The Extreme Vertices Model (EVM) for Orthogonal Polyhedra

A. Aguilera
D. Ayala

Report LSI–97–6–R

The Extreme Vertices Model (EVM) for Orthogonal Polyhedra.

A. Aguilera and D. Ayala.
Universitat Politècnica de Catalunya .
Barcelona, Spain.
aguilera@goliat.upc.es, ayala@lsi.upc.es
January, 1997.

LSI-97-6-R

Abstract

Orthogonal Polyhedra offer a worthy-to-explore simplification. In this work we propose a specific model for representing orthogonal polyhedra that allows simple and robust algorithms for performing the most usual and demanding tasks, such as closed and regularized boolean operations; solid splitting and other set membership classification operations; and so on. These algorithms have much lower complexities than their counterparts for general objects have, and they also avoid floating-point computations.

1 Introduction.

In several disciplines, such as solid modeling and computational geometry, it is very usual to start studying problems on simpler classes of polyhedra rather than on the general case. The most usually chosen class is the convex polyhedra class. Convexity enables the use of efficient and simple algorithms [Preparata85], [Edelsbrunner87]. Orthogonal Polyhedra (OP, from now on) are a less used simple class. Nevertheless, some works have been published dealing with or using them. In [Juan89] a B-Rep to CSG conversion algorithm is presented that works for a restricted class of OP. The obtained CSG expression is a Peterson-style formula and the restricted class are the acyclic OP. In [Juan95] the same author extends the domain for a certain class of cyclic OP. In [Montani91] an octree to B-Rep conversion algorithm is presented and an OP is obtained. In [Ayala95] an algorithm that simplifies geometry is presented for the particular case of OP; a more complex algorithm is needed for the general case of polyhedra [Andújar96].

Boxes, which are both convex and orthogonal, have been widely used in many applications [Preparata85], [Hoffmann89a], [Samet89] and they have also been most widely used as geometric bounds (e.g., for approximating and evaluating CSG models). In [Preparata85] a large collection of problems dealing with orthogonal polygons (2D-OP), represented as union of rectangles, can be found.

In this work we define a specific model to represent orthogonal polyhedra, the Extreme Vertices Model (EVM). We have also developed simple and robust algorithms that work on OP represented in this model. We consider all the OP geometry (i.e., faces and edges), as well as the geometric elements (planes, lines and boxes) with which we operate oriented in the same iso-oriented coordinate system (except in sec. 6.3).

This work is inspired on the fact that OP offer a worthy-to-explore simplification over general polyhedra. Plane and line equations are very simple. Most of the major operations, like conversion algorithms between EVM and other representation schemes and set membership classification can be done on EVM in linear time. This is an important improvement over the general case, in which some of these operations involve quadratic complexity.

EVM represents OP in a complete and compact way. The model is complete because we can infer from it all the topological and geometric information of the polyhedron, and it is compact because it only stores some of the OP vertices.

Although input data (i.e., vertices' coordinates) can be floating-point values, no time-consuming floating-point arithmetic is ever performed, so there are no propagation errors. All results are obtained by just classifying vertices' coordinates of the initial data.

This paper is arranged as follows. Section 2 introduces the needed topological and geometrical terminology, then it applies this terminology on OP and finally presents an exhaustive vertex analysis for OP. Section 3 defines the Extreme Vertices Model and its basic associated operations. Section 4 presents a boolean operations algorithm and section 5 describes four splitting methods on EVM. Section 6 introduces the Ordered Union of Disjoint Boxes model (OUoDB) and discusses the Point and Line in Polyhedra problems. Section 7 presents some methods for converting EVM to and from other representation schemes for solids, like Octrees, OUoDB and B-Rep. Section 8 shows how to obtain an OPP's perimeter, area and volume; and finally, our conclusions and paths for future work are in Section 9.

2 Background.

In this section we introduce the needed topological and geometrical terminology, which is then applied on OP. Finally, an exhaustive vertex analysis for OP is presented.

2.1 Terminology.

- A point p of a set S in the d-dimensional Euclidean space E^d is called an interior point of S if there exists an open d-dimensional neighborhood that consists of points in S only. A point p is called a boundary point of S if it is not an interior point. The set $B(S)$ of all boundary points of S, and the set $I(S)$ of all the interior points of S, are defined as the boundary and the interior of S respectively.

The relationship between a boundary point and its neighboring points of a set S in E^d is described by three characterizations [Tang91a]:

- A point p in $B(S)$ is called a two-manifold point if it has a d-dimensional neighborhood such that the subset of the points of S contained in that neighborhood is topologically equivalent to a hemisphere. [Weiler86]

- A point p in $B(S)$ is called a pseudo-manifold point if every d-dimensional neighborhood of it contains some points in I(S).

- A point p in $B(S)$ is called a non-manifold point if it has a d-dimensional neighborhood such that the subset of points of S contained in that neighborhood entirely belongs to $B(S)$.

A pseudo-manifold point is a relaxation of a two-manifold point, i.e., it only requires that every neighborhood of the points contains some interior points of the set, but with no topological constraint on the neighborhoods. According to these three characterizations, a point set S in E^d is [Tang91a]:

- a two-manifold set if every point in $B(S)$ is a two-manifold point.
- a pseudo-manifold set if every point in $B(S)$ is a pseudo-manifold point.
- a non-manifold set if $B(S)$ contains some non-manifold points. [Tang91a]

This means that two-manifold sets are a particular case of pseudo-manifold sets. On the other hand, and because objects must be homogeneously three dimensional within this work, non-manifold sets are immediately excluded from our consideration.

- A regular set (r-set) is a bounded, closed, regular and semi-analytic set; A regular set coincides with the closure of its interior. [Requicha80], [Rossignac91].

- A polyhedron is defined by a finite set of plane polygons such that every edge of a polygon is shared by exactly one other polygon (adjacent polygons). The vertices and edges of the polygons are the vertices and edges of the polyhedron; the polygons are the faces of the polyhedron. [Preparata85]

- 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. [Tang91a]

- 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 two-manifold vertex is the apex of more than one cone of faces. [Rossignac91].

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). Finally, a non-manifold polyhedron is a non homogeneously three-dimensional object, i.e., it has "dangling" faces or edges. [Rossignac91], [Tang91a]. See figure 2.1.

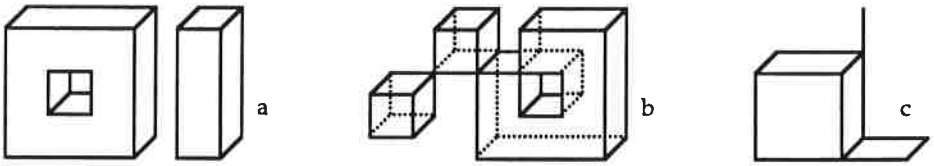


Figure 2.1: a) A (two-manifold) polyhedron. b) A pseudo-polyhedron.
c) A non-manifold polyhedron.

Also note that polyhedra do not need to be composed of simply connected regions (any component may have holes), but the various components of it must be totally disjoint. In the other hand, note that regions in pseudo-polyhedra whose interiors are disjoint may share common vertices or edges, but not common faces [Rossignac91]. A B-Rep for pseudo-polyhedra that stores the geometrical information of vertices and faces' normal vectors, along with the topological relations: E->V, F->E and E->F is presented in [Tang91a].

- A halfspace is the portion of E^d lying on one side of a hyperplane [Preparata85].

We will be dealing with no more than three dimensions ($d \leq 3$), so a hyperplane is just a plane in E^3, a straight line in E^2, and a single point in E^1.

- A halfspace H is a supporting halfspace of a closed subset S of E^d if $H \cap S \neq \emptyset$ and $S \subset H$.

- If a vertex of a pseudo-polyhedron has a supporting halfspace, then it is called locally supportable. Furthermore, if the supporting halfspace of the vertex also supports the polyhedron, the vertex is said to be globally supportable.

- If the complement of the vertex has a supporting halfspace, it is called complementary supportable.

- Finally, if neither the vertex nor the complement of the vertex has a supporting halfspace, the vertex is said to be non-supportable [Tang91b], [Juan95].

It is well known that set theoretic boolean operations are not closed for two-manifold sets (nor for pseudo-manifold sets) since a non-manifold set may result. So, regularized boolean operations are used to maintain the homogeneous three dimensionality, and the domain will be extended to pseudo-manifold sets to end up with closed operations.

$$P \cup^* Q = c(I(P \cup Q))$$
$$P \cap^* Q = c(I(P \cap Q))$$
$$P -^* Q = c(I(P - Q))$$
$$P \otimes^* Q = c(I(P \otimes Q))$$

Where $\cup^*$, $\cap^*$, $-^*$ and $\otimes^*$ are the regularized Union, Intersection, Difference and Exclusive OR respectively, defined as the closure of the interior of the corresponding set theoretic boolean operation. In this work, only regularized boolean operations will always be performed; however, to improve readability, the standard symbols $\cup$, $\cap$, $-$ and $\otimes$ will be used instead of $\cup^*$, $\cap^*$, $-^*$ and $\otimes^*$, respectively.

2.2 Orthogonal Polyhedra.

Orthogonal polyhedra are polyhedra with all their edges oriented in three orthogonal directions. We can also define orthogonal pseudo-polyhedra and non-manifold orthogonal polyhedra in the same way as for general polyhedra. The examples of figure 2.1 are indeed orthogonal polyhedra.

In this work we will concentrate on orthogonal pseudo-polyhedra (OPP from now on, while OP will refer to the restricted two-manifold orthogonal polyhedra).

OPP imply a restriction of general polyhedra concerning with the geometry. Thus in an OPP:

- all planes and lines are parallel to three orthogonal axes.
- a two-manifold edge is adjacent to two faces, whereas a non-manifold edge is adjacent to exactly four faces; and
- the number of incident edges for any vertex ranges from three to six, although in an OP this number can only be three, four or six [Juan95].

These geometric characteristics make OPP be a more restricted class than general polyhedra. However, concerning with the topology, OPP do not imply any restriction at all. OPP allow any number of rings on faces, holes (of any genus) and shells.

2.3 Vertex analysis for OPP.

An OPP may be decomposed into a collection of quasi-disjoint boxes; i.e., boxes joined by a lower dimensional element (a face, an edge or a vertex). See [Foley90], for example.

Definition 2.1 Two quasi-disjoint boxes (rectangles in the 2D case) are well-aligned if no vertex of one of them lies in the middle of an edge or face of the other. That is, the whole joining lower dimensional element (face or edge) is common to both boxes.

Let us first analyze 2D-OPP (orthogonal pseudo-polygons). A set of well-aligned rectangles determines a 2D-OPP whose vertices must coincide with some of the rectangles' vertices. Each of these rectangles' vertices can be considered as the origin of a local coordinate system, and they may belong to up to four rectangles (one for each local quadrant). The OPP's

vertices are determined according to the presence or absence of each of this four surrounding rectangles. There are $2^4 = 16$ possible combinations which, by applying rotational symmetries, may be grouped on six equivalence classes (configurations) shown on figures 2.2 (*a* to *f*).

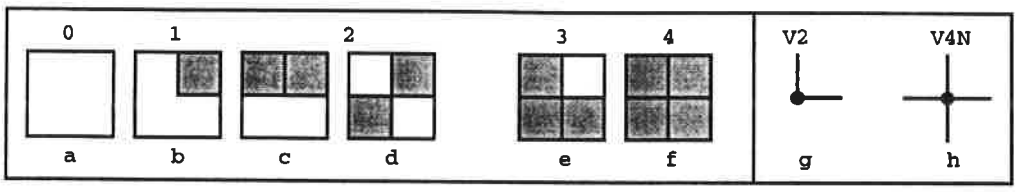


Figure 2.2: (a to f) Possible configurations with 0 to 4 surrounding rectangles.
g) Two-manifold vertex. h) Non-manifold vertex.

From this analysis it results that there are only two types of vertices in a 2D-OPP: the two-manifold vertex with two incident edges, and the non-manifold vertex with four incident edges. They will be referred as V2 and V4N (the "N" stands for Non-manifold), see figures 2.2 (g and h). Figure 2.2 shows that both configurations (b) and (e) represent a V2; configuration (d) represents a V4N; but configurations (a), (c) and (f) represent no vertex. Every vertex of a two-manifold (i.e., simple) orthogonal polygon is V2, while an orthogonal pseudo-polygon has also a V4N wherever its boundary intersects itself.

Let's now analyze 3D-OPP. A set of well-aligned boxes determines an OPP whose vertices must coincide with some of the boxes' vertices. Similarly, each of these boxes' vertices may belong to up to eight boxes (on each local octant). The OPP's vertices are determined according to the presence or absence of each of the surrounding eight boxes. There are $2^8 = 256$ possible combinations which, by applying rotational symmetries, may be grouped into 22 equivalence classes (configurations) [Srihari81], shown on figure 2.3. Grouping complementaries leads to the 14 basic patterns [Lorensen87], also called major cases [vanGelder94], figures *a* to *n*.

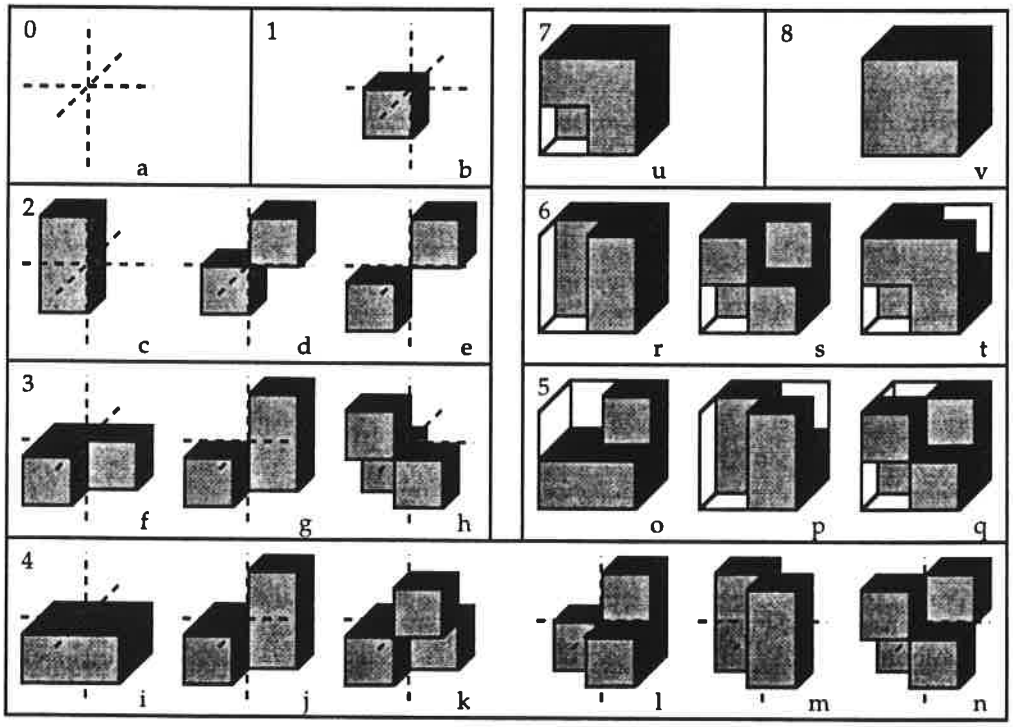


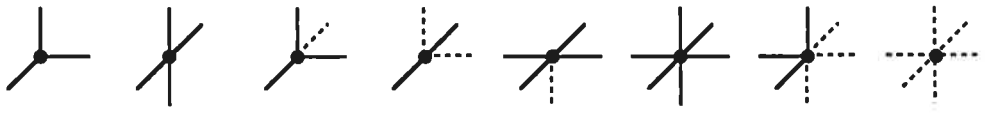
Figure 2.3: Possible configurations with 0 to 8 surrounding boxes.

Note that each configuration, except a, c, i, m, r and v, represents a vertex for the final OPP, since these exceptions respectively represent a point: outside the OPP, on a two-manifold edge, on a face, on a non-manifold edge, on a two-manifold edge, and an interior point. Also only

configurations b, d and f represent locally supportable vertices, while configurations o, s and u represent complementary supportable vertices.

From the analysis of figure 2.3 vertices can be classified on eight types depending on the number of two-manifold and non-manifold edges incident to them. Figure 2.4 shows these types which will be referred as V3, V4, V4N1, V4N2, V5N, V6, V6N1 and V6N2.

Types V3, V4 and V6 correspond to vertices that can appear in an OP, except the cases corresponding to configuration (e) and its symmetric (t). These exceptions correspond to a non-manifold vertex since it is the apex of two cones of faces and they can only appear in OPP. The remaining five types of vertices have non-manifold incident edges and therefore they also can only appear in OPP.



	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
Corresponding configurations:	b,f,o,u	j	g, p	k	d, s	e, l, t	h, q	n
Supportable vertices	b, f	-	-	-	d	-	-	-
Complementary supportable	o, u	-	-	-	s	-	-	-

Figure 2.4: Vertex classification according to its number of incident edges.
(dashed lines represent non-manifold edges)

Finally and concerning with supportability, in OP only V3 are (complementary or locally) supportable; the remaining (V4 and V6) are not supportable [Juan95]. In OPP only V3 and V5N are supportable; the remaining, are not (see figures 2.3 &2.4).

3 The Extreme Vertices Model.

In this section the Extreme Vertices Model (EVM) for OP is described, and then this description is extended for OPP. Finally, basic operations on EVM are also presented.

As a convention, every linear or planar element will be axis-prefixable, which means that the element is parallel to an axis according to its prefix (x-, y-, or z-) for linear elements, or perpendicular to the prefixed axis for planar elements; e.g., "x-edge" stands for an edge parallel to the x axis, while "z-face" refer s to a face perpendicular to the z-axis.

3.1 The Extreme Vertices Model (EVM) for two-manifold OP.

Definition 3.1 *A brink (or extended-edge) is the longest uninterrupted segment, built out of a sequence of collinear and contiguous two-manifold edges of an OP. Similarly (extending the concept of brink to a surface), an extended-face is the maximal set of coplanar and contiguous faces, joined by lower dimensional elements (vertices or edges). (Brinks and extended-faces are axis-prefixable.)*

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 figure 3.1, center). Even though edges $\overline{ab}$ and $\overline{cd}$ are collinear in the example of this figure, they do not constitute a single brink because they do not belong to a unique uninterrupted segment. Similarly, every extended-face consists of one or more faces (see figure 3.1, right).

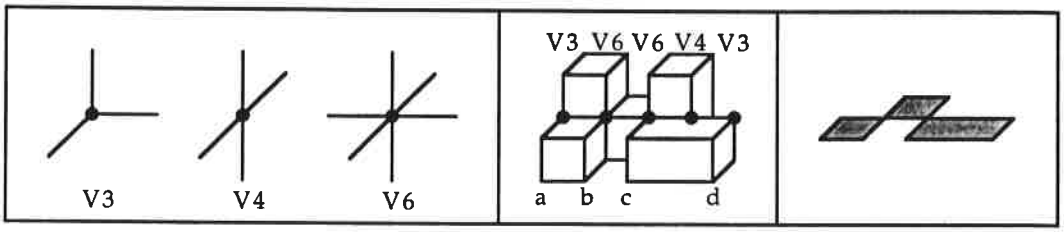


Figure 3.1. (left): Edges meeting at a V3, V4 and V6 vertex. (center): An OP with a brink having four edges and five vertices; the Extreme Vertices are V3 while the inner are V4 and V6. (There are also five other brinks of two edges each). (right): An OP's extended-face.

Edges meeting at a V3 vertex are all linearly independent whereas edges meeting at V4 or V6 vertices are not. All four (six) edges meeting at a V4 (V6) vertex belong to two (three) perpendicular directions, that is, they are members of two (three) perpendicular brinks and, hence, they appear as two (three) couples of collinear edges (see figure 3.1). Also every V4 or V6 incident edge has a neighbor edge in the brink corresponding to its direction. In other words, any V4 or V6 is the only common point of two neighbor edges of a same brink.

Lemma 3.1 *In a brink both ending vertices are V3 and the remaining (interior) are V4 or V6.*

Proof: Because all the edges meeting at vertices V4 or V6 have a neighbor edge in the same brink, such vertices cannot appear at the end of any brink. And because any edge meeting at vertices V3 does not have any neighbor edge in the same brink, such vertices must appear only at the end of brinks (see figure 3.1). ■

Definition 3.2 *We will call Extreme Vertices (EV) of an OP to the ending (or extreme) vertices of all the OP brinks, i.e., the V3 vertices of the OP.*

Definition 3.3 *We define the Extreme Vertices Model (EVM) for OP as a model that only stores all their Extreme Vertices. Moreover EVM(P) will denote the Extreme Vertices Model of an orthogonal polyhedron P.*

Lemma 3.2 *Let P be an OP and OH(P) be its iso-oriented orthogonal hull or minimum bounding box. Then, only a subset of V3 vertices of P lies on the boundary of OH(P) and, therefore, all V4 and V6 vertices lie in the interior of OH(P).*

Proof: Vertices V3 are locally or complementary supportable and vertices V4 and V6 are non-supportable. Only locally supportable vertices of an OP can lie on its minimum bounding box. ■

Let $VX = \{x_1, x_2, \ldots, x_{nx}\}$ be the ordered set of different values for the x coordinates of every Extreme Vertex, nx being the total number of different x values (and similarly VY and VZ for its y and z coordinates, with sizes ny and nz).

Lemma 3.3 For Every vertex V4 or V6 with coordinates (x_i, y_i, z_i), $x_i \in VX - \{x_1, x_{nx}\}$ (and analogously for its y and z coordinates).

Proof: Vertices V4 (V6) are in the interior of two (three) perpendicular brinks (in fact they are the intersection of these brinks), so their coordinates can be obtained from the coordinates of the ending vertices of those brinks, then $x_i \in VX$, $y_i \in VY$ and $z_i \in VZ$. However, from lemma 3.2: $x_1 < x_i < x_{nx}$, also $y_1 < y_i < y_{ny}$ and $z_1 < z_i < z_{nz}$. ■

Theorem 3.1 *The Extreme Vertices Model is a valid B-Rep model for OP.*

Proof: To prove this theorem we must prove that all the geometry, topology and correct orientation of the OP boundary can be obtained from the EVM.

Concerning with geometry, from lemma 3.3, all coordinates of vertices V4 and V6 appear as coordinates of the Extreme Vertices (V3). Then, although vertices V4 and V6 do not appear in the model, they can be inferred from it.

Concerning with topology and orientation, in section 7.5 a conversion algorithm from EVM to a hierarchical B-rep model is presented that will prove this theorem. ■

Definition 3.4 *A plane of vertices of an OP is the set of vertices lying on a plane perpendicular to a main axis. We will also refer as line of vertices (within a plane of vertices) to the set of vertices lying on a line parallel to a main axis. (both of them are axis-prefixable.)*

edge	$\subseteq$	brink	$\subseteq$	line of vertices
face	$\subseteq$	extended - face	$\subseteq$	plane of vertices

Let Q be a point set, then every point $q \in Q \subset E^3$ defines three possible orthogonal lines of vertices and three possible orthogonal planes of vertices incident to q.

Lemma 3.4 *Every line (and plane) of vertices has an even number of Extreme Vertices.*

Proof: There can be any number of brinks within a given line of vertices and each one is defined by a couple of consecutive Extreme Vertices within the given line. Therefore the number of Extreme Vertices within any line and plane of vertices is even. ■

Corollary 3.1 *Every EVM contains an even number of Extreme Vertices.*

Definition 3.5 *A strip is the region between two consecutive planes (lines) of vertices.*

Definition 3.6 *A section is the 2D(1D)-OP resulting from the intersection between a 3D(2D)-OP and an orthogonal plane (line) perpendicular to a main axis which must not coincide with any plane (line) of vertices. (sections are axis-prefixable.)*

The example on figure 3.2 shows a 3D graph (a) of the point set representing an EVM. Its x-brinks (b), y-brinks (c) and z-brinks (d) can be found by matching up contiguous vertices on the appropriate lines of vertices. Moreover, the z-faces and z-extended-faces (e) can be built by linking x-brinks & y-brinks together. y-extended-faces (f) and x-extended-faces (g) can be built in a similar way; also in (g) some z-lines of vertices are shown. Finally, linking all brinks together leads to a wire-frame model (h) for the represented OP. Its three x-sections (i) (the shaded polygons, where the middle one is composed by two polygons), as well as the y-sections & z-sections (not shown), and the polyhedron itself (j) can be found using algorithms discussed later in this work. Note that this OP has a V4 and a V6, circled in (j), which do not belong to the EV set, so they are not marked in the remaining (a) to (i) figures.

If an OP contains a V4 (V6) vertex, it will have one (three) extended-face(s) composed by two faces joined by that vertex. Figure 3.2 shows in (j) a V4 and a V6, and in (e) two extended-faces each composed by two faces: the upper one contains the V4 and the lower one, the V6. The remaining two extended-faces joining in this V6 vertex are shown in in (f) and in (g).

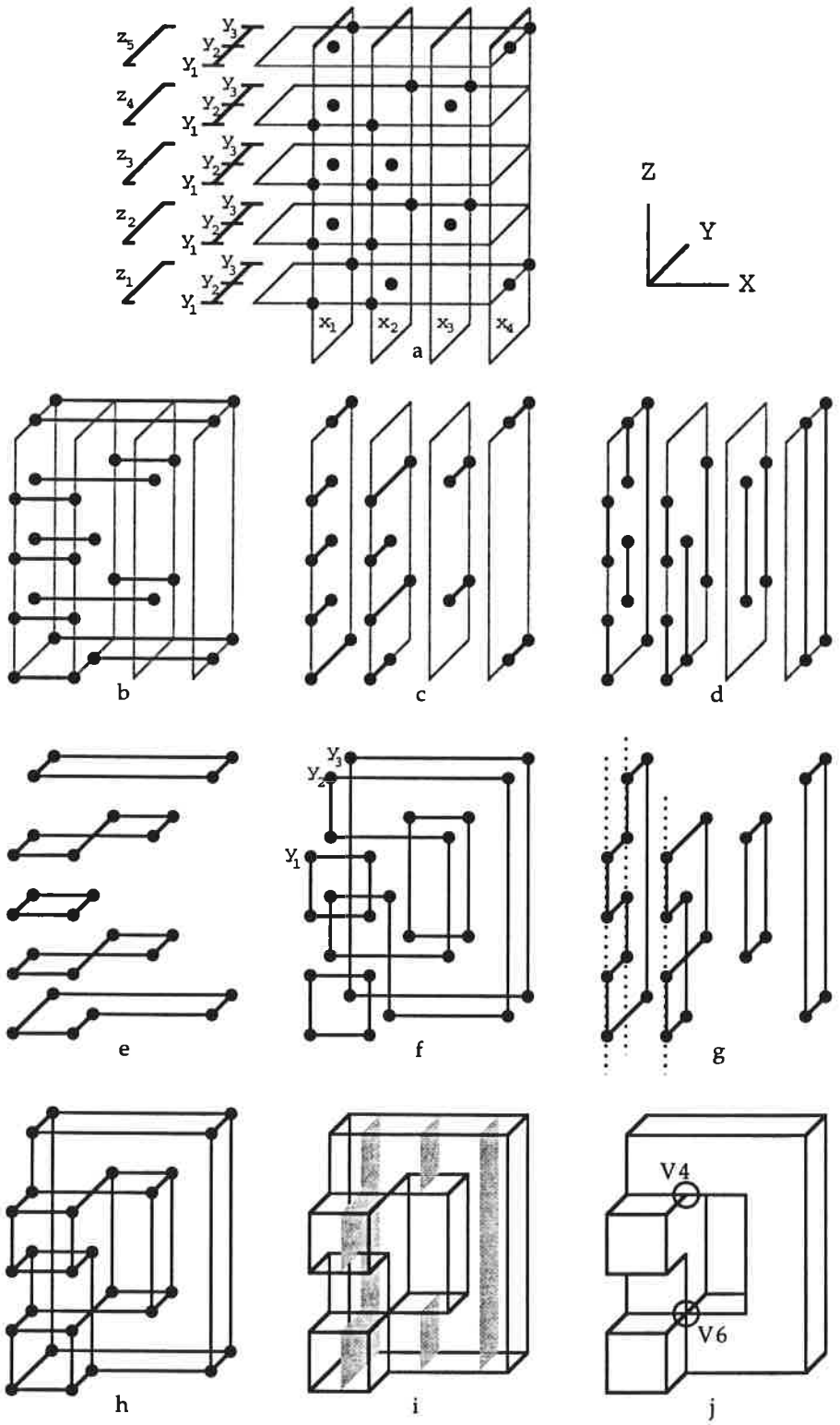


Figure 3.2: A complete example of an two-manifold OP with a V4 and a V6.
(see text for details)

3.2 Extending the Extreme Vertices Model to the pseudo-manifold case.

Let us first analyze 2D-OPP (orthogonal pseudo-polygons). It can easily be shown that, in a 2D-OPP's brink, V2 are the Extreme Vertices, while V4N are the inner ones (see figure 3.3 right). So, a 2D-EVM would only store V2 vertices while V4N can be directly obtained by intersecting brinks. Note that every vertex of a two-manifold polygon is an ExtremeVertex, so edges and brinks are, in this case, equivalent.

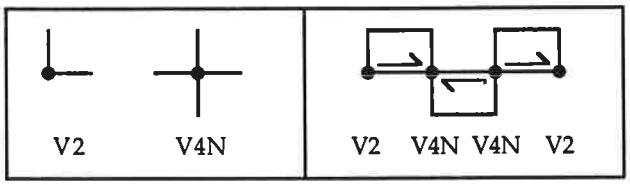


Figure 3.3. (left): Edges meeting at a V2 and V4N vertex. (right): A 2D-OPP's brink containing three edges and four vertices; the Extreme Vertices are V2 while the inner ones are V4N. (there are also two other brinks of two edges each).

Here we need to emphasize that, although edges' direction vectors of a 2D-OPP's brink are collinear, they may have opposite directions in order to fulfill the Jordan theorem, (see figure 3.3, right). In a similar way, even though all quasi-disjoint faces' normal vectors of an extended face are parallel, they may have opposite directions. Faces of a 3D-OP extended-face can only be joined by a V4 or V6, and their normal vectors point to the same direction when joined by a V6, but to opposite directions when joined by a V4. (see figure 3.2 (j), for an example).

Let's now analyze 3D-OPP. For a natural extension and ignoring non-manifold edges since they do not belong to brinks, note that vertices V4N1 and V6N1 would be just like V3; V4N2 would be points inside a single edge; V5N would behave like V4; And V6N2 would be isolated points (see figure 2.4). Then two-manifold edges meeting at a V3, V4N1 and V6N1 are all linearly independent whereas two-manifold edges meeting at V4, V4N2, V5N or V6 are not, and they can be grouped in one, two or three couples of collinear edges; in other words, any V4, V4N2, V5N or V6 is the only common point of two neighbor edges of a same brink.

Based on all these considerations, we can now extend to OPP all definitions, lemmas and theorems stated for the two-manifold case. The number inside parenthesis refers to the corresponding definition, lemma or theorem for the two-manifold case of the preceding section. Lemmas and theorems won't be proved again, since their proofs would be practically the same (except lemma 3.5). Also, definition 3.1 for brinks and extended-faces stays the same.

Lemma 3.5 (3.1) *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.*

Proof: 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, therefore those vertices mark the end of brinks in all three orthogonal directions. Any V4, V4N2, V5N or V6 is the only common point of two neighbor 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; therfore neither this vertex itself does. ∎

Definition 3.7 (3.2) *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.*

Definition 3.8 (3.3) *We define the Extreme Vertices Model (EVM) for OPP as a model that only stores all their Extreme Vertices. Moreover EVM(P) will denote the EVM of an orthogonal pseudo-polyhedron P.*

Lemma 3.6 (3.2) *Let P be an OPP and OH(P) be its iso-oriented orthogonal hull or minimum bounding box. Then, only a subset of vertices V3 and V5N of P lies on the boundary of OH(P) and, therefore, all V4, V4N1, V4N2, V6, V6N1 and V6N2 vertices lie in the interior of OH(P).*

Proof: Same as Lemma 3.2, but also V5N is supportable. ■

Let VX = $\{x_1, x_2, \ldots, x_{nx}\}$ be the ordered set of different values for the x coordinates of every Extreme Vertex (V3, V4N1 or V6N1), nx being the total number of different x values (and similarly VY and VZ for its y and z coordinates, with sizes ny and nz).

Lemma 3.7 (3.3) *For* every non extreme vertex (V4, V4N2, V5N, V6 and V6N2) with coordinates (x_i, y_i, z_i), $x_i \in VX$, $y_i \in VY$ and $z_i \in VZ$.

Proof: Vertices V4, V5N andV6 are in the interior of two or three perpendicular brinks, so their coordinates can be obtained from the ending vertices's coordinates of those brinks, then $x_i \in VX$, $y_i \in VY$ and $z_i \in VZ$. A vertex V4N2 is in the interior of one brink, so two of its three coordinates can be known in a similar way. The remaining one, as well as all three coordinates of a vertex V6N2, can be obtained using the fact that the other end of a non-manifold edge can be either a vertex whith known coordinates (V4N1, V5N or V6N1) , or again, a V4N2 or V6N2. In this case, by lemma 3.6 neither of these vertices lie on OH(P), thus another non-manifold edge of the new vertex can be used to repeat this procedure until a vertex whith known coordinates is reached. ■

Theorem 3.2 (3.1) *The Extreme Vertices Model is a valid B-Rep model for OPP.*

Proof: Same as Theorem 3.1■

The remaining concepts for OPP (lines and planes of vertices, strips, and sections) do not differ from OP, neither do the properties of Lemma 3.4 and Corollary 3.1 regarding the even number of Extreme Vertices, so we will not expose them here again.

Lemma 3.8 *An Extreme Vertex of a 3D-OPP (2D-OPP) is surrounded by an odd number of well-aligned boxes (rectangles), as defined in section 2.3. An even number number of surrounding boxes (rectangles) either defines a non-extreme vertex, or does not define any vertex at all.*

Proof: The proof comes from the exhaustive analysis of vertices done in section 2.3 (see figure 2.2 for the 2D case and figures 2.3 and 2.4 for the 3D case). ■

Corollary 3.2 *Every Extreme Vertex of an OPP has an odd number of incident faces.*

Proof: The proof also comes from the exhaustive analysis of vertices done in section 2.3 (see table in 2.4). ■

According to definition 3.1, a brink cannot contain any non-manifold edge, therefore even cases like an uninterrupted mixture of non-manifold and two-manifold collinear contiguous edges will not be considered as a single brink. Figure 3.4 shows in (a) an example of this mixture. Even though edges $\overline{AB}$, $\overline{BC}$ and $\overline{CD}$ are collinear and form an uninterrupted segment, they do not constitute a single brink because $\overline{BC}$ is a non-manifold edge. Also note that vertices *A* and *D* are V3, while vertices *B* and *C* are V4N1; so all of them are Extreme Vertices, therefore they define two brinks, not just one.

In figure 3.4 (b) the pseudo-polyhedron is disassembled to show the extended-faces. Two of them are composed by two faces, joined by a (non-manifold) edge, and whose normal vectors have opposite directions. Figure 3.4 (c) shows the z-extended-face. and figure 3.4 (d) shows the y-extended-face. (these two extended-faces correspond to the shaded ones in figure 3.4 (b); and the non-manifold edge is dashed).

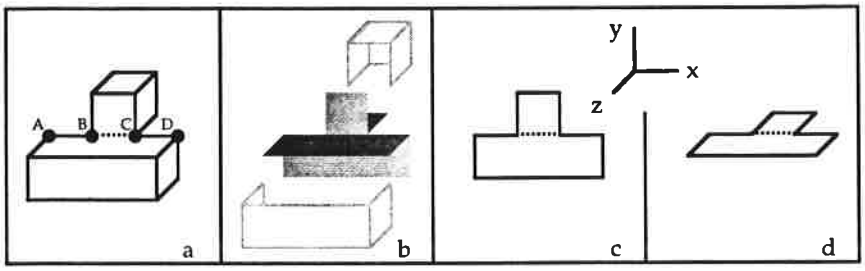


Figure 3.4: a) Example of an uninterrupted mixture of non-manifold and two-manifold collinear edges, they do not constitute a single brink. b) disassembly of the pseudo-polyhedron. c) z-extended-face. d) y-extended-face. (the non-manifold edge is dashed)

We have shown that EVM can represent OP and OPP. A simple example (see figure 3.5) will show that non-manifold polyhedra cannot be represented using an EVM:

The top row in figure 3.5 shows an (U-shaped) OP and its corresponding EVM. It also shows that the of y-brinks' lengths are *a* and *b* (*b*<*a*). The middle y-plane of vertices contains four vertices: A, B, C and D; while vertices A', B', C' and D' lie on the lowest y-plane of vertices. If the middle y-plane of vertices is lowered by decreasing the y-coordinate of points A, B, C and D simultaneously (thus increasing value b) then the polyhedron will keep its topological form as long as value *b* does not reach *a*.

The middle row in figure 3.5 shows the "expected" non-manifold polyhedron when value b reaches value a. However, the set of 16 vertices of this EVM constructs the polyhedron shown on the right side. This OP is the regulariztion of the "expected" non-manifold one.

The bottom row in figure 3.5 shows that by lengthening the y-brinks to a size beyond the length of a, then an OPP is obtained.

Observe that each EVM of these three examples have sixteen V3 vertices. These are all the OP vertices of the top and middle row. The OPP on the bottom row has also four V5N vertices.

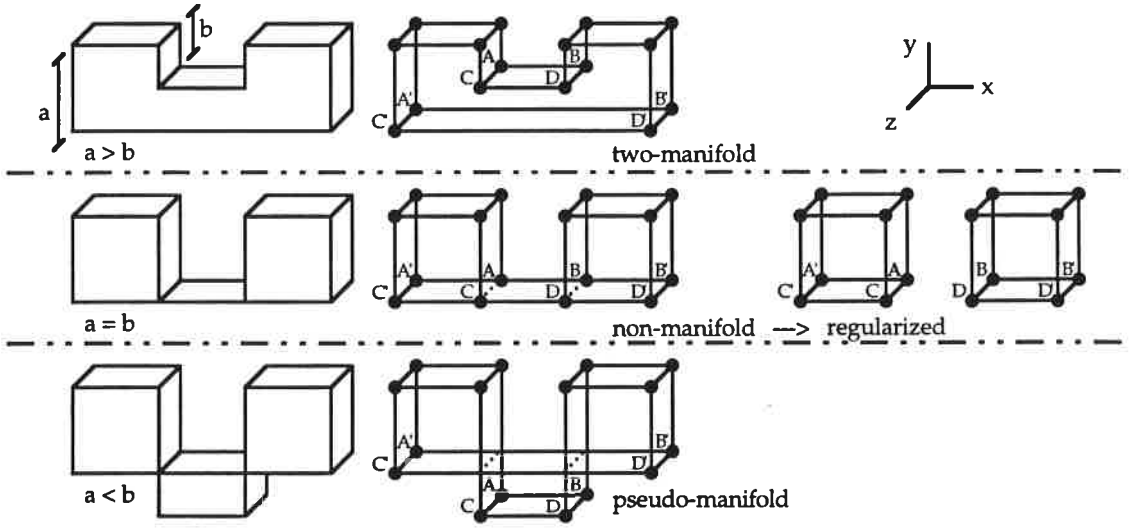


Figure 3.5: Two-manifold and pseudo-manifold OP can be represented using a EVM, but non-manifold OP cannot, since a regularized OPP will be interpreted instead. (see text for details)

In order to describe operations on EVM, we will define a more elaborated EVM which we will refer as ABC-sorted EVM.

3.3 The ABC-sorted EVM.

Definition 3.9 *An ABC-sorted EVM is an EVM where its Extreme Vertices are sorted first by coordinate A, then by B and then by C.*

EVM can be sorted on six different ways: XYZ, XZY, YXZ, YZX, ZXY and ZYX. For example, in a ZXY-sorted EVM, its Extreme Vertices are arranged in their z-planes of vertices (i.e., with the same z coordinate). In each such a plane they are arranged in their y-lines of vertices (i.e., with the same x coordinate). Finally they appear as y intervals (see figure 3.6).

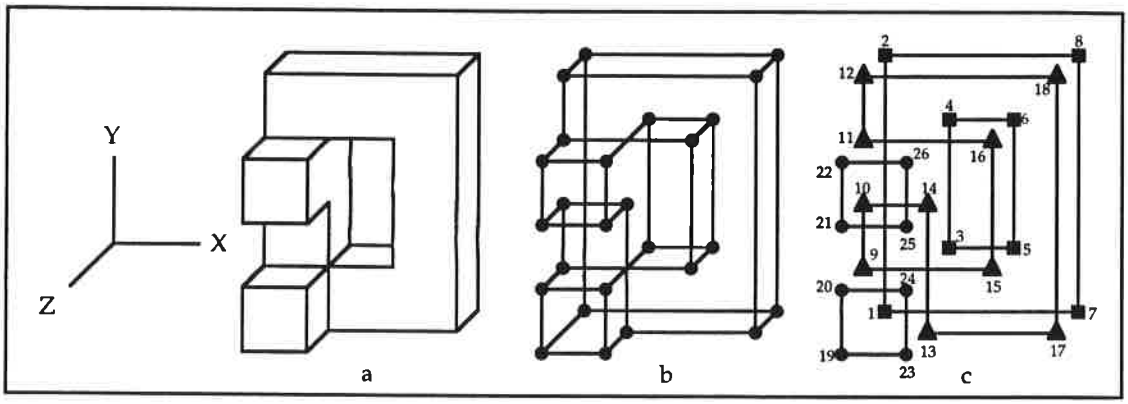


Figure 3.6: ZXY-sorted EVM. a) A hidden line representation of an OP with one V6, one V4 and 26 V3s. b) Its corresponding wire-frame representation. c) This representation shows the order number for each V3 vertex. It also shows the model's three z-planes of vertices (with different marks)

All the orthogonal planes intersecting an OPP in the same strip give the same section. Hence, every strip has its representing section. Furthermore, as an OPP can be interpreted as a sequence of strips, we can define the sequence of sections for an OPP.

An ABC-sorted EVM is a sequence of planes (lines) of vertices. The number np of elements in this sequence is the number of different A-coordinates in the model. The number ns of sections is $ns = np + 1$ because the empty sections S_0 and S_{np} are also considered. Figure 3.7 shows the sections and planes of vertices for an OP.

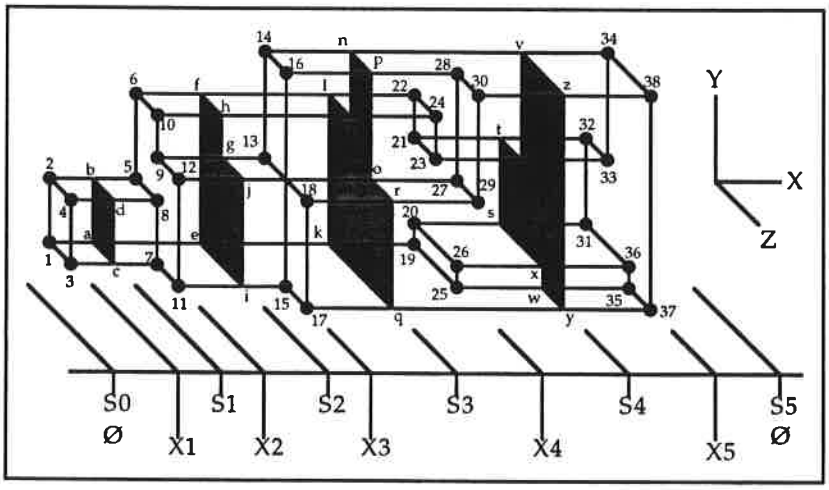


Figure 3.7: This OP is represented in a XZY-sorted EVM. It has 5 x-planes of vertices, X1 to X5. Each of them corresponds to the set of vertices with the same coordinate X. The shadowed polygons are its four sections S1 to S4. We also consider two more empty sections, S0 and S5.

An ABC-sorted EVM can represent n-dimensional OPP (n≤3) by taking into account the last n coordinates. Then planes and lines of vertices of an OPP will be represented also in this model. Moreover, as a section is actually an OPP, 1D and 2D sections will also be represented in this model.

So, we defined the **ABCsorted** type with the following operations:

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

PROCEDURE PutPlv(INPUT plv: ABCsorted; I/O P: ABCsorted, dim: INTEGER)
{Appends a plane (dim=2) or a line (dim=1) of vertices to 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}

PROCEDURE PutBrink(INPUT Vb, Ve: Vertex; I/O P: ABCsorted)
{Appends to an EVM a brink defined by its Extreme Vertices
(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 End(P: ABCsorted) RETURN Boolean
{Returns TRUE if the end of P has been reached}

PROCEDURE SetCoord(I/O P: ABCsorted; INPUT Coord: REAL, dim: INTEGER)
{Sets the A (dim=2) or the B (dim=1) coordinate to Coord on every vertex
of a plane (line) of vertices P}

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

3.4 The Exclusive OR operation on EVM.

The Exclusive OR operation, $\otimes$, over two sets A and B, which is defined as: $A \otimes B = (A \cup B) - (A \cap B)$, or equivalently as: $A \otimes B = (A - B) \cup (B - A)$, can be easily carried out on EVM, as stated by the following theorem:

Theorem 3.3 *Let P and Q be two d-dimensional OPP having $EVM(P)$ and $EVM(Q)$ as their respective models , then $EVM(P \otimes Q) = EVM(P) \otimes EVM(Q)$.*

See [Aguilera98] for a proof.

It should be noted that while P, Q and $P \otimes Q$ are infinite point sets, $EVM(P)$, $EVM(Q)$ and $EVM(P) \otimes EVM(Q)$ are finite point sets.

Now, we define the function `MergeXor`, which performs an XOR between two ABC-sorted EVM. Since the model is sorted, this function consists on a merging-like algorithm (as suggested by its name), and thus, it will run on linear time.

```
FUNCTION MergeXor(P, Q: ABCsorted; dim: INTEGER) RETURN ABCsorted
{Applies the Exclusive OR operation to the vertices of P and Q and
returns the resulting set (using only the last dim coordinates of each
vertex and ignoring the remaining)}
```

Corollary 3.3 *Let P and Q be d-dimensional disjoint or quasi-disjoint OPP having $EVM(P)$ and $EVM(Q)$ as their respective models, then $EVM(P \cup Q) = EVM(P) \otimes EVM(Q)$.*

Proof: If P and Q are disjoint or quasidisjoint OPP then $P \cap Q = \varnothing$, so $P \otimes Q = (P \cup Q) - \varnothing = P \cup Q$ thus, by theorem 3.3, $EVM(P \cup Q) = EVM(P) \otimes EVM(Q)$. ■

Corollary 3.4 *Let P and Q be two d-dimensional OPP such that $P \supseteq Q$, with $EVM(P)$ and $EVM(Q)$ as their respective models, then $EVM(P - Q) = EVM(P) \otimes EVM(Q)$ (i.e., the complement of Q with respect to P).*

Proof: If $Q \subseteq P$ then $Q - P = \varnothing$, thus $P \otimes Q = (P - Q) \cup \varnothing = P - Q$; then, by theorem 3.3, $EVM(P - Q) = EVM(P) \otimes EVM(Q)$. ■

For general boolean operations see section 4.

3.5 Computing sections from planes (lines) of vertices.

Any section S_i of an OPP is computed by doing an exclusive OR between its previous section S_{i-1} and its previous plane (line) of vertices plv_i:

$$S_i = S_{i-1} \otimes plv_i, \quad \forall i \in [1, np], \text{ wich can also be written as } S_i = \overset{i}{\underset{k=1}{\otimes}} plv_k$$

where

$S_0 = \varnothing, \quad S_{np} = \varnothing.$ (Note that this implies $S_1 = plv_1$ and $S_{np-1} = plv_{np}$)

Then we define the corresponding function GetSection:

```
FUNCTION GetSection(S, plv: ABCsorted; dim: INTEGER)
RETURN ABCsorted
{returns the next section of an OPP whose previous section is S. This
function works for dimension 2 or 1. If dim=2 (dim=1), plv is the
previous plane (line) of vertices and S is a 2D (1D) section}

      RETURN(MergeXor(S, plv, dim))
ENDFUNCTION
```

Thus, an algorithm that computes the sequence of sections of an OPP from its EVM using the previous functions would be as follows:

```
PROCEDURE EVM_to_SectionSequence(INPUT P:  ABCsorted; dim: INTEGER;
                                 OUTPUT S[]:ABCsorted; coord[]:REAL);
{If dim=2 (dim=1) then builds the sequence of 2D (1D) sections S[] of an
orthogonal polyhedron (polygon) P.}

   i := 0;
   S[i]:= IniEVM();
   plv := ReadPlv(P, dim);
   WHILE NOT End(P) DO
      i := i+1;
      S[i] := GetSection(plv, S[i-1], dim);
      coord[i] := GetCoord(plv, dim);
      plv := ReadPlv(P, dim);
   ENDWHILE;

ENDPROCEDURE
```

The call to `GetSection` whithin this algorithm, when used on an XYZ- or XZY-sorted EVM, for example, will ignore the first (x) coordinate of every Extreme Vertex. In other words all vertices lying on each x-line of vertices will be considered as the same point.This is true because it uses function `MergeXor` (with `dim=2`), which uses only the last `dim`, i.e., the last 2 coordinates (*y* and z) of each Extreme Vertex. Finally `dim=2` because x-sections and x-planes of vertices (which are 2D-OP) are XORed using function `MergeXor`.

For examle, in the XYZ-sorted EVM of figure 3.7, vertices 9, 13, 23 & 33, which have the same y and z coordinate, will be considered as the same (2D) point across all x-sections and x-planes of vertices. Thus vertex 9 will define vertex g on S2 (in the same x-line of vertices); vertex 13 will remove it so it won't appear on S3; vertex 23 will redefine it as vertex u in S4; and finally vertex 33 will remove it to the final S5 empty section. All this takes place on the same x-line of vertices.

3.6 Computing planes (lines) of vertices from sections.

A plane (line) of vertices P_i of an OPP is computed by doing an exclusive OR between its previous S_{i-1} and next S_i sections:

$$P_i = S_{i-1} \otimes S_i, \quad \forall i \in [1, np]$$

Then we define the corresponding function GetPlv:

```
FUNCTION GetPlv(Si, Sj: ABCsorted; dim: INTEGER) RETURN ABCsorted
{This function also works for dimensions 2 or 1. If dim=2 (dim=1), then
Si and Sj are 2D (1D) consecutive sections, and it returns the
corresponding plane (line) of vertices between Si and Sj}

      RETURN(MergeXor(Si, Sj, dim)
ENDFUNCTION
```

Actually, this function does the same that the GetSection function, i.e., an exclusive OR between two sets of vertices, but as they are conceptually different we will use both of them.

Thus, an algorithm that computes the EVM from a sequence of sections of an OPP and their corresponding coordinates in the next higher dimension, using the previous functions would be as follows:

```
FUNCTION SectionSequence_to_EVM(S[]:ABCsorted;coord[]:REAL;dim:INTEGER)
RETURN ABCsorted
{If dim=2 (dim=1) then from the sequence of 2D (1D) sections S[],
it builds the EVM of an orthogonal polyhedron (polygon) P.}

   P := IniEVM();
   FOR each section i DO
      plv := GetPlv(S[i-1], S[i], dim);
      SetCoord(plv, coord[i], dim);
      PutPlv(plv, P, dim);
   ENDFOR;
   RETURN(P);
ENDFUNCTION
```

3.7 Condition for a point set to be a valid EVM.

The following theorem gives a necessary and sufficient condition for a finite point set to be a valid EVM:

Theorem 3.4 *Let Q be a finite point set in E^d, then*

Q is a valid EVM for some pseudo-polyhedron P, i.e., Q=EVM(P)

$$\Longleftrightarrow$$

Every possible line of vertices, in each dimension of Q, holds an even number of points of Q.

Proof:

$\Rightarrow$ Lemma 3.4 proves that every line of vertices of an EVM holds an even number of Extreme Vertices.

$\Leftarrow$ If every possible line of vertices, in each dimension of Q, holds an even number of points, then the last (and external) section of P, S_{np}, perpendicular to any axis, and whose EVM is computed as

$$EVM(S_{np}) = \bigotimes_{k=1}^{np} EVM(plv_k) \qquad \text{(according to Section 3.5)}$$

will indeed be empty ($EVM(S_{np}) = \varnothing$), and thus P will be a valid (and bounded) OPP. ■

Function XOR cancels out every pair of Extreme Vertices of different planes of vertices that lie on a same line of vertices perpendicular to those planes of vertices. Thus:

- if every line of vertices contains an even number of points, then the XOR function will produce the empty set as the last section of P, S_{np}, perpendicular to the given line of vertices. Conversely,

- if a line of vertices contains an odd number of points then the XOR function will not produce the empty set as the last section of P, S_{np}, perpendicular to the given line of vertices

4 The Boolean Operations Algorithm.

Now, we are able to present the boolean operations algorithm. The algorithm merges two OPP, say P and Q, represented in the same ABC-sorted EVM, in such a way that the corresponding planes of vertices become also merged. We consider all the resulting strips, some of them will correspond to untouched strips of P or Q and only one section will have to be considered. However some other strips will correspond to a part of a P strip and a part of a Q strip with their corresponding sections. The algorithm considers this sections as 2D-OPP and operates them in the same way.

We can explain the algorithm as follows. The sequence of sections for objects P and Q are computed. Then these sections are merged in order to compute the sequence of sections of the R resulting object. Finally, from this sequence of sections, the ABC-sorted EVM of the resulting object R is obtained. Nevertheless, the implemented algorithm does not work in this sequential form; it actually works in a wholly merged form and only needs to store one section for each of the P and Q operands and two consecutive sections for the result R. Then, the algorithm is O(n) as merging-like algorithms are.

The corresponding Boolean Operations Algorithm can be stated as:

```
TYPE Object = ENUM {P, Q} ENDTYPE
FUNCTION OpBool(P, Q: ABCsorted; {the input objects     }
                dim : INTEGER;   {dimension of P and Q }
                op  : BoolOp)    {the Boolean operation}
               RETURN ABCsorted
   VAR
      s[P..Q]: ABCsorted;    {s[P], s[Q]: current sections of P, Q     }
      obj: Object;           {the current selected object              }
      plvi,plvo: ABCsorted; {input & output planes (lines) of vertices}
      Coord: Real;           {the common coordinate of the current plv }
      R, sRprevious, sRcurrent: ABCsorted; {R and two of its sections }
   ENDVAR

   IF dim = 1 THEN
      RETURN(OpBool1D(P, Q, op))
   ELSE
      dim:= dim-1
      s[P]:= IniEVM()
      s[Q]:= IniEVM()
      R:= IniEVM()
      sRcurrent:= IniEVM()
      GetPlane(P, Q, dim, plvi, Coord, obj)
      WHILE NOT End(P) AND NOT End(Q) DO
         S[obj]:= GetSection(plvi, S[obj], dim)
         sRprevious:= sRcurrent
         sRcurrent:= OpBool(s[P], s[Q], dim, op)
         plvo:= GetPlv(sRprevious, sRcurrent, dim)
         SetCoord(plvo, Coord, dim)
         PutPlv(plvo, R, dim)
         GetPlane(P, Q, dim, plvi, Coord, obj)
      ENDWHILE
      WHILE NOT End(P) DO
         PutBool(plvi, R, op); plvi:= ReadPlv(P, dim)
      ENDWHILE
      WHILE NOT End(Q) DO
         PutBool(plvi, R, op); plvi:= ReadPlv(Q, dim)
      ENDWHILE
      RETURN(R)
   ENDIF
ENDFUNCTION
```

- Function `OpBool1D` performs 1D boolean operations between P and Q that now are collinear lines of vertices.

- Procedure `GetPlane` gets the next plane (line) of vertices, `plvi`, of P or Q, with its common coordinate `Coord`, and to which of these objects, `obj`, it belongs. The plane (line) of vertices `plvi` is obtained using function `ReadPlv` and its common coordinate `Coord` using function `GetCoord` (see section 3.3). This procedure works as in a merging process.

- Functions `GetSection` and `GetPlv` perform an exclusive OR between the sets of vertices of their operands (see sections 3.5 & 3.6).

- `OpBool` works for 3D-OPP (dim=3) and for 2D-OPP (dim=2). The recursive case of this procedure is a merging-like algorithm.

- When the end of one of the objects is reached then the main iteration finishes, and the remaining planes (lines) of vertices of the other object are either appended or not to the resulting object depending on the boolean operation considered. Procedure `PutBool` performs this appending process.

Figure 4.1 shows a 2D example, figure 4.2 shows the corresponding running example and figure 4.3 shows a 3D example.

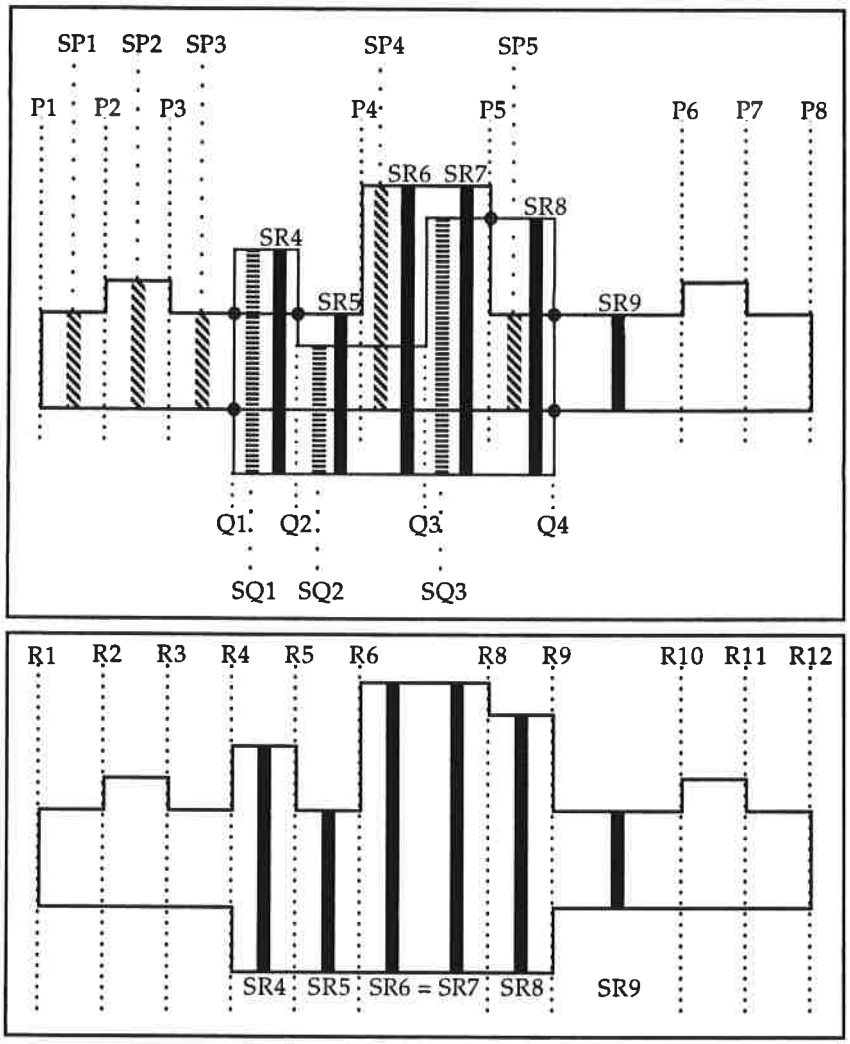


Figure 4.1: 2D Boolean Operations example.

plvi	s[P]	s[Q]	sRprevious	sRcurrent	plvo
P1	SP1 = P1	Ø	Ø	SP1	R1 = P1
P2	SP2	Ø	SP1	SP2	R2 = P2
P3	SP3	Ø	SP2	SP3	R3 = P3
Q1	SP3	SQ1 = Q1	SP3	SR4 = SP3 ∪ SQ1	R4
Q2	SP3	SQ2	SR4	SR5 = SP3 ∪ SQ2	R5
P4	SP4	SQ2	SR5	SR6 = SP4 ∪ SQ2	R6
Q3	SP4	SQ3	SR6	SR7 = SP4 ∪ SQ3	R7= Ø
P5	SP5	SQ3	SR7	SR8 = SP5 ∪ SQ3	R8
Q4	SP5	SQ4 = Ø	SR8	SR9 = SP5 ∪ SQ4	R9
P6					R10 = P6
P7					R11 = P7
P8					R12 = P8

Figure 4.2: Boolean Operations running example. `End(Q)` is `TRUE` when `Q4` is selected. We can observe that `SR6 = SR7` since `SP4 ∪ SQ2 = SP4 ∪ SQ3`, thus making `R7 = Ø`.

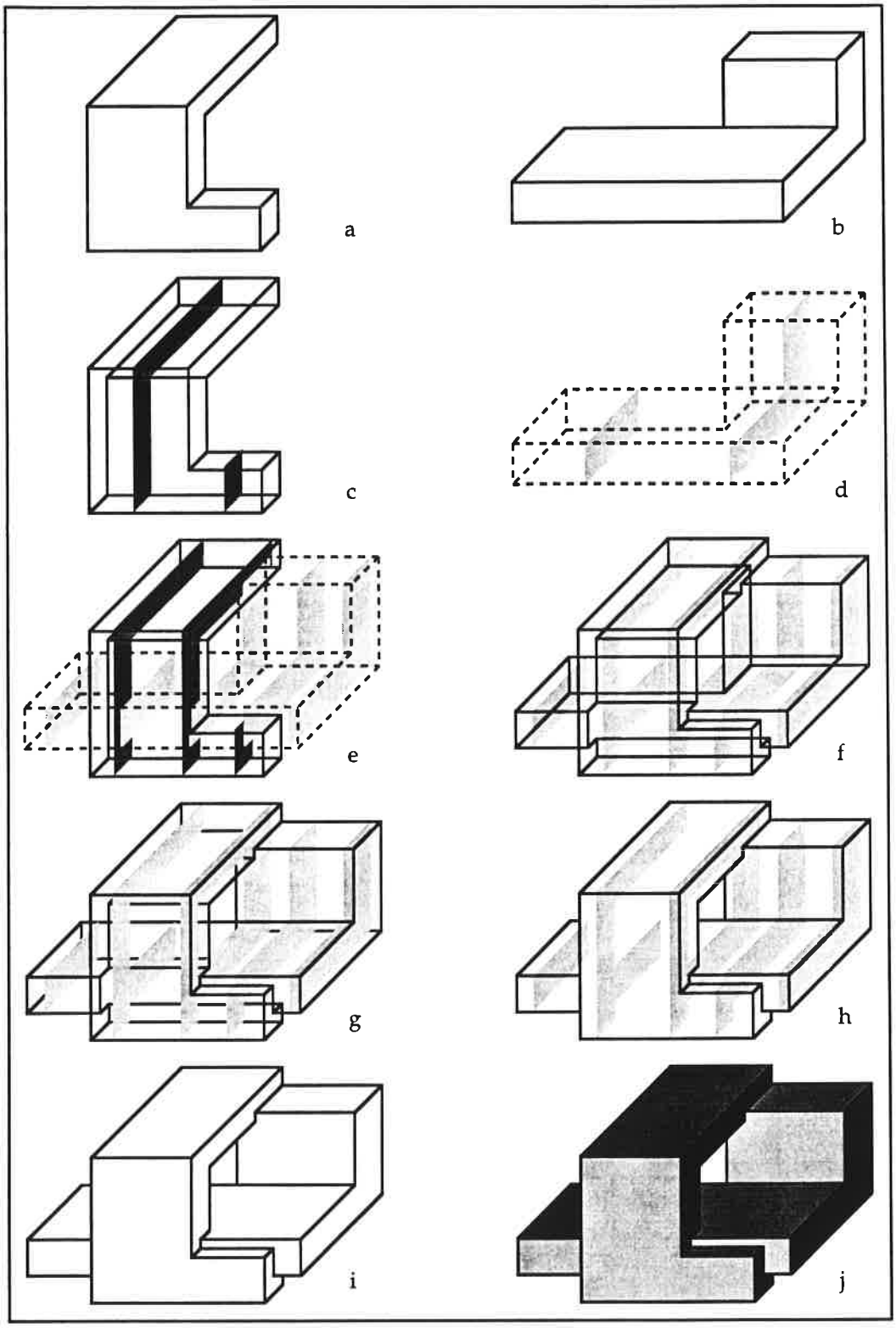


Figure 4.3: A 3D boolean operations example.

5 Splitting Algorithms.

In this section four algorithms are presented that solve the splitting problem between an OPP P and an orthogonal plane SP. All of them compute two resulting OPP: one of them Q, corresponding to the IN halfspace and the other one R, corresponding to the OUT halfspace. All four algorithms work on ABC-sorted EVM and run in O(n) time.

The first three algorithms are respectively used when SP is perpendicular to the A, B or C axis, according to the ABC-sorted EVM. However, they will be presented in a more convenient order. The fourth algorithm has no restricted use, so it may be freely used.

5.1 Splitting plane perpendicular to the C axis.

This algorithm is based on the classical one [Mäntyla88] dealing correctly with ON cases. This splitting algorithm only considers those brinks parallel to the C axis which appear as consecutive couples of vertices in the ABC-sorted model.

Let $V_b = V_{2k-1}$ and $V_e = V_{2k}$ be the beginning and ending vertices of the k^{th} brink parallel to the C axis defined in an ABC-sorted EVM. It can easily be shown that both V_b and V_e have the same A and B coordinates, and the C-coordinate is less in V_b than in V_e. Thus, we can easily know, 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 computed, then V_b and V_i will be assigned to Q; while V_i and V_e to R.

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

```
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}

  Q := IniEVM();
  R := IniEVM();
  ReadBrink(P, Vb, Ve)
  WHILE NOT End(P) DO
    IF  IN(Vb) and ( IN(Ve)   or  ON(Ve)) THEN PutBrink(Vb, Ve, Q);
    IF (ON(Vb)  or  OUT(Vb)) and OUT(Ve)  THEN PutBrink(Vb, Ve, R);
    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 computations. Let for example, $x = x_p$ be the plane equation, and $V_b = (x_1, y, z)$ and $V_e = (x_2, y, z)$ be the vertices in a ZYX or YZX-model, then $V_i = (x_p, y, z)$. Figure 5.1 (left) shows an OP (top) which is to be split. Note that only horizontal brinks are considered. Circles show the intersection points between C-brinks and SP, each of them will generate an Extreme Vertex for Q and another one for R. Also note that one of those circles corresponds to an existing V4, and another one to a V6. The result is shown below, where the dots correspond to newly generated Extreme Vertices.

5.2 Splitting plane perpendicular to the A axis.

This algorithm is based on the fact that when SP is perpendicular to the A axis, then the given ABC-sorted EV set can be divided, without disturbing its order, into three subsets:

- The first one contains the IN vertices of the EV set with respect to SP, which go to Q.
- The second one holds the ON vertices (when SP coincides with a plane of vertices of P).
- The last subset contains the OUT vertices which go to R.

Any of these subsets may be empty. Moreover, section 3.5 showed that the first and last planes of vertices of an OPP coincide with its first and last internal sections, i.e., $S_1 = plv_1$ and $S_{np-1} = plv_{np}$. So this algorithm applies this property to the resulting OPP. It first reads planes of vertices from P, and starts building Q with them, but also uses them to keep track of the current section of P. Then two cases might occur:

- SP does not coincide with any plane of vertices (i.e., SP splits $strip_k$), then S_k will be set as the last plane of vertices for Q, and also as the first one for R.
- SP coincides with plv_k, then S_{k-1} will be set as the last plane of vertices for Q, and S_k as the first one for R. (see figure 5.1, center; where SP coincides with P_3).

```
PROCEDURE SplitA(INPUT P: ABCsorted, SP: plane; OUTPUT Q, R: ABCsorted)
{Splits object P by plane SP (perpendicular to the A axis) into objects
Q and R. Sec is the current section of P}

  Q := IniEVM();                    { Initial planes of vertices go to Q}
  Sec := IniEVM();
  dim := GetDim(P)-1;
  plv := ReadPlv(P, dim);
  WHILE NOT End(P) AND CommonCoord(plv)<CommonCoord(SP) DO
     PutPlv(plv, Q, dim);
     Sec := GetSection(plv, Sec, dim);
     plv := ReadPlv(P, dim);
  ENDWHILE;
  SetCoord(Sec, CommonCoord(SP), dim);
  PutPlv(Sec, Q, dim);                  { Last plane of vertices for Q }

  R := IniEVM();
  IF NOT End(P) THEN
     IF CommonCoord(plv)=CommonCoord(SP) THEN
       Sec := GetSection(plv, Sec, dim);
       SetCoord(Sec, CommonCoord(SP), dim);
       plv := ReadPlv(P, dim);
     ENDIF;
     PutPlv(Sec, R, dim);               { First plane of vertices for R }

     WHILE NOT End(P) DO                {   The remaining plv goes to R }
       PutPlv(plv, R, dim);
       plv := ReadPlv(P, dim);
     ENDWHILE;
  ENDIF;
ENDPROCEDURE
```

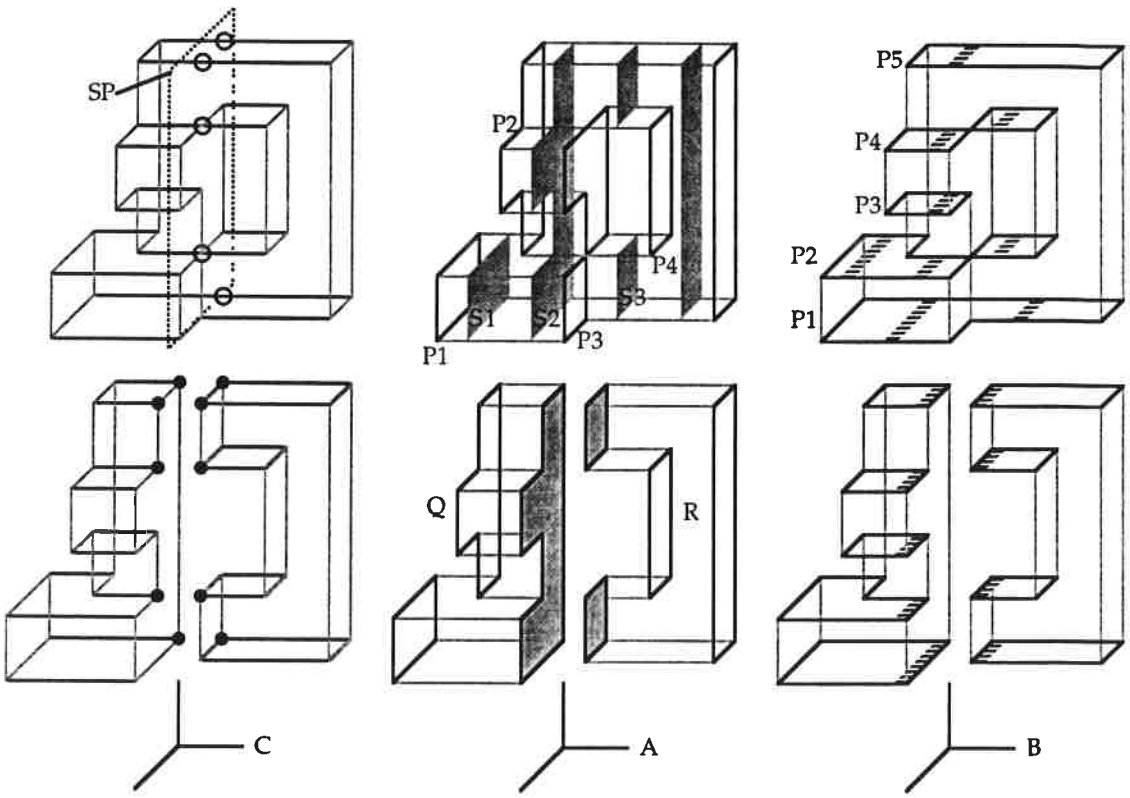


Figure 5.1 (left to right): Splitting an OPP with a plane perpendicular to the C, A and B axis. The splitting plane coincides with a plane of vertices which also contains a V4 and a V6. (See text for more details)

5.3 Splitting plane perpendicular to the B axis.

In this case, planes of vertices are processed as a 2D case of the previous method, since each one may intersect with SP. See figure 5.1 (right).

```
PROCEDURE SplitB(INPUT P: ABCsorted, SP: plane; OUTPUT Q, R: ABCsorted)
{Splits object P by plane SP (perpendicular to the B axis) into objects
Q and R. Sec is the current section of P}

  Q := IniEVM();
  R := IniEVM();
  dim := GetDim(P)-1;

  plv := ReadPlv(P, dim);
  WHILE NOT End(P) DO
    SplitA(plv, SP, Qplv, Rplv);
    PutPlv(Qplv, Q, dim);
    PutPlv(Rplv, R, dim);
    plv := ReadPlv(P, dim);
  ENDWHILE;

ENDPROCEDURE
```

5.4 Splitting plane perpendicular to any axis.

This method does not depend of the splitting plane orientation. It first finds the minimum iso-oriented bounding box B_P of the polyhedron P. Then splits this box with SP obtaining B_Q and B_R, which are, respectively, bounding boxes for Q and R (but not necessarily their minimum ones). Finally Q and R can be obtained by intersecting P with B_Q and B_R. (see Figure 5.2)

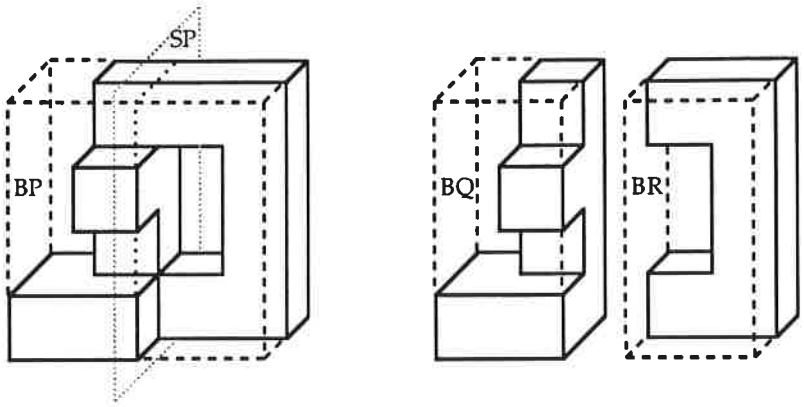


Figure 5.2. (left): A 3D example with a V4 and a V6 vertex (which do not belong to the EVM) and the splitting plane coincides with a plane of vertices. (right): The resulting objects.

6 Point and Line in Polyhedra.

An algorithm for determining if a given point is in the interior (IN), exterior (OUT) or boundary (ON) of an OPP, when stored in an EVM, can be easily constructed. However, every method we tried has lead us to an O(n) algorithm. With a faster algorithm in mind we opted for an alternate scheme for modeling OPP, so we developed a special cell-decomposition model which we call "Ordered Union of Disjoint Boxes".

6.1 The Ordered Union of Disjoint Boxes Model (OUoDBM).

An OPP can be represented as a list of Disjoint Boxes which (when properly sorted) will allow simpler and faster algorithms. This model has proven that it complements EVM rather than compete against it.

From an ABC-sorted EVM we can obtain an ABC-Ordered Union of Disjoint Boxes Model (ABC-OUoDBM) for an OPP P. This model is the set of boxes obtained by:

a) splitting P at every internal plane of vertices of P perpendicular to the A-axis, thus obtaining an ordered sequence A-sections; and

b) splitting each A-section at every one of its internal plane of vertices perpendicular to the B-axis, thus obtaining a sorted sequence of Disjoint Boxes.

As these boxes are created, they are stored in a list. Only two vertices for each box need to be stored, so an ABC-OUoDBM consists in a list of pairs of vertices, which are the minimum and maximum coordinate values for the corresponding box.

The way this list is constructed makes that in this list, boxes are sorted according to the A-values, then to the B-values, and finally to the C-values.

$$Box_k := \langle left, bottom, rear, right, top, front \rangle$$

The reader should note that, in the 2D case, this conversion is equivalent to finding the set of 1D sections of the given OPP and performing an appropriate extrusion with them. Also, because this model stores only two vertices per box, it may be even more compact than the corresponding EVM. Future work will analyze and compare these storing sizes.

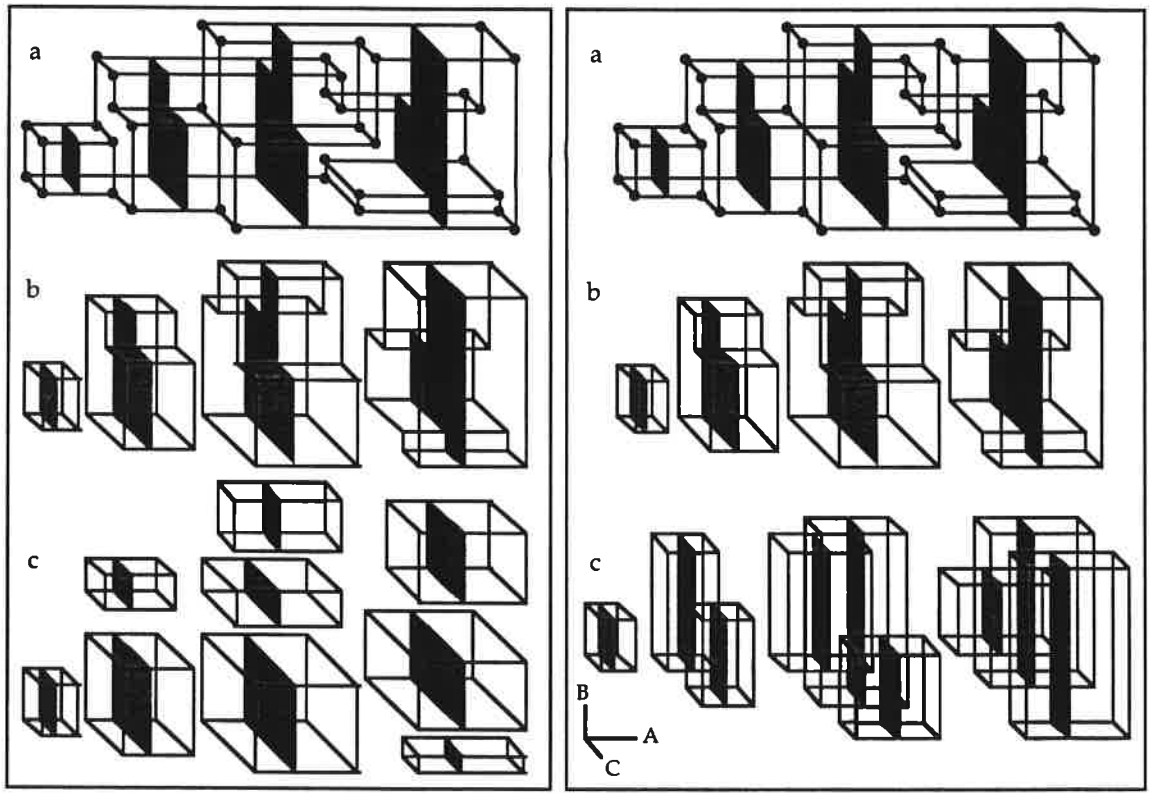


Figure 6.1: Construction of an ABC (left) and ACB (right) Ordered Union of Disjoint Boxes Model. All shaded polygons represent the A-sections.

This model is well suited for displaying an OPP using a back-to-front or similar methods.

6.2 Point in Polyhedra.

By using the OUoDB model, the classification of a point against an OPP is reduced to find a box within the model that contains the given point (IN case), or to prove that no such box exists in the model (OUT case). The ON case will result when the given point lies on a face of only one box, on edges of up to three boxes, or when the point is the vertex of up to seven boxes. Since the model is ordered, binary search can be used and thus, it can be done in O(log n) time.

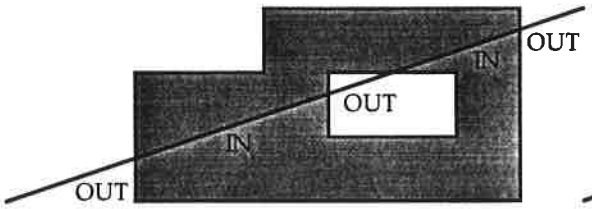
In [Aguilera96] an OPP is proposed as geometric bounds in CSG. Therefore the point in polyhedra algorithm described here, can be very effective for evaluating a CSG model.

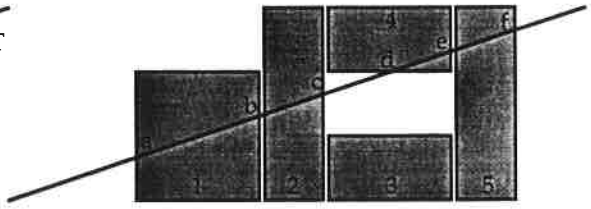
6.3 Line in Polyhedra.

The OUoDB model can also be used for classifying a line against an OPP. The algorithm can simply return whether the line intersects, or furthermore, report the IN and OUT segments. This algorithm can be very useful for many applications like ray-tracing and others.

There are many fast (O(1)) ray-box intersection algorithms published elsewhere, so all we have to do is use one of them with each box within the OUoDB model. This would lead us to an O(n) algorithm.

For example, figure 6.2 (left) shows a line intersecting an OPP. To its right, the same OPP is represented as an Ordered Union of (five) Disjoint Boxes. All ray entry and exit points to and from the OPP can be computed by finding each intersected box entry and exit points; then performing an XOR over them and thus discarding duplicities (an exit from a box and a re-entry into a neighbor box), as shown in figure 6.3.





Box number	Entry point	Exit point
1	a	b
2	b	c
3	-	-
4	d	e
5	e	f

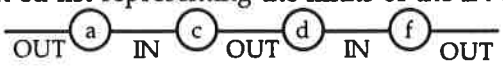
sorted entry/exit points

a b b c d e e f

XOR-ed entry/exit points

a c d f

Figure 6.2. A line intersecting an OPP (top left).The OUoDB model for the same OPP (top right) showing the entry and exit points for each box. (bottom left) Table with the entry and exit points for each box. Then entry/exit points sorted across the intersecting line (bottom right) and finally the XOR-ed list representing the limits of the IN and OUT segments:



Warning: Except for orthogonal lines, computing these entry and exit points for general lines requires floating-point arithmetic, which has been avoided in this work until now.

7 Conversion algorithms between EVM and other solid models.

This section deals with the process of converting the EVM to and from other solid models. The first part covers conversions from spatial-occupancy enumeration models to the EVM. This work does not include conversions from the EVM to those models, because general OPP does not always decompose into identical cells arranged in a fixed, regular grid. However, future work will present algorithms that compute OPP approximations on those models from an EVM. The second part presents algorithms for converting the EVM to and from B-Rep models.

It is convenient to note here that, although input data (i.e., vertices' coordinates) are floating-point values, no time-consuming floating-point arithmetic has ever been performed so far. So there have been no propagation errors. All results are obtained by just classifying vertices' coordinates of the initial data. However, when converting from other models into EVM, we have to be aware of two important things:

a) The foreign model must represent a valid orthogonal (pseudo-)polyhedron, otherwise the obtained result (if any) will not be meaningful at all.

b) If, for any reason, OPP's vertices had to be computed in the foreign model (for instance, if they were not directly available in it), they might contain round-off (or other kind of) errors. In this case some corrective procedures, such as data discretization, will have to be performed.

However, once a potential EV set is obtained, it can be validated using theorem 3.4.

7.1 Spatial-Occupancy Enumeration models to EVM.

A spatial-occupancy enumeration model is a set of black and white cells or nodes. Each cell is a convex OP (i.e., a box and often a cube). The set of black cells represents an OPP P whose vertices coincide with some of the black cells' vertices. A spatial-occupancy enumeration model should provide means for generating a list of all eight vertices of each black cell.

Each of these vertices may be common to (surrounded by) up to eight black cells. So, according to Lemma 3.8, if a vertex is surrounded by an odd number of black cells (well-aligned boxes) then it is an Extreme Vertex. Thus, a spatial-occupancy enumeration model to EVM conversion algorithm would be as simple as collecting every vertex that appears in an odd number of times in such a list, and discarding the remaining ones.

Alternatively, since black nodes are (quasi-)disjoint cubic OP, then by corollary 3.3:

$$EVM(P) = EVM\left(\bigcup_k BlackNode_k\right) = \bigotimes_k EVM\left(BlackNode_k\right)$$

and, since all eight vertices of a cube are Extreme Vertices, then all we have to do is list all eight vertices of every black node and collect every vertex that appears in an odd number of times in such a list, and discarding the remaining ones.

This provides a method for converting spatial-occupancy enumeration models to EVM. However, as stated before, a spatial-occupancy enumeration model should provide means for generating a list of all eight vertices for each black cell. The following gives some clues for some spatial-occupancy enumeration models (OUoDB and Octree):

7.2 Listing OUoDB vertices.

If Box_k := <left, bottom, rear, right, top, front> the all eight vertices are:

1)	(left, bottom, rear)	5)	(right, bottom, rear)
2)	(left, bottom, front)	6)	(right, bottom, front)
3)	(left, top, rear)	7)	(right, top, rear)
4)	(left, top, front)	8)	(right, top, front)

7.3 Listing black nodes' vertices for Octrees.

The following code lists all eight vertices for each of the Octree's black nodes. `Q` is a reference (pointer) to the structure; `width` is the length of the node under consideration whose minimum coordinates are: `x`, `y` and `z`.

```
Procedure ListOctreeVertices(Q: Octree; width, x, y, z: Integer);
Begin
   If NodeType(Q)=Black Then
      For k:=0 to 7 do
         Begin
            (x1, y1, z1) := OffsetVertex(x, y, z, k, width);
            Write(x1, y1, z1);
         End
   Else If NodeType(Q) = Gray Then
      For k:=0 to 7 do
         Begin
            (x1, y1, z1) := OffsetVertex(x, y, z, k, width/2);
            ListOctreeVertices(Son(Q, k), width/2, x1, y1, z1);
         End
End;
```

As an example, the following function does this vertex offsetting.

```
Function OffsetVertex(x, y, z, k, dist: Integer);
Begin
   (x1, y1, z1) := (x, y, z);
   If Odd(k) Then x1:=x+dist;
   k:=int(k/2);
   If Odd(k) Then y1:=y+dist;
   k:=int(k/2);
   If Odd(k) Then z1:=z+dist;
   Return(x1, y1, z1);
End;
```

7.4 B-Rep to EVM.

A boundary representation of an OPP must be able to produce a list of vertices for each face. If a vertex appears in an odd number of such lists then it belongs to an odd number of faces. So, by corollary 3.2, it is an Extreme Vertex. Thus, a B-Rep to EVM algorithm would be as simple as collecting every vertex that appears in an odd number of such lists, and discarding the remaining ones.

Warning: Be aware of points (a) and (b) explained at beginning of section 7.

7.5 EVM to B-Rep.

This section shows how to compute a Boundary-Representation (B-Rep) of an OPP from its EVM. As it will be shown, all the information needed for a valid B-Rep can be obtained from the EVM, and this will prove theorem 3.1 from section 3.1.

The algorithm returns a list of polygons which represent the border for faces and holes. Each polygon is given as a sequence of vertices. It produces counterclockwise boundaries for faces and clockwise boundaries for holes, as viewed from the OPP's outside. First the 2D case will be analyzed, then we will extend to the 3D case.

The algorithm for the 2D case makes use of a two-step approach: It first converts the 2D-EVM into an Ordered Union of Disjoint Boxes (OUoDB) scheme, as stated in section 6. Then the boundary of the represented regions is reconstructed by working on the OUoDB data structure.

We shall note that the 2D-OUoDB scheme is very close to a data structure known as Parallel Connected Strips, or PCS for short; see [Montani91]. In that work, an algorithm that converts PCS to 4-directions Freeman codes is given, and it is the basis of the 2D-EVM to B-Rep conversion algorithm.

In the following Pascal code, `left[k]`, `bottom[k]`, `right[k]` and `top[k]` are the limits for the k^{th} box of `Boxes`; where $k \in [1, n]$; while `bottomF[k]` and `topF[k]` flag that the corresponding border has already been reconstructed. Procedure `OUoDB_to_BREP` finds every box j with an untraced bottom or top border and calls `TraceOUoDB` (with the correct direction) which traces and constructs a complete face border or hole respectively. During this call, top or bottom edges of a box are marked when traversed, thus avoiding endless border reconstruction.

```
Procedure OUoDB_to_BREP(Boxes: OUoDBM);
Begin
  With Boxes Do
    For j := 1 To n Do
       Begin
        If   bottomF[j] = False Then TraceOUoDB(j, goRight)
        Else If topF[j] = False Then TraceOUoDB(j, goLeft);
       End;
End;
```

Procedure `TraceOUoDB` uses the bottom edge of a box when going to the right, or the top edge when going to the left. So it first sets `OriginVertex` as the bottom-left or top-right corner accordingly, then it reconstructs the border until this `OriginVertex` is reached again.

```
Procedure TraceOUoDB(j, dir);
Begin
  Case dir Of
    goRight: OriginVertex := SetVertex(left[j], bottom[j]); {vertex a}
    goLeft : OriginVertex := SetVertex(right[j], top[j]);   {vertex c}
  End;

  Repeat
    Case dir Of
      goRight: TraceVertexRIGHT(j, dir, NewVertex);
      goLeft : TraceVertexLEFT (j, dir, NewVertex);
    End;
  Until NewVertex = OriginVertex;
  Output('End of polygon.');
End;
```

Procedure TraceVertexRIGHT is called every time a bottom edge is to be traced from left to right (see Figure 7.1). Then, it either traces a vertical edge (cases 1 to 4), or none (case 5). Finally it determines whether the next horizontal direction will be to the left (cases 1 & 3), or to the right (cases 2, 4 & 5), if the next edge is whether the top or the bottom of a box, accordingly. In the first case the next call will be to procedure TraceVertexLEFT, while in the second case, it'll be again to procedure TraceVertexRIGHT. Note that if a non-manifold vertex is reached, then the algorithm does not jump to another box; that is, if top[j]=bottom[k] then case 1 is used instead of case 2; if bottom[j]=top[k] then case 1 or 2 is used instead of case 3 or 4; Finally, if top[j-1]=bottom[k] then case 4 is used instead of case 3.

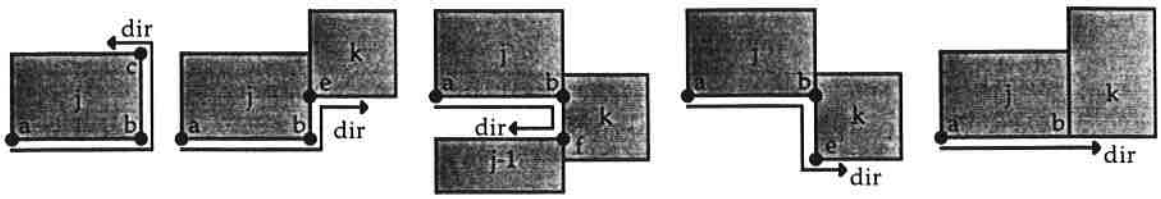


Figure 7.1. Cases 1 to 5 may occur after tracing bottom edge of box j, from left to right.

```
Procedure TraceVertexRIGHT(j, dir, NewVertex);
Begin
  Mark(bottomF[j]);
  NewVertex := SetVertex(right[j], bottom[j]);          {vertex b }

  k := j + 1;                                           {find box k}
  While (k<=n) & (right[j]=right[k]) Do incr(k);
  While (k<=n) & (right[j]=left[k]) & (bottom[j]>=top[k]) Do incr(k);

  If (k>n) | (right[j]<left[k]) | (top[j]<=bottom[k]) Then {case  1}
    Begin
      OutputVertex(NewVertex);                          {output b }
      NewVertex := SetVertex(right[j], top[j]);         {vertex c }
      OutputVertex(NewVertex);                          {output c }
      dir := goLeft;                                    {go left  }
    End
  Else If bottom[j] < bottom[k] Then                          {case  2}
    Begin
      OutputVertex(NewVertex);                          {output b }
      NewVertex := SetVertex(left[k], bottom[k]);       {vertex e }
      OutputVertex(NewVertex);                          {output e }
      j := k;                                           {use box k}
    End
  Else If bottom[j] > bottom[k] Then                    {case  3  or  4}
    Begin
      OutputVertex(NewVertex);                          {output b }
      If (j>1) & (right[j-1]=right[j]) & (top[j-1]>bottom[k]) Then
        Begin                                                     {case  3}
          NewVertex:=SetVertex(right[j-1], top[j-1]);   {vertex f }
          dir := goLeft;                                {go left  }
          j := j - 1                                    {box j-1  }
        End
      Else                                                        {case  4}
        Begin
          NewVertex := SetVertex(left[k], bottom[k]);   {vertex e }
          j := k;                                       {use box k}
        End;
      OutputVertex(NewVertex);                          {output f or e}
    End
  Else                                                        {case  5}
    j := k;                                             {use box k}
End;
```

Similarly, procedure `TraceVertexLEFT` is called every time a top edge is to be traced from right to left, then cases 6 to 10 may occur (see Figure 7.2). This procedure is identical to procedure `TraceVertexRIGHT` concerning its structure (only some reference names must be swapped, like: top and bottom, left and right, incr & decr, etc.) so it will not be included here.

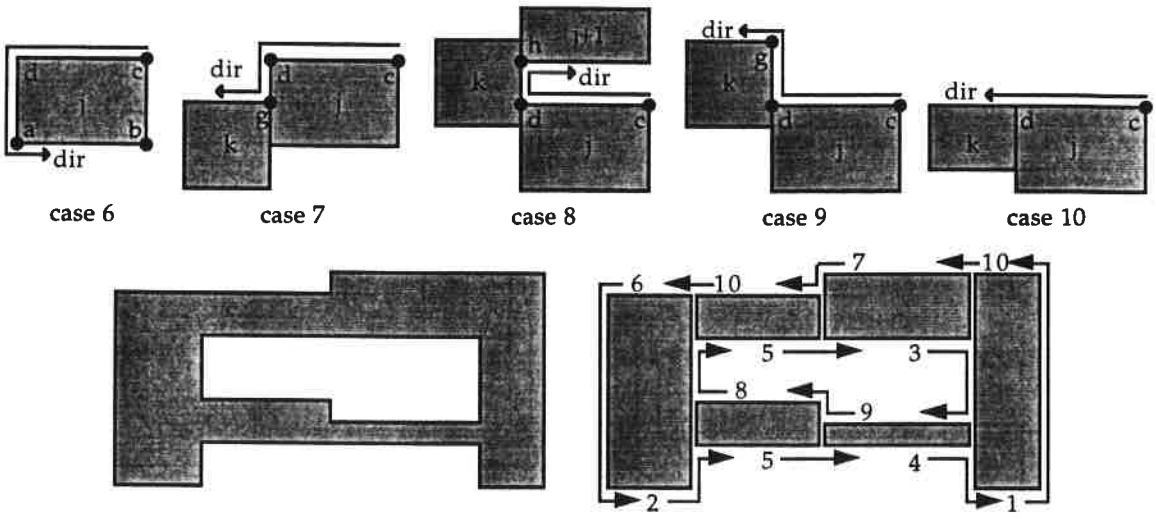


Figure 7.2. Top: Cases 6 to 10 may occur after tracing top edge of box j, from right to left. Bottom: Example of an OP (left) and its OUoDBM (right) showing the applied cases.

Although these two procedures are enough for the 2D case, they are not for the 3D case. (it needs twelve!). Nevertheless, the implemented algorithm does not work in this form; it actually has only one of such procedures, and differences are handled through parameters.

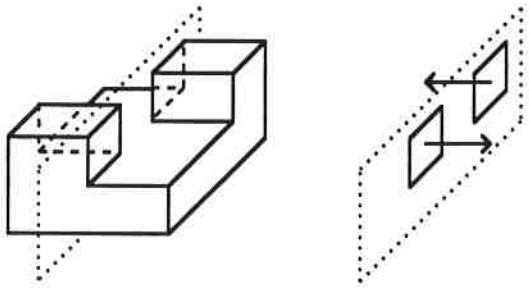


Figure 7.3: Example of an OP with two faces in the same plane of vertices, whose (parallel) normal vectors point to opposite directions.

Let us now analyze the 3D case. It has been shown that every face lies on a plane of vertices. Planes of vertices are easy-to-obtain 2D-OPP and thus, are suitable for applying the 2D-EVM to B-Rep conversion algorithm. However, faces with opposite normal vectors, lying on the same plane of vertices, will be wrongly traversed in the same direction. (see Figure 7.3). Therefore we addressed this problem in a different way, where the set of OPP's sections plays an important role in the conversion process.

It has been shown (see section 3.6) that any plane of vertices P_i of an OPP is computed by doing an exclusive OR between its previous S_{i-1} and next S_i sections:

$$P_i = S_{i-1} \otimes S_i, \quad \forall i \in [1, np]$$

Where P_i is an EVM representing the set of coplanar faces lying on the given plane of vertices. Moreover:

$$S_{i-1} \otimes S_i = (S_{i-1} - S_i) \cup (S_i - S_{i-1}).$$

We respectively call $(S_{i-1} - S_i)$ and $(S_i - S_{i-1})$ the forward and backward differences of consecutive sections S_{i-1} and S_i. Moreover those differences are quasi-disjoint to each other i.e., its regularized intersection is empty: $(S_{i-1} - S_i) \cap (S_i - S_{i-1}) = \emptyset$, so they define a partition for the set of faces lying on the given plane of vertices P_i.

Within this partition, forward differences $(S_{i-1} - S_i)$ are the sets of faces whose normal vectors point to the positive side of the main axis perpendicular to P_i, while backward differences $(S_i - S_{i-1})$ are the sets of faces whose normal vectors point to the negative side. Figure 7.4 shows a complete example.

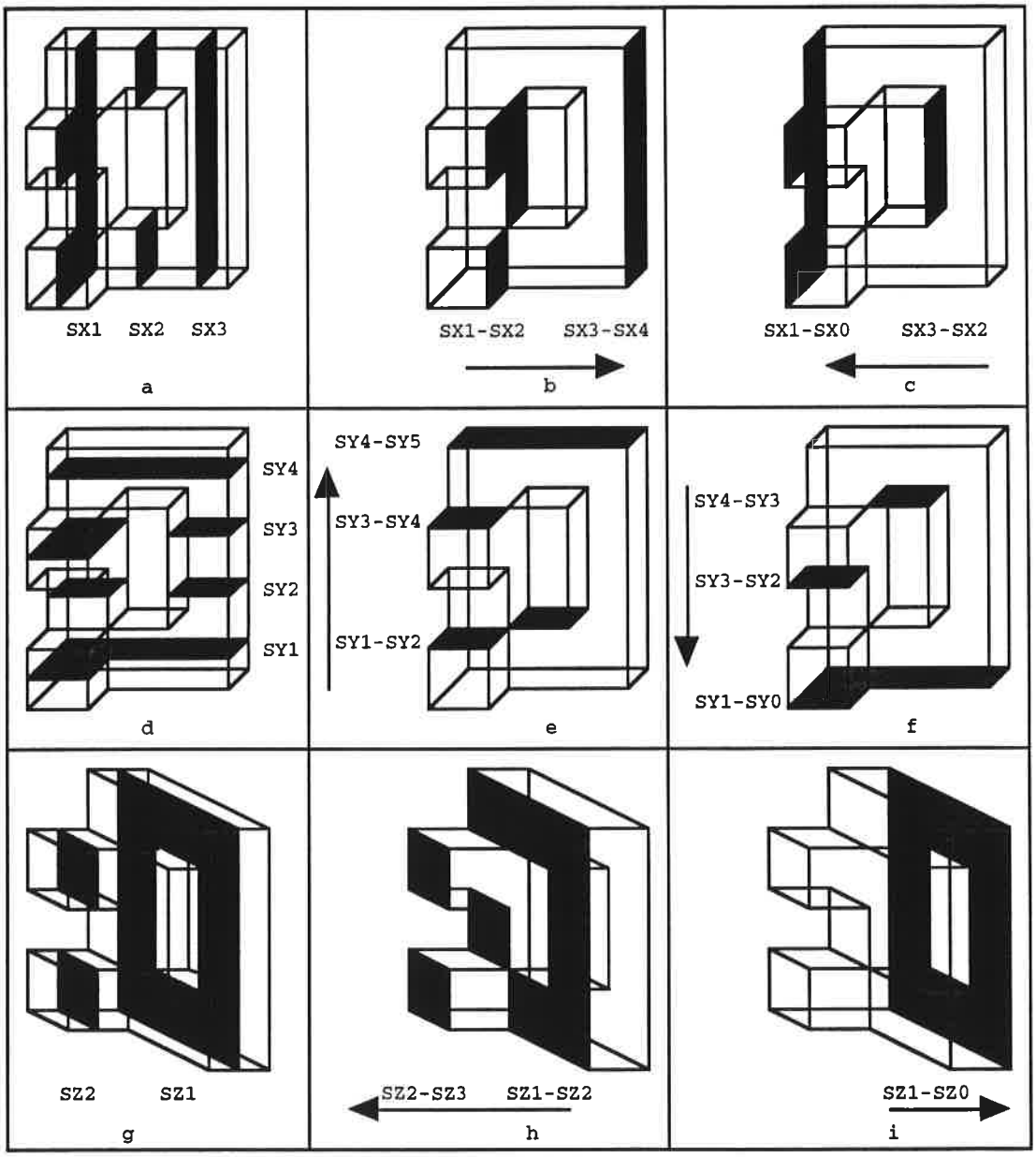


Figure 7.4: Computing faces from an EVM. (a) Sections perpendicular to X.
(b) Non-empty forward x-differences. (c) Non-empty backward x-differences.
(d) Sections perpendicular to Y. Non-empty forward (e) and backward (f) y-differences.
(g) Sections perpendicular to Z. Non-empty forward (h) and backward (i) z-differences.

Thus an algorithm that converts EVM to B-Rep is outlined as follows, where:

- `ABCsort(P, axis)` will sort `P` as a XYZ, YZX or ZXY-sorted EVM, according to `axis`.
- `OUoDB(P, axis)` will convert `P` as an Ordered Union of Disjoint Boxes, sorted first by axis `axis`. (see section 6.1).
- `EVM_to_SectionSequence` is described in section 3.5; `OpBool` in section 4; and `SetCoord` in section 3.3.

```
PROCEDURE EVM_to_BREP(P: ABCsorted)
  FOR axis in ['X', 'Y', 'Z'] do
    ABCsort(P, axis);
    EVM_to_SectionSequence(P, 2, S[]; coord[]);
    FOR k:=1 to Nsections DO
      ForwardDiff := OpBool(S[k-1], S[k],   3, '-');
      BckwardDiff := OpBool(S[k],   S[k-1], 3, '-');
      SetCoord(ForwardDiff, coord[k], 2);
      SetCoord(BckwardDiff, coord[k], 2);
      Boxes := OUoDB(ForwardDiff, axis);
      OUoDB_to_BREP(Boxes, PositiveSide, axis);
      Boxes := OUoDB(BckwardDiff, axis);
      OUoDB_to_BREP(Boxes, NegativeSide, axis);
    ENDFOR
  ENDFOR
ENDPROCEDURE
```

8 Perimeter, Surface and Volume of an OPP.

The perimeter of a 2D-OPP can be found by summing up the length of all its brinks. The following function does this. By using recursion, it can easily be extended to sum up all brinks of a 3D-OPP. This extension does not sum up non-manifold edges, since they are not included in brinks.

```
FUNCTION Perimeter(P: ABCsorted, dim: INTEGER)
{Sums all Brinks' length together of P}
  dim := dim-1; Perim := 0; LastSection := IniEVM();  { initialize }
  plv:=ReadPlv(P, dim);
  WHILE NOT End(P) DO                                  { process a plv }
    CurrCoord := GetCoord(plv, dim)
    CurrSection := GetSection(LastSection, plv, dim);

    ReadBrink(LastSection, Vb, Ve);              { This loop adds the   }
    WHILE NOT End(LastSection) DO                { last A-Strip's width }
      Perim := Perim + 2*(CurrCoord-LastCoord); { for each EV found in }
      ReadBrink(LastSection, Vb, Ve);            { last A-section.      }
    ENDWHILE
                              { (before current plv) }
    IF dim>1 THEN Perim := Perim + Perimeter(plv, dim)
    ELSE
      ReadBrink(plv, Vb, Ve);
      WHILE NOT End(plv) DO                  { This loop adds the length }
        Perim := Perim + Distance(Vb, Ve); { of each B-brink (2D-case) }
        ReadBrink(plv, Vb, Ve);            { or each C-brink (3D-case) }
      ENDWHILE
    ENDIF
    LastCoord := CurrCoord;
    LastSection := CurrSection;
    plv:=ReadPlv(P, 2);
  ENDWHILE
  RETURN(Perim);
ENDFUNCTION
```

The surface of a 2D-OPP can easily be found if it is expressed as an Union of Disjoint Boxes, then the algorithm only needs to add the surface of all these boxes together, else the surface of these boxes can be added as they are computed within the converting process. The surface of a 3D-OPP's boundary can be computed using the following expression:

$$\sum_{k=1}^{N_s}\left[Area(S_k - S_{k-1}) + dist(plv_{k+1} - plv_k)\cdot Perim(S_k) + Area(S_k - S_{k+1})\right]$$

where

- N_S is the number of A-Sections (A-Strips) in the given OPP.
- $S_k - S_{k-1}$ and $S_k - S_{k+1}$ are the backward and forward differences (see fig. 7.4) of consecutive A-Sections S_{k-1}, S_k and S_{k+1}. That is, they are the set of A-faces bounding A-$Strip_k$ whose normal vectors point, respectively, to the negative and positive side of the main axis A.
- $Area(S_k - S_{k-1})$ and $Area(S_k - S_{k+1})$ are those A-faces' surface.
- $Perim(S_k)$ is the perimeter of section S_k
- $dist(plv_{k+1} - plv_k)$ is the width of A-$Strip_k$.

Finally, the volume of an OPP can easily be computed if it is expressed as an Union of Disjoint Boxes, then the algorithm only needs to add all these boxes' volumes together.

Warning: These three algorithms may require floating-point arithmetic, which has been avoided in this work until now.

9 Conclusions and Future Work.

This work presents new schemes for representing orthogonal polyhedra: the Extreme Vertex Model, the Sections Sequence of an OPP, and the Ordered Union of Disjoint Boxes. They include simple and robust algorithms for performing the more demanding tasks.

Set Membership Classification processes were developed, which include classifying OPP against OPP (boolean operations); OPP against orthogonal plane (splitting); orthogonal plane against OPP (sections); line against OPP and point against OPP (point and line in polyhedra). The following table shows all these possibilities:

<table>
<tr><th>Vs.</th><th>1D-OP</th><th>2D-OPP</th><th>3D-OPP</th><th></th></tr>
<tr><td>point</td><td>Point in Segment
(IN ON OUT)</td><td>Point in Polygon
(IN ON OUT)</td><td>Point in Polyhedra
(IN ON OUT)</td><td><-- 0D-results</td></tr>
<tr><td>general line</td><td rowspan="2"></td><td>Segment classifi-
cation (IN OUT)</td><td rowspan="2">Segment classifi-
cation (IN OUT)</td><td rowspan="2"></td></tr>
<tr><td>orthogonal line</td><td>1D-Sections</td></tr>
<tr><td>1D-OP</td><td>1D-Boolean
Operations</td><td></td><td></td><td><-- 1D-results</td></tr>
<tr><td>general plane</td><td rowspan="2"></td><td rowspan="2"></td><td>2D-General
Sections</td><td rowspan="2"></td></tr>
<tr><td>orthogonal plane</td><td>2D-Orthogonal
Sections</td><td><-- 2D-results</td></tr>
<tr><td>2D-OPP</td><td>2D-Splitting
(2D-OPP vs. line)</td><td>2D-Boolean
Operations</td><td></td><td></td></tr>
<tr><td>3D-OPP</td><td></td><td>3D-Splitting
(3D-OPP vs. plane)</td><td>3D-Boolean
Operations</td><td><-- 3D-results</td></tr>
</table>

Classifying non orthogonal elements with OPP are a common task in many applications, like rendering. The line vs. OPP classification is included here. We are working in classifying general planes against OPP. However, we are not interested in classifying an OPP against a general plane since non-orthogonal polyhedra may result.

Boolean operations are also well suited for the EVM. We proposed in an earlier work, OP as geometric bounds in CSG (see [Aguilera96]). That work can be neatly enhanced by using the point and line in polyhedra algorithms here described. Other potential applications suitable to use EVM are: OPP as geometric bounds in B-Rep, Collision detection and Features recognition.

Because the Ordered Union of Disjoint Boxes Model only stores two vertices per box, it may be even more compact than its corresponding EVM. Future work will analyze and compare these storing sizes.

This work does not include conversions from the EVM to spatial-occupancy enumeration models, because general OPP does not always decompose into identical cells arranged in a fixed, regular grid. However, future work will present algorithms that compute OPP approximations on those models from an EVM.

References.

[Aguilera96]:
A. Aguilera and D. Ayala. *Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry*. Technical Report LSI-96-64-R. LSI-Universitat Politècnica de Catalunya, 1996.

[Aguilera97]:
A. Aguilera and D. Ayala. *The Extreme Vertices Model for Orthogonal Polyhedra*. Technical Report LSI-97-6-R. LSI-Universitat Politècnica de Catalunya, 1996.

[Aguilera98] :
A. Aguilera. *Polyhedrical Orthogonal Approximation: Study and Application*. Ph. D. Thesis. Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, (1998, in preparation).

[Andújar96]:
C. Andújar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Solé. Automatic generation of multiresolution boundary representations. *Eurographics'96*, Vol. 15, 1996.

[Ayala95]:
D. Ayala, C. Andújar, and P. Brunet. Automatic simplification of orthogonal polyhedra. In D. Fellner, editor, *Modeling, Virtual Worlds, Distributed Graphics: Proceedings of the International MVD'96 Workshop*. Infix, 1995.

[Edelsbrunner87]:
H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Velarg, 1987.

[Foley90]:
J. D. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*, Second Edition. Addison-Wesley Publishing Company, 1990.

[Hoffmann89]:
C. M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kauffmann Publishers, Inc., 1989.

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

[Juan95]:
Domain extension of isothetic polyhedra with minimal CSG representation. *Computer Graphics Forum*, (5): 281-293, 1995.

[Lorensen87]:
W. E. Lorensen and H. Cline. Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21 (4): 44-50, 1987

[Mäntyla88]:
M. Mäntyla. *An Introduction to Solid Modeling*. Computer Scientific Press, 1988.

[Montani91] :
C. Montani and R. Scopigno. Quadtree/Octree to boundary conversion. In *Graphics Gems II*, chapter IV.7, pages 202-218. Academic Press, Inc. 1991.

[Preparata85]:
F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Velarg, 1985.

[Requicha80]:
A. G. Requicha. Representations for rigid solids: theory, methods and systems. *ACM Computing Surveys*, 12 (4): 437-464.

[Rossignac91]:
J. R. Rossignac and A. G. Requicha. Constructive Non-Regularized Geometry. *CAD*, 23(1): 21-32, 1991.

[Samet89]:
H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley Publ., Reading, MA, 1989.

[Srihari81]:
S. N. Srihari. Representation of three-dimensional digital images. *ACM Computing Surveys*, 13 (1): 399-424. 1981

[Tang91a]:
K. Tang and T. Woo. Algorithmic aspects of alternating sum of volumes. Part 1: Data structure and difference operation. *CAD*, 23(5): 357-366, 1991.

[Tang91b]:
K. Tang and T. Woo. Algorithmic aspects of alternating sum of volumes. Part 2: Non-convergence and its remedy. *CAD*, 23(6): 435-443, 1991.

[vanGelder94]:
A. van Gelder and J. Wilhelms. Topological Considerations in Isosurface Generation. *ACM Transactions on Graphics*, 13(4): 337-375, 1994.

[Weiler86]:
K. Weiler. *Topological Structures for Geometric Modeling*. Ph. D. Dissertation. Rensselaer Polytechnic Institute, USA. 1986.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1997

LSI–97–1–R "On the Number of Descendants and Ascendants in Random Search Trees", Conrado Martínez and Helmut Prodinger.

LSI–97–2–R "On the Epipolar Geometry and Stereo Vision", Blanca García de Diego.

LSI–97–3–R "Solving Incidence and Tangency Constraints in 2D", Núria Mata.

LSI–97–4–R "*Designer:* A Tool to Design and Model Workflows", Camilo Ocampo, Pere Botella.

LSI–97–5–R "OBJECTFLOW: A Modular Workflow Management System", Camilo Ocampo, Pere Botella.

LSI–97–6–R "The Extreme Vertices Model (EVM) for Orthogonal Polyhedra", A. Aguilera and D. Ayala.

LSI–97–7–R "An Improved Master Theorem for Divide-and-Conquer Recurrences", Salvador Roura.

Hardcopies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es

See also the Department WWW pages, `http://www-lsi.upc.es/www/`