Efficient Computation of Regularized Boolean Operations on the Extreme Vertices Model in the n-Dimensional Space (nD-EVM)

Ricardo Pérez-Aguila
Instituto de Electrónica y Computación (IEC)
Universidad Tecnológica de la Mixteca (UTM)
ricardo.perez.aguila@gmail.com

Abstract
The objective behind this work is to describe the Extreme Vertices Model in the n-Dimensional Space (nD-EVM) and the way it represents nD Orthogonal Pseudo Polytopes (nD-OPPs) by considering only a subset of their vertices: the Extreme Vertices. There are presented the Regularized Boolean Operators and the way they assure the dimensional homogeneity. Algorithms for performing Regularized Boolean Operations under the nD-EVM will be presented and details about an experimental time complexity analysis will be given in order to sustain the efficiency shared by the model when these operations take place through the described procedures. Finally, it is shown, as an example of application, an alternative representation for 3D datasets (voxelizations). The proposal establishes to model datasets as four-dimensional polytopes where the fourth dimension corresponds to color. Then, it will be shown how the 4D representation is concisely expressed through the EVM.

Keywords
n-Dimensional Polytopes, Computational Geometry, Geometrical and Topological Modeling, Polytopes Representation Schemes.

1. Introduction
The objective behind this work is to describe the Extreme Vertices Model in the n-Dimensional Space (nD-EVM) and the way it represents nD Orthogonal Pseudo Polytopes (Section 2) by considering only a subset of their vertices: the Extreme Vertices (Section 3). The Extreme Vertices Model (3D-EVM) was originally presented, and widely described, by Aguilera & Ayala for representing 2-manifold Orthogonal Polyhedra [1] and later considering both Orthogonal Polyhedra (3D-OPs) and Pseudo-Polyhedra (3D-OPPs) [2]. This model has enabled the development of simple and robust algorithms for performing the most usual and demanding tasks on solid modeling, such as closed and regularized Boolean operations, solid splitting, set membership classification operations and measure operations on 3D-OPPs. It is natural to ask if the EVM can be extended for modeling n-Dimensional Orthogonal Pseudo-Polytopes (nD-OPPs). In this sense, some experiments were made, by Pérez-Aguila & Aguilera [8], where the validity of the model was assumed true in order to represent 4D and 5D-OPPs. Finally, in [9] was formally proved that the nD-EVM is a complete scheme for the representation of nD-OPPs. The meaning of complete scheme was based in Requicha's set of formal criteria that every
scheme must have rigorously defined: Domain, Completeness, Uniqueness and Validity. Although the EVM of an nD-OPP has been defined as a subset of the nD-OPPs vertices, there is much more information about the polytope hidden within this subset of vertices. There will be described basic procedures and algorithms in order to obtain this information (Sections 3.2, 5 and 6).

The Section 7 is oriented to describe the way the nD-EVM performs some of the key operations between polytopes: Closed and Regularized Boolean Operations. These operations provide a way for combining polytopes in order to compose new ones. There are presented the Regularized Boolean Operators and the way they assure the dimensional homogeneity (Section 4). In Section 7.1, an algorithm for performing Regularized Boolean Operations under the nD-EVM will be described, and in Section 7.2, details about an experimental time complexity analysis, originally presented in [9] & [11], will be given in order to sustain the efficiency shared by the nD-EVM when these operations take place through the described procedures.

Finally, it is presented in Section 8, as an example of application, an alternative representation for 3D datasets (voxelizations). The proposal, originally described in [12], establishes to model datasets as four-dimensional polytopes where the fourth dimension corresponds to color. Then, it will be shown how the 4D representation is concisely expressed through the EVM.

2. The n-Dimensional Orthogonal Pseudo-Polytopes

Definition 2.1: A Singular n-Dimensional Hyper-Box in \( \mathbb{R}^n \) is the continuous function

\[ I^n : [0,1]^n \rightarrow [0,1]^n \]

\[ x \sim I^n(x) = x \]

Definition 2.2: For all \( i, 1 \leq i \leq n \), the two singular (n-1)D hyper-boxes \( I^n_{(i,0)} \) and \( I^n_{(i,1)} \) are defined as follows: If \( x \in [0,1]^{n-1} \) then

- \( I^n_{(i,0)}(x) = I^n(x_1,\ldots,x_{i-1},0,x_i,\ldots,x_{n-1}) = (x_1,\ldots,x_{i-1},0,x_i,\ldots,x_{n-1}) \)
- \( I^n_{(i,1)}(x) = I^n(x_1,\ldots,x_{i-1},1,x_i,\ldots,x_{n-1}) = (x_1,\ldots,x_{i-1},1,x_i,\ldots,x_{n-1}) \)

Definition 2.3: In a general singular nD hyper-box \( c \) the \( (i,\alpha) \)-cell is defined as

\[ c_{(i,\alpha)} = c \circ I^n_{(i,\alpha)} \]

Definition 2.4: The orientation of a cell \( c \circ I^n_{(i,\alpha)} \) is given by \((-1)^{\alpha+i} \).

Definition 2.5: An (n-1)D oriented cell is given by the scalar-function product

\[ (-1)^{i+\alpha} \cdot c \circ I^n_{(i,\alpha)} \]

Definition 2.6: A formal linear combination of singular general kD hyper-boxes, \( 1 \leq k \leq n \), for a closed set \( A \) is called a k-chain.
Definition 2.7 [15]: Given a singular nD hyper-box \( I^n \) the (n-1)-chain, called the boundary of \( I^n \), is defined by
\[
\partial(I^n) = \sum_{i=1}^{n} \left( \sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot I_{(i,\alpha)}^n \right)
\]

Definition 2.8 [15]: Given a singular general nD hyper-box \( c \) the (n-1)-chain, called the boundary of \( c \), is defined as
\[
\partial(c) = \sum_{i=1}^{n} \left( \sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot c \circ I_{(i,\alpha)}^n \right)
\]

Definition 2.9 [15]: The boundary of an n-chain \( \sum c_i \), where each \( c_i \) is a singular general nD hyper-box, is given by
\[
\partial\left(\sum c_i\right) = \sum \partial(c_i)
\]

Definition 2.10: A collection \( c_1, c_2, \ldots, c_k \), 1 \( \leq k \leq 2^n \), of general singular nD hyper-boxes is a combination of nD hyper-boxes if and only if
\[
\left[ \bigcap_{\alpha=1}^{\delta} c_{\alpha}([0,1]^n) = (0,\ldots,0) \right] \wedge \left( \forall i, j, 1 \leq i, j \leq k \right) \left( c_i([0,1]^n) \neq c_j([0,1]^n) \right)
\]

In the above definition the first part of the conjunction establishes that the intersection between all the nD general singular hyper-boxes is the origin, while the second part establishes that there are not overlapping nD hyper-boxes.

Definition 2.11: An n-Dimensional Orthogonal Pseudo-Polytope \( p \), or just an nD-OPP \( p \), will be an n-chain composed by nD hyper-boxes arranged in such way that by selecting a vertex, in any of these hyper-boxes, it describes a combination of nD hyper-boxes (Definition 2.10) composed up to \( 2^n \) hyper-boxes.

Describing nD-OPPs as union of disjoint nD hyper-boxes is very useful because in the following propositions there will be considered geometrical and/or topological local analysis over such vertices and their respective incident hyper-boxes.

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

Definition 3.1: Let \( c \) be a combination of hyper-boxes in the nD space. An Odd Edge will be an edge with an odd number of incident hyper-boxes of \( c \).

Definition 3.2: 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.

Definition 3.3: The Extreme Vertices of an nD-OPP \( p \) are the ending vertices of all the brink in \( p \).
The brinks in an nD-OPP p can be classified according to the main axis to which they are parallel. Since the extreme vertices mark the end of brinks in the n orthogonal directions, is that any of the n possible sets of brinks parallel to Xi-axis, 1 ≤ i ≤ n, produce to the same set EV(p).

**Theorem 3.1** [9]: A vertex of an nD-OPP p, n ≥ 1, when is locally described by a set of surrounding nD hyper-boxes, is an extreme vertex if and only if it is surrounded by an odd number of such nD hyper-boxes.

**Theorem 3.2** [9]: Any extreme vertex of an nD-OPP, n ≥ 1, when is locally described by a set of surrounding nD hyper-boxes, has exactly n incident linearly independent odd edges.

**Definition 3.4:** Let p be an nD-OPP. A kD couplet of p, 1 < k < n, is the maximal set of kD cells of p that lies in a kD space, such that a kD cell e0 belongs to a kD extended hypervolume if and only if e0 belongs to an (n-1)D cell present in ∂p).

Let Q be a finite set of points in \( \mathbb{R}^3 \). In [2] was defined the ABC-sorted set of Q as the set resulting from sorting Q according to coordinate A, then to coordinate B, and then to coordinate C. For instance, a set Q can be ABC-sorted is six different ways: \( X_1X_2X_3 \), \( X_1X_3X_2 \), \( X_2X_1X_3 \), \( X_2X_3X_1 \), \( X_3X_1X_2 \) and \( X_3X_2X_1 \). Now, let p be a 3D-OPP. According to [2] the Extreme Vertices Model of p, EVM(p), denotes to the ABC-sorted set of the extreme vertices of p. Then EVM(p) = EV(p) except by the fact that EV(p) is not necessarily sorted. In this work we will assume that the coordinates of extreme vertices in the Extreme Vertices Model of an nD-OPP p, EVM_n(p) are sorted according to coordinate X_1, then to coordinate X_2, and so on until coordinate X_n. That is, we are considering the only ordering \( X_1...X_{i-1}X_i...X_n \) such that \( i-1 < i, 1 < i ≤ n \).

**Definition 3.5:** Let p be an nD-OPP. The Extreme Vertices Model of p, denoted by EVM_n(p), is defined as the model that stores to all the extreme vertices of p.

**3.1 Sections and Slices of nD-OPPs**

**Definition 3.6:** The Projection Operator for (n-1)D cells, points, and set of points is respectively defined as follows:

- Let \( c(I_{i,a_i}^n(x)) = (x_1, ..., x_n) \) be an (n-1)D cell embedded in the nD space. \( \pi_j(c(I_{i,a_i}^n(x))) \)

  will denote the projection of the cell \( c(I_{i,a_i}^n(x)) \) onto an (n-1)D space embedded in nD space whose supporting hyperplane is perpendicular to Xi-axis:

  \[ \pi_j(c(I_{i,a_i}^n(x))) = (x_1, ..., \hat{x}_j, ..., x_n) \]

- Let \( v = (x_1, ..., x_n) \) 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: \( \pi_j(v) = (x_1, ..., \hat{x}_j, ..., x_n) \)

- Let Q be a set of points in \( \mathbb{R}^n \). The projection of the points in Q, denoted by \( \pi_j(Q) \), is given by 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 cases \( \hat{x}_j \) is the coordinate corresponding to Xi-axis to be suppressed.
Definition 3.7: Consider an nD-OPP p:

- Let \( np_i \) be the number of distinct coordinates present in the vertices of p along \( X_i \)-axis, \( 1 \leq i \leq n \).
- Let \( \Phi_i^j(p) \) be the \( k \)-th \((n-1)\)D couplet of p which is perpendicular to \( X_i \)-axis, \( 1 \leq k \leq np_i \).

Definition 3.8: A Slice is the region contained in an nD-OPP p between two consecutive couplets of p. \( \text{Slice}_i^k(p) \) will denote to the \( k \)-th slice of p which is bounded by \( \Phi_i^j(p) \) and \( \Phi_i^{k+1}(p) \), \( 1 \leq k < np_i \).

Definition 3.9: 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 \( X_i \), \( 1 \leq i \leq n \), which not coincide with any \((n-1)\)-couplet of p. A section will be called external or internal section of p if it is empty or not, respectively. \( \text{Section}_i^k(p) \) will refer to the \( k \)-th section of p between \( \text{Slice}_i^k(p) \) and \( \text{Slice}_{i+1}^k(p) \), \( 1 \leq k < np_i \).

3.2 Computing Couplets and Sections

Theorem 3.3 [9]: The projection of the set of \((n-1)\)D-couplets, \( \pi_i(\Phi_i^j(P)) \), of an nD-OPP P, can be obtained by computing the regularized XOR (\( \otimes \)) between the projections of its previous \( \pi_i(S_{i-1}^k(P)) \) and next \( \pi_i(S_i^k(P)) \) sections, i.e.,

\[
\pi_i(\Phi_i^j(P)) = \pi_i(S_{i-1}^k(P)) \otimes * \pi_i(S_i^k(P)), \ \forall k \in [1, np_i]
\]

Theorem 3.4 [9]: The projection of any section, \( \pi_i(S_i^k(p)) \), of an nD-OPP p, can be obtained by computing the regularized XOR between the projection of its previous section, \( \pi_i(S_{i-1}^k(P)) \), and the projection of its previous couplet \( \pi_i(\Phi_i^j(P)) \).

4. Regularized Boolean Operations

Independently of the scheme we consider for the representation of nD polytopes, it should be feasible to combine them in order to compose new objects [4]. One of the most common methods to combine polytopes are the set theoretical Boolean operations, as the union, difference, intersection and exclusive OR. However, the application of an ordinary set theoretical Boolean operation on two polytopes does not necessarily produce a polytope. For example, the ordinary intersection between two cubes with only a common vertex is a point. Instead of using ordinary set theoretical Boolean operators, the Regularized Boolean Operators ([13] & [14]) will be used. The practical purpose of regularization of polytope models is to make them dimensionally homogeneous [16]. The regularization operation can be defined as Regularized(S) = Closure(Interior(S)) which results in a closed regular set.

---

Ricardo Pérez-Aguila, August 2009
Each regularized Boolean operator is defined in function of an ordinary operator in the following way: \( A \odot^* B = \text{Closure} \left( \text{Interior}(A \odot B) \right) \). In such way we will have:

\[
\begin{align*}
A \cup^* B &= \text{Closure} \left( \text{Interior}(A \cup B) \right) & \text{Regularized Union} \\
A \cap^* B &= \text{Closure} \left( \text{Interior}(A \cap B) \right) & \text{Regularized Intersection} \\
A \otimes^* B &= \text{Closure} \left( \text{Interior}(A \otimes B) \right) & \text{Regularized Exclusive OR} \\
A \, \neg^* B &= \text{Closure} \left( \text{Interior}(A \, \neg B) \right) & \text{Regularized Difference}
\end{align*}
\]

These operators are defined as the closure of the interior of the corresponding set theoretical Boolean operation ([6] & [14]). In this way, the regularized operations between polytopes always will generate polytopes [16] or a null object (the empty set). Recapturing the previous example, the regularized intersection between two cubes with a common vertex is precisely the null object.

5. The Regularized XOR operation on the nD-EVM

**Theorem 5.1** [2]: Let \( p \) and \( q \) be two \( n \)-D-OPPs having \( \text{EVM}_n(p) \) and \( \text{EVM}_n(q) \) as their respective EVMs in \( n \)-D space, then

\[
\text{EVM}_n(p \otimes^* q) = \text{EVM}_n(p) \otimes \text{EVM}_n(q).
\]

This result allows expressing a formula for computing \( n \)-D-OPPs sections from couplets and vice-versa, by means of their corresponding Extreme Vertices Models. These formulae are obtained by combining **Theorem 5.1** with **Theorem 3.3**; and **Theorem 5.1** with **Theorem 3.4**, respectively:

**Corollary 5.1** [2]: \( \text{EVM}_{n-1}(\pi_i(\Phi_{n-1}^i(p))) = \text{EVM}_{n-1}(\pi_i(S_{n-1}^i(p))) \otimes \text{EVM}_{n-1}(\pi_i(S_{n-1}^i(p))) \)

**Corollary 5.2** [2]: \( \text{EVM}_{n-1}(\pi_i(S_{n-1}^i(p))) = \text{EVM}_{n-1}(\pi_i(S_{n-1}^i(p))) \otimes \text{EVM}_{n-1}(\pi_i(\Phi_{n-1}^i(p))) \)

Consider the following useful result that provides a way for expressing a specific case of the Regularized Union by means of Regularized XOR:

**Corollary 5.3** [2]: Let \( p \) and \( q \) be two disjoint or quasi disjoint \( n \)-D-OPPs having \( \text{EVM}_n(p) \) and \( \text{EVM}_n(q) \) as their respective Extreme Vertices Models, then

\[
\text{EVM}_n(p \cup^* q) = \text{EVM}_n(p) \otimes \text{EVM}_n(q)
\]

6. Basic Algorithms for the nD-EVM

According to **Sections 3** and **4** it is possible to define the following primitive operations which are based in the functions originally presented in [2]:

**Output**: An empty \( n \)-D-EVM.

**Procedure** \texttt{InitEVM}()

\{
 \text{Returns the empty set.}
\}
Function MergeXOR performs an XOR between two nD-EVMs, that is, it keeps all vertices belonging to either EVM\textsubscript{n}(p) or EVM\textsubscript{n}(q) and discards any vertex that belongs to both EVM\textsubscript{n}(p) and EVM\textsubscript{n}(q). Since the model is sorted, this function consists on a simple merging-like algorithm, and therefore, it runs on linear time [2]. Its complexity is given by O(Card(EVM\textsubscript{n}(p)) + Card(EVM\textsubscript{n}(q))) since each vertex from EVM\textsubscript{n}(p) and EVM\textsubscript{n}(q) needs to be processed just once. Moreover, according to Theorem 5.1, the resulting set corresponds to the regularized XOR operation between p and q.

\begin{algorithm}
\textbf{Algorithm 6.1. Computing }EVM\textsubscript{n-1}(\pi_1(S'_1(p)))\textbf{ as }EVM\textsubscript{n-1}(\pi_1(S'_{k-1}(p))) \otimes EVM\textsubscript{n-1}(\pi_1(\Phi'_1(p)))\textbf{.}
\begin{itemize}
\item \textbf{Input:} An (n-1)D-EVM corresponding to section S.
\item \textbf{Output:} An (n-1)D-EVM corresponding to couplet hvl.
\end{itemize}
\item Procedure GetSection(EVM S, EVM hvl)
// Returns the projection of the next section
// of an nD-OPP whose previous section is S.
return MergeXOR(S, plv)
\end{algorithm}
**Algorithm 6.2.** Computing $EVM_{n-1}\left(\pi_i\left(\Phi_k\left(p\right)\right)\right)$ as $EVM_{n-1}\left(\pi_i\left(S_{k,i}^i\left(p\right)\right)\right)\otimes EVM_{n-1}\left(\pi_i\left(S_{k,j}^i\left(p\right)\right)\right)$

**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 $S_j$)
// Returns the projection of the couplet between consecutive sections $S_i$ and $S_j$.  
return MergeXOR($S_i$, $S_j$)
end-of-procedure

Algorithm 6.3. Computing the sequence of sections from an nD-OPP represented through the nD-EVM.

**Input:** An nD-EVM $p$.

**Procedure** EVM_to_SectionSequence(EVM $p$)
EVM hvl  // Current couplet.  
EVM $S_i$, $S_j$  
hvl = InitEVM( )  
$S_i$ = InitEVM( )  
$S_j$ = InitEVM( )  
hvl = ReadHvl(p)
while (Not(EndEVM($p$)))
  $S_j$ = GetSection($S_i$, hvl)
  Process($S_i$, $S_j$)
  $S_i$ = $S_j$  
hvl = ReadHvl($p$)  // Read next couplet.
end-of-while
end-of-procedure

**Corollary 7.1** [2]: Let $p$ and $q$ be two nD-OPPs and $r = p \text{ op}\ast q$ where $\text{op}\ast$ is in $\{\cup\ast, \cap\ast, \neg\ast, \otimes\ast\}$. Then  
$$\pi_i\left(S_{k}^i(r)\right) = \pi_i\left(S_{k}^i(p)\right) \text{ op}\ast \pi_i\left(S_{k}^i(q)\right)$$

Moreover, if all these sections lie in the same (n-1)D hyperplane then  
$$S_{k}^i(r) = S_{k}^i(p) \text{ op}\ast S_{k}^i(q)$$

Now we present the following

**Theorem 7.1** [2]: A regularized Boolean operation, $\text{op}\ast$, where $\text{op}\ast \in \{\cup\ast, \cap\ast, \neg\ast, \otimes\ast\}$, over two nD-OPPs $p$ and $q$, both expressed in the nD-EVM, can be carried out by means of the same $\text{op}\ast$ applied over their own sections, expressed through their Extreme Vertices Models, which are (n-1)D-OPPs.
This result leads into a recursive process for computing the Regularized Boolean operations using the nD-EVM, which descends on the number of dimensions. The base or trivial case of the recursion is the 1D-Boolean operations which can be performed using direct methods.

Algorithm 7.1. Computing Regularized Boolean Operations on the EVM.

**Input:** The nD-OPPs p and q expressed in the nD-EVM.
The number n of dimensions and the Boolean operation op.
**Output:** The output nD-OPP r, such that r = p op* q, codified as an nD-EVM.

**Procedure** BooleanOperation(EVM p, EVM q, BooleanOperator op, int n)

1. EVM sP, sQ // Current sections of p and q respectively.
2. EVM hvl // I/O couplet.
3. boolean fromP, fromQ // flags for the source of the couplet hvl.
4. CoordType coord // the common coordinate of couplets.
5. EVM r, sRprev, sRcurr // nD-OPP r and two of its sections.
6. If (n = 1) then // Basic case
   return BooleanOperation1D(p, q, op)
7. else
   n = n - 1
   sP = InitEVM( )
   sQ = InitEVM( )
   sRcurr = InitEVM( )
   NextObject(p, q, coord, fromP, fromQ)
   While (Not(EndEVM(p)) and Not(EndEVM(q)))
      If (fromP = true) then
         hvl = ReadHvl(p)
         sP = GetSection(sP, hvl)
      end-of-if
      If (fromQ = true) then
         hvl = ReadHvl(q)
         sQ = GetSection(sQ, hvl)
      end-of-if
      sRprev = sRcurr
      sRcurr = BooleanOperation(sP, sQ, n, op) // Recursive call
      hvl = GetHvl(sRprev, sRcurr)
      SetCoord(hvl, coord)
      PutHvl(hvl, r)
   end-of-while
   while (Not(EndEVM(p)))
      hvl = ReadHvl(p)
      PutBool(hvl, r, op)
   end-of-while
   while (Not(EndEVM(q)))
      hvl = ReadHvl(q)
      PutBool(hvl, r, op)
   end-of-while
   return r
end-of-else
end-of-procedure
7.1. The Boolean Operations Algorithm for the nD-EVM

This section describes the algorithm originally presented in [2] for performing regularized Boolean operations. Let \( p \) and \( q \) be two nD-OPPs represented through the nD-EVM, and let \( \text{op}^* \) be a Boolean operator in \( \{ \cup^*, \cap^*, \neg^*, \otimes^* \} \). The algorithm computes the resulting nD-OPP \( r = p \text{ op}^* q \), and it is based on Theorem 7.1. Note that \( r = p \otimes^* q \) can also be trivially performed using Theorem 5.1. The idea behind the algorithm is the following:

- The sequence of sections from \( p \) and \( q \), perpendicular to \( X_i \)-axis, can be obtained first, based in Theorem 3.4.

- Then, according to Corollary 7.1, every section of \( r \) can recursively be computed as
  \[
  S_i^k(r) = S_i^k(p) \text{ op}^* S_i^k(q).
  \]

- Finally, \( r \) can be obtained from its sequence of sections, perpendicular to \( X_i \)-axis, according to Theorem 3.3.

Nevertheless, Algorithm 7.1 does not work in this sequential form. It actually works in a wholly merged form in which it only needs to store one section for each of the operands \( p \) and \( q \), and two consecutive sections for the result \( r \).

The following are some functions present in Algorithm 7.1 but not defined previously:

- Function \( \text{BooleanOperation1D} \) performs 1D Boolean operations between \( p \) and \( q \) that are two 1D-OPPs.

- Procedure \( \text{NextObject} \) considers both input objects \( p \) and \( q \) and returns the common \textit{coord} value of the next \textit{hvl} to process, using function \( \text{GetCoord} \). It also returns two flags, \textit{fromP} and \textit{fromQ}, which signal from which of the operands (both inclusive) is the next \textit{hvl} to come.

- The main loop of procedure \( \text{BooleanOperation} \) gets couplets from \( p \) and/or \( q \), using function \( \text{GetSection} \). These sections are recursively processed to compute, according to Corollary 7.1, the corresponding section of \( r \), \( s_{\text{Rcurr}} \). Since two consecutive sections, \( s_{\text{Rprev}} \) and \( s_{\text{Rcurr}} \), are kept, then the projection of the resulting \textit{hvl}, is obtained by means of function \( \text{GetHvl} \) and then, it is correctly positioned by procedure \( \text{SetCoord} \).

When the end of one of the polytopes \( p \) or \( q \) is reached then the main iteration finishes and the remaining couplets of the other polytope are either appended or not to the resulting polytope depending on the Boolean operation considered. Procedure \( \text{PutBool} \) performs this appending process.
7.2 Performance of Boolean Operations under the nD-EVM:
An Experimental Time Complexity Analysis

The temporal efficiency of Algorithm 7.1 was evaluated from a statistical point of view. We proceed as follows (in [9] & [11] a more detailed treatment of this analysis is given):

• Our testing considered \( n = 2, 3, 4, 5 \).
• For each \( n \), 16,000 random nD-OPPs were generated according to the following procedures:
  o Given two hypervoxelizations (see Section 8 for details about this representation scheme for polytopes) representing nD-OPPs \( g_1 \) and \( g_2 \) we obtain their respective nD-EVMs namely \( \text{EVM}_n(g_1) \) and \( \text{EVM}_n(g_2) \). According to Theorem 3.1, if a vertex is surrounded by an odd number of occupied hypervoxels then it is an Extreme Vertex. Thus, a hypervoxelization to nD-EVM conversion consists on collecting every vertex that belongs to and odd number of hypervoxels, and discarding the remaining vertices.
  o Given the Regularized Boolean Operator \( \text{op}^* \) we perform both \( g_1 \text{op}^* g_2 \) and \( \text{EVM}_n(g_1 \text{op}^* g_2) \).
  o Let \( \text{EVM}_n(r) \) be the output given by Algorithm 7.1, i.e., \( \text{EVM}_n(r) = \text{EVM}_n(g_1 \text{op}^* g_2) \). Let \( r' \) be the result provided by Boolean operation \( \text{op}^* \) between hypervoxelizations of nD-OPPs \( g_1 \) and \( g_2 \). As a mechanism for controlling possible errors in our implementations we obtain \( \text{EVM}_n(r') \) and verify that all the 16,000 generated nD-OPPs satisfied \( \text{EVM}_n(r') = \text{EVM}_n(r) \). The time for comparison \( \text{EVM}_n(r') = \text{EVM}_n(r) \) is not added to the registered execution times.
• The considered Boolean operations are Regularized Intersection, Union and XOR. In the case corresponding to XOR operation we have tested the same 8,000 pairs of generated nD-OPPs with the Algorithm MergeXOR in order to compare its efficiency with Algorithm 7.1.
• The units for the time measures are given in nanoseconds.
• The evaluations were performed with a CPU Intel Celeron (900 Mhz) and 256 Mbytes in RAM.
• The algorithms were implemented using the Java Programming Language under the Software Development Kit 1.5 provided by Sun Microsystems.
• Once the generation of nD-OPPs has finished and the algorithms were evaluated we proceed with a statistical analysis in order to find a trendline of the form \( t = ax^b \), where \( x = \text{Card}(\text{EVM}_n(g_1)) + \text{Card}(\text{EVM}_n(g_2)) \), that fits as good as possible to our measures in order to provide an estimation of the temporal complexity of the evaluated algorithms for each value of \( n \). The quality of the approximation curve is assured by computing the \( R^2 \) value known as the coefficient of determination. It is well known that \( R^2 \in [0, 1] \) and it reveals how closely the estimated values for the trendline correspond to our time measures. According to the literature, our trendlines are most reliable when its \( R^2 \) is at or near 1.
• Finally and starting from the obtained data we propose an approximation surface for temporal complexity of Algorithm 7.1 for each considered Boolean operation. Such surface which will be a function of two variables: the number \( x \) of Extreme Vertices in the input polytopes and the number \( n \) of dimensions.

Ricardo Pérez-Aguila, August 2009
According to the obtained results we have the following observations:

- Performing intersections has a lesser cost respect to unions. This phenomenon was previously identified in [2] for the 3D case. Although both operations are performed by the same algorithm, the way the polytopes are processed is distinct. As pointed out by [2], Algorithm 7.1 has three processing stages labeled as stage A, stage B and stage C (see Figure 1). Only one of the two involved nD-OPPs is present at stages A and C, with trivial recursive calls at stage A, and no recursive calls at stage C. If the involved Boolean operation is an intersection then the result is empty at those stages, thus almost no work is done at stage A, and no work at all is done at stage C. Any way, stage B will deal with both operands, but the recursive calls at this stage will also have stages A, B and C. Unions, on the other hand, produce Boolean results at all three stages [2].

- Performing Regularized XOR operation is more efficient by using MergeXOR function instead of Algorithm 7.1. We have commented previously that MergeXOR has a linear complexity execution time because it considers extreme vertices in both input polytopes and discards those vertices present in both polytopes, as established in Theorem 5.1. Moreover, execution time of MergeXOR is not affected by the dimensionality of the input polytopes. In our experiment we had 2D, 3D, 4D and 5D-OPPs with 0 to approximately 9,000 extreme vertices and although its dimensionality is distinct, its cardinality is the same.

- The time complexity of Algorithm 7.1 increases according to the dimensionality of the input nD-OPPs. This situation is easy to deduce because the number of recursivity levels depends of the number of dimensions ([9] & [11]).

It is natural to infer that execution time of Algorithm 7.1 depends on two variables: the cardinality x of the nD-EVMs associated to the input polytopes, and the number n of dimensions. Using the measures obtained in the described experiments we computed approximation surfaces, i.e., functions from $\mathbb{N}^2$ to $\mathbb{R}$, that provide us an estimation of the execution time to expect given the number of extreme vertices and the number of dimensions. In Table 1 we present approximation surfaces of the form $t = ax^b n^c$ for Intersection and Union operations and their respective coefficients of determination; in the case for XOR operation we present a function of the form $t = ax^b n^b + c$ (The function $t = 4506.37 x^{1.1819} n^{1.4462}$ was also found for XOR operation, however its coefficient of determination was 0.8357. We decided to propose an alternative form for this specific case in order to provide a more precise estimation).
Table 1. Approximation surfaces for estimating execution times for Boolean operations using Algorithm 7.1.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Approximation Surface</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>( t = 4,271.11 \times 1^{1.1737} )</td>
<td>4,271.11</td>
<td>1.1737</td>
<td>1.0862</td>
<td>0.9234</td>
</tr>
<tr>
<td>Union</td>
<td>( t = 16,698.63 \times 1^{1.0821} )</td>
<td>16,698.63</td>
<td>1.0821</td>
<td>1.0607</td>
<td>0.9221</td>
</tr>
<tr>
<td>XOR</td>
<td>( t = 483.17 \times 1^{1.4161} + 108,263,080 )</td>
<td>483.1716</td>
<td>1.4161</td>
<td>108,263,080</td>
<td>0.9260</td>
</tr>
</tbody>
</table>

It can be observed in these equations that the surface approximating execution time for Intersection operation preserves the property identified before: Its execution time is lesser that execution time of Union operation and even XOR operation. We can then identify, as shown in Table 1, that the exponents associated to the number of vertices vary between 1 and 1.5. This experimentally identified complexity for our algorithm provides elements to determine the temporal efficiency when we perform some operations between nD-OPPs represented through the nD-EVM.

8. Application: Representing 3D Datasets through 4D-OPPs

The representation of a polytope through a scheme of Hyperspatial Occupancy Enumeration is essentially a list of identical hyperspatial cells occupied by the polytope. Specific types of cells, called hypervoxels [5] are hyper-boxes (hypercubes, for example) of a fixed size that lie in a fixed grid in the n-dimensional space. Jonas defines two kinds of hypervoxels [5]:

- Centered Hypervoxel: an n-dimensional hyper-box whose dimensions are given by \( x_1 \text{Side}, x_2 \text{Side}, ..., x_n \text{Side} \) and it is represented by the coordinates of its centroid.
- Shifted Hypervoxel: whose characteristics are same that those for the centered hypervoxel, except that its representation is given by some of its \( 2^n \) vertices.

By instantiation, we know that a 2D hypervoxel is a pixel while a 3D hypervoxel is a voxel; the term rexel is suggested for referencing a 4D hypervoxel [5].

The collection of hyper-boxes can be codified as an n-dimensional array \( C_{x_1,x_2,...,x_n} \). The array will represent the coloration of each hypervoxel. If \( C_{x_1,x_2,...,x_n} = 0 \), the empty hypervoxel \( C_{x_1,x_2,...,x_n} \) represents an unoccupied region from the n-dimensional space. If \( C_{x_1,x_2,...,x_n} = k \neq 0 \), where \( k \) is in a given color scale (black & white, grayscale, RGB, etc.), the occupied hypervoxel \( C_{x_1,x_2,...,x_n} \) represents an used region from the n-dimensional space with intensity \( k \). In fact, the set of occupied cells defines an orthogonal polytope \( p \) whose vertices coincide with some of the occupied cells’ vertices.

By using the representation through an array the spatial complexity of a hypervoxelization is at least \( \prod_{i=1}^{n} m_i \) where \( m_i \), \( 1 \leq i \leq n \), is the length of the grid along the \( X_i \)-axis. For example, a three-dimensional grid with \( m_1 = m_2 = m_3 = 1000 \) requires to store 1 billion
(1×10^9) voxels. Moreover, according to the used color scale each voxel will have a storing requirement. For example, if the color space is RGB then each voxel will require three bytes for codifying its corresponding intensity.

The conversion of a voxelization to a 4D polytope which, at its time, is converted to the 4D-EVM, is in fact a straight procedure. The proposal, originally described in [12], is oriented to color spaces with three or more values (a color scale based in black and white values, for example, is not considered). If it is the case the datasets have binary intensity values then refer to [10] where a more appropriate representation is described. It is assumed the intensities of each voxel are expressed as a single value. First, each voxel is extruded towards the fourth dimension where its intensity-plus-one value is now considered as its X_4 coordinate (coordinates X_1, X_2, and X_3 correspond to the original voxels' coordinates). See Figure 2. Let us call xf to the set composed by the 4D hyperprisms (the extruded voxels) of the extruded 3D dataset.

![Figure 2. Extruding a voxel towards the fourth dimension.](image)

The result is a 4D hyperprism whose height is given by the voxel's intensity color.

Let pr_i be a 4D hyper prism in xf and npr the number of prisms in that set (this number is in fact equal to the number of occupied voxels in the original dataset). Since all the hyperprisms in xf are quasi disjoint 4D-OPPs, we can easily obtain the 4D-OPP and its respective extreme vertices of the whole 4D extruded dataset by computing the regularized union of all the hyperprisms in xf. Then we have to apply Corollary 5.3:

\[ EVM_4(F) = \bigotimes_{i=1}^{npr} EVM_4(pr_i \in xf) \]

Where F is the 4D-OPP that represents the union of all the hyperprisms in xf. By this way, it is obtained a representation for a 3D Dataset through a 4D-OPP and the EVM.

Now, we will describe some results related to the conversion from voxelizations to our proposed representation. Such voxelizations correspond to “real world” grayscale datasets taken from the MoViBio Research Group [7], and the University of Tübingen’s Project VolRen [17]. The Table 2 show the measures obtained, and originally presented in [12], when we converted 3D voxelizations, taken from the mentioned research groups, to 4D-OPPs and the EVM. The descriptions corresponding to the set of objects being modeled in each voxelization are given also in Table 2 as well as the total number of voxels and the total number of extreme vertices in each representation.

---

Ricardo Pérez-Aguila, August 2009
Table 2. 3D Datasets used for conversion to 4D-OPPs and finally expressed through the 4D-EVM.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Voxelization size</th>
<th>4D-EVM Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aneurism</td>
<td>Rotational x-ray scan of the arteries of the right half of a human head. An aneurism is present.</td>
<td>(256 × 256 × 256) ≡ 16,777,216</td>
<td>1,128,404</td>
</tr>
<tr>
<td>Bonsai</td>
<td>Computed tomography of a bonsai tree.</td>
<td>(256 × 256 × 256) ≡ 16,777,216</td>
<td>10,530,120</td>
</tr>
<tr>
<td>Foot</td>
<td>Rotational C-arm x-ray scan of a human foot. Tissue and bone are present in the dataset.</td>
<td>(256 × 256 × 256) ≡ 16,777,216</td>
<td>9,368,452</td>
</tr>
<tr>
<td>Skull</td>
<td>Rotational computer arm x-ray scan of a human skull.</td>
<td>(256 × 256 × 256) ≡ 16,777,216</td>
<td>3,002,256</td>
</tr>
</tbody>
</table>

Now, let \( p \) be a dataset expressed under a voxelization with size \((x_1\text{Size} \times x_2\text{Size} \times x_3\text{Size})\) and with \( \text{EVM}_4(p) \) as its corresponding EVM. Consider the ratio

\[
\frac{x_1\text{Size} \cdot x_2\text{Size} \cdot x_3\text{Size}}{\text{Card}(\text{EVM}_4(p))}
\]

As can be seen, the idea behind such ratio is to express the number of times the quantity of voxels in the original representation of the object \( p \) is greater than the number of extreme vertices in its corresponding representation through the 4D-EVM. For example, consider model \textit{Skull} (Table 2). Its source voxelization has size \((256 \times 256 \times 256)\) which implies that we require to store 16,777,216 voxels. The 4D-EVM associated to \textit{Skull} has 3,002,256 extreme vertices (see Table 2). Hence, our proposed ratio gives us the value 5.58 which implies that the number of stored voxels that belong to the original representation of the object is precisely 5.58 times greater than the number of obtained extreme vertices [12]. The Table 3 shows the ratio Number-of-voxels/Number-of-Extreme-Vertices for the models described in Table 2. The value shared by our ratio depends on the topology and geometry of the objects being modeled, but it shows to us the conciseness, related to storing requirements, when we represent such objects through the EVM.

\[Ricardo\;Pérez-Aguila,\;August\;2009\]
Table 3. The ratio Number-of-voxels/Number-of-Extreme-Vertices for the 3D voxelizations shown in Table 2.

<table>
<thead>
<tr>
<th>Object</th>
<th>Voxelization Size (Number of voxels)</th>
<th>Card(EVM(_4(p))) (Number of extreme vertices)</th>
<th>(\frac{x_1\cdot Size\cdot x_2\cdot Size\cdot x_3\cdot Size}{Card(EVM(_4(p)))})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aneurism</td>
<td>16,777,216</td>
<td>1,128,404</td>
<td>14.86</td>
</tr>
<tr>
<td>Bonsai</td>
<td>16,777,216</td>
<td>10,530,120</td>
<td>1.59</td>
</tr>
<tr>
<td>Foot</td>
<td>16,777,216</td>
<td>9,368,452</td>
<td>1.79</td>
</tr>
<tr>
<td>Skull</td>
<td>16,777,216</td>
<td>3,002,256</td>
<td>5.58</td>
</tr>
</tbody>
</table>

The importance behind a “real world” 3D dataset is the information can be obtained about it. If our 3D models are represented through the 4D-EVM then, according to the procedures mentioned in Section 3.2, computing their sections (Corollary 5.2, Algorithm 6.1) will provide us a classification of the elements in the original model according to their intensities [12]. Given an 8-bits grayscale dataset then it is possible to obtain at most 256 3D sections from its corresponding 4D-OPP. Such sections are those perpendicular to the axis associated to color.

It is usually common that intensities are associated to physical properties. For example, in a medical dataset intensities could refer to certain tissues, hence, by computing 3D sections we obtain, for a given section, only those parts of the dataset with the same type of tissue. The Table 4 presents some 3D sections obtained from the dataset Foot. The first section shown in Table 4 presents the part of the dataset Foot whose voxels correspond to soft tissue and skin. The remaining sections presented in the same table show the different types of bone present in the dataset. As commented previously, the type of material is, in this case, defined by the color intensity of the voxels in the original dataset.

Table 4. Some 3D sections extracted from the 4D-EVM associated to dataset Foot.

The projection of each section, perpendicular to the axis associated to color, in a 4D-EVM, is in fact a 3D-EVM which in time corresponds to a binary voxelization because it only contains material of the same type. Hence, it is possible to apply Corollary 5.2 to one of these 3D sections in order to obtain their associated 2D sections. These 2D sections will describe the interior of the modeled homogeneous object with the objective to perform the appropriate analyses according to the application.

The Table 5 shows some 2D sections corresponding to one of the 3D sections from the model Bonsai which was originally presented in Table 2. The table shows 10 of these 2D sections which allow observing a part of the object which is composed by the same type of material (or in other words, the representation of those voxels in the original dataset with the same color intensity).
Table 5. Visualizing some 2D sections perpendicular to X₁-axis which were processed from one 3D section of the 4D-OPP associated to the dataset Bonsai (Table 2).

<table>
<thead>
<tr>
<th>Section 1</th>
<th>Section 30</th>
<th>Section 45</th>
<th>Section 60</th>
<th>Section 75</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Section 1" /></td>
<td><img src="image2.png" alt="Section 30" /></td>
<td><img src="image3.png" alt="Section 45" /></td>
<td><img src="image4.png" alt="Section 60" /></td>
<td><img src="image5.png" alt="Section 75" /></td>
</tr>
<tr>
<td>Section 120</td>
<td>Section 165</td>
<td>Section 180</td>
<td>Section 195</td>
<td>Section 255</td>
</tr>
<tr>
<td><img src="image6.png" alt="Section 120" /></td>
<td><img src="image7.png" alt="Section 165" /></td>
<td><img src="image8.png" alt="Section 180" /></td>
<td><img src="image9.png" alt="Section 195" /></td>
<td><img src="image10.png" alt="Section 255" /></td>
</tr>
</tbody>
</table>

9. Conclusions and Future Work

In this work we have described the **Extreme Vertices Model in the n-Dimensional Space (nD-EVM)**. The Extreme Vertices Model allows representing nD-OPPs by means of a single subset of their vertices: the **Extreme Vertices**. **Sections 2 to 7** are in fact a very brief description of the capabilities of the model because we have developed simple and robust algorithms, besides the ones presented in this work, for performing the most usual and demanding tasks on polytopes modeling such as measure operations (computing nD content, and computing (n-1)D content of the boundary of an nD-OPP), boundary extraction, and set membership classification operations (see [2] and [9] for more details). **In Section 7.2** the efficiency of the nD-EVM algorithm for computing Boolean Operations was evaluated from a statistical point of view. In such analyses we proposed approximation surfaces, originally presented in [9] & [11], that fit as good as possible to the measures we obtained from the execution times of that algorithm. Such surfaces depend on two parameters: the number of input extreme vertices and the number of dimensions. **Table 1** summarizes the execution times of the Regularized Boolean Operations Algorithm that was described in **Section 7.1**. In all the equations associated to our approximation surfaces we have that by fixing the number of dimensions our functions become dependent only of one variable: the number of input extreme vertices. By this way we can then identify, as shown in **Table 1**, that the exponents associated to the number of vertices vary between 1 and 1.5. This experimentally identified complexity for our algorithm provides us elements to determine the temporal efficiency when we perform some Regularized Boolean Operations between nD-OPPs represented through the nD-EVM. Moreover, in **Section 6** was presented a specific algorithm for computing XOR in linear time: the MergeXOR algorithm. The algorithm’s linear complexity is a valuable property when it is observed the nD-EVM’s intensive use of Regularized XOR in several fundamental operations (for example, computing sections and couplets of an nD-OPP, **section 3.2**).
On the other hand, in **Section 8** we described a way for representing 3D datasets through the nD-EVM. It is well known that voxelizations are the native way in which some datasets are represented and stored. Moreover, common 3D datasets such as the found in medical applications, for example, are required to have a high degree of precision because of the importance of the information obtained from them. However, to more precision usually a cost in spatial complexity must be paid. In this last sense, efficient procedures for compression are required such that the precision in the datasets is not compromised or at least it is affected as minimum as possible. **Section 8** described some results, originally presented in [12], related to the conversion from voxelizations to our specific implementation of nD-EVM when n=4. The presented evaluations lead to establish that an important level of conciseness is obtained when such voxelizations are expressed according a four-dimensional context. Moreover, it is possible to interrogate the 4D-EVM associated to a dataset in order to obtain useful information.

Finally, we conclude by mentioning about the development of “real world” practical applications under the context of the nD-EVM, which are widely discussed and modeled in [9]. These practical applications, through we have showed the versatility of application of the model, consider: (1) the representation and manipulation of 2D and 3D color animations; (2) a method for comparing images oriented to the evaluation of volcanoes’ activity; (3) the way the nD-EVM enhances Image Based Reasoning; and finally, (4) an application to collision detection between 3D objects. As previously commented, details and results, about these four applications can be found in [9].

10. References


