Università degli Studi di Roma Tre

Dipartimento di Informatica e Automazione

Via della Vasca Navale, 79 – 00146 Roma, Italy

Dimension-Independent BSP (2): Boundary to Interior Mapping

Claudio Baldazzi *Alberto Paoluzzi*

RT-DIA-29-97 **Dicembre 1997**

Università di Roma Tre,
Via della Vasca Navale, 79
00146 Roma, Italy.

ABSTRACT

In this paper we discuss a CSG/BSP algorithm to perform the conversion from the boundary to the interior of d-dimensional polyhedra. Both a d-dimensional polyhedral point-set and its boundary $(d-1)$-faces are here represented as BSP trees. In this approach no structure, no ordering and even no orientation is required for such boundary BSP trees. In particular it is shown that the interior point-set may be implicitly represented as the Boolean XOR of unbounded polyhedral "stripes" of dimension d, which are bijectively associated to the $(d-1)$-faces of the d-polyhedron. A set of quasi-disjoint convex cells which partitionate the polyhedron interior may be computed by explicitly evaluating such CSG tree with XOR operations on the non-leave nodes and with BSP (stripe) trees on the leave nodes.

1 Introduction

Problem statement The computational problem addressed in this paper can be stated as follows: "Given a BSP representation of the $(d-1)$-dimensional boundary of a regular polyhedron of dimension d, generate both an implicit (CSG/BSP) and an explicit (decompositive) representation of some partition of its interior with convex cells".

The implicit representation is here given as a CSG tree with all XOR operations on the non-leaf nodes and with so-called "stripe-BSP" trees on the leaf nodes. The explicit representation as a standard BSP tree will be simply obtained by just traversing such implicit representation and by executing the requested Boolean operations.

For a remind of basic concepts about BSP trees and a complete discussion of the notation used here, see the companion paper [2], where the converse transformation between the standard BSP decomposition of the interior of a regular d-polyhedron and the collection of BSP trees associated to its $(d-1)$-dimensional boundary faces is discussed.

Approach It is well known, from the Jordan theorem in the plane, that the number of intersection points between a semi-infinite line and the boundary of a 2D polygon is either odd or even, depending on the fact that the ray is shot from either the interior or the exterior region. This result is actually dimension-independent, and states that the number of intersections between an affine halfspace of dimension 1 starting at a point and a closed hypersurface is either odd or even depending on the position of the point with respect to the interior of the hypersurface.

The algorithms given in this paper are actually motivated by the property, similar in some sense to the Jordan theorem, that the XOR of a number n of instances of a set A is either the empty set or the set A itself, depending on the parity (even or odd, respectively) of n. So, we show that a BSP tree of the interior of a regular d-polyhedron may be computed by evaluating a multiple XOR expression having as arguments the unbounded "BSP-stripes" associated to the $(d-1)$-faces of the polyhedron.

Due to commutativity and associativity properties of the XOR operation, no preliminary ordering of the $(d-1)$-faces is needed to compute the result. This approach is hence well suited for parallel implementations, since the computing load can be easily distributed between different processors.

Previous work In several applications of computer graphics (e.g. in GIS and architectural CAD) it is often necessary to transform a closed and possibly unconnected polyline into the plane polygon it is boundary of. This transformation is often accomplished by

computing a (possibly constrained) Delaunay triangulation [3, 7, 6].

A boundary to CSG local XOR formula for generating a Boolean expression of a *simple*, i.e. non self-intersecting, 2D polygon was given by Guibas *et al.* in [5]. Such approach only works with simple polygons because it is necessary to decide if any vertex is either convex or concave. Dobkin *et al.* in [4] give monotone (i.e. with no complementation) CGS formulæ for *simple* and *manifold* polygons in $O(n \log n)$ time, were n is the number of edges. They also prove, by giving counterexamples, that such formulæ do not always exist for 3D polyhedra.

The transformation discussed in this paper can also be considered a multidimensional boundary to CSG mapping. In [11, 12] Shapiro and Vossler discusses several algorithms to perform both ways such a transformation in 3D, either in a linear or quadratic algebraic environment. An algorithm for boundary to BSP conversion of polyhedra starting from a standard boundary representation in 3D, hence by using adjacency information, was given by Thibault and Naylor in [13]. Boolean operations over BSP trees strongly resemble to CSG trees, so that the Naylor's approach to Booleans [8] can certainly be considered a mixed CSG/BSP approach.

Results The method here proposed seems interesting for various reasons. First, conversely than the known approaches, it can be uniformly applied to polyhedra of whatever intrinsic dimension. Also it allows for independent processing of $(d-1)$-faces, and hence it is easily parallelizable. More importantly, no ordering and even no orientation of the boundary faces is required in order to generate the BSP of the interior. Furthermore, it does not require any representation of the topology of the input object. In 2D, our (monotone and linear time) method may be applied to non simple and non manifold polygons. Also, it has provable correctness. Finally, if the incidence relations between faces of dimension k and $k-1$ are known, $1 \leq k \leq d$, then the algorithm can be executed iteratively, so allowing for reconstructing a d-polyhedron from its $(d-k)$-faces.

Paper preview In Section 2 some basic concepts are given for the computing machinery described in the paper, together with a short outline of the presented approach. In Section 3 new concepts and operations needed to implement the algorithms presented in this paper are introduced. In particular, we formally define the concepts of *BSP extrusion*, *section*, *projection* and the concept of *stripe* generated by an embedded BSP tree. In Section 4 it is discussed how to generate a *XOR tree* representation of the input polyhedron and how to compute the explicit BSP decomposition of the interior by traversing the XOR tree. In Section 5 two complete examples are presented, and the computational performance is

discussed with reference to empirical results, together with some implementation details. In the Conclusion section some possible extensions of the presented approach are outlined.

2 Preliminary definitions and background

The notation used in this paper is introduced in [2], according to the original definitions given by Naylor [9] and by Naylor *et al.* [8]. Some preliminary definitions from [2] are quickly recalled here for sake of readability. In the present paper we extend such definitions, relative to *regular BSP* trees, to consider also *embedded BSP* trees.

In this paper we often need to distinguish between affine subspaces of dimension $n-1$ (hyperplanes or covectors) and the corresponding normal vectors. At this purpose we denote as h_f the affine hull of the f face and with h^f its normal vector. Conversely, we always use subscripts for coordinate representations of both vectors and covectors.

2.1 Regular BSP trees

A *regular* BSP (Binary Space Partition) tree defined on a set of hyperplanes in Euclidean d-dimensional space E^d establishes a hierarchical partitioning of such space. Each node ν of such a binary tree is associated to a convex and possibly unbounded region of E^d denoted by R_ν. The two sons of an internal node ν are denoted as $below(\nu)$ and $above(\nu)$, respectively. Leaves correspond to unpartitioned regions, which are either OUT (*empty*) or IN (*full*) *cells*. Each internal node ν of the tree is associated with a *partitioning hyperplane* h_ν, which intersects the interior of the region R_ν. The hyperplane h_ν subdivides R_ν into three subsets:

1. the subregion $R^0_\nu = R_\nu \cap h_\nu$. This point-set has affine support of dimension $d-1$;
2. the subregion $R^-_\nu = R_\nu \cap h^-_\nu$, where h^-_ν is the negative halfspace of h_ν. The halfspace h^-_ν is associated with the edge $(\nu, below(\nu))$ of the BSP tree. The region R^-_ν is associated with the *below* subtree of ν, i.e. $R^-_\nu = R_{below(\nu)}$;
3. the subregion $R^+_\nu = R_\nu \cap h^+_\nu$, where h^+_ν is the positive halfspace of h_ν. The halfspace h^+_ν is associated with the tree edge $(\nu, above(\nu))$. The region R^+_ν is associated with the *above* subtree, i.e. $R^+_\nu = R_{above(\nu)}$.

We suppose that dimension-independent *regularized* [10] set operations of *union* ($|$), *intersection* ($\&$), *difference* ($-$) and *symmetric difference* ($\wedge$), also called XOR, are available for regular BSP trees, according to the algorithm of Naylor *et al.* [8], as implemented by Baldazzi [1] with Linear Programming techniques.

Node region Given a node ν in a regular BSP tree, the region R_ν is defined as the intersection of the closed halfspaces on the path from the root to ν. More formally, the region described by any node ν is: $R_\nu = \cap_e h_e^\pm$, $e \in E(\nu)$, where $E(\nu)$ is the edge set on the path from the root to ν and $h_e^\pm$ is the halfspace associated to the edge e.

Halfspace BSP Let denote with H^- and H^+ the elementary BSP trees with root h and leaves IN, OUT and OUT, IN, respectively. Clearly, the two trees H^- and H^+ are a BSP representation of the two halfspaces delimited by the root hyperplane h.

2.2 Embedded BSP trees

An embedded BSP tree of dimension (d, n) is a hierarchical partitioning of a d-dimensional affine space (embedded in the Euclidean space E^n) with $(d-1)$-dimensional affine subspaces.

A regular BSP tree describes a set of either full or empty *solid* cells. Conversely, an embedded BSP tree can be considered as a set of (linear) curves, surfaces, and so on. Notice that any embedded BSP tree is regular, i.e. solid, in the relative topology of its affine hull.

We define here *representation space* $\mathcal{BSP}^{d,n}$ the set of all the embedded BSP trees which partitionate some affine subspace of dimension d in Euclidean space E^n. The *BSP representation scheme* used in this paper is a mapping

$$\mathrm{BSP} : \mathcal{P}^{d,n} \to \mathcal{BSP}^{d,n},$$

where $\mathcal{P}^{d,n}$ is the set of homogeneously dimensional d-polyhedra in E^n.

Notice that the set of regular BSP trees of dimension d coincides with the subset $\mathcal{BSP}^{d,d}$ of embedded BSP trees.

Section BSP and Face BSP There is an useful relationship between regular BSP trees of dimension d and the set of pairs made by an affine transformation and a $(d-1)$-dimensional BSP tree embedded in E^d. In particular, the so-called *section-trees* and *face-trees* introduced in the companion paper [2] can be described by giving a pair $(M_f, \mathrm{BSP}(f))$, where f is a regular polyhedron in $\mathcal{P}^{d-1,d-1}$ and

$$M_f : E^d \to E^d : h_d \mapsto h_f$$

is an invertible affine transformation which maps the coordinate subspace h_d, with equation $x_d = 0$, into the affine hull h_f of the face f.

Boundary BSP Accordingly, a *Boundary BSP* associated to a solid and regular d-polyhedron P in E^d can be defined as a set of pairs

$$\{(M_f, \mathrm{BSP}(f)) \mid f \in F(P)\},$$

where $F(P)$ is the set of $(d-1)$-faces of P.

3 Operations

In this section we introduce some concepts and operations needed to define and/or to implement the algorithms presented in this paper. In particular, we formally define the concepts of BSP *extrusion*, *section*, *projection* and *stripe* generated by an embedded BSP tree.

Let be given an origin and an orthonormal basis $\{e^i\}$ in E^d, and denote with e_i the coordinate hyperplane normal to e^i.

Definition 1 (BSP Extrusion) *Let A be a regular tree in $\mathcal{BSP}^{d,d}$. We call* BSP extrusion *a mapping η between regular trees defined as follows:*

$$\eta : \mathcal{BSP}^{d,d} \to \mathcal{BSP}^{d+1,d+1} : A \mapsto A \times E,$$

where E denotes the one-dimensional Euclidean space. So, the notation $\eta(A)$ will denote the BSP tree associated to the partition of E^{d+1} defined as the set of cells

$$\{b_i \mid b_i = a_i \times E\}$$

where a_i is any d-cell in the partitioning of E^d induced by A.

Notice that the computer representation of $\eta(A)$ is very similar to the representation of A, since the implicit inequalities which define the partitioning of the embedding space do not change. E.g., notice that the equation of a straight line in E^2, let $ax + by + c = 0$, also represents the vertical plane of E^3 which projects on the line. Only the information field which records the dimension of the embedding space needs to be updated, so that the extrusion operation is performed in constant time $O(1)$.

Definition 2 (BSP Section) *Let A be a tree in $\mathcal{BSP}^{d,d}$, and consider an hyperplane $h_0 + h_1 x_1 + \ldots + h_d x_d = 0$ in E^d, where $h = (h_0, h_1, \ldots, h_d) \in \Re^{d+1}$. We call* BSP section *a mapping*

$$\sigma : \Re^{d+1} \times \mathcal{BSP}^{d,d} \to \mathcal{BSP}^{d-1,d}$$

where the image $\sigma_h(A) \equiv (T, B)$ of the tree A is computed according to the algorithm discussed in [2], and where $T \in Lin^{d+1}$, $B \in \mathcal{BSP}^{d-1,d-1}$ and $T(e_d) = h$.

Definition 3 (BSP Projection) *Consider a tree A in $\mathcal{BSP}^{d-1,d}$, with affine support h. We call* BSP Projection *a non degenerate mapping*

$$\pi_d : \mathcal{BSP}^{d-1,d} \to \mathcal{BSP}^{d-1,d-1}$$

such that $(\sigma_h \circ \eta \circ \pi_d)(A) = A$.

The projection tree $\pi_d(A)$ is computed as follows. For each hyperplane h_ν in A, h_ν is substituted by the element of the hyperplane bundle

$$h + \lambda h_\nu = 0, \qquad \lambda \in \Re,$$

whose normal is perpendicular to the unit vector e^d. In other words, given a tree $A \in \mathcal{BSP}^{d-1,d}$ with support hyperplane h, each hyperplane h_ν in A is transformed into the member of the bundle generated by h and h_ν, which is perpendicular to e^d. This operation will result in a BSP tree projected in the coordinate subspace e_d, with equation $x_d = 0$. The output tree $\pi_d(A)$ will hence belong to $\mathcal{BSP}^{d-1,d-1}$.

Definition 4 (Stripe BSP) *Consider a tree A in $\mathcal{BSP}^{d-1,d}$, with affine support h. We call* Stripe BSP *the tree $\Sigma(A) \in \mathcal{BSP}^{d,d}$ defined as the intersection of the BSP halfspace H^+ (defined in Section 2), with the BSP tree obtained by extruding the projection of A onto the coordinate subspace h^d:*

$$\Sigma : \mathcal{BSP}^{d-1,d} \to \mathcal{BSP}^{d,d} : A \mapsto H^+ \ \& \ (\eta \circ \pi_d)(A)$$

The orientation of the affine support of the input tree must be carefully checked in this case. Let be given the tree $A \in \mathcal{BSP}^{d-1,d}$, embedded in a hyperplane h_f of E^d. As a preliminary test, check for the "external" orientation of h by computing the dot product $h^f \cdot e^d$, where h^f and e^d are the normal vector to h_f and the last element in the Euclidean base $\{e^i\}$, respectively. If the dot product is negative, then h_f must be multiplied by -1.

A degenerate case occurs when $h^f \cdot e^d = 0$, i.e. when the affine support of A is perpendicular to the coordinate subspace e_d. In such a case the resulting Stripe BSP is simply the EMPTY tree, i.e. the tree with only one node labeled as OUT. Such degenerate case will be excluded by any further consideration.

Example 1

In Figure 1a the BSP of a 3D solid in general position in E^3 is shown. This example solid is used to show the generation of a *Stripe BSP* associated to its bottom face. In particular, Figure 1b shows both the bottom face f and the projection $\pi_3(BSP(f))$. In Figure 1c the stripe BSP generated by f is displayed.

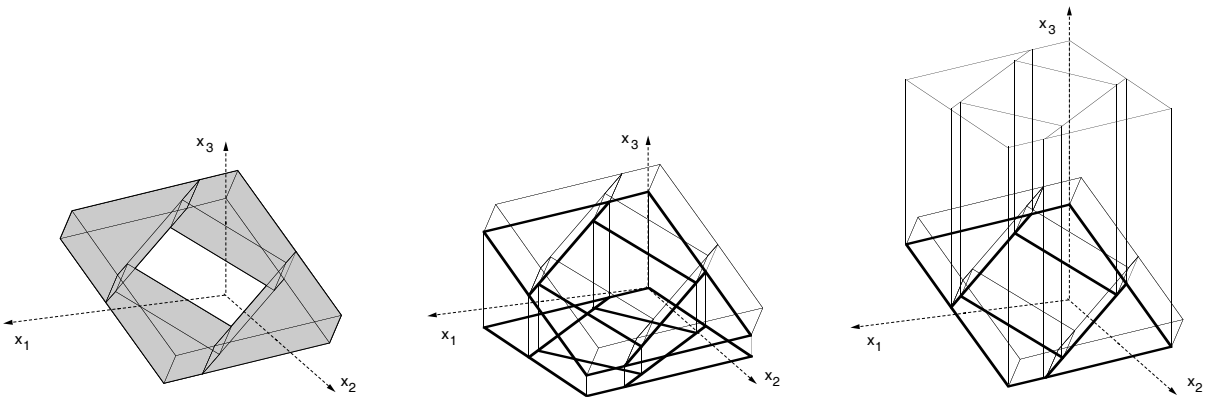

Figure 1: (a) The BSP decomposition of a 3D solid in general position. (b) A face BSP and its π_3 projection. (c) The stripe BSP associated to the bottom face.

4 Algorithms

The goal of the algorithms discussed in this section is to transform a *boundary BSP*, i.e. an unordered collection of *face BSP* trees and embedding maps, which describes the boundary of a d-polyhedron P in E^d, into a regular tree $\mathrm{BSP}(P) \in \mathcal{BSP}^{d,d}$ which describes a cell decomposition of P.

Geometric approach The idea underlying the algorithm discussed in this paper can be easily understood if considered with reference to its 2D instance. Given a closed and possibly unconnected polyline p in E^2, a CSG tree is generated, where each leave node is an unbounded stripe region S_i of E^2 and each non-leave node is a regularized XOR operation. In particular, the leave nodes S_i are bijectively associated to the polyline edges e_i. Each region S_i is defined as the intersection of three affine subspaces, and can be seen as an unbounded vertical stripe. As a consequence of the Jordan theorem the polygon P, which equates the topological closure of the interior of p, can be represented as the Boolean XOR of all such stripes.

If the leaves of any CSG tree associated to such multiple XOR expression are represented as BSP trees, then the evaluation of the CSG produces a BSP tree which partitionates both the polygon P and the embedding space E^2 in convex cells. An explicit representation of the cells as subsets of linear inequalities can be produced by traversing the BSP tree and collecting, for each IN (i.e. full) cell of it, the set of linear inequalities associated to the current path on the tree.

Algorithm preview Two main steps can be identified in the computation. First, the CSG with all XOR operations in the non-leaf nodes and Stripe BSP trees in the leaf nodes must be constructed, in particular by computing the Stripe BSP associated to the boundary faces. Then such CSG tree must be evaluated, by properly traversing the tree and executing the Boolean operations.

1. ***Stripe-BSP computation.*** For each face f of P, a regular tree $\mathrm{BSP}(S_f)$ is properly built starting from the embedded tree $(M_f, \mathrm{BSP}(f))$. S_f is the solid unbounded stripe generated by the following steps.

   (a) *Embedding* the face tree $\mathrm{BSP}(f) \in \mathcal{BSP}^{d-1,d-1}$ in E^d space. The result of this operation is a BSP tree $F \in \mathcal{BSP}^{d-1,d}$. An example of this operation is given in Figure 2b.

   (b) *Projecting* the tree F into a coordinate subspace of dimension $d-1$. The projection is performed into the coordinate subspace e_d, normal to the last reference axis. This operation is implemented by evaluating, for each $h_i \in F$, the hyperplane bundle formed by the affine subspace h_f which contains the face and by the hyperplane h_i which bounds the face. In each bundle of hyperplanes the element $\hat{h}_i$ orthogonal to e_d must be chosen. See Figure 2c.

   (c) *Extruding* the projected tree generated at step (b). See Figure 2d.

   (d) *Intersecting* the extruded projection with one of halfspace BSP (H_f^+ or H_f^-) associated to the affine support h_f of the f face. The choice is done according to the sign of the dot product between the h^f vector and e^d. The result is shown in Figure 2e.

   The corresponding STRIPE algorithm is detailed in Figure 3. If h_f is orthogonal to the coordinate subspace e_d, then the resulting Stripe BSP is simply the EMPTY tree, with only one node labeled as OUT.

2. ***XOR-tree evaluation.*** The Boolean XOR of all such stripes is evaluated, so giving the desired BSP representation of P

$$\mathrm{BSP}(P) = \bigwedge_{f \in F(P)} \mathrm{BSP}(S_f), \tag{1}$$

   where $F(P)$ is the set of boundary faces of P and $\mathrm{BSP}(S_f)$ is the BSP representation of the solid unbounded stripe associated to the face f, as returned by the *STRIPE* algorithm.

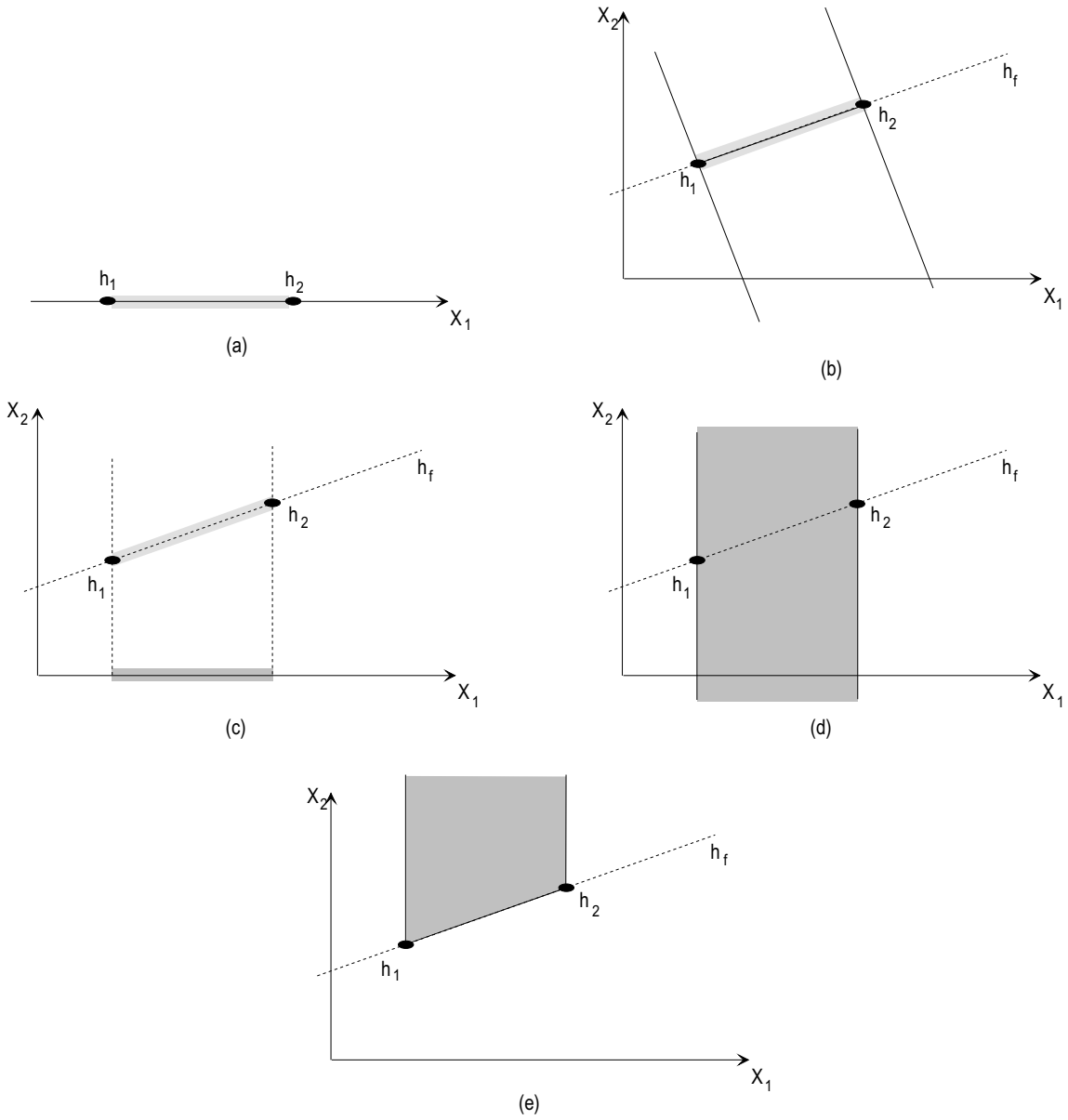

Figure 2: The stripe algorithm. (a) Input data: $(M_f, \mathrm{BSP}(f))$, where f is a regular polyhedron in $\mathcal{P}^{d-1,d-1}$ and $M_f : E^d \to E^d : e_d \mapsto h_f$ is the embedding mapping. (b) Embedded tree. (c) Projected tree. (d) Extruded projection. (e) Unbounded stripe generated by intersection with an halfspace BSP.

```
STRIPE( (M_f, BSP(f)): BSP face ): BSP tree;
/* M_f embedding matrix, BSP(f) ∈ BSP^{d-1,d-1} */

    T: BSP tree;

    if ( h^f · e^d == 0 )
        T = EMPTY;
    else {
        /* Embedding */
        Multiply all hyperplanes of f for the matrix M_f;
        T = BSP(f);          /* T ∈ BSP^{d,d} */

        /* Projection followed by Extrusion */
        T = (η ∘ π )(T);

        /* Intersection with halfspace BSP */
        if ( h^f · e^d > 0 )
            T = H_f^+  &  T;
        else
            T = H_f^-  &  T;

    }
    return T;
end
```

Figure 3: The *STRIPE* algorithm.

The choice of $\hat{h}_i$ orthogonal to e_d must provide a coherent orientation for the above and below subspaces. In other words the points of h_f in the below subspace of h_i must remain in the below subspace of $\hat{h}_i$, and analogously for points in the above subspaces.

This search of $\hat{h}_i$ from a bundle can be visualized as a rotation of h_i into the new hyperplane $\hat{h}_i$. The coherent orientation is guaranteed if the rotation is performed maintaining greater than 0 the dot product between the normal h^i and the normal $\hat{h}^i$. In 2D and 3D this results in a rotation with angle $\alpha < \pi/2$.

Notice that $h^i \cdot \hat{h}^i = 0$ implies that $h_i \perp \hat{h}_i$. But it is also: (a) $\hat{h}_i \perp e_d$ by construction and (b) $h_i \perp h_f$ by definition, after the embedding. Hence, for the transitivity of the orthogonality relationship, it is also $h_f \perp e_d$. So, the resulting S_f stripe collapses to the EMPTY tree and will not be taken into any subsequent account.

Left and balanced traversal Since the XOR operation is associative and commutative, the Formula (1) can be computed in several ways, using different binary trees to codify the multiple XOR expression. E.g., the expression:

$$B = A_1 \wedge A_2 \wedge A_3 \wedge A_4,$$

where $A_1, \ldots, A_4$ are BSP trees, can be evaluated by using either a left-associative rule

$$B = ((A_1 \wedge A_2) \wedge A_3) \wedge A_4, \tag{2}$$

or a balanced set of parentheses:

$$B = (A_1 \wedge A_2) \wedge (A_3 \wedge A_4). \tag{3}$$

At least two different evaluation algorithms may therefore be introduced, which correspond to the linear left evaluation (2) and to the balanced (divide et impera) approach of equation (3). The right associative strategy is analogous to the left associative one, and will not be considered. The two corresponding LXOR and QXOR algorithms are shown in Figures 4 and 5, respectively. In the following section their computational performance is discussed with reference to empirical results.

Linear accumulation algorithm (LXOR) The LXOR algorithm requires as input the *faces* array of face-BSP trees in $\mathcal{BSP}^{d-1,d}$ and their number n. LXOR returns as output a BSP tree in $\mathcal{BSP}^{d,d}$. The approach is extremely easy. The face-BSP trees in the *faces* array are iteratively transformed into a Stripe-BSP. At the same time the XOR of the current stripe against an accumulator variable of BSP type is performed. The algorithm returns the final value of such accumulator.

```
LXOR( faces: array of BSP faces; n: number of faces ): BSP tree;
    T: BSP tree;

    T = STRIPE( faces[0] );
    for (i = 1 to n-1 )
        T = XOR( T, STRIPE( faces[i] ) );

    return T;
end
```

Figure 4: The *LXOR* algorithm.

Divide et impera algorithm (QXOR) A divide et impera algorithm may be devised by recursively evaluating the XOR of two BSP trees in $\mathcal{BSP}^{d,d}$ which result from the evaluation of the first and second half of a multiple XOR expression, respectively. Conversely that in the LXOR case, most part of the computation is performed by merging with a XOR operation two generic BSP trees, and not by merging one such a tree and a BSP-stripe. The implementation is given in two steps. The main procedure *QXOR* starts the computation by calling the recursive procedure *QXOPproc* with the actual value of parameters.

5 Examples

In this section some simple 2D and 3D examples are presented, in order to ease the understanding of the discussed methods. First we show how to represent a 2D stripe associated to a line segment in $\mathcal{P}^{1,2}$, then we show the behavior of the *LXOR* algorithm with input in $\mathcal{P}^{1,2}$ and $\mathcal{P}^{2,3}$, respectively.

Example 2 (2D Stripe)
First notice that an horizontal stripe $\Sigma(BSP(a_i))$, with $a_i \in \mathcal{P}^{1,2}$, is defined as the intersection of three halfspaces, and is represented as a very simple BSP tree where each internal node is an halfspace and only one leaf is coded as a IN cell (see Figure 6). Notice that $\det M \leq 0$ represents the internal halfspace supporting the edge (v, w), where

$$M = \begin{bmatrix} x & y & 1 \\ x_v & y_v & 1 \\ x_w & y_w & 1 \end{bmatrix}$$

```
QXOR( faces: array of BSP faces; n: number of faces ): BSP tree;
    return QXORPROC( faces, 0, n-1 );
end

QXORPROC( faces: array of BSP faces; i, j: faces index ): BSP tree;
    T1, T2: BSP tree;
    if (j-i == 0)
        return STRIPE( faces[i] );
    else if (j-i == 1)
        T1 = STRIPE( faces[i] );
        T2 = STRIPE( faces[j] );
        return XOR( T1, T2 );
    else
        T1 = QXORPROC( faces, i, (i+j)/2 );
        T2 = QXORPROC( faces, 1+(i+j)/2, j );
        return XOR( T1, T2 );
end
```

Figure 5: The *QXOR* algorithm.

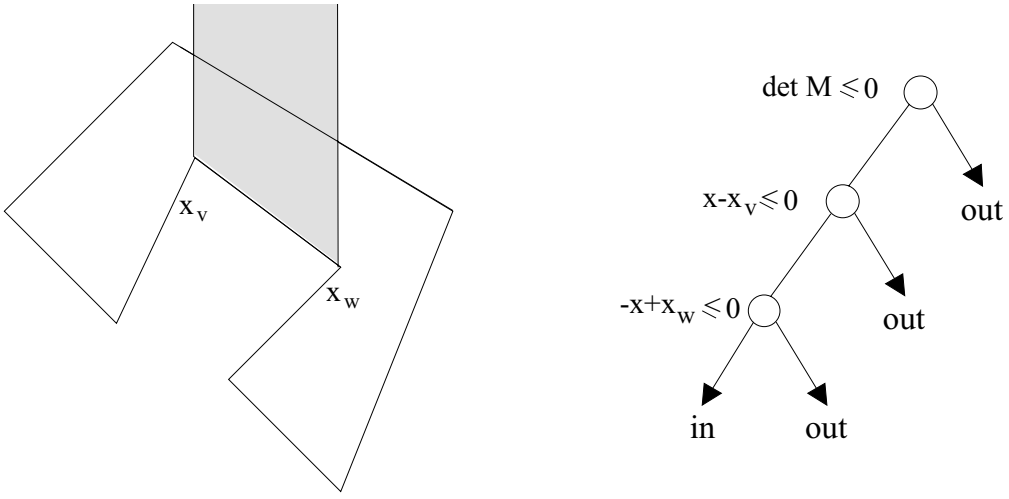

Figure 6: (a) Each edge of the polyline generated a unbounded stripe. (b) Each stripe is represented by a simple unbalanced BSP tree.

gives the equation of the line for the points v and w.

Example 3 (2D Polyline $\rightarrow$ 2D Polygon)
In this example we trace all the steps of the boundary-to-interior transformation applied to a polygon boundary $P \in \mathcal{P}^{1,2}$, in such a way that $LXOR(P) = Q \in \mathcal{P}^{2,2}$, with $\partial Q = P$. An unconvex, unconnected and with internal "lakes" and "islands" example polyline is therefore illustrated in Figure 7, in order to look at the behavior of the $LXOR$ algorithm.

Example 4 (3D Boundary $\rightarrow$ 3D Interior)
As we have seen in the previous sections, a regular BSP of a d-polyhedron can be obtained from the boundary BSP, i.e. from the unordered collection of its face-BSP pairs. In this example we show the reconstruction of the interior of a polyhedron $P \in \mathcal{P}^{3,3}$, starting from $\mathrm{BSP}(f_i)$, with $f_i \in F(P)$. In Figure 8 the set of non-degenerate polyhedral stripes $\mathrm{BSP}^{-1}(\Sigma(\mathrm{BSP}(f_i)))$ is shown. Notice that the BSP representation of all the vertical faces of P, i.e. those parallel to e^3, degenerate (i.e. are not regular) and are not taken into account. In Figure 9 the polyhedral result of the computation of the iterative step

$$T_i = T_{i-1} \wedge \Sigma(\mathrm{BSP}(f_i))$$

is shown, where $\Sigma(\mathrm{BSP}(f_i))$ is the stripe associated to the face f_i and where T_i is the accumulator variable, according to the LXOR algorithm. Notice that it is assumed $T_0 =$ EMPTY.

Experimental performance The algorithms proposed in this paper has been implemented and tested in a PC environment with Pentium 166 processor and Visual C++ under Windows NT. The test experiments aimed to reconstruct solid representations for a class of 3D objects of very complex topology starting from a BSP representation of their boundary faces. In order to compare the results of the mapping from boundary to interior with those of the converse algorithm [2], we choose to reconstruct the regular BSP for the same 3D grid of orthogonal rods used in Reference [2]. A picture of this object for rod number $n = 9$ is given in Figure 10.

The test objects were defined by computing the Boolean union of a number of rods variable from 4^3 to 9^3. Two different sets of starting configurations were tested. The first set of test objects (denoted as $4c$–$9c$ in Table 1) were given parallel to the reference frame. The results, i.e. both the computation time and the cell number, of the second set of test (denoted as $4r$–$9r$ in Table 1) are conversely obtained as the average of 10 tests with three variable rotation angles around the coordinate axis.

The columns of Table 1, which refer to algorithm LXOR, give in the order:

1. The number of rods for side (variable from 4 to 9) and the type of configuration: c = centered; r = rotated.

2. The number of input faces, i.e. the number of embedded face BSP.

3. The number of both IN and OUT cells in the regular input tree, i.e. before of the extraction of the face BSP trees.

4. The time in seconds needed to generate the Boundary BSP on the same computing environment, by using the converse algorithm [2].

5. The number of both IN and OUT cells in the regular output tree, i.e. after the boundary to interior conversion.

6. The reconstruction average time, given in seconds.

7. The number of BSP nodes processed in the unit time by our implementation of the discussed algorithm.

It is possible to see that whereas the LXOR algorithm performance is very good in the centered case, so that to produce even some optimization of the input BSP tree, the computation time and the cell number is much higher in the rotated configuration. Anyway, it may be interesting to note that the reconstruction time is about three times than the generation of the Boundary BSP.

A similar set of experiments has been executed to test the performance of Algorithm QXOR. The results are displayed in Table 2. It is interesting to note that LXOR seems to perform a little better than QXOR, but this may depend on the nature of the processed objects.

6 Conclusion

This paper has discussed a transformation between a boundary and a decompositive BSP representation of dimension-independent polyhedra. The presented algorithms can be used to generate a cell-decomposition of a d-polyhedron starting from the BSP trees of its $(d-1)$-dimensional boundary faces. Such an approach may be useful to give insight into the problem of computing non-regularized Booleans over multidimensional polyhedra, which is our ultimate goal. Since the presented approach does not depend from neither the orientation nor the ordering of boundary faces, it easily allows for parallel implementations. An open issue remains the optimization of the cell number of the output

Table 1: Test experiments for LXOR algorithm.

Grid	Face num.	Input tree	Gen. faces time	Output tree	Reconst. time	Nodes/RTime
4c	24	457(112,117)	13	449(104,121)	1	449
5c	30	909(225,230)	42	887(210,234)	4	221.7
6c	36	1593(396,401)	110	1553(372,405)	8	194.1
7c	42	2557(637,642)	252	2495(602,646)	22	113.4
8c	48	3849(960,965)	556	3761(912,969)	48	78.3
9c	54	5517(1377,1382)	1121	5399(1314,1386)	97	55.6
4r	24	457(112,117)	13	3957(850,1129)	32	123.6
5r	30	909(225,230)	42	9231(2015,2601)	126	73.2
6r	36	1593(396,401)	110	18449(4089,5136)	372	49.6
7r	42	2557(637,642)	252	33187(7463,9131)	984	33.7
8r	48	3849(960,965)	556	55113(12540,15017)	2249	24.5
9r	54	5517(1377,1382)	1121	85839(19684,23236)	4583	18.7

Table 2: Test experiments for QXOR algorithm.

Grid	Face num.	Input tree	Gen. faces time	Output tree	Reconst. time	Nodes/RTime
4c	24	457(112,117)	13	481(120,121)	1	481
5c	30	909(225,230)	42	1001(235,266)	3	333.6
6c	36	1593(396,401)	110	1797(444,455)	8	224.6
7c	42	2557(637,642)	252	2835(700,718)	15	189
8c	48	3849(960,965)	556	4145(1104,969)	25	165.8
9c	54	5517(1377,1382)	1121	5979(1476,1514)	60	99.6
4r	24	457(112,117)	13	5181(945,1646)	32	161.9
5r	30	909(225,230)	42	12511(2339,3917)	132	94.8
6r	36	1593(396,401)	110	24645(4675,7648)	368	66.9
7r	42	2557(637,642)	252	42929(8272,13193)	929	46.2
8r	48	3849(960,965)	556	70313(13700,21457)	2128	33
9r	54	5517(1377,1382)	1121	110817(22034,33375)	4906	22.5

tree. It is the authors' opinion that a strong optimization for this kind of BSP trees is possible at relatively low cost.

References

[1] Baldazzi, C. *Boolean Set Operations on Multidimensional Polyhedra.* Graduation Thesis, Università "La Sapienza", Rome, Italy, February 1996. (In Italian).

[2] Baldazzi, C., and Paoluzzi, A. *Dimension-Independent BSP (1). Section and Interior-to-Boundary Mapping.* To appear on *International Journal of Shape Modeling*, 1997.

[3] De Floriani, L., and Puppo, E. Hierarchical triangulation for multiresolution surface description. *Computer Graphics*, 14(4), 363-411, Oct. 1995. Proc. of ACM Siggraph'95.

[4] Dobkin, D., Guibas, L., Hershberger, J., and Snoeyink, J. An efficient algorithm for finding the CSG representation of a simple polygon. *Algoritmica*, (10), 1993, 1–23.

[5] Guibas, L., Ramshaw, L. and Stolfi, J. A kinetic framework for computational geometry. *Proc. of the 24th IEEE Symposium on Foundations of Computer Science*, 1983, IEEE, 100–111.

[6] Guibas, L., Stolfi, J. Primitives for the manipulation of general subdivision and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2), 74-123, 1985

[7] Lee, D. T., Schachter, B. Two algorithms for constructing Delaunay triangulations. *International Journal of Computers and Information Science*, 9(3), 219-242, 1980

[8] Naylor, B. F., Amanatides, J., and Thibault, W. Merging BSP trees yields polyhedral set operations. *Computer Graphics*, 24(4), August 1990, 115–124. (Proc. of ACM Siggraph'90).

[9] Naylor, B. F. Binary space partitioning trees as an alternative representation of polytopes. *Computer Aided Design*, 22(4), April 1990, 250–252.

[10] Requicha, A. A. G. Representations for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4), December 1980, 437–464.

[11] Shapiro, V., Vossler, D. L. Construction and optimization of CSG representation. *Computer Aided Design*, 23(1), 4-20, Jan./Feb. 1991.

[12] Shapiro, V., Vossler, D. L. Separation for Boundary to CSG Conversion. *Computer Graphics*, 12(1), January 1993, 35–55. (Proc. of ACM Siggraph'93).

[13] Thibault, W., and Naylor, B. F. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(4), July 1987, 153–162. (Proc. of ACM Siggraph'87).

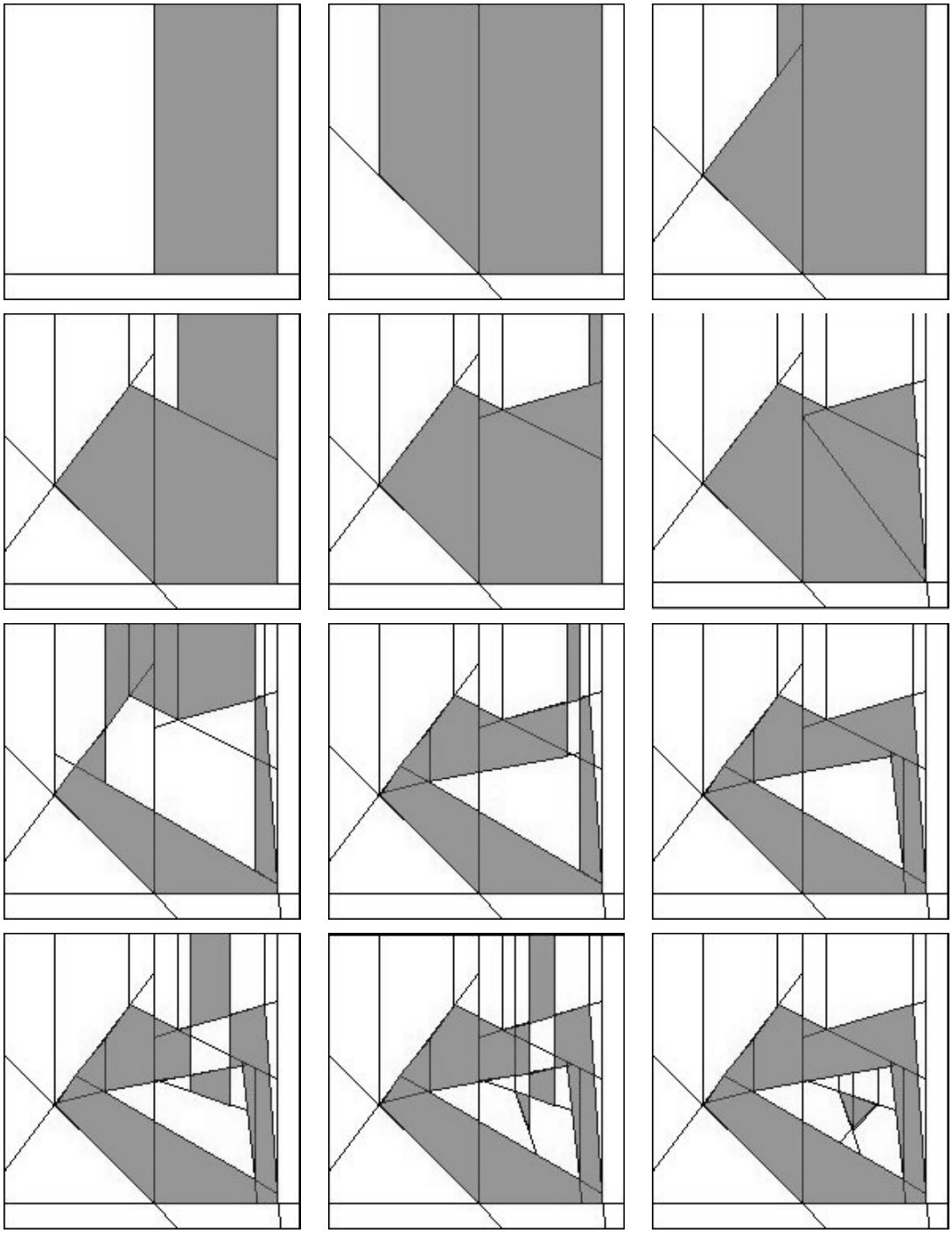

Figure 7: Incremental construction of a 2D polygon by using the LXOR algorithm.

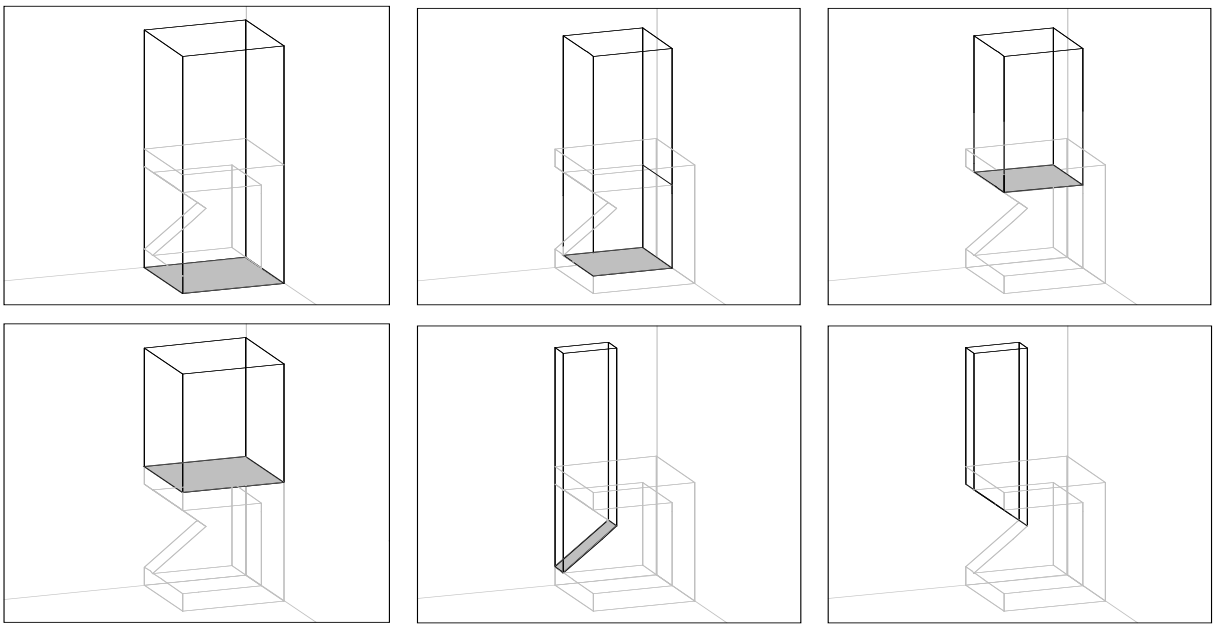

Figure 8: Non degenerate BSP-stripes associated to the 2-faces of the solid.

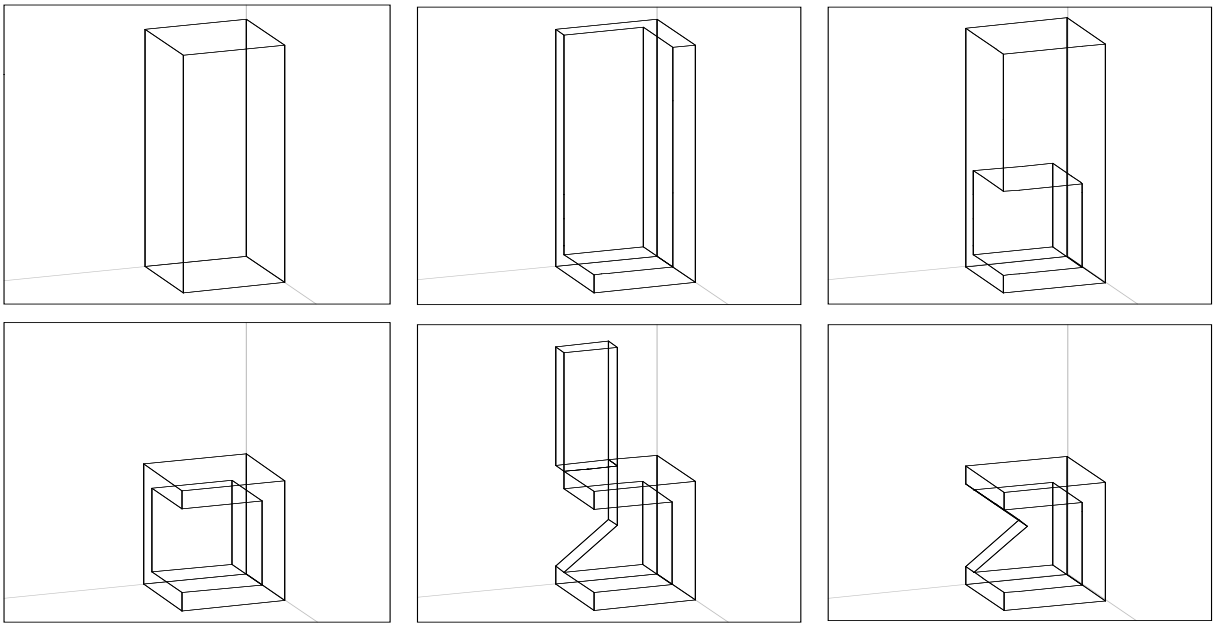

Figure 9: Results of a progressive XOR with the face stripes.

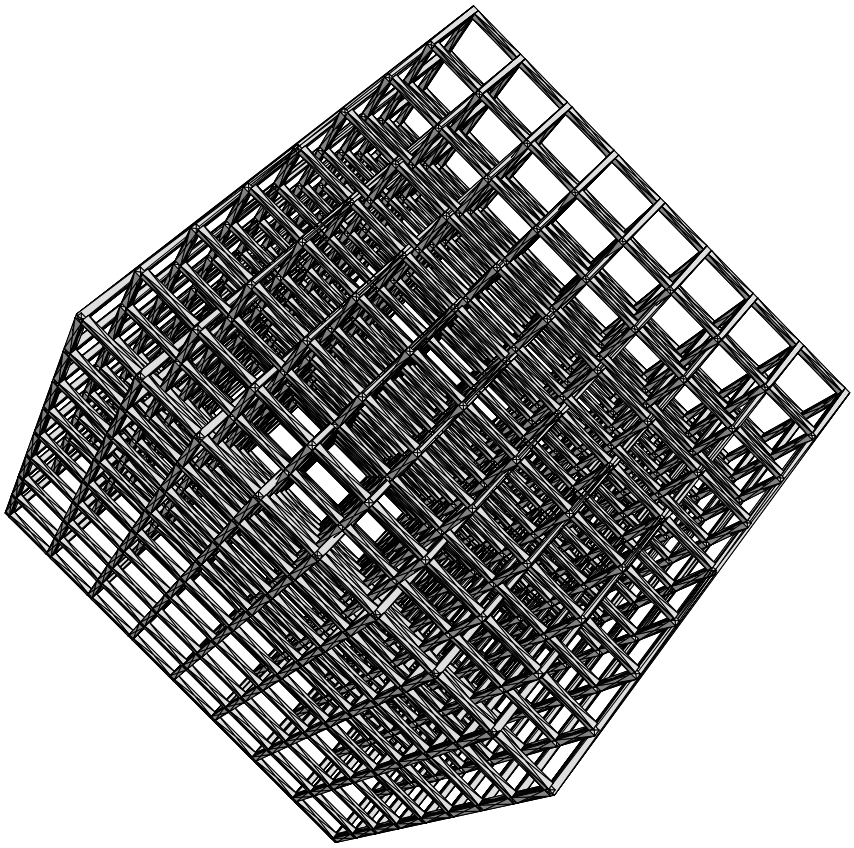

Figure 10: The polyhedron used for the performance evaluation. In this figure is shown a grid with $9 \times 9 \times 9$ rods.