

Research Article

Efficient Boundary Extraction from Orthogonal Pseudo-Polytopes: An Approach Based on the nD -EVM

Ricardo Pérez-Aguila

Computer Engineering Institute, The Technological University of the Mixteca (UTM), Carretera Huajuapán-Acatlilma Km. 2.5, Huajuapán de León, 69000 Oaxaca, Mexico

Correspondence should be addressed to Ricardo Pérez-Aguila, ricardo.perez.aguila@gmail.com

Received 9 December 2010; Accepted 9 March 2011

Academic Editor: Tak-Wah Lam

Copyright © 2011 Ricardo Pérez-Aguila. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This work is devoted to contribute with two algorithms for performing, in an efficient way, connected components labeling and boundary extraction from orthogonal pseudo-polytopes. The proposals are specified in terms of the extreme vertices model in the n -dimensional space (nD -EVM). An overview of the model is presented, considering aspects such as its fundamentals and basic algorithms. The temporal efficiency of the two proposed algorithms is sustained in empirical way and by taking into account both lower dimensional cases (2D and 3D) and higher-dimensional cases (4D and 5D).

1. Introduction

The extreme vertices model in the n -dimensional space (nD -EVM) is a representation scheme whose dominion is given by the n -dimensional orthogonal pseudo-polytopes, that is, polytopes that can be seen as a finite union of iso-oriented nD hyperboxes (e.g., hypercubes). The fundamental notions of the model, for representing 2D and 3D-OPPs, were originally established by Aguilera and Ayala in [1, 2]. Then, in [3], it was formally proved the model is capable of representing and manipulating nD -OPPs. The nD -EVM has a well-documented repertory of algorithms (see [1, 3–7]) for performing tasks such as Regularized Boolean operations, set membership classification, measure interrogations (content and boundary content), morphological operations (e.g., erosion and dilation), geometrical and topological interrogations (e.g., discrete compactness), among other algorithms currently being in development. Furthermore, conciseness, in terms of spatial complexity, is one the nD -EVM's main properties, because the model only requires storing a subset of a polytope's

vertices, the extreme vertices, which finally provides efficient temporal complexity of our algorithms. Sections 2 and 3 are summaries of results originally presented in [1, 3]. For the sake of brevity, and because we aim for a self-contained paper, some propositions are only enunciated. Their corresponding proofs and more specific details can be found in [1, 3]. In Section 2, the main concepts and basic algorithms behind the nD -EVM are described, while Section 3 presents an algorithm for extracting forward and backward differences of an nD -OPP.

Section 4 presents the first main contribution of this work: An EVM-based algorithm for computing connected components labeling over an nD -OPP. It is well known connected components labeling is one of the most important tools in image processing and computer vision. The main idea is to assign labels to the pixels in an image such that all pixels that belong to a same connected component have the same label [8]. As pointed out by [9], it is essential for the specification of efficient methods that identify and label connected components in order to process them separately for the appropriate analysis and/or operations. Several approaches have been proposed in order to achieve such labeling, and some of them are applicable in multidimensional contexts (e.g., see [10, 11]), but there is a step that is commonly present in the majority of the methodologies: the scanning step [12]. Such a step determines a label for the current processed pixel via the examination of those neighbor pixels with an assigned label. Suzuki et al. group the methods for connected components labeling in four categories [13].

- (i) Methods that require multiple passes over the data: the scanning step is repeated successively until each pixel has a definitive label.
- (ii) Two-pass methods: they scan the image once to assign temporal labels. A second pass has the objective of assigning the final labels. They make use of lookup lists or tables in order to model relationships between temporal and/or equivalent labels [9].
- (iii) Methods that use equivalent representations of the data: their strategy is based on using a more suitable representation based on hierarchical tree structures (quadrees, octrees, etc.) instead of the image's raw data in order to speed up the labeling [10, 14].
- (iv) Parallel algorithms.

In [7], Rodríguez and Ayala describe an EVM-based algorithm for connected components labeling. It works specifically for 2D and 3D-OPPs. First, they obtain a particular partitioning from the 3D-EVM known as ordered union of disjoint boxes (OUODB), which is, in fact, a special kind of cell decomposition [7]. Then, once the OUODB partition has been achieved, it follows a process which is based in the two-pass approach. We take into account some ideas presented by Rodríguez and Ayala, but our proposal deals with a higher-dimensional context, that is, with nD -OPPs, and it works directly with their corresponding nD -EVM representations.

Section 5 describes the second contribution of this work: a methodology for extracting kD boundary elements from an nD -OPP represented through the nD -EVM. It is well known that a boundary model for a 3D solid object is a description of the faces, edges, and vertices that compose its boundary together with the information about the connectivity between those elements [15]. However, the boundary representations can be recursively applied not only to solids, surfaces, or segments but to n -dimensional polytopes [16]. For Putnam and Subrahmanyam, a polytope's boundary representation can be seen as a *boundary tree* [17].

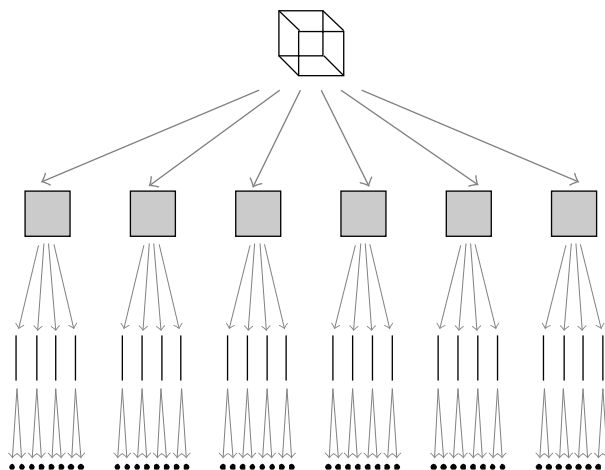


Figure 1: The boundary tree associated to a 3D cube.

In the tree, each node is split into a component for each element that it bounds. An element (vertex, edge, etc.) will be represented several times inside the tree, one for each boundary that it belongs to. See Figure 1 for a cube's boundary tree.

The way we extract kD boundary elements from an nD -OPP represented through the nD -EVM will be reminiscent to the reconstruction of the boundary tree associated with such nD -OPP. Moreover, our methodology is strongly sustained in the use of our proposed connected components labeling algorithm (Section 4) and the procedures described in Sections 2 and 3. Some study cases will be presented in order to appreciate the information our proposed procedure can share. Finally, in Section 6, we determine, in empirical way, the time complexity of the algorithm presented in Section 5 in order to provide evidence of its efficiency specifically when higher-dimensional nD -OPPs are considered.

2. The Extreme Vertices Model in the n -Dimensional Space (nD -EVM)

In Section 2.1, some conventions and preliminary background related directly with orthogonal polytopes will be introduced. In Section 2.2, the foundations of the nD -EVM will be established. Section 2.3 presents some basic algorithms under the EVM.

2.1. The n -Dimensional Orthogonal Pseudo-Polytopes (nD -OPPs)

The following definitions ((a) to (i)) belong to Spivak [18].

- (a) A *singular n -dimensional hyperbox* in \mathbb{R}^n is given by the continuous function $I^n : [0, 1]^n \rightarrow [0, 1]^n$ such that $I^n(x) = x$.
- (b) For all $i, 1 \leq i \leq n$, if $x \in [0, 1]^{n-1}$, then the two singular $(n-1)D$ hyperboxes $I_{(i,0)}^n$ and $I_{(i,1)}^n$ are given by

$$\begin{aligned} I_{(i,0)}^n(x) &= I^n(x_1, \dots, x_{i-1}, 0, x_i, \dots, x_{n-1}) = (x_1, \dots, x_{i-1}, 0, x_i, \dots, x_{n-1}), \\ I_{(i,1)}^n(x) &= I^n(x_1, \dots, x_{i-1}, 1, x_i, \dots, x_{n-1}) = (x_1, \dots, x_{i-1}, 1, x_i, \dots, x_{n-1}). \end{aligned} \tag{2.1}$$

- (c) A *general singular k -dimensional hyperbox* in the closed set $A \subset \mathbb{R}^n$ is defined as the continuous function $c : [0, 1]^k \rightarrow A$.
- (d) Given a general singular n D hyperbox c the composition $c_{(i,\alpha)} = c \circ I_{(i,\alpha)}^n$ defines an $(i,)\alpha$ -cell.
- (e) The *orientation* of an $(n-1)$ D cell $c \circ I_{(i,\alpha)}^n$ is given by $(-1)^{\alpha+i}$, while an $(n-1)$ D *oriented cell* is given by the scalar-function product $(-1)^{i+\alpha} \cdot c \circ I_{(i,\alpha)}^n$.
- (f) A formal linear combination of singular general k D hyperboxes, $1 \leq k \leq n$, for a closed set A is called a k -chain.
- (g) Given a singular n D hyperbox I^n , the $(n-1)$ -chain, called the *boundary of I^n* , is defined as

$$\partial(I^n) = \sum_{i=1}^n \left(\sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot I_{(i,\alpha)}^n \right). \quad (2.2)$$

- (h) For a singular general n D hyperbox c , we define the $(n-1)$ -chain, called the *boundary of c* , by

$$\partial(c) = \sum_{i=1}^n \left(\sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot c \circ I_{(i,\alpha)}^n \right). \quad (2.3)$$

- (i) The *boundary of an n -chain $\sum c_i$* , where each c_i is a singular general n D hyperbox, is given by

$$\partial\left(\sum c_i\right) = \sum \partial(c_i). \quad (2.4)$$

Based on the above Spivak's definitions, we have the elements to establish our own precise notion of orthogonal polytope. A collection $c_1, c_2, \dots, c_k, 1 \leq k \leq 2^n$, of general singular n D hyperboxes is a *combination of n D hyperboxes* if and only if

$$\left[\bigcap_{\alpha=1}^k c_\alpha([0, 1]^n) = \left(\underbrace{0, \dots, 0}_n \right) \right] \wedge [(\forall i, j, i \neq j, 1 \leq i, j \leq k) (c_i([0, 1]^n) \neq c_j([0, 1]^n))]. \quad (2.5)$$

The first part of the conjunction establishes that the intersection between all the n D general singular hyperboxes is the origin, while the second part establishes that there are not overlapping n D hyperboxes. Finally, we will say an n -dimensional orthogonal pseudopolytope p , or just an n D-OPP p , is an n -chain composed by n D hyperboxes arranged in such way that by selecting a vertex, in any of these hyperboxes, it describes a combination of n D hyperboxes composed up to 2^n hyperboxes.

2.2. nD -EVM's Fundamentals

Let c be a combination of hyperboxes in the n -dimensional space. An *odd adjacency edge* of c , or just an *odd edge*, will be an edge with an odd number of incident hyperboxes of c . Conversely, if an edge has an even number of incident hyperboxes of c , it will be called *even adjacency edge* or just an *even edge*. A *brink* or *extended edge* is the maximal uninterrupted segment, built out of a sequence of collinear and contiguous odd edges of an nD -OPP. By definition, every odd edge belongs to brinks, whereas every brink consists of m edges, $m \geq 1$, and contains $m + 1$ vertices. Two of these vertices are at either extreme of the brink and the remaining $m - 1$ are interior vertices. The ending vertices of all the brinks in p will be called *extreme vertices of an nD -OPP p* and they are denoted as $EV(p)$. Finally, we have that any extreme vertex of an nD -OPP, $n \geq 1$, when is locally described by a set of surrounding nD hyperboxes, has exactly n incident linearly independent odd edges.

Let p be an nD -OPP. A kD *extended hypervolume* of p , $1 < k < n$, denoted by $\phi(p)$, is the maximal set of kD cells of p that lies in a kD space, such that a kD cell e_0 belongs to a kD extended hypervolume if and only if e_0 belongs to an $(n - 1)D$ cell present in $\partial(p)$, that is,

$$(e_0 \in \phi(p)) \iff (\exists c, c \text{ belongs to } \partial(p)) (e_0([0, 1]^k) \subseteq c([0, 1]^{n-1})). \quad (2.6)$$

Given an nD -OPP, we say the *extreme vertices model* of p , denoted by $EVM_n(p)$, is defined as the model as only stores to all the extreme vertices of p . We have that $EVM_n(p) = EV(p)$ except by the fact that coordinates of points in $EV(p)$ are not necessarily sorted. In general, it is always assumed that coordinates of extreme vertices in the extreme vertices model of an nD -OPP p , $EVM_n(p)$, have a fixed coordinates ordering. Moreover, when an operation requires manipulating two EVMs, it is assumed both sets have the same coordinates ordering.

The *projection operator* for $(n - 1)D$ cells, points, and sets of points is, respectively specified as follows.

- (i) Let $c(I_{(i,\alpha)}^n(x)) = (x_1, \dots, x_n)$ be an $(n - 1)D$ cell embedded in the nD space. $\pi_j(c(I_{(i,\alpha)}^n(x)))$ will denote the projection of the cell $c(I_{(i,\alpha)}^n(x))$ onto an $(n - 1)D$ space embedded in nD space whose supporting hyperplane is perpendicular to X_j -axis; that is, $\pi_j(c(I_{(i,\alpha)}^n(x))) = (x_1, \dots, \hat{x}_j, \dots, x_n)$.
- (ii) Let $v = (x_1, \dots, x_n)$ a point in \mathbb{R}^n . The projection of that point in the $(n - 1)D$ space, denoted by $\pi_j(v)$, is given by $\pi_j(v) = (x_1, \dots, \hat{x}_j, \dots, x_n)$.
- (iii) Let Q be a set of points in \mathbb{R}^n . The projection of the points in Q , denoted by $\pi_j(Q)$, is defined as the set of points in \mathbb{R}^{n-1} such that $\pi_j(Q) = \{p \in \mathbb{R}^{n-1} : p = \pi_j(x), x \in Q \subset \mathbb{R}^n\}$.

In all the three above cases, \hat{x}_j is the coordinate corresponding to X_j -axis to be suppressed.

For an nD -OPP p , we will say that np_i denotes the number of distinct coordinates present in the vertices of p along X_i -axis, $1 \leq i \leq n$. Let $\Phi_k^i(p)$ be the k th $(n - 1)D$ extended hypervolume, or just a $(n - 1)D$ *Couplet*, of p which is perpendicular to X_i -axis, $1 \leq k \leq np_i$.

A *slice* is the region contained in an nD -OPP p between two consecutive couplets of p . $\text{Slice}_k^i(p)$ will denote to the k th slice of p , which is bounded by $\Phi_k^i(p)$ and $\Phi_{k+1}^i(p)$, $1 \leq k < np_i$. A *section* is the $(n - 1)D$ -OPP, $n > 1$, resulting from the intersection between an nD -OPP p and a $(n - 1)D$ hyperplane perpendicular to the coordinate axis X_i , $n \geq i \geq 1$, which dose not

coincide with any $(n - 1)$ D-couplet of p . A section will be called *external* or *internal* section of p if it is empty or not, respectively. $S_k^i(p)$ will refer to the k th section of p between $\Phi_k^i(p)$ and $\Phi_{k+1}^i(p)$, $1 \leq k < np_i$. Moreover, $S_0^i(p)$ and $S_{np_i}^i(p)$ will refer to the empty sections of p before $\Phi_1^i(p)$ and after $\Phi_{np_i}^i(p)$, respectively.

The projection of the set of $(n - 1)$ D couplets, $\pi_i(\Phi_k^i(P))$, $1 \leq i \leq n$, of an n D-OPP p can be obtained by computing the regularized XOR (\otimes^*) between the projections of its previous $\pi_i(S_{k-1}^i(p))$ and next $\pi_i(S_k^i(p))$ sections; that is, $\pi_i(\Phi_k^i(p)) = \pi_i(S_{k-1}^i(p)) \otimes^* \pi_i(S_k^i(p))$, for all $k \in [1, np_i]$. On the other hand, the projection of any section, $\pi_i(S_k^i(p))$, of an n D-OPP p can be obtained by computing the regularized XOR between the projection of its previous section, $\pi_i(S_{k-1}^i(p))$, and the projection of its previous couplet $\pi_i(\Phi_k^i(p))$. Or, equivalently, by computing the regularized XOR of the projections of all the previous couplets; that is, $\pi_i(S_k^i(p)) = \otimes_{j=1}^{*k} \pi_i(\Phi_j^i(p))$.

2.2.1. The Regularized XOR Operation on the n D-EVM

Let p and q be two n D-OPPs having their respective representations $EVM_n(p)$ and $EVM_n(q)$, then $EVM_n(p \otimes^* q) = EVM_n(p) \otimes EVM_n(q)$. This result, combined with the procedures presented in the previous section for computing sections from couplets and vice versa, allows expressing a formula for computing them by means of their corresponding extreme vertices models. That is,

- (i) $EVM_{n-1}(\pi_i(\Phi_k^i(p))) = EVM_{n-1}(\pi_i(S_{k-1}^i(p))) \otimes EVM_{n-1}(\pi_i(S_k^i(p)))$,
- (ii) $EVM_{n-1}(\pi_i(S_k^i(p))) = EVM_{n-1}(\pi_i(S_{k-1}^i(p))) \otimes EVM_{n-1}(\pi_i(\Phi_k^i(p)))$.

2.2.2. The Regularized Boolean Operations on the n D-EVM

Let p and q be two n D-OPPs and $r = p \text{ op}^* q$, where op^* is in $\{\cup^*, \cap^*, -^*, \otimes^*\}$. Then, $\pi_i(S_k^i(r)) = \pi_i(S_k^i(p)) \text{ op}^* \pi_i(S_k^i(q))$. This implies a regularized Boolean operation, op^* , where $\text{op}^* \{\cup^*, \cap^*, -^*, \otimes^*\} \in$, over two n D-OPPs p and q , both expressed in the n D-EVM, can be carried out by means of the same op^* applied over their own sections, expressed through their extreme vertices models, which are $(n - 1)$ D-OPPs. This last property leads to a recursive process, for computing the regularized Boolean operations using the n D-EVM, which descends on the number of dimensions. The base or trivial case of the recursion is given by the 1D-Boolean operations which can be achieved using direct methods. The XOR operation can also be performed according to the result described in the above subsection.

2.3. Basic Algorithms for the n D-EVM

Before going any further, we say X_A -axis is the n D space's coordinate axis associated to the first coordinate present in the vertices of $EVM_n(p)$. For example, given coordinates ordering $X_1 X_2 X_3$, for a 3D-OPP, then $X_A = X_1$.

The following primitive operations are, in fact, based in those originally presented in [1] see Algorithm 1.

As can be observed, algorithms *MergeXor*, *GetSection*, and *GetHvl* are clearly sustained in the results presented in Section 2.2. The algorithm *GetSection*, via *MergeXor* algorithm,

```

Output: An empty  $nD$ -EVM.
Procedure InitEVM( )
{ Returns the empty set. }
Input: An  $(n - 1)D$ -EVM hvl embedded in  $nD$  space.
Input/Output: An  $nD$ -EVM  $p$ 
Procedure PutHvl(EVM hvl, EVM  $p$ )
{ Appends an  $(n - 1)D$  couplet hvl, perpendicular to  $X_A$ -axis, to  $p$ . }
Input: An  $nD$ -EVM  $p$ 
Output: An  $(n - 1)D$ -EVM embedded in  $(n - 1)D$  space.
Procedure ReadHvl(EVM  $p$ )
{ Extracts next  $(n - 1)D$  couplet perpendicular to  $X_A$ -axis from  $p$ . }
Input: An  $nD$ -EVM  $p$ 
Output: A Boolean.
Procedure EndEVM(EVM  $p$ )
{ Returns true if the end of  $p$  along  $X_A$ -axis has been reached. }
Input/Output: An  $(n - 1)D$ -EVM  $p$  embedded in  $(n - 1)D$  space.
Input: A coordinate coord of type CoordType
(CoordType is the chosen type for the vertex coordinates)
Procedure SetCoord(EVM  $p$ , CoordType coord)
{ Sets the  $X_A$ -coordinate to coord on every vertex of the  $(n - 1)D$ 
couplet  $p$ . For coord = 0, it performs the projection  $\pi_A(p)$ . }
Input: An  $(n - 1)D$ -EVM  $p$  embedded in  $nD$  space.
Output: A CoordType
Procedure GetCoord(EVM  $p$ )
{ Gets the common  $X_A$  coordinate of the  $(n - 1)D$  couplet  $p$ . }
Input: An  $nD$ -EVM  $p$ .
Output: A CoordType
Procedure GetCoordNextHvl(EVM  $p$ )
{ Gets the common  $X_A$  coordinate of the next available  $(n - 1)D$  couplet of  $p$ . }
Input: Two  $nD$ -EVMs  $p$  and  $q$ .
Output: An  $nD$ -EVM
Procedure MergeXor(EVM  $p$ , EVM  $q$ )
{ Applies the ordinary Exclusive OR operation to the vertices of
 $p$  and  $q$  and returns the resulting set. }
Input: An  $(n - 1)D$ -EVM corresponding to section  $S$ .
An  $(n - 1)D$ -EVM corresponding to couplet hvl.
Output: An  $(n - 1)D$ -EVM.
Procedure GetSection(EVM  $S$ , EVM hvl)
{ return MergeXor( $S$ , hvl) }
Input: An  $(n - 1)D$ -EVM corresponding to section  $S_i$ .
An  $(n - 1)D$ -EVM corresponding to section  $S_j$ .
Output: An  $(n - 1)D$ -EVM.
Procedure GetHvl(EVM  $S_i$ , EVM  $S_j$ )
{ return MergeXor( $S_i$ ,  $S_j$ ) }
Input: Two  $nD$ -EVMs  $p$  and  $q$ .
The Boolean Operation op to apply over  $p$  and  $q$ .
The number  $n$  of dimensions.
Output: An  $nD$ -EVM.
Procedure BooleanOperation(EVM  $p$ , EVM  $q$ , BooleanOperator op, int  $n$ )
{ Returns as output an  $nD$ -OPP  $r$ , codified as an  $nD$ -EVM, such that
 $r = p \text{ op}^* q$ . }

```

Algorithm 1: Basic procedures under the EVM.

returns the projection of the next section of an nD -OPP between its previous section and couplet [1, 3]. On the other hand, the algorithm *GetHvl* returns the projection of the couplet between consecutive input sections S_i and S_j [1, 3].

The algorithm *BooleanOperation* computes the resulting nD -OPP $r = p \text{ op}^* q$, where op^* is in $\{\cup^*, \cap^*, -, \otimes^*\}$ (note that $r = p \otimes^* q$ can also be trivially performed using *MergeXor* function). The basic idea behind the procedure is the following.

- (i) The sequence of sections from p and q , perpendicular to X_A -axis, are obtained first.
- (ii) Then, every section of r can recursively be computed as

$$\pi_i(S_k^i(r)) = \pi_i(S_k^i(p)) \text{ op}^* \pi_i(S_k^i(q)). \quad (2.7)$$

- (iii) Finally, r 's couplets can be obtained from its sequence of sections, perpendicular to X_A -axis.

In fact, *BooleanOperation* can be implemented in a wholly merged form in which it only needs to store one section for each of the operands p and q (sP and sQ , resp.), and two consecutive sections for the result r ($sRprev$ and $sRcurr$) [1]. The idea is to consider a main loop that gets sections from p and/or q , using function *GetSection*. These sections, sP and sQ , are recursively processed to compute the corresponding section of r , $sRcurr$. Since r 's two consecutive sections, $sRprev$ and $sRcurr$, are kept, then the projection of the resulting couplet, is obtained by means of function *GetHvl*, and then, it is correctly positioned in r 's EVM by procedure *SetCoord* [1]. When the end of one of the polytopes p or q is reached, then the main loop finishes, and the remaining couplets of the other polytope are either appended or not to the resulting polytope r depending on the Boolean operation considered. More specific details about this implementation of *BooleanOperation* algorithm can be found in [1] or [3].

3. Forward and Backward Differences of an nD -OPP

Because it is well known that regularized XOR operation over two sets A and B can be expressed as $A \otimes^* B = (A -^* B) \cup^* (B -^* A)$, then we have that given an nD -OPP p , and specifically one of its couplets

$$\begin{aligned} \pi_i(\Phi_k^i(p)) &= \pi_i(S_{k-1}^i(p)) \otimes^* \pi_i(S_k^i(p)) = \left(\pi_i(S_{k-1}^i(p)) -^* \pi_i(S_k^i(p)) \right) \\ &\cup^* \left(\pi_i(S_k^i(p)) -^* \pi_i(S_{k-1}^i(p)) \right), \end{aligned} \quad (3.1)$$

the expressions $\pi_i(S_{k-1}^i(p)) -^* \pi_i(S_k^i(p))$ and $\pi_i(S_k^i(p)) -^* \pi_i(S_{k-1}^i(p))$ will be called *forward* and *backward differences* of the projection of two consecutive sections $\pi_i(S_{k-1}^i(p))$ and $\pi_i(S_k^i(p))$, and they will be denoted by $FD_k^i(p)$ and $BD_k^i(p)$, where X_i -axis is perpendicular to them.

One interesting characteristic, described in [1], of forward and backward differences of a 3D-OPP, is that forward differences are the sets of faces, on a couplet, whose normal vectors point to the positive side of the coordinate axis perpendicular to such couplet. Similarly, backward differences are the sets of faces, on a couplet, whose normal vectors point to the negative side of the coordinate axis perpendicular to such couplet. By this way, it is provided a procedure for obtaining the correct orientation of faces in a 3D-OPP when it is converted from 3D-EVM to a boundary representation.

In the context of an nD -OPP p , there are methods to identify normal vectors in the $(n - 1)D$ cells included in p 's boundary. For example, simplicial combinatorial topology provides methodologies assuming a polytope is represented under a simplexation. Such methods operate under the fact the $n + 1$ vertices of an nD simplex are labeled and sorted. Such sorting corresponds to an odd or an even permutation. By taking n vertices from the $n + 1$ vertices of the nD simplex, we get the vertices corresponding to one of its $(n - 1)D$ -cells. In this point, usually, a set of entirely arbitrary rules are given to determine the normal vector to such $(n - 1)D$ cells; see for example, [19, 20]. Such rules establish, according to the parity of the permutation, if the assigned normal vector points towards the interior of the polytope or outside of it.

On the other hand, there are works that consider the determination of normal vectors by taking into account properties of the cross product and the vectors that compose the basis of nD space. For example, Kolcun [21] provides one of such methodologies; however, it is also dependent of polytopes are represented through a simplexation. In this last sense, forward

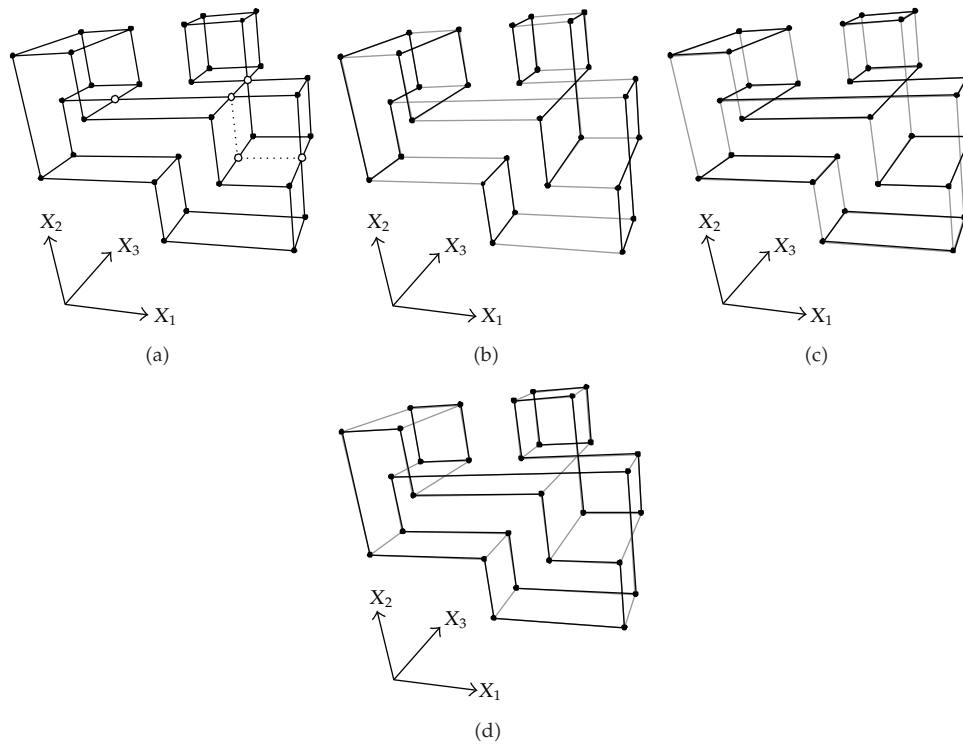


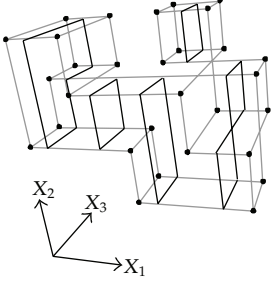
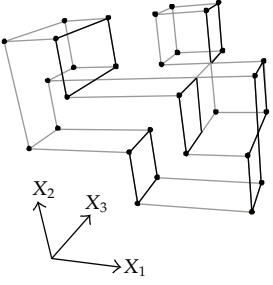
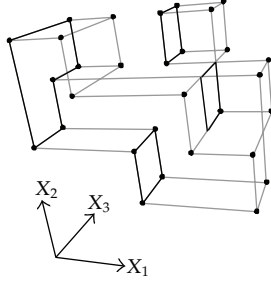
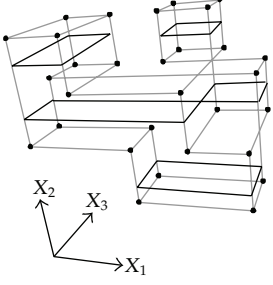
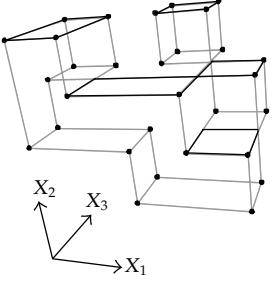
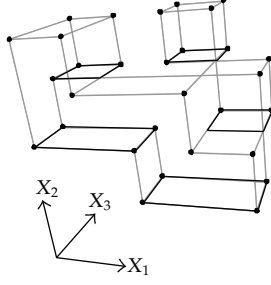
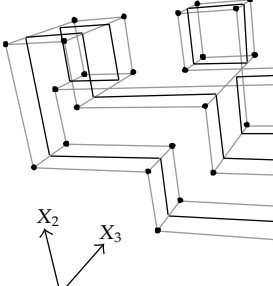
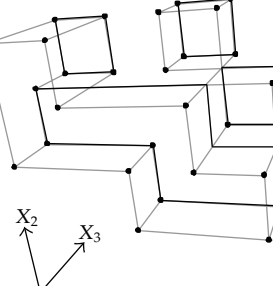
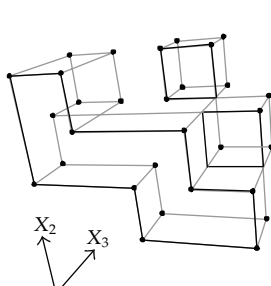
Figure 2: (a) A 3D-OPP with its extreme (black) and nonextreme vertices (white). (b) Couplets perpendicular to X_1 -axis. (c) Couplets perpendicular to X_2 -axis. (d) Couplets perpendicular to X_3 -axis.

and backward differences will provide us a powerful tool, because, in fact, given an n D-OPP p , forward differences $FD_k^i(p)$ are the sets of $(n-1)$ D cells lying on $\Phi_k^i(p)$, whose normal vectors point to the positive side of the coordinate axis X_i , while backward differences $BD_k^i(p)$ are the sets of $(n-1)$ D cells also lying on $\Phi_k^i(p)$ but whose normal vectors point to the negative side of the coordinate X_i -axis.

Figure 2 shows a 3D-OPP and its corresponding sets of 2D-couplets perpendicular to X_1 , X_2 , and X_3 -axes (Figures 2(b), 2(c), 2(d)). According to Figure 2(a), the OPP has 5 non-extreme vertices which are marked in white. Three of them have four coplanar incident odd edges; another one has six incident odd edges, and the last one has exactly two incident collinear odd edges. The remaining vertices, actually the extreme vertices, have exactly three linearly independent odd edges. Table 1 shows the extraction, for the 3D-OPP shown in Figure 2(a), of its faces and their correct orientation through forward and backward differences. The first row shows sections perpendicular to X_1 -axis. Through them, we compute forward differences $FD_1^1(p)$ to $FD_3^1(p)$ in order to obtain the faces whose normal vector points towards the positive side of X_1 -axis. On the other hand, these same sections share us to compute backward differences $BD_1^1(p)$ to $BD_3^1(p)$, which are composed by the set of faces whose normal vector points towards the negative side of X_1 -axis. In a similar way, the remaining two rows of Table 1 show sections perpendicular to X_2 and X_3 -axes and their respective forward and backward differences.

An algorithm for computing forward and backward differences consists of obtaining projections of sections of the input polytope and processing them through *BooleanOperation*

Table 1: Forward and backward differences of a 3D-OPP (see text for details).

Sections	Forward differences	Backward differences
		
		
		

algorithm by computing regularized difference between two consecutive sections. Algorithm 2 implements these simple ideas in order to compute the forward and backward differences in an nD -OPP p represented through the nD -EVM. Its output will consist of two sets: the first set FD contains the $(n - 1)D$ -EVMs corresponding to forward differences in p , while the second set BD contains the $(n - 1)D$ -EVMs corresponding to backward differences in p . Algorithm 2 will be useful when we describe our procedure for extracting kD boundary elements of an nD -OPP which is represented through the nD -EVM. Such procedure will be described in Section 5.

4. A Connected Components Labeling Algorithm under the nD -EVM

Now, as one of the main contributions of this work, we will describe our proposed algorithm for performing connected components labeling in an nD -OPP p expressed under the nD -EVM. We think our algorithm can be positioned in Suzuki's second category (see Section 1), because it assigns temporal labels for elements embedded in the current processed slice of p ,

```

Input: An  $n$ D-EVM  $p$ .
          The number  $n$  of dimensions.
Output: A set FD containing the  $(n - 1)$ D-EVMs of Forward Differences in  $p$ .
          A set BD containing the  $(n - 1)$ D-EVMs of Backward Differences in  $p$ .
Procedure GetForwardBackwardDifferences(EVM  $p$ , int  $n$ )
  FD =  $\emptyset$  // FD will store  $(n - 1)$ D-EVMs corresponding to Forward Differences.
  BD =  $\emptyset$  // BD will store  $(n - 1)$ D-EVMs corresponding to Backward Differences.
  EVM hvl // Current couplet.
  EVM  $S_i, S_j$  // Previous and next sections about hvl.
  EVM FDcurr // Current Forward Difference.
  EVM BDcurr // Current Backward Difference.
  hvl = InitEVM()
   $S_i$  = InitEVM()
   $S_j$  = InitEVM()
  while(Not(EndEVM( $p$ )))
    hvl = ReadHvl( $p$ )// Read next couplet.
     $S_j$  = GetSection( $S_i$ , hvl)
    FDcurr = BooleanOperation( $S_i, S_j$ , Difference,  $n - 1$ )
    BDcurr = BooleanOperation( $S_j, S_i$ , Difference,  $n - 1$ )
    FD = FD  $\cup$  {FDcurr} // The new computed Forward Difference is added to FD.
    BD = BD  $\cup$  {BDcurr} // The new computed Backward Difference is added to BD.
     $S_i$  =  $S_j$ 
  end-of-while
  return {FD, BD}
end-of-procedure

```

Algorithm 2: Computing forward and backward differences in a polytope p represented through an n D-EVM.

$\text{Slice}_k^i(p)$, but when the next slice is processed, $\text{Slice}_{k+1}^i(p)$, it is possible some of these labels change taking place a second pass over $\text{Slice}_k^i(p)$ which assigns new labels. Two properties of interest of our algorithm are pointed out.

- (i) It processes only two consecutive slices of p at a time. In fact, such slices are not explicitly manipulated but their corresponding representative $(n - 1)$ D sections.
- (ii) The algorithm descends recursively over the number of dimensions in such way that it is determined for each $(n - 1)$ D section of p its corresponding $(n - 2)$ D components. By this way, the $(n - 2)$ D components of two consecutive $(n - 1)$ D sections are manipulated and used for determining the labels to assign.

The idea behind the proposed methodology considers to process sequentially each $(n - 1)$ D section of p . Let S_j be the current processed section of p . We obtain, via a recursive call, its $(n - 1)$ D components. Let S_i be the section before S_j . For each $(n - 1)$ D component cS_j in S_j and for each $(n - 1)$ D component cS_i in S_i , it is performed their regularized intersection $cS_j \cap^* cS_i$. According to the result, one of three cases can arise.

Case 1. (i) If the intersection is not empty and cS_j does not belong to any component of p , then cS_j is added to the component of p , where the current component of S_i , cS_i , belongs.

Case 2. (i) If the intersection is not empty but cS_j is already associated to a component of p , then such component and the component where cS_i is contained are united to form only one new component for p . It could be the case that both components to unite are, in fact, the same. In this situation, no action is performed. On the other hand, if the union is achieved, then all

the references pointing to the original nD components where cS_i and cS_j belonged must be updated in order to point to the new component.

Case 3. (i) If the intersection with all components of S_i is empty, then cS_j does not belong to any of the currently identified components of p . Therefore, cS_j belongs to, and starts, a new nD component.

Tables 2 and 3 show an example where the above ideas are applied on a 2D-OPP. Such OPP is composed of 22 extreme vertices and has five internal sections which are perpendicular to X_1 -axis. Once all sections are processed, it is determined the existence of two components.

A more specific implementation of our procedure is given in Algorithm 3. In order to distinguish between Cases 1 and 2, the algorithm uses a Boolean flag (*withoutComponent*). When Case 1 is achieved, then cS_j , the current component of section S_j , is already associated to a component of p . Therefore, the flag's value is inverted for indicating this last property and, for instance, sharing that only Case 2 could be reached in the subsequent iterations. If for all components of S_i , the section before S_j , the intersection with cS_j was empty, then the flag never changed its original value. In this way, Case 3 can be identified.

Our implementation considers the use of lists for controlling and manipulating references between $(n - 1)D$ sections' components and their associated nD components. We assume that such lists are manipulated and accessed through the following primitive functions.

- (i) *Size (list)*: returns the number of elements in the list.
- (ii) *List (i)*: returns the element at position i of the list, $0 \leq i < \text{Size}(\text{list})$.
- (iii) *Update (list, i, nV)*: the list is updated by substituting element in position i with the new element nV .
- (iv) *Append (list, v)*: an element v is appended at the end of the list.
- (v) *Insert (list, i, v)*: an element v is inserted at position i of the list.
- (vi) *Remove (list, i)*: it is removed from the list the element at position i .
- (vii) *IndexOf (list, v)*: returns a negative integer if v is not present in the list. Otherwise, an index between 0 and $\text{Size}(\text{list})-1$ is returned indicating the first occurrence of v .
- (viii) *DeleteAll (list)*: deletes all the elements contained in the list.

Algorithm 3 considers the use of five lists.

- (i) *componentsSj (componentsSi)*: the $(n - 1)D$ components of section $S_j(S_i)$. The $(n - 1)D$ -EVM associated to the j th (i th) component of section $S_j(S_i)$ is obtained via *componentsSj(j) (componentsSi(i))*.
- (ii) *indexesSj (indexesSi)*: a list of indexes. $k_1 = \text{indexesSj}(j)$ ($k_2 = \text{indexesSi}(i)$) denotes the j th (i th) component of section $S_j(S_i)$ belongs to the k_1 th (k_2 th) component of nD -OPP p .
- (iii) *componentsP*: the components of input nD -OPP p . *components(k)* returns a list of $(n - 1)D$ sections which describe the k th nD component of p .

Let cS_i be the i th component of section S_i and cS_j be the j th component of section S_j . Both cS_i and cS_j were, respectively, obtained from *componentsSi(i)* and *componentsSj(j)*. The lists are used and manipulated according the considered algorithm's case.

Table 2: Connected component labeling over a 2D-OPP (Part 1).

<p>$S_1^1(p)$</p>	<p>The projection of section $S_1^1(p)$ has the 1D component $\overline{a_1 b_1}$. Because $S_0^1(p)$ is an empty section. Then, Case 3: $\overline{a_1 b_1}$ belongs to the new component 0.</p>	
<p>$S_1^1(p)$ $S_2^1(p)$</p>	<p>The projection of section $S_2^1(p)$ has four 1D components: $\overline{a_2 b_2}$, $\overline{c_2 d_2}$, $\overline{e_2 f_2}$, and $\overline{g_2 h_2}$. $\overline{g_2 h_2} \cap \overline{a_1 b_1} \neq \emptyset$, and $\overline{g_2 h_2}$ does not belong to any component of p. Then, Case 1: $\overline{g_2 h_2}$ now belongs to component 0 (the component which $\overline{a_1 b_1}$ belongs).</p>	
	<p>$\overline{e_2 f_2} \cap \overline{a_1 b_1} \neq \emptyset$, and $\overline{e_2 f_2}$ does not belong to any component of p. Then, Case 1: $\overline{e_2 f_2}$ now belongs to component 0.</p>	
	<p>$\overline{c_2 d_2} \cap \overline{a_1 b_1} = \emptyset$. Case 3: $\overline{c_2 d_2}$ belongs to new component 1.</p>	
	<p>$\overline{a_2 b_2} \cap \overline{a_1 b_1} = \emptyset$. Case 3: $\overline{a_2 b_2}$ belongs to new component 2.</p>	

- (i) In Case 1, $componentsP(indexesSi(i))$ returns the component of p , where cS_i belongs. cS_j is incorporated to such component by appending it in such list. Finally, via $Append(indexesSj, indexesSi(i))$, it is now established that cS_j belongs to the same component than cS_i , because they have the same component's index.
- (ii) In Case 2, we obtain the components, where cS_i and cS_j belong through $componentsP(indexesSi(i))$ and $componentsP(indexesSj(j))$ (if $indexesSi(i)$ and $indexesSj(j)$ are equal, then, as commented previously, no action is performed). All the elements

Table 3: Connected component labeling over a 2D-OPP (Part 2).

	<p>Projection of section $S_3^1(p)$ has two 1D components: $\overline{a_3b_3}$ and $\overline{c_3d_3}$.</p>	
	<p>$\overline{c_3d_3} \cap^* \overline{g_2h_2} \neq \emptyset$, and $\overline{c_3d_3}$ does not belong to any component of p. Then, Case 1: $\overline{c_3d_3}$ now belongs to component 0.</p>	
	<p>$\overline{c_3d_3} \cap^* \overline{e_2f_2} \neq \emptyset$, and $\overline{c_3d_3}$ is associated to component 0. Then, Case 2: $\overline{e_2f_2}$ also belongs to component 0. No action takes place.</p>	
	<p>$\overline{c_3d_3} \cap^* \overline{c_2d_2} \neq \emptyset$, and $\overline{c_3d_3}$ belongs to component 0. Case 2: $\overline{c_2d_2}$ is associated to component 1. Therefore, components 0 and 1 are united in the new component 3.</p>	
	<p>$\overline{a_3b_3} \cap^* \overline{a_2b_2} \neq \emptyset$, and $\overline{a_3b_3}$ does not belong to any component of p. Case 1: $\overline{a_3b_3}$ is now associated to component 2.</p>	
	<p>The projection of section $S_4^1(p)$ is composed by segment $\overline{a_4b_4}$.</p>	
	<p>$\overline{a_4b_4} \cap^* \overline{a_3b_3} \neq \emptyset$, and $\overline{a_4b_4}$ does not belong to any component of p. Therefore, Case 1: $\overline{a_4b_4}$ now belongs to component 2.</p>	
	<p>The projection of section $S_5^1(p)$ is composed by segment $\overline{a_5b_5}$.</p>	
	<p>$\overline{a_5b_5} \cap^* \overline{a_4b_4} \neq \emptyset$, and $\overline{a_5b_5}$ does not belong to any component of p. Then, Case 1: $\overline{a_5b_5}$ is now associated to component 2.</p>	

in the first component are transferred to the second component. The position where the first component resided in the list *componentsP* is now updated to *NULL*. Via, *Update(indexesSi, i, indexesSj(j))*, it is established that cS_i belongs to the same component like cS_j . Finally, it must be verified if there are $(n - 1)$ D components of S_i and S_j whose value in *indexesSi* and *indexesSj* refers to the now nullified position in *componentsP*. If it is the situation, then their indexes must be updated with the value in *indexesSj(j)*.

- (iii) In Case 3, a new component for n D-OPP p is created. The new list, containing only to cS_j , is appended in list *componentsP*. The value given by the size of list *componentsP* minus one is appended in list *indexesSj*. Such value corresponds to the component's index for cS_j .

When the algorithm finishes the processing of sections, each list in *componentsP* is a list of $(n - 1)$ D sections or a *NULL* element. If a *NULL* element is found, then it must be removed. Otherwise, we obtain the corresponding n D-EVM, which at its time corresponds to a component of the input n D-OPP p . It is possible the $(n - 1)$ D sections contained in a given list are not correctly ordered. With the purpose of determining the correct sequence, in Algorithm 3, when a $(n - 1)$ D-OPP is appended, the common X_A coordinates of the previous and next couplets are also introduced with it.

Consider a 5D-OPP q generated by the union of the following six 5D hypercubes:

- (i) $C_1 = [0, 1] \times [0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$,
- (ii) $C_2 = [0, 1] \times [1, 2] \times [0, 1] \times [0, 1] \times [0, 1]$,
- (iii) $C_3 = [0, 1] \times [2, 3] \times [0, 1] \times [0, 1] \times [0, 1]$,
- (iv) $C_4 = [1, 2] \times [2, 3] \times [0, 1] \times [0, 1] \times [0, 1]$,
- (v) $C_5 = [2, 3] \times [2, 3] \times [1, 2] \times [1, 2] \times [1, 2]$,
- (vi) $C_6 = [3, 4] \times [2, 3] \times [0, 1] \times [1, 2] \times [1, 2]$.

Figure 3(a) shows q 's 5D-4D-3D-2D projection. Hypercubes C_5 and C_6 have a common (3D) volume; hypercube C_5 shares an edge with hypercube C_4 , and hypercubes C_1 to C_4 compose a *J-shaped* form. Its representation via the 5D-EVM required 92 extreme vertices. Figures 3(b) and 3(c) show its 4D couplets and sections, perpendicular to X_1 -axis, respectively. Each 4D section of q is composed of only one component (this is determined via a recursive call of Algorithm 3). Remember that $S_0^1(q) = \emptyset$, hence, its intersection with section $S_1^1(q)$ is also empty and, therefore, $S_1^1(q)$ is associated to 5D component 0. Next, the regularized intersections between projections of sections $S_1^1(q)$ and $S_2^1(q)$ is also empty. This implies that $S_2^1(q)$ is associated with 5D component 1. The same result is obtained when computing regularized intersection between projections of sections $S_2^1(q)$ and $S_3^1(q)$. Component 2 has to $S_3^1(q)$. Finally, the intersection between $S_3^1(q)$ and $S_4^1(q)$ corresponds to a 4D hyperbox, which implies that $S_4^1(q)$ is also associated to component 2. As a final result, Algorithm 3 identified three 5D components in q .

- (i) Component 0 corresponds to hypercube C_6 (Figure 4(a)).
- (ii) Component 1 corresponds to hypercube C_5 (Figure 4(b)).
- (iii) Component 2 corresponds to the union of hypercubes C_1 to C_4 (Figure 4(c)).

Previously, we mentioned hypercubes C_5 and C_6 shared a 3D volume. When computing regularized intersection between $S_1^1(q)$ and $S_2^1(q)$ (their corresponding sections in q) the result


```

Input: An  $nD$ -OPP  $p$ .
           The number  $n$  of dimensions.
Output: A list containing the EVMs which describe the  $nD$  components of  $p$ .
Procedure ConnectedComponentsLabeling(EVM  $p$ , int  $n$ )
  if( $n = 1$ ) then
    return ConnectedComponentsLabeling1D( $p$ ) // Base case.
  else
    EVM hvl // Current  $(n - 1)D$  couplet of  $p$ .
    EVM  $S_i, S_j$  // Previous and next  $(n - 1)D$  sections about hvl.
     $S_i = \text{InitEVM}()$ 
    List components  $S_i$  // The  $(n - 1)D$  components of section  $S_i$ .
    List indexes  $S_i$  // A list of indexes.
    List components  $S_j$  // The  $(n - 1)D$  components of section  $S_j$ .
    List indexes  $S_j$  // A list of indexes.
    List componentsP // The  $nD$  components of input polytope  $p$ .
    CoordType currCoord = getCoordNextHvl( $p$ )
    hvl = ReadHvl( $p$ )
    CoordType nextCoord = getCoordNextHvl( $p$ )
    while(Not(EndEVM( $p$ )))
       $S_j = \text{GetSection}(S_i, \text{hvl})$ 
      components  $S_j = \text{ConnectedComponentsLabeling}(S_j, n - 1)$  // Recursive call.
      for(int  $j = 0, j < \text{Size}(\text{components } S_j), j = j + 1$ )
        EVM  $cS_j = \text{components } S_j(j)$ 
        boolean withoutComponent = true
        for(int  $i = 0, i < \text{Size}(\text{components } S_i), i = i + 1$ )
          EVM  $cS_i = \text{components } S_i(i)$ 
          EVM  $c = \text{BooleanOperation}(cS_j, cS_i, \text{Intersection}, n - 1)$ 
          if(Not(IsEmpty( $c$ )) and (withoutComponent = true)) then
            // CASE 1
            withoutComponent = false
            List componentToUpdate = componentsP(indexes  $S_i(i)$ )
            Append(componentToUpdate, { $cS_j, \text{currCoord}, \text{nextCoord}$ })
            Append(indexes  $S_j, \text{indexes } S_i(i)$ )
          else if(Not(IsEmpty( $c$ )) and (withoutComponent = false)) then
            // CASE 2
            if(indexes  $S_i(i) \neq \text{indexes } S_j(j)$ ) then
              List component1 = componentsP(indexes  $S_i(i)$ )
              List component2 = componentsP(indexes  $S_j(j)$ )
              for(int  $k = 0, k < \text{Size}(\text{component1}), k = k + 1$ )
                Append(component2, component1( $k$ ))
              end-of-for
              Update(componentsP, indexes  $S_i(i)$ , NULL)
              for(int  $k = 0, k < \text{Size}(\text{indexes } S_i), k = k + 1$ )
                if(indexes  $S_i(k) = \text{indexes } S_i(i)$ ) then
                  Update(indexes  $S_i, k, \text{indexes } S_j(j)$ )
                end-of-if
              end-of-for
              for(int  $k = 0, k < \text{Size}(\text{indexes } S_j), k = k + 1$ )
                if(indexes  $S_j(k) = \text{indexes } S_i(i)$ ) then
                  Update(indexes  $S_j, k, \text{indexes } S_j(j)$ )
                end-of-if
              end-of-for
              Update(indexes  $S_i, i, \text{indexes } S_j(j)$ )
            end-of-if
          end-of-if
        end-of-for
      if(withoutComponent = true) then
        // CASE 3
        List newComponent
        Append(newComponent, { $cS_j, \text{currCoord}, \text{nextCoord}$ })
        Append(componentsP, newComponent)
        Append(indexes  $S_j, \text{Size}(\text{componentsP}) - 1$ )
      end-of-if
    end-of-for
     $S_i = S_j$ 
    components  $S_i = \text{components } S_j$ 
    indexes  $S_i = \text{indexes } S_j$ 
    currCoord = GetCoordNextHvl( $p$ )
    hvl = ReadHvl( $p$ ) // Read next couplet.
  nextCoord = GetCoordNextHvl( $p$ )
end-of-while
// End of sections' processing.
int  $i = 0$ 
while( $i < \text{Size}(\text{componentsP})$ )
  if(componentsP( $i$ ) = NULL) then
    Remove(componentsP,  $i$ )
     $i = i - 1$ 
  else
    List currentListOfSections = componentsP( $i$ )
    EVM evmComponent = getEVMforListOfSections(currentListOfSections,  $n$ )
    Update(componentsP,  $i$ , evmComponent)
  end-of-if
   $i = i + 1$ 
end-of-while
return componentsP
end-of-if
end-of-procedure

```

Algorithm 3: Connected component labeling over an nD -OPP expressed in the nD -EVM.

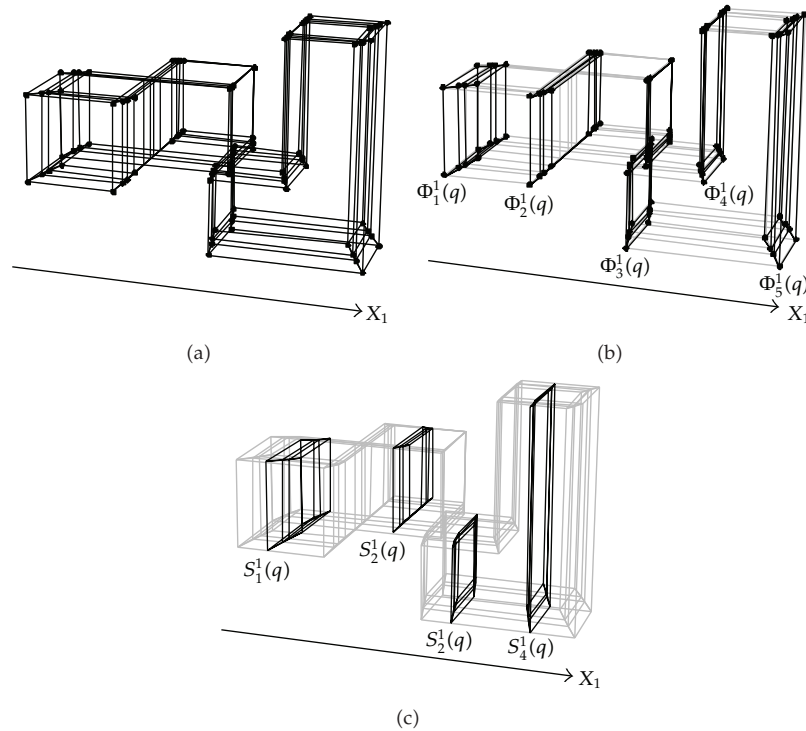


Figure 3: A 5D-OPP. (a) 5D-4D-3D-2D projection. (b) 4D Couplets perpendicular to X_1 -axis. (c) 4D Sections perpendicular to X_1 -axis.

was the empty set. This is due to the use of regularized boolean operations: operations between nD polytopes must be dimensionally homogeneous; that is, they must produce as result another nD polytope or the null object [22]. Hypercubes C_5 and C_4 shared a (1D) edge, but the regularized intersection between their corresponding sections, in q , $S_2^1(q)$ and $S_3^1(q)$, also produced a null object. A distinct situation arises when the regularized intersection between $S_3^1(q)$ and $S_4^1(q)$ is analyzed: Algorithm 3 determined that they are associated with the same component.

The above example has illustrated a property of Algorithm 3. When the foundations behind our method were established, it was commented the algorithm process, not in an explicit way, consecutive slices of the input polytope p . Each nD slice of p has its corresponding representative $(n - 1)D$ section for which Algorithm 3 determines its corresponding $(n - 1)D$ components. Let c_1 and c_2 be *nonempty* $(n - 1)D$ components such that c_1 is associated to $(n - 1)D$ section $S_j^i(p)$, while c_2 is associated to $(n - 1)D$ section $S_{j+1}^i(p)$. The fact that c_1 and c_2 have empty regularized intersection implies (1) that their corresponding nD regions are disjoint and, therefore, they have null adjacencies, or (2) that some parts of its corresponding slices have a kD adjacency; that is, some kD elements, $k = 0, 1, 2, \dots, n - 2$, are, completely or partially, shared. In both cases, Algorithm 3 uses this kind of adjacencies, via the result of regularized intersection between c_1 and c_2 , for determining that the corresponding nD regions are disjoint and, for instance, associate them with different components. Hence, a nD component shared by Algorithm 3 corresponds to those parts of the input nD -OPP that are united via $(n - 1)D$ adjacencies. The way Algorithm 3

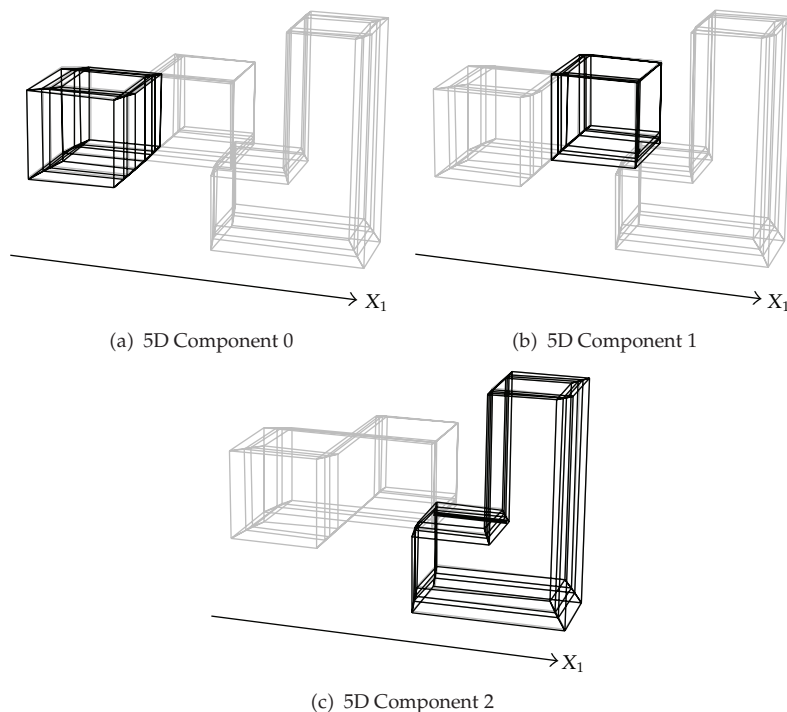


Figure 4: The three components identified by applying Algorithm 3 over the 5D-OPP from Figure 3(a).

breaks a polytope will be very useful in the following sections, specifically when we present our methodology for identifying k D boundary elements.

5. Extracting k D Boundary Elements from an n D-OPP

This section presents the second main contribution of this work. As commented in Section 1, the way we extract k D boundary elements from an n D-OPP represented through the n D-EVM will be reminiscent to the reconstruction of the boundary tree associated to such n D-OPP. According to Section 3, forward and backward differences can be computed through Algorithm 2. In fact, all $FD_k^i(p)$ and $BD_k^i(p)$ from an n D-OPP are $(n-1)$ D-OPPs embedded in $(n-1)$ D space, because the projection operator is used in their corresponding definitions: $FD_k^i(p) = \pi_i(S_{k-1}^i(p)) -^* \pi_i(S_k^i(p))$ and $BD_k^i(p) = \pi_i(S_k^i(p)) -^* \pi_i(S_{k-1}^i(p))$. If such forward and backward differences were computed through our proposed algorithm, then they are expressed as $EVM_{n-1}(FD_k^i(p))$ and $EVM_{n-1}(BD_k^i(p))$. If we apply again Algorithm 2 to such $(n-1)$ D-OPPs, we will get new forward and backward differences that correspond to the $(n-2)$ D oriented cells on the boundary of such $(n-1)$ D-OPPs. These new forward and backward differences are themselves $(n-2)$ D-OPPs represented through the EVM. Hence, by applying again Algorithm 2 to them, we obtain their associated $(n-3)$ D oriented cells grouped as forward and backward differences. This procedure generates a recursive process which descends in the number of dimensions. In each recursivity level, we obtain forward and backward differences associated to the input k D-OPPs, $1 \leq k \leq n$. The base case is

```

Input: An  $k$ D-OPP  $p$ 
The number  $k$  of dimensions
A list boundaryElements containing  $n$  lists.
A list previousCoordinates for completing the  $n$  coordinates of
 $(k - 1)$ -coordinates points.
Procedure GetBoundaryElements(EVM p,int k,List boundaryElements,List previousCoordinates)
  if ( $k \geq 2$ ) then
    EVM FDcurr, BDcurr
    List  $v = \text{GetSetVX1}(p)$ 
    List identifiedBoundaryElements = boundaryElements (k - 1)
    List vertices = boundaryElements(0)
    {FD, BD} = GetForwardBackwardDifferences(p, k)
    for ( $i = 0, i < \text{Size}(\text{FD}), i = i + 1$ ) // Forward Differences are processed.
      EVM FDcurr = FD(i)
      List compsFDcurr = ConnectedComponentsLabeling(FDcurr, k - 1)
      CoordType commonCoord = v(i)
      for ( $j = 0, j < \text{Size}(\text{compsFDcurr}), j = j + 1$ )
        EVM currComp = compsFDcurr(j)
        List verticesCurrComp
        for each Extreme Vertex  $eV$  in currComp do
          List point = getNdPoint(eV, commonCoord, previousCoordinates)
          int index = IndexOf(vertices, point)
          if ( $\text{index} < 0$ ) then // A new point has been discovered.
            Append(vertices, point)
             $\text{index} = \text{Size}(\text{vertices}) - 1$ 
          end-of-if
          Append(verticesCurrComp, index)
        end-of-for
        int index = IndexOf(identifiedBoundaryElements, verticesCurrComp)
        if ( $\text{index} < 0$ ) then // A new  $(k - 1)$ D boundary element has been found.
          Append(identifiedBoundaryElements, verticesCurrComp)
        end-of-if
        Append(previousCoordinates, commonCoord)
        // Recursive call for identifying new  $(k - 2)$ D boundary elements.
        GetBoundaryElements(currComp, k - 1, boundaryElements, previousCoordinates)
        Remove(previousCoordinates,  $\text{Size}(\text{previousCoordinates}) - 1$ )
      end-of-for
    end-of-for
    for ( $i = 0, i < \text{Size}(\text{BD}), i = i + 1$ ) // Backward differences are processed.
      EVM BDcurr = BD(i)
      List compsBDcurr = ConnectedComponentsLabeling(BDcurr, k - 1)
      CoordType commonCoord = v(i)
      for ( $j = 0, j < \text{Size}(\text{compsBDcurr}), j = j + 1$ )
        EVM currComp = compsBDcurr(j)
        List verticesCurrComp
        for each Extreme Vertex  $eV$  in currComp do
          List point = getNdPoint(eV, commonCoord, previousCoordinates)
          int index = IndexOf(vertices, point)
          if ( $\text{index} < 0$ ) then // A new point has been discovered.
            Append(vertices, point)
             $\text{index} = \text{Size}(\text{vertices}) - 1$ 
          end-of-if
          Append(verticesCurrComp, index)
        end-of-for
        int index = IndexOf(identifiedBoundaryElements, verticesCurrComp)
        if ( $\text{index} < 0$ ) then // A new  $(k - 1)$ D boundary element has been found.
          Append(identifiedBoundaryElements, verticesCurrComp)
        end-of-if
        Append(previousCoordinates, commonCoord)
        // Recursive call for identifying new  $(k - 2)$ D boundary elements.
        GetBoundaryElements(currComp, k - 1, boundaryElements, previousCoordinates)
        Remove(previousCoordinates,  $\text{Size}(\text{previousCoordinates}) - 1$ )
      end-of-for
    end-of-for
  end-of-if
end-of-procedure

```

Algorithm 4: Extracting k D boundary elements from an OPP expressed in the EVM.

reached when $n = 1$. In this situation, the boundary of a 1D-OPP is described by the beginning and ending extreme vertices of each one of its composing segments.

Algorithm 4 implements the above-proposed ideas in order to identify and to extract k D boundary elements from a n D-OPP. Input parameters for our algorithm require the EVM associated with the input n D-OPP p , the number k of dimensions, and a list, where the boundary elements of p are going to be stored. Such a list, called *boundaryElements*, is assumed to be a list of lists such that *boundaryElements*(k) returns the list that contains the identified k D elements, $k = 0, 1, 2, \dots, n - 1$. When $k = n$, then we have the main call with n D-OPP p , $n \geq 2$, and list *boundaryElements* initialized with n empty lists (if $n = 1$, then the boundary elements of p are precisely its extreme vertices which can be directly extracted from

the $EVM_1(p)$. When $2 \leq k < n$, then the input OPP is, in fact, a kD boundary element of p and *boundaryElements* contains the elements (vertices, edges, faces, etc.) discovered prior to such recursive call. In the algorithm's basic case, when $k = 1$, no action is performed, because the input 1D-OPP has only as boundary elements the vertices that bound its corresponding segment. The vertices in such 1D-OPP have been previously identified, because the algorithm always receives and uses a list of discovered vertices. Such a list is located in the input *boundaryElements* at position zero, and it is updated, as required, along main and all recursive calls.

Specifically, Algorithm 4 performs the discovery of kD boundary elements, for an input kD -OPP p , in the following way.

- (i) We obtain the list, where the discovered $(k-1)D$ boundary elements are going to be stored: *boundaryElements* $(k-1)$. We obtain the list of previously discovered vertices of p : *boundaryElements*(0).
- (ii) We compute, through Algorithm 2, p 's forward and backward differences perpendicular to X_A -axis (the axis associated with the first coordinate of the vertices in the input EVM).
- (iii) For each element in the list of forward (backward) differences are computed, via Algorithm 3, its corresponding components. Each component is expressed as a $(k-1)D$ -EVM.

Now, each component in the currently processed forward (backward) difference is processed in the following way.

- (i) We initialize a list which will store the vertices that form the component (*verticesCurrComp*).
- (ii) Because the component is an OPP embedded in $(k-1)D$ space, the vertices in its EVM only contain $k-1$ coordinates. The points' coordinates are completed first by incorporating them into the common X_A coordinate that the k -coordinates extreme vertices, that define the forward (backward) difference to which the component belongs, have. The remaining $n-k$ coordinates are taken from the list *previousCoordinates* (such a list is received as input by the algorithm). The function *GetNdPoint* performs the completion of the points' coordinates in such a way that we obtained vertices with n coordinates.
- (iii) Each point (now with n coordinates) from the component is evaluated in order to determine if it has been previously discovered. In this sense, it is verified if the point is present in the list of identified vertices. If function *IndexOf* returns a negative index, then it implies a new vertex has been discovered. In this case, the point is appended in the list of vertices. Its corresponding new index is given by the size of such a list minus one. In any case, the point's index is inserted in the component's list of vertices (*verticesCurrComp*). It is important to take into account that points listed in *verticesCurrComp* appear in the order they were discovered by our algorithm; hence, points' ordering in such a list do not necessarily follow the orientation of the component they belong to.

Once all the points in the EVM of the current component have been processed, we continue, on one side, to determine if it has been previously discovered, and on the other side, to use it for the discovering of new $(k-2)D$ boundary elements. Both tasks are, respectively, achieved in the following way.

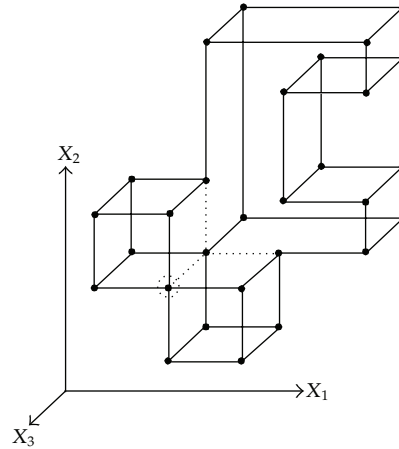


Figure 5: A 3D-OPP. The dotted lines correspond to even edges. The vertex with the dotted circle is a nonextreme vertex.

- (i) It is verified if the component's list of vertices ($verticesCurrComp$) is an element of the list containing the previously identified $(k - 1)$ D boundary elements. If the function $IndexOf$ returns a negative index, then a new $(k - 1)$ D boundary element has been discovered and list $verticesCurrComp$ is appended in the list of $(k - 1)$ D elements.
- (ii) It is performed a recursive call to the algorithm by using as input the component's $(k - 1)$ D-EVM. Such a recursive call will identify $(k - 2)$ D boundary elements of n D-OPP p .

Consider the 3D-OPP p shown in Figure 5. Suppose that the ordering of the coordinates of 3D-EVM of p is $X_1X_2X_3$. All the vertices of p are extreme except that with a dotted circle, because it has five incident edges and four of them are odd edges, while the fifth one, which is parallel to X_3 -axis, is, in fact, an even edge; hence, it does not belong to any brink of p . There are another two even edges in p : one of them is parallel to X_1 -axis while the other to X_2 -axis. Tables 4 and 5 show the way Algorithm 4 discovers faces, perpendicular to X_1 -axis, and vertices of p (we recall that our algorithm lists boundary elements' vertices in the order they are discovered). Table 4 corresponds to the processing of forward differences. $FD_1^1(p)$ is composed by only one component and its 2D-EVM shares the discovering of four vertices. Three of them correspond to projections of extreme vertices of p , because they have only 2 coordinates, while the fourth one precisely corresponds to the projection of the nonextreme vertex indicated in Figure 5. The vertices are completed, in order to have three coordinates, by annexing them the common coordinate of its corresponding forward difference $FD_1^1(p)$. By this way, vertices v_1, v_2, v_3 , and v_4 , and the face they define, are initially discovered. Via Algorithm 3, $FD_2^1(p)$ is separated in two components. Each component corresponds to a new face defined by four vertices. Both faces and the eight vertices are, respectively, added to the lists of discovered faces and vertices.

Table 5 shows the processing of backward differences. $BD_2^1(p)$ corresponds to a 2D-OPP in which six of its seven vertices are extreme vertices. When applying Algorithm 3 over $BD_2^1(p)$, we obtained two components. The extreme vertices of $BD_2^1(p)$ are present in the components, but a new vertex appear which is product of the separation of the original

Table 4: Identifying faces, of the 3D-OPP from Figure 5, through forward differences and Algorithm 4 (see text for details). Faces' vertices are listed in the order they are discovered).

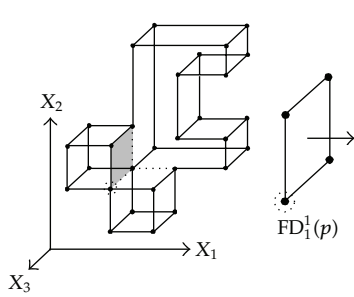
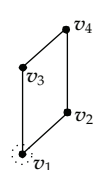
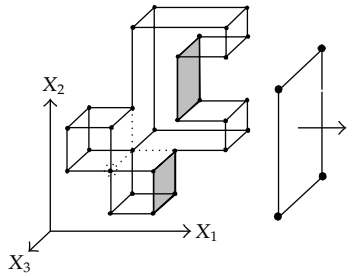
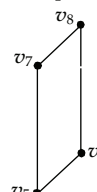
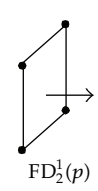
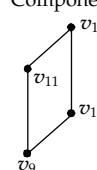
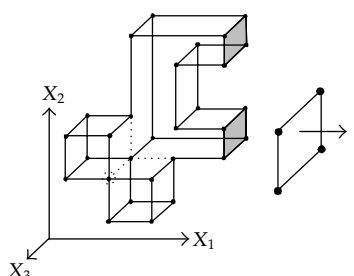

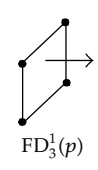
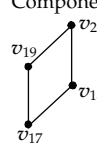
	Forward difference	Connected components labeling	Discovered vertices and faces
			v_1, v_2, v_3, v_4 (Vertex v_1 was not in $EVM_3(p)$). $V(p) = \{v_1, v_2, v_3, v_4\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}\}$
		Component 1 	v_5, v_6, v_7, v_8 $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}\}$
		Component 2 	$v_9, v_{10}, v_{11}, v_{12}$ $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}\}$
		Component 1 	$v_{13}, v_{14}, v_{15}, v_{16}$ $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}, \{v_{13}, v_{14}, v_{15}, v_{16}\}\}$
		Component 2 	$v_{17}, v_{18}, v_{19}, v_{20}$ $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{20}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}, \{v_{13}, v_{14}, v_{15}, v_{16}\}, \{v_{17}, v_{18}, v_{19}, v_{20}\}\}$

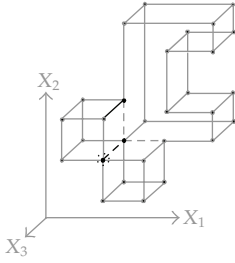
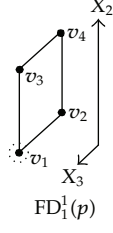
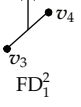

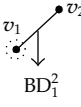

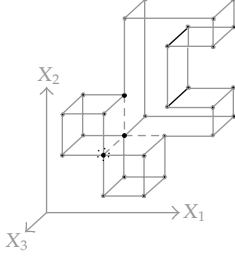
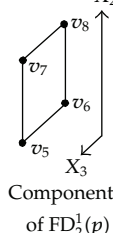
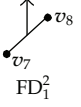

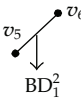

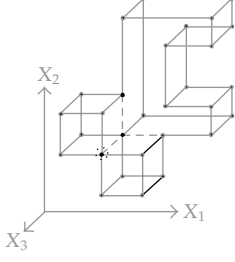
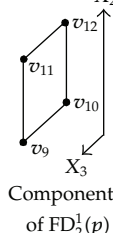
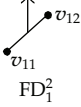

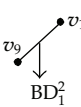

Table 5: Identifying faces, of the 3D-OPP from Figure 5, through backward differences and Algorithm 4 (see text for details. Faces' vertices are listed in the order they are discovered).

	Backward difference	Connected components labeling	Discovered vertices and faces
<p>A 3D-OPP is shown in a coordinate system with axes X_1, X_2, and X_3. To its right is a 2D graph representing the backward difference $BD_1^1(p)$, which consists of a square with an additional vertex and edges extending from the top and right sides.</p>	<p>A graph representing Component 1, consisting of a vertical edge between vertices v_2 and v_{25}, and a path of vertices v_2, v_{26}, v_{27} connected by edges.</p>	$v_{21}, v_{22}, v_{23}, v_{24}$ $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}, \{v_{13}, v_{14}, v_{15}, v_{16}\}, \{v_{17}, v_{18}, v_{19}, v_{20}\}, \{v_{21}, v_{22}, v_{23}, v_{24}\}\}$	
<p>The same 3D-OPP is shown. To its right is a 2D graph representing $BD_2^1(p)$, which is a more complex graph with multiple vertices and edges, including a path of vertices v_1, v_2, v_{28}, v_{29}.</p>	<p>Component 1</p> <p>Component 2</p> <p>Component 1 graph: A vertical edge between v_2 and v_{25}, and a path v_2, v_{26}, v_{27}.</p> <p>Component 2 graph: A path of vertices v_1, v_2, v_{28}, v_{29}.</p>	$v_2, v_{25}, v_{26}, v_{27}$ (Vertex v_2 was discovered when processing $FD_1^1(p)$) $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{27}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}, \{v_{13}, v_{14}, v_{15}, v_{16}\}, \{v_{17}, v_{18}, v_{19}, v_{20}\}, \{v_{21}, v_{22}, v_{23}, v_{24}\}, \{v_2, v_{25}, v_{26}, v_{27}\}\}$ v_1, v_2, v_{28}, v_{29} (Vertices v_1 and v_2 were discovered when processing $FD_1^1(p)$) $V(p) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{27}, v_{28}, v_{29}\}$ $F(p) = \{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_{11}, v_{12}\}, \{v_{13}, v_{14}, v_{15}, v_{16}\}, \{v_{17}, v_{18}, v_{19}, v_{20}\}, \{v_{21}, v_{22}, v_{23}, v_{24}\}, \{v_2, v_{25}, v_{26}, v_{27}\}, \{v_1, v_2, v_{28}, v_{29}\}\}$	

face. Such a vertex (v_2), in fact, was previously identified when processing $FD_1^1(p)$ (Table 4). nonextreme vertex v_1 is present in the second component: it was also identified when $FD_1^1(p)$ was processed. The remaining vertices in both components, and the faces they define, are appended to their corresponding lists.

Each one of the 2D components in the forward and backward differences of p is used as input for the recursive calls of Algorithm 4. These calls have the objective of identifying edges perpendicular to X_2 -axis. Table 6 presents the way Algorithm 4 discovers of some of these edges. When $FD_1^1(p)$ is sent as input its corresponding 1D forward and backward differences, perpendicular to X_2 -axis, are computed. Two edges are identified: one is defined by vertices v_3 and v_4 (an odd edge), while the other is defined by vertices v_1 and v_2 . Edge $\{v_1, v_2\}$ is one of the three even edges that 3D-OPP p has. According to previous sections,

Table 6: Identifying some edges, of the 3D-OPP from Figure 5, through Algorithm 4 (see text for details).

	Input 2D-OPP	1D Forward and backward differences	Connected components labeling	Discovered edges
				$\{v_3, v_4\}$ $E(p) = \{\{v_3, v_4\}\}$
				$\{v_1, v_2\}$ (even edge) $E(p) = \{\{v_3, v_4\}, \{v_1, v_2\}\}$
				$\{v_7, v_8\}$ $E(p) = \{\{v_3, v_4\}, \{v_1, v_2\}, \{v_7, v_8\}\}$
				$\{v_5, v_6\}$ $E(p) = \{\{v_3, v_4\}, \{v_1, v_2\}, \{v_7, v_8\}, \{v_5, v_6\}\}$
				$\{v_{11}, v_{12}\}$ $E(p) = \{\{v_3, v_4\}, \{v_1, v_2\}, \{v_7, v_8\}, \{v_5, v_6\}, \{v_{11}, v_{12}\}\}$
				$\{v_9, v_{10}\}$ $E(p) = \{\{v_3, v_4\}, \{v_1, v_2\}, \{v_7, v_8\}, \{v_5, v_6\}, \{v_{11}, v_{12}\}, \{v_9, v_{10}\}\}$

nonextreme vertices are ignored by the EVM. Moreover, when brinks are obtained from an EVM, even edges remain unidentified, because they are not part of brinks. Tables 4 and 6 exemplify situations where these types of former boundary elements are obtained after computing forward and backward differences.

When considering nD orthogonal polytopes, $n \geq 4$, Algorithm 4 is clearly capable of identifying nonextreme vertices and even edges. Furthermore, it is capable of discovering those kD boundary elements that cannot be described via brinks. Consider the 4D-OPP p

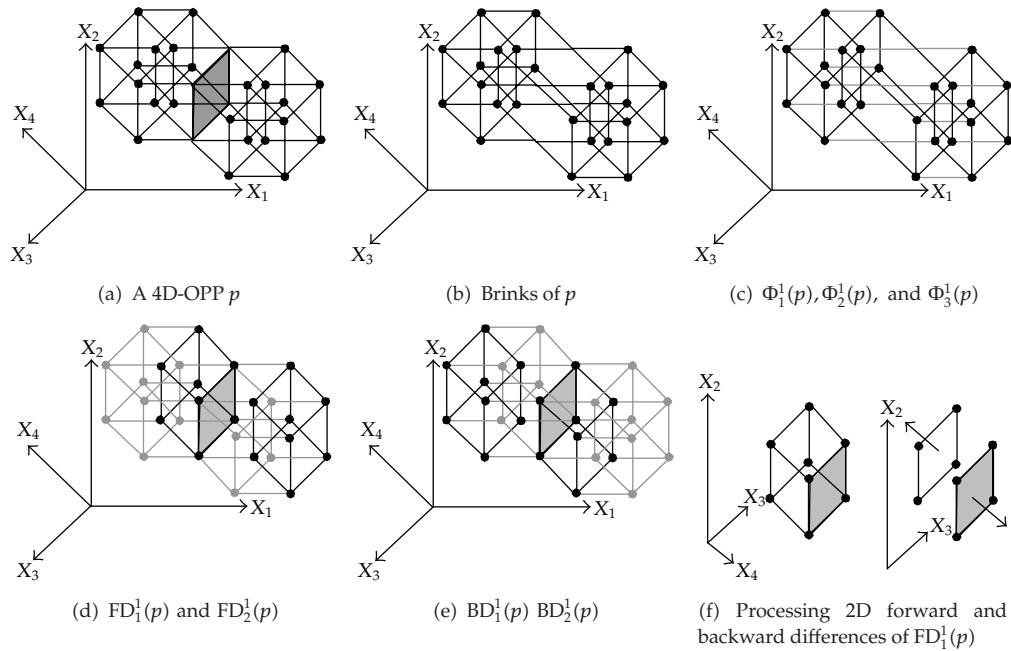


Figure 6: Discovery of a specific face in a 4D-OPP via Algorithm 4 (see text for details).

presented in Figure 6(a). Such polytope can be seen as two hypercubes sharing a face (which is shaded in gray). The face is composed by four nonextreme vertices and four even edges. When p 's brinks, parallel to X_1 to X_4 -axes, are obtained (Figure 6(b)), the shared face's elements are not present, because the 4D-EVM only stores extreme vertices which in time define brinks which are formed only by odd edges. Figure 6(c) presents the three 3D couplets, perpendicular to X_1 -axis, of p . Each couplet is formed by a volume. The three volumes appear to intersect, but this is an effect of the 4D-3D-2D projections used. Figure 6(d) presents p 's two 3D forward differences perpendicular to X_1 -axis. When processing $FD_1^1(p)$, Algorithm 4 detects a 3D boundary volume defined by eight vertices: four of them correspond to extreme vertices of p , while the other four correspond to the vertices of the shared face: such vertices did not belong to $EVM_4(p)$. These four vertices are detected again when processing backward difference $BD_2^1(p)$ (see Figure 6(e)). Three-dimensional forward difference $FD_1^1(p)$ is used, in a recursive call, as input for Algorithm 4 (see Figure 6(f)). When its 2D forward and backward differences, perpendicular to X_4 -axis, are processed and specifically when processing its only backward difference, the shared face is formerly discovered and stored in the list of p 's boundary faces. The edges of such face are formerly discovered when this face is then sent as input, in a new recursive call, of Algorithm 4.

We conclude this section by mentioning that it is important to take into account the coordinates ordering in an EVM determines which boundary elements are identified. Suppose that we use the sorting $X_1X_4X_2X_3$. Algorithm 2 will extract forward and backward differences perpendicular to X_1 -axis. Then, in turn, Algorithm 4 shares to identify 3D boundary volumes perpendicular precisely to X_1 -axis (see Figures 6(d) and 6(e)). When each one of these volumes is sent as input in the corresponding recursive calls, we have input 3D-EVMs sorted as $X_4X_2X_3$. Hence, there are identified, again, because of the use of Algorithm 2, faces perpendicular to X_4 -axis. In this phase, we are working with volumes

embedded in a 3D space, because of the use of projection operator which has removed the X_1 -coordinate. But, when the source 4D Space is taken into account, it can be determined, in fact, such faces are perpendicular to the plane described by X_1 and X_4 axes, because they precisely belong to volumes perpendicular to X_1 -axis (see Figure 6(f)). When each face, expressed as a 2D-EVM, is sent again as input, we identify edges perpendicular to X_2 -axis, formally, edges perpendicular to the 3D hyperplane described by X_1 , X_4 , and X_2 axes. Some applications could require to have access to boundary elements which cannot be identified via a given coordinates ordering. In this sense, a new sorting of the corresponding EVM is required, but it can be achieved in efficient time respect to the number of extreme vertices in the corresponding EVM. By this way, for example, coordinates sorting $X_3X_4X_1X_2$ will share to identify 3D volumes perpendicular to X_3 -axis, faces perpendicular to the plane X_3X_4 , and edges perpendicular to the 3D hyperplane $X_3X_4X_1$.

6. An Experimental Time Complexity Analysis for Boundary Extraction under the n D-EVM

The following key points define the conditions over which the execution time of Algorithm 4 was measured.

- (i) The units for the time measures are given in nanoseconds.
- (ii) The evaluations were performed with a CPU Intel Core Duo (2.40GHz) and 2 Gigabytes in RAM.
- (iii) Our algorithms were implemented using the Java Programming Language under the Software Development Kit 1.6.0.
- (iv) Our testing consider $n = 2, 3, 4, 5$.
- (v) For each n , we have generated 10,000 random n D-OPPs. Each generated n D-OPP g was, respectively, expressed as $EVM_n(g)$ and sent as input to Algorithm 4.
- (vi) In the 2D, 3D, 4D, and 5D cases, the considered coordinates ordering were X_1X_2 , $X_1X_2X_3$, $X_1X_2X_3X_4$, and $X_1X_2X_3X_4X_5$, respectively. This implies that, as specified in Section 5, we identified first $(n - 1)$ D boundary elements perpendicular to X_1 -axis, then we identified $(n - 2)$ D boundary elements perpendicular to X_2 -axis, and so on.
- (vii) Once the generation of n D-OPPs has finished and the algorithm was evaluated, we proceed with a statistical analysis in order to find a trendline of the form $t = ax^b$, where $x = \text{Card}(EVM_n(g))$, that fits as good as possible to our measures in order to provide an estimation of the temporal complexity of Algorithm 4 for each value of n . The quality of the approximation curve is assured by computing the coefficient of determination R^2 .

Table 7 shows some statistics related to our generated polytopes. In Figure 7 the behavior of Algorithm 4 with our testing sets of n D-OPPs can be visualized. In the same chart the corresponding trendline for each value of n whose associated equations are shown in Table 8 can be also appreciated.

We can then observe in the obtained trendlines (Table 8) that the exponents associated with the number of vertices vary between 1 and 1.5. These experimentally identified complexities for our Algorithm 4 provide us elements to determine its temporal efficiency when we extract boundary elements from an n D-OPP expressed as an EVM.

Table 7: Some statistical characteristics of the sets of random n D-OPPs for testing of Algorithm 4.

n	Max card (EVM $_n(g)$)	Min card (EVM $_n(g)$)	Median card (EVM $_n(g)$)	Average card (EVM $_n(g)$)	Standard deviation card (EVM $_n(g)$)
2	7,016	4	3,682	3,633.35	2,004.88
3	6,742	0	5,334	4,695.53	1,772.10
4	6,830	0	5,362	4,727.48	1,781.45
5	7,310	0	5,392	4,825.45	1,876.69

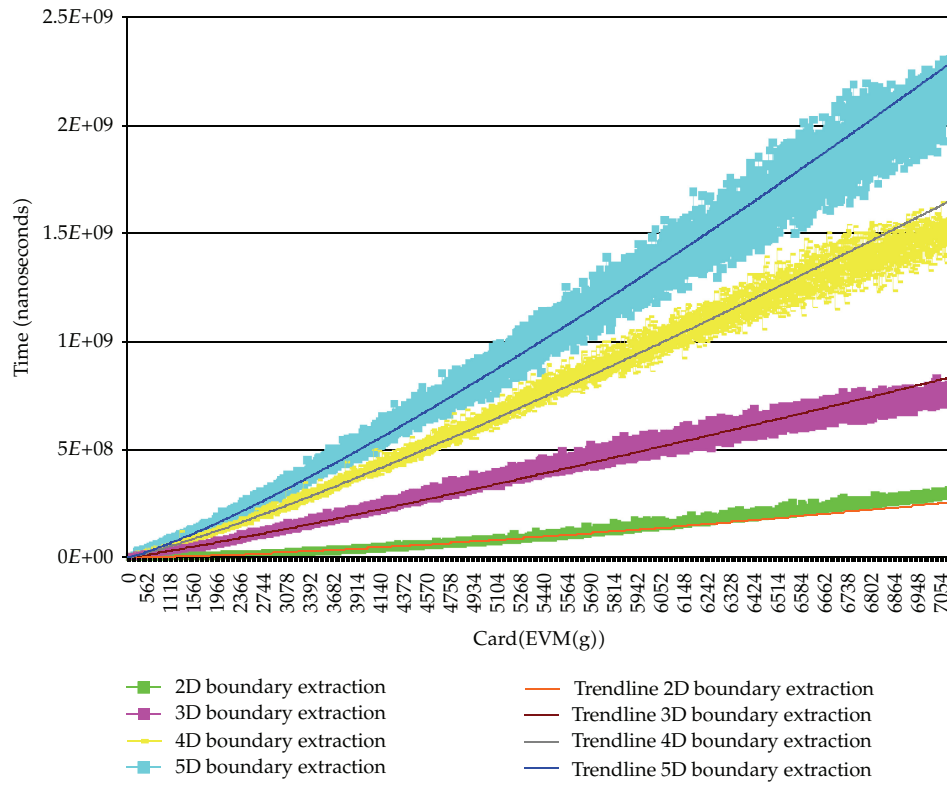


Figure 7: Execution times of Algorithm 4 for 2D, 3D, 4D, and 5D-OPPs and their corresponding trendlines.

Table 8: Equations associated to the trendlines that describe execution time of Algorithm 4 in the cases with $n = 2, 3, 4, 5$.

n	Trendline $t = ax^b$	a	b	R^2
2	$t = 484.02x^{1.4313}$	484.02	1.4313	0.9866
3	$t = 27,391x^{1.1214}$	27,391	1.1214	0.9962
4	$t = 33,300x^{1.1744}$	33,300	1.1744	0.9835
5	$t = 34,957x^{1.2047}$	34,957	1.2047	0.9658

7. Concluding Remarks and Future Work

In this work, we have provided two contributions: an algorithm for computing connected components labeling and an algorithm for extracting boundary elements. The procedures deal with the dominion of the n -dimensional orthogonal pseudo-polytopes, and they are specified in terms of the extreme vertices model in the n -dimensional space. As seen before, both algorithms are sustained in the basic methodologies provided by the nD -EVM. As commented in Section 2, some of such basic algorithms in time are sustained in the *MergeXor* algorithm. It computes the regularized XOR between two nD -OPPs expressed in the EVM but in terms of the ordinary XOR operation. This implies *MergeXor*, as its name indicates, is implemented as a merging procedure that only takes in account those extreme vertices present in the first or second input EVMs but not in both. Therefore, it can be specified as a merging-like procedure with an execution linear time given by the sum of the cardinalities of the input EVMs [1]. On one hand, this has as a consequence the temporal efficiency of the basic EVM algorithms. On the other hand, as seen in Section 6 and from an empirical point of view, we have elements to sustain the efficiency of the algorithms provided here.

As part of our future work, and particularly, immediate applications for Algorithms 3 and 4, we comment that we will attack the problem related to the automatic classification of video sequences. In [23], we describe a methodology for representing 2D color video sequences as 4D-OPPs embedded in a 4D color-space-time geometry. Our idea is the use of geometrical and topological properties, such as boundary descriptions, connected components, and discrete compactness, in order to determine similarities and differences between sequences and for instance to classify them. In [5], part of this work has been boarded using only discrete compactness, and the obtained results are promising. We are going to incorporate the information provided by Algorithms 3 and 4 in order to determine if they can contribute to enhance the results.

References

- [1] A. Aguilera, *Orthogonal polyhedra: study and application*, Ph.D. thesis, Universitat Politècnica de Catalunya, 1998.
- [2] A. Aguilera and D. Ayala, "Orthogonal polyhedra as geometric bounds in constructive solid geometry," in *Proceedings of the 4th ACM Siggraph Symposium on Solid Modeling and Applications (SM '97)*, pp. 56–67, ACM, Atlanta, Ga, USA, 1997.
- [3] R. Pérez-Aguila, *Orthogonal polytopes: study and application*, Ph.D. thesis, Universidad de las Américas-Puebla (UDLAP), 2006.
- [4] R. Pérez-Aguila, "Modeling and manipulating 3D datasets through the extreme vertices model in the n -dimensional space (nD -EVM)," *Research in Computer Science*, vol. 31, pp. 15–24, 2007.
- [5] R. Pérez-Aguila, "Computing the discrete compactness of orthogonal pseudo-polytopes via their n -EVM representation," *Mathematical Problems in Engineering*, vol. 2010, Article ID 598910, 28 pages, 2010.
- [6] J. Rodríguez and D. Ayala, "Erosion and dilation on 2D and 3D digital images: a new size-independent approach," *Vision Modeling and Visualization*, pp. 143–150, 2001.
- [7] J. Rodríguez and D. Ayala, "Fast neighborhood operations for images and volume data sets," *Computers and Graphics*, vol. 27, no. 6, pp. 931–942, 2003.
- [8] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, vol. 13, no. 4, pp. 471–494, 1966.
- [9] S. Marchand-Maillet and Y. M. Sharaiha, *Binary Digital Image Processing*, Academic Press Inc., San Diego, Calif, USA, 2000.
- [10] H. Samet and M. Tamminen, "efficient component labeling of images of arbitrary dimension represented by linear bintrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 579–586, 1988.

- [11] K. Sandfort and J. Ohser, "Labeling of n -dimensional images with choosable adjacency of the pixels," *Image Analysis & Stereology*, vol. 28, no. 1, pp. 45–61, 2009.
- [12] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," in *Progress in Biomedical Optics and Imaging*, vol. 5747 of *Proceedings of SPIE*, no. 3, pp. 1965–1976, 2005.
- [13] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding*, vol. 89, no. 1, pp. 1–23, 2003.
- [14] H. Samet, "Connected component labeling using quadtrees," *Journal of the Association for Computing Machinery*, vol. 28, no. 3, pp. 487–501, 1981.
- [15] A. A. Requicha, "Representations for rigid solids: theory, methods, and systems," *Computing Surveys*, vol. 12, no. 4, pp. 437–464, 1980.
- [16] H. Hansen and N. Christensen, "A model for n -dimensional boundary topology," in *Proceedings of the 2th Symposium on Solid Modeling and Applications*, pp. 65–73, Montreal, Canada, 1993.
- [17] L. K. Putnam and P. A. Subrahmanyam, "Boolean operations on n -dimensional objects," *IEEE Computer Graphics and Applications*, vol. 6, no. 6, pp. 43–51, 1986.
- [18] M. Spivak, *Calculus on Manifolds. A Modern Approach to Classical Theorems of Advanced Calculus*, W. A. Benjamin, Inc., New York, NY, USA, 1965.
- [19] G. McCarty and G. S. Yound, *Topology*, Dover Publications Inc., New York, NY, USA, 2nd edition, 1988.
- [20] G. L. Naber, *Topological Methods in Euclidean Spaces*, Dover Publications Inc., Mineola, NY, USA, 2000.
- [21] A. Kolcun, "Visibility criterion for planar faces in 4D," in *Proceedings of Spring Conference on Computer Graphics (SCCG '04)*, pp. 216–219, Budmerice, Slovakia, 2004.
- [22] T. Takala, "A taxonomy on geometric and topological models," *Computer Graphics and Mathematics*, pp. 146–171, 1992.
- [23] R. Pérez-Aguila, "Representing and visualizing vectorized videos through the extreme vertices model in the n -dimensional Sspace (nD-EVM)," *Journal Research in Computer Science*, vol. 29, pp. 65–80, 2007.



Hindawi

Submit your manuscripts at
<http://www.hindawi.com>

