Model (EVM) for OPP as a model that only stores all their EV. Finally, an ABC-sorted EVM is an EVM where its EV are sorted first by coordinate *A*, then by *B*, and then by *C*. An EVM can be sorted on six different ways: *XYZ*, *XZY*, *YXZ*, *YZX*, *ZXY*, and *ZYX*. From now on, EVM(P) will denote the ABC-sorted EVM of an OPP *P*, since most of the definitions and results (although not all of them) require of this ABC-ordering. Although the EVM has been defined for 3D-OPP, it is also defined for 2D-OPP and 1D-OPP [Aguil98b].

Theorem 1. *The Extreme Vertices Model is a complete B-Rep model for OPP.*

Coordinate values for non-extreme vertices can be directly obtained from coordinates of EV by intersecting brinks. Moreover, all remaining geometric and topological information about an OPP can also be obtained from its EVM. For a formal proof of this theorem see [Aguil98b].

Let $V_b = V_{2k-1}$ and $V_e = V_{2k}$, for k = 1, 2,… be two consecutive vertices within an ABC-sorted EVM(P), then V_b and V_e are the beginning and ending vertices of the k^{th} *C*-brink defined in an ABC-sorted EVM. (*C*-brinks refers to those brinks parallels to the *C* axis.)

A plane of vertices of an OPP is the set of EV lying on a plane perpendicular to a coordinate axis (i.e., the 2D-EVM of the faces on that plane, see Fig. 4.c). We will also refer as line of vertices (within a plane of vertices) to the set of EV lying on a line parallel to a coordinate axis (i.e., the 1D-EVM of collinear brinks). From now on, plv will refer to both planes and lines of vertices for orthogonal polyhedra and polygons), respectively.

3.2 The ABC-sorted Type, and Interface for the EVM.

According to the above, the ABCsorted type can be defined, along with the following primitive operations:

```
FUNCTION InitEVM() RETURN ABCsorted
{ Returns an empty ABC-sorted EVM }

PROCEDURE PutBrink(INPUT
Vb,Ve:Vertex; I/O P: ABCsorted)
{ Appends to an EVM a brink defined
  by its Extreme Vertices Vb & Ve
  (i.e., appends two consecutive
  Extreme Vertices to the EVM P) }

PROCEDURE ReadBrink(INPUT
P:ABCsorted; OUTPUT Vb, Ve: Vertex)
{ Reads next brink (or pair of
  Extreme Vertices) from an EVM P.}
```

```
FUNCTION ReadPlv(P: ABCsorted,
dim: INTEGER) RETURN ABCsorted
{ Extracts next plane (dim=2) or
  line (dim=1) of vertices from an
  EVM P. That is, the set of
  Extreme Vertices with the same A
  (or A & B) coordinate values. }

FUNCTION EndEVM(P: ABCsorted)
RETURN Boolean
{ Returns TRUE if the end of P has
  been reached. }

FUNCTION GetCoord(P: ABCsorted,
dim: INTEGER) RETURN CoordType
{ Gets the common A (dim=2) or B
  (dim=1) coordinate of a plane
  (line) of vertices P. }
```

Where `CoordType` is the chosen type for the vertex coordinates (`INTEGER`, `REAL`, `DOUBLE`, etc.).

4 SET MEMBERSHIP CLASSIFICATIONS ON THE EVM

4.1 Point in Polyhedra.

The chosen method is an adaptation from the well-known *crossings test* [Shimr62], also known as the *parity*, or *even-odd test* [Prepa85, Mänty88]. This test leads to the fastest algorithm without any preprocessing [Haine94].

The adaptation consists in considering a semi-line parallel to the *A*-axis (*B*-axis for the 2D case), starting from -∞ and ending at the test point. Then a sweep-plane(line)-like process is performed, which starts with an OUT condition and updates it whenever the OPP boundary is crossed (or just touched). All three (IN, ON, and OUT) conditions are handled. When the semi-line goes through a plane of vertices the process is similarly repeated for this plv, i.e., it is a recursive process in the dimension. The trivial (or base) case is for dim = 1, where the test is performed for a line of vertices.

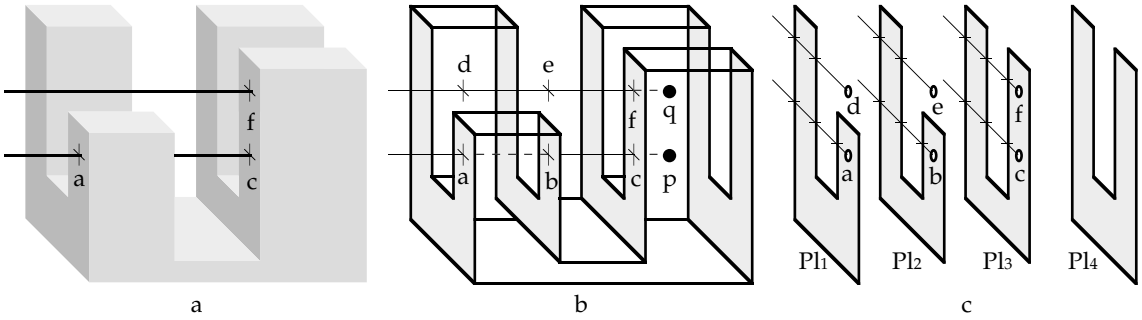


Figure 5: Two points *p* and *q* are to be tested against an OPP. A semi-line from -∞ and ending at *p* goes through Pl_1, Pl_2 and Pl_3 (the supporting planes of the OPP's faces) at points *a, b,* and *c*, respectively; while a semi-line from -∞ and ending at *q* goes through those planes at *d, e*, and *f*. Points *a* to *f* are recursively tested against the corresponding faces. In this example, points *d* and *e* are outside the corresponding faces (OUT) so the OPP's boundary is not crossed at these two points. Any plv that is beyond the test point (like Pl_4) or is not perpendicular to the semi-line will not need to be processed at all.

The problem in the ON cases, when the semi-line passes through one or more vertices and edges can be ignored by considering the semi-line to be a half-plane divider, with one of the half-planes including the semi-line's points. In other words, whenever the semi-line passes through a vertex, the vertex, and the corresponding edge on the semi-line, are always classified as being infinitesimally above it. In this way, no vertices or edges are considered lying on the semi-line, and the resulting code is both simpler and faster.

Moreover, in order to handle the ON cases correctly, the ON condition is further subdivided as ONIN and ONOUT subconditions, which correspond to the ON condition being infinitesimally IN or OUT, respectively. Fig. 6 shows a 1D example (left) and some 2D examples (right) which include all possible transition cases. Finally, transition cases are sumarized in Table 1.

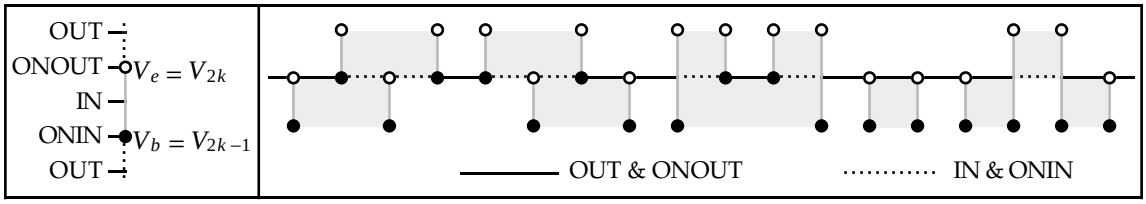


Figure 6: Edges and vertices lying on the semi-line (ON cases) are further classified as ONIN or ONOUT according to the classification (IN or OUT) of the corresponding point being infinitesimally above it. **Left)** 1D example. **Right)** 2D examples.

	OUT	ONOUT	IN	ONIN
OUT	OUT	ONOUT	IN	ONIN
ONOUT	ONOUT	OUT	ONIN	IN
IN	IN	ONIN	OUT	ONOUT
ONIN	ONIN	IN	ONOUT	OUT

Table 1: The IN, OUT, ONIN and ONOUT transition rules.

INresult	ONresult	Meaning
FALSE	FALSE	OUT
FALSE	TRUE	ONOUT
TRUE	FALSE	IN
TRUE	TRUE	ONIN

Table 2: Boolean values code.

Algorithm 1: Point in EVM.

This algorithm receives an ABC-point `pt` and an ABC-sorted EVM `P`, and produces two Boolean results: `INresult` and `ONresult`, whose combined values produce all four possible results as shown in Table 2.

It recursively testes whether the ray crosses any `plv` previous to the test point. Since each EV is processed at most three times (once at each recursive level), then this algorithm runs in linear time. The corresponding *Point in EVM* Algorithm can be stated as:

```
PROCEDURE PointInEVM( INPUT pt: ABCpoint; P: ABCsorted; dim: INTEGER;
                      OUTPUT INresult, ONresult:BOOLEAN);
   VAR
      plv: ABCsorted;      { current plane (line) of vertices        }
      INflag, ONflag,      { IN & ON flags returned by recursive call}
      PtInPl: BOOLEAN;     { TRUE if pt lies on the plv just read    }
      plvCoord: CoordType; { for saving the plv common coordinate    }
   ENDVAR

   IF dim = 1 THEN PtInLine(pt, P, INresult, ONresult)   {trivial case}
   ELSE
      dim := dim-1;
      PtInPl := FALSE;     { assume pt not lying on Pl }
      INresult := FALSE;   { initialize result as OUT  }
      ONresult := FALSE;   { initialize result as OUT  }
      plv := ReadPlv(P, dim);
      plvCoord := GetCoord(plv, dim);
      WHILE NOT EndEVM(P) AND plvCoord ≤ PtCoord(pt, dim) DO
         IF plvCoord = PtCoord(pt, dim) THEN PtInPl := TRUE ENDIF
         PointInEVM(pt, plv, dim, INflag, ONflag); { recursive call }
         IF INflag THEN INresult := NOT INresult ENDIF
         IF ONflag THEN ONresult := NOT ONresult ENDIF
         plv := ReadPlv(P, dim);
      ENDWHILE

      IF PtInPl AND INflag THEN ONresult := TRUE ENDIF

   ENDIF
ENDPROCEDURE
```

Where procedure `PtInLine` returns:

- ONIN, if $pt = V_b = V_{2k-1}$, (*pt* is the beginning vertex of the k^{th} *C*-brink, for some *k*.)
- ONOUT, if $pt = V_e = V_{2k}$, (*pt* is the ending vertex of the k^{th} *C*-brink, for some *k*.)
- IN, if $V_b < pt < V_e$, (*pt* is in the interior of the k^{th} *C*-brink, for some *k*.)
- OUT, otherwise,

as described in first column of Table 1, and the returned result is coded according to Table 2.

4.2 Testing a Plane against an OPP.

A test for determining whether a general plane intersects an OPP *P*, can be developed for the EVM, using the fact that a plane intersects *P* iff it intersects any of the OPP brinks, i.e., if the Extreme Vertices of a brink are at either side of the plane. Moreover, *C*-brinks can be trivially tested for this intersection in linear time, since they are defined by two consecutive vertices $V_b = V_{2k-1}$ and $V_e = V_{2k}$.

Different ABC-sortings of the EVM can be used for testing other brinks. Thus, an O(n Log n) preprocess (the sort) is needed to test any number of different planes against P, each in linear time.

Furthermore, the intersection of the plane with P (i.e., the section) can be computed in the following way. As each brink is tested, its intersection point with the plane can be computed (this process requires floating-point arithmetic), then the intersecting points of any two brinks with a common EV can be joined in a domino-like procedure, effectively obtaining the contour of the section.

4.3 Splitting an OPP with a plane perpendicular to the *C* axis.

This section presents an algorithm for the classification of an OPP P against splitting plane SP perpendicular to the C axis. It computes two resulting OPP: one of them Q, corresponding to the IN half-space, and the other one R, corresponding to the OUT half-space.

This algorithm is based on the fact that the Extreme Vertices of each of the resulting objects Q and R will be a subset of *EVM(P)*, except for some new Extreme Vertices that could be created, and they will lie on SP. Also, since SP is perpendicular to the C axis, then neither A-brinks nor B-brinks of P can ever be split by SP, only some C-brinks (those whose Extreme Vertices are at either side of SP). Therefore, this splitting algorithm only considers those brinks parallel to the C axis, and they appear as consecutive couples of vertices in the ABC-sorted model, so it runs in linear time.

If SP is perpendicular to other axis then a suitable ABC-sorting must be applied to the model prior to this process. See [Aguil97] for other splitting algorithms that can be directly applied when SP is perpendicular to other axis.

Let $V_b = V_{2k-1}$ and $V_e = V_{2k}$ be the beginning and ending vertices of the k^{th} C-brink defined in an ABC-sorted EVM. Since both V_b and V_e have the same A and B coordinates, while the C coordinate is less in V_b than in V_e, then, it can easily be known, for each of the vertices V_b and V_e, if it is IN, OUT or ON with respect to the splitting plane SP (i.e., in the negative or positive halfspaces of SP, or on SP itself) by just comparing its C-coordinates with the plane equation.

Then only two cases may occur:

a) Both vertices lie in the same halfspace of SP, or one of them is ON the SP. Then both of them will be assigned to the same Q or R resulting object.

b) Each vertex belongs to a different halfspace. In this case a new vertex V_i at the intersection between the brink and the splitting plane is obtained. Then, V_b and V_i will be assigned to Q, while V_i and V_e will be assigned to R.

According to these two cases, the corresponding Splitting Algorithm can be stated as:

Algorithm 2: Splitting an ABC-sorted OPP P with a C-Splitting Plane SP.

```
PROCEDURE SplitC(INPUT P: ABCsorted, SP: plane; OUTPUT Q, R: ABCsorted)
{Splits object P by plane SP (perpendicular to the C axis) into objects Q
  and R}

  VAR Vb, Ve, Vi: ABCpoint ENDVAR

  Q := InitEVM();
  R := InitEVM();

  ReadBrink(P, Vb, Ve);
  WHILE NOT EndEVM(P) DO
    IF  IN(Vb) AND ( IN(Ve)   OR  ON(Ve)) THEN PutBrink(Vb, Ve, Q) ENDIF
    IF (ON(Vb)  OR  OUT(Vb)) AND OUT(Ve)  THEN PutBrink(Vb, Ve, R) ENDIF
    IF  IN(Vb) AND  OUT(Ve) THEN
      Intersect(Vb, Ve, SP, Vi);
      PutBrink(Vb, Vi, Q);
      PutBrink(Vi, Ve, R);
   ENDIF
   ReadBrink(P, Vb, Ve);
  ENDWHILE
ENDPROCEDURE
```

Note that the procedure `Intersect` obtains V_i without any computation. Let for example, $x = x_p$ be the plane equation, and also let $V_b = (x_1, y, z)$ and $V_e = (x_2, y, z)$ be the beginning and ending vertices of a brink in a ZYX or YZX-sorted model, then $V_i = (x_p, y, z)$.

Fig. 7.a shows an OP P that is to be split . Note that only horizontal brinks (shown as solid lines) need to be considered. Circles show the intersection points between *C*-brinks and SP, each of them will generate two Extreme Vertices, one for Q and one for R.

Also note that one of those circles corresponds to an existing V4, and another one to a V6 (SP coincides with a plane of vertices of P). The result is shown in Fig. 7.b, where the dots correspond to the newly generated Extreme Vertices.

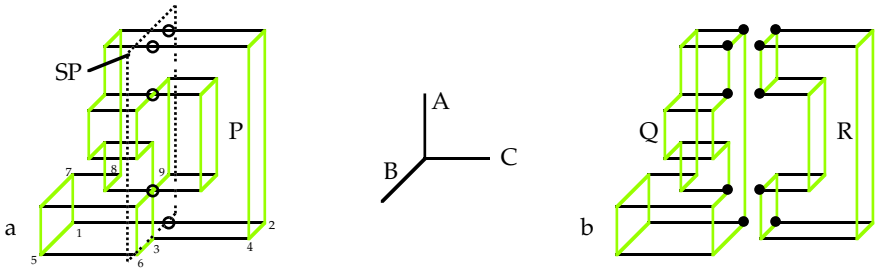


Figure 7: (a) An OPP that is going to be split by plane *SP* perpendicular to the *C*-axis. . **(b)** The resulting OPPs *Q* and *R*. (see text for more details).

5 PERFORMANCE COMPARISON

This section provides theoretical and some experimental comparisons of known methods (of linear complexity) vs. the proposed one. The performance of the respective algorithms is better in the EVM than in other methods mainly because of the following facts:

- The complexities of known methods are linear with respect to the total number of vertices $O(nV)$ in the polyhedron, as opposed to the proposed algorithms which are linear with respect to the number of Extreme Vertices $O(nEV)$. Obviously $nEV \le nV$, but very often $nEV << nV$.
- No time-consuming floating-point arithmetic is ever performed in the EVM algorithms.

  Note that EVM algorithms are tailored (limited) to handle OPPs only.

The worst case, when $nEV = nV$, the EVM algorithms equal the performance of known methods. On the other hand, the best cases correspond to a case study, found in [Aguil98b], where a succession of OPPs W_k, $k \ge 2$, is defined. Each OPP, W_k, is built inside a cubic bounding box or orthogonal hull $OH(W_k)$, with side of k units of length, and composed of the maximum number of unit-cubes joined by edges, but with the minimum number of Extreme Vertices. See Fig. 8.

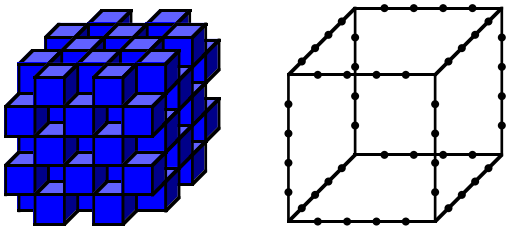


Figure 8: Example of W_5 and its EVM.

The set of Extreme Vertices for each W_k is constructed by dividing each edge of $OH(W_k)$ into k segments, then these splitting points are all the Extreme Vertex of W_k, when k is odd. When k is even, four additional points are required and they must be placed at corners of the bounding box $OH(W_k)$ such that no edge of $OH(W_k)$ has Extreme Vertices at both ends. So W_k has $12(k-1)$, or, $12(k-1)+4$ Extreme Vertices of type V3 (if k is odd or even, respectively); $6(k-1)^2$ vertices of type V5N on the faces of $OH(W_k)$; and $(k-1)^3$ vertices of type V6N2 in the interior of $OH(W_k)$. Therefore, $nEV = O(k)$ while $nV = O(k^3)$, in fact it is shown in [Aguil98b], that $nEV = O(\sqrt[3]{nV})$, therefore $nEV << nV$.

In Fig. 9, a graph shows the behavior of the *Point-In-Polyhedron* algorithms testing 500 random points generated inside the bounding box of each OPP. For this test, random OPPs, with the desired number of vertices were generated as the union of random boxes.

"Known methods" in Fig. 9 stands for the *crossings test,* where a semi-line parallel to the *x* axis is supposed to be tested against all the polyhedron faces. This test, however, has been adapted to OPPs and only those faces perpendicular to the semi-line were tested, therefore, every vertex in the OPP is processed exactly once. Also, the set of vertices is assumed to be sorted in a sequence describing the polygonal faces being perpendicular to the semi-line, thus the algorithm runs in a time proportional to the total number of vertices. Both "EVM (Average)" and "EVM (Minimum)" in Fig. 9 stand for the behavior of algorithm 1, where the set of vertices is assumed to be XYZ-sorted. The first one represents the average behavior of 50 random polyhedra for each number of vertices, while the second is the result of applying it to the succession of OPPs W_k described above.

6 CONCLUSIONS

The EVM is a highly concise model for OPP that allows simpler and faster algorithms for Set Membership Classification in linear time (some of them may require an *O(n log n)* preprocess).

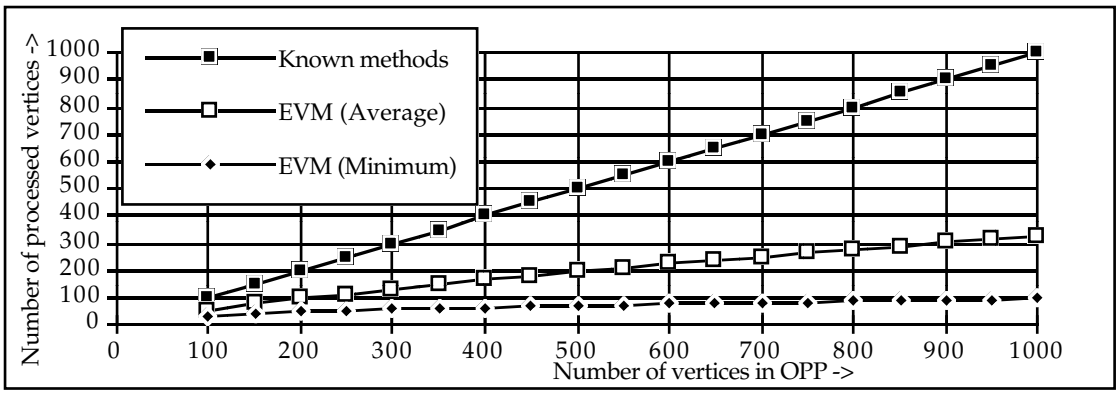


Figure 9: Experimental comparison of known methods with the proposed one.

The experimental results show that the average EVM performance is quite better than other methods, and not too far from the optimum (the W_k case study).

The above methods use the vertex list as their only data structure. Faster *O(log n)* methods have been developed elsewhere [Haine94], but they need to do a preprocess to generate an alternate polyhedron representation and/or additional efficiency structures. Similarly in [Aguil98a], other OPP representation and data structure are proposed, that allows us to do most of the set membership classifications in time *O(log n)*.

7 ACKNOWLEDGMENTS

The present work has been partially supported by CICYT grants TIC-95-630-C03.

8 REFERENCES

[Aguil96] Aguilera,A, Ayala,D: *Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry*. In Hoffman,C, Bronsvort,W, editors, *Fourth ACM Siggraph Symposium on Solid Modeling and Applications*, Vol.4, pp.56-67, 1997. Also as Technical Report LSI-96-64-R. Universitat Politècnica de Catalunya, 1996.

[Aguil97] Aguilera,A, Ayala,D: *El Modelo de Vértices Extremos (EVM) para Poliedros Ortogonales. VII Congreso Español de Informática Gráfica (CEIG '97)*, Memorias del Congreso (written in Spanish), Vol.7, pp.111-125, Barcelona, España, 1997. Also as *The Extreme Vertices Model for Orthogonal Polyhedra*. (written in English) Technical Report LSI-97-6-R. LSI-Universitat Politècnica de Catalunya, 1997.

[Aguil98a] Aguilera,A, Ayala,D: *Domain Extension for the Extreme Vertices Model (EVM) and Set Membership Classification.* To appear in the proceedings of the CSG '98, Information Geometers Ltd., 1998.

[Aguil98b] Aguilera,A: *Orthogonal Polyhedra: Study and Application*. Ph.D. Thesis. LSI-Universitat Politècnica de Catalunya, 1998.

[Haine94] Haines,E: *Point in Polygon Strategies.* In Heckbert,P, editor, *Graphics Gems IV*. pp.24-46. Academic Press, Boston, 1994.

[Juan89] Juan-Arinyo,R: *On Boundary to CSG and Extended Octree to CSG Conversions*. In Strasser,W, editor, *Theory and Practice of Geometric Modeling*, pp. 349-367. Springer-Velarg, 1989.

[Loren87] Lorensen,W, Cline,H: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics*, Vol.21, No.4, pp.44-50, 1987

[Mänty88] Mäntylä,M: *An Introduction to Solid Modeling.* Computer Scientific Press, 1988.

[Prepa85] Preparata,F, Shamos,M: *Computational Geometry: an Introduction*. Springer-Velarg, 1985.

[Rossi91] Rossignac,J, Requicha,A: *Constructive Non-Regularized Geometry. Computer - Aided Deign*, Vol.23, No.1, pp. 21-32, 1991.

[Shimr62] Shimrat,M: *Algorithm 112: Position of point relative to polygon. Communications of the ACM*, Vol.5, p.434, 1962.

[Sriha81] Srihari,S: *Representation of Three-Dimensional Digital Images. ACM Computing Surveys*, Vol.13, No.1, pp.399-424, 1981

[Tang91] Tang,K, Woo,T: *Algorithmic Aspects of Alternating Sum of Volumes. Part 1: Data Structure and Difference Operation. Computer-Aided Deign*, Vol.23, No.5, pp.357-366, 1991.

[vGeld94] van Gelder,A, Wilhelms,J: *Topological Considerations in Isosurface Generation. ACM Transactions on Graphics*, 13 (4): 337-375, 1994.