Modeling and Manipulating 3D Datasets through the Extreme Vertices Model in the n-Dimensional Space (nD-EVM)

Ricardo Pérez-Aguila

Universidad Tecnológica de la Mixteca
Carretera Huajuapan-Acatlima Km. 2.5.
Huajuapan de León, Oaxaca 69000, México
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 n-Dimensional Orthogonal Pseudo-Polytopes (nD-OPP's) by considering only a subset of their vertices: the Extreme Vertices. Once we have the elements for managing nD-OPP's through the nD-EVM, we will exemplify the way 3D datasets can be represented through the model. These examples are analyzed in order to compare storage requirements according to their respective representations through voxelizations and through the nD-EVM. The basic algorithms on the nD-EVM provide us useful information about the polytopes being modeled through our scheme.

Keywords: n-Dimensional Orthogonal Polytopes Modeling, Geometrical and Topological Interrogations, 3D Voxelizations Compression, Computational Geometry.

1 Introduction and Problem Statement

Coxeter defines an **n-Dimensional Euclidean Polytope** Π_n as a finite region of n-dimensional Euclidean space enclosed by a finite number of (n-1)-dimensional hyperplanes [3]. The finiteness of the region implies that the number N_{n-1} of bounding hyperplanes satisfies the inequality $N_{n-1}>n$. The part of the polytope that lies on one of these hyperplanes is called a cell. Each cell of a Π_n is an (n-1)D polytope, Π_{n-1}. The cells of a Π_{n-1} are Π_{n-2}'s, and so on; we thus obtain a descending sequence of elements Π_{n-3}, Π_{n-4}, ... , Π_3 (a volume), Π_2 (a polygon), Π_1 (an edge), Π_0 (a vertex) [3].

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 [4] 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 [4]:

- Centered Hypervoxel: an n-dimensional hyper-box whose dimensions are given by x_1Side, x_2Side, ..., x_nSide 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 [4].

The collection of hyperboxes can be codified as an n-dimensional array $C_{x_1,x_2,...,x_n}$ of binary data.

The array will represent the coloration of each hypervoxel:

- If $C_{x_1,x_2,...,x_n}=1$ the **black** hypervoxel $C_{x_1,x_2,...,x_n}$ represents an occupied region from the nD space.
- If $C_{x_1,x_2,...,x_n}=0$ the **white** hypervoxel $C_{x_1,x_2,...,x_n}$ represents an empty region from the nD space.

In fact, the set of black cells represents an nD-OPP p whose vertices coincide with some of the black cells' vertices.

By using the representation through a binary array, the computation of some operations just control the operations between bits for all the elements. It is well known, for example, that Boolean set operations are trivial under this scheme, however, the spatial complexity of an hypervoxelization is $\prod m_i$ 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.

It is well known that hypervoxelizations 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

Industrial Informatics.
Research in Computing Science 31, 2007, pp. 15-24

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. It is clear that the union of the black cells in an nD hypervoxelization defines an nD polytope, in fact, an nD Orthogonal Pseudo-Polytope (nD-OPP). 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-OPP's by considering only a subset of their vertices: the Extreme Vertices (**Section 2**). The Extreme Vertices Model (3D-EVM) was originally presented, and widely described, in [1] for representing 2-manifold Orthogonal Polyhedra and later considering both Orthogonal Polyhedra (3D-OP's) and Pseudo-Polyhedra (3D-OPP's) [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-OPP's. 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, in [6], where the validity of the model was assumed true in order to represent 4D and 5D-OPPs. Finally, in [7] 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 criterions 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-OPP's vertices, there is much more information about the polytope hidden within this subset of vertices. We will describe basic procedures and algorithms in order to obtain this information (**Sections 2.6, 2.7** and **2.8**). Once we have the elements for managing nD-OPP's through the nD-EVM, we will describe the way 3D datasets can be represented through the model (**Section 3**). We will show some examples and we will compare storage requirements according to their respective representations through voxelizations and through the nD-EVM. We will indicate the way the basic algorithms on the nD-EVM provide useful information about the polytopes being modeled under our scheme.

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

2.1 Preliminary Background: n-Dimensional Orthogonal Pseudo-Polytopes

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

$$I^n: [0,1]^n \rightarrow [0,1]^n$$
$$x \sim I^n(x) = x$$

For a general singular kD hyper-box c we will define the boundary of c.

Definition 2.2: *For all i,* $1 \le i \le n$, *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*

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

Definition 2.3: *In a general singular nD hyper-box c we define the (i,α)-cell as* $c_{(i,\alpha)} = c \circ I^n_{(i,\alpha)}$

The next definitions indicate what we consider as the orientation of an (n-1)D cell.

Definition 2.4: *The orientation of an (n-1)D 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,* $1 \le k \le n$, *for a closed set A is called a k-chain.*

Definition 2.7 [8]: *Given a singular nD hyper-box* I^n *we define the (n-1)-chain, called the boundary of* I^n, *by* $\partial(I^n) = \sum_{i=1}^{n} \left(\sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot I^n_{(i,\alpha)} \right)$

Definition 2.8 [8]: *Given a singular general nD hyper-box c we define the (n-1)-chain, called the boundary of c, by* $\partial(c) = \sum_{i=1}^{n} \left(\sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot c \circ I^n_{(i,\alpha)} \right)$

Definition 2.9 [8]: *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*

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

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: *We say that 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, we have that such vertex describes a combination of nD hyper-boxes (**Definition 2.10**) composed up to* 2^n *hyper-boxes.*

Describing nD-OPP's as union of disjoint nD hyper-boxes in such way that by selecting a vertex, in any of these hyper-boxes, we have that such vertex is surrounded up to 2^n hyper-boxes, will be very useful because in the following propositions we consider geometrical and/or topological local analysis over such vertices and their respective incident hyper-boxes.

2.2 The nD-EVM: Foundations

Definition 2.12: *Let c be a combination of hyper-boxes in the n-Dimensional space. An Odd Edge will be an edge with an odd number of incident hyper-boxes of c.*

Definition 2.13: *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 2.14: *We will call Extreme Vertices of an nD-OPP p to the ending vertices of all the brinks in p. EV(p) will denote to the set of Extreme Vertices of 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 X_i-axis, $1 \leq i \leq n$, produce to the same set EV(p).

Definition 2.15: *Let p be an nD-OPP. $EV_i(p)$ will denote to the set of ending or extreme vertices of the brinks of p which are parallel to x_i-axis, $1 \leq i \leq n$.*

Theorem 2.1 [7]: *A vertex of an nD-OPP p, $n \geq 1$, 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 2.2 [7]: *Any extreme vertex of an nD-OPP, $n \geq 1$, when is locally described by a set of surrounding nD hyper-boxes, has exactly n incident linearly independent odd edges.*

Definition 2.16: *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 e_0 belongs to a kD extended hypervolume if and only if e_0 belongs to an (n-1)D cell present in $\partial(p)$.*

Theorem 2.3 [7]: *Let p be an nD-OPP with its associated sets $EV_1(p)$, $EV_2(p)$, ..., $EV_{n-1}(p)$, $EV_n(p)$. Then $EV_1(p) = EV_2(p) = \ldots = EV_{n-1}(p) = EV_n(p)$.*

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 $X_1 \ldots X_i \ldots X_n$ such that $i-1 < i$, $1 < i \leq n$.

Definition 2.17: *Let p be an nD-OPP. We will define the Extreme Vertices Model of p, denoted by $EVM_n(p)$, as the model as only stores to all the extreme vertices of p.*

Theorem 2.4 [7]: *Card(EV$_i$(p))=Card(EV(p))=Card(EVM$_n$(p)) is an even number, $1 \le i \le n$.*

2.3 Sections and Slices of nD-OPP's

Definition 2.18: *We define the Projection Operator for (n-1)D cells, points, and set of points respectively as follows:*

- *Let $c(I^n_{(i,a)}(x)) = (x_1,..,x_n)$ be an (n-1)D cell embedded in the nD space. $\pi_j\left(c(I^n_{(i,a)}(x))\right)$ will denote the projection of the cell $c(I^n_{(i,a)}(x))$ onto an (n-1)D space embedded in nD space whose supporting hyperplane is perpendicular to X_j-axis: $\pi_j\left(c(I^n_{(i,a)}(x))\right) = (x_1,...,\hat{x}_j,...,x_n)$*
- *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 $\mathbb{R}^n$. We define the projection of the points in Q, denoted by $\pi_j(Q)$, as the set of points in $\mathbb{R}^{n-1}$ such that $\pi_j(Q) = \left\{p \in \mathbb{R}^{n-1} : p = \pi_j(x),\ x \in Q \subset \mathbb{R}^n\right\}$*

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

Definition 2.19: *Consider an nD-OPP p:*

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

Definition 2.20: *A Slice is the region contained in an nD-OPP p between two consecutive couplets of p. $Slice^i_k(p)$ will denote to the k-th slice of p which is bounded by $\Phi^i_k(p)$ and $\Phi^i_{k+1}(p)$, $1 \le k < np_i$.*

Definition 2.21: *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, $1 \le i \le n$, which 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. $S^i_k(p)$ will refer to the k-th section of p between $\Phi^i_k(p)$ and $\Phi^i_{k+1}(p)$, $1 \le k < np_i$.*

2.4 Computing Couplets and Sections

Theorem 2.5 [7]: *The projection of the set of (n-1)D-couplets, $\pi_i\left(\Phi^i_k(P)\right)$, of an nD-OPP P, can be obtained by computing the regularized XOR (⊗) between the projections of its previous $\pi_i\left(S^i_{k-1}(P)\right)$ and next $\pi_i\left(S^i_k(P)\right)$ sections, i.e., $\pi_i\left(\Phi^i_k(P)\right) = \pi_i\left(S^i_{k-1}(P)\right) \otimes^* \pi_i\left(S^i_k(P)\right),\ \forall k \in [1, np_i]$*

Theorem 2.6 [7]: *The projection of any section, $\pi_i\left(S^i_k(p)\right)$, of an nD-OPP p, can be obtained by computing the regularized XOR between the projection of its previous section, $\pi_i\left(S^i_{k-1}(p)\right)$, and the projection of its previous couplet $\pi_i\left(\Phi^i_k(p)\right)$.*

2.5 The Regularized XOR operation on the nD-EVM

Theorem 2.7 [2]: *Let p and q be two nD-OPP's having $EVM_n(p)$ and $EVM_n(q)$ as their respective EVM's in nD space, then $EVM_n(p \otimes^* q) = EVM_n(p) \otimes EVM_n(q)$.*

This result allows expressing a formula for computing nD-OPP's sections from couplets and vice-versa, by means of their corresponding Extreme Vertices Models. These formulae are obtained by combining **Theorem 2.7** with **Theorem 2.5**; and **Theorem 2.7** with **Theorem 2.6**, respectively:

Corollary 2.1 [2]: $EVM_{n-1}\left(\pi_i(\Phi^i_k(p))\right) = EVM_{n-1}\left(\pi_i(S^i_{k-1}(p))\right) \otimes EVM_{n-1}\left(\pi_i(S^i_k(p))\right)$

Corollary 2.2 [2]: $EVM_{n-1}\left(\pi_i(S^i_k(p))\right) = EVM_{n-1}\left(\pi_i(S^i_{k-1}(p))\right) \otimes EVM_{n-1}\left(\pi_i(\Phi^i_k(p))\right)$

2.6 Basic Algorithms for the nD-EVM

According to **Sections 2.2** to **2.4** we can define the following primitive operations which are based in the functions originally presented in [2]:

```
Output: An empty nD-EVM.
Procedure InitEVM( )
{ Returns the empty set.                    }
```

```
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 X1-axis from p.          }
```

```
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: Integer or
Real)
Procedure SetCoord(EVM p, CoordType
coord)
{ Sets the X1-coordinate to coord on
  every vertex of the (n-1)D couplet
  p. For coord = 0 it performs the
  projection π1(p).                         }
```

```
Input:  An nD-EVM p
Output: A Boolean.
Procedure EndEVM(EVM p)
{ Returns true if the end of p
  along X1-axis has been reached.           }
```

```
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,
  which is perpendicular to
  X1-axis, to p.                            }
```

```
Input:  Two nD-EVM's p and q.
Output: An nD-EVM
Procedure MergeXor(EVM p, EVM q)
{ Applies the Exclusive OR
  operation to the vertices of
  p and q and returns the
  resulting set.                            }
```

Function MergeXor performs an XOR between two nD-EVM's, that is, it keeps all vertices belonging to either $EVM_n(p)$ or $EVM_n(q)$ and discards any vertex that belongs to both $EVM_n(p)$ and $EVM_n(q)$. 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 $O(Card(EVM_n(p)) + Card(EVM_n(q))$ since each vertex from $EVM_n(p)$ and $EVM_n(q)$ needs to be processed just once. Moreover, according to **Theorem 2.7**, the resulting set corresponds to the regularized XOR operation between p and q.

From the above primitive operations, the **Algorithms 2.1** and **2.2** may be easily derived. The **Algorithm 2.3** computes the sequence of sections of an nD-OPP p from its nD-EVM using the previous functions [2]. It sequentially reads the projections of the (n-1)D couplets *hvl* of the polytope p. Then it computes the sequence of sections using function *GetSection*. Each pair of sections S_i and S_j (the previous and next sections about the current *hvl*) is processed by a generic processing procedure (called *Process*), which performs the desired actions upon S_i and S_j.

```
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)
    // Returns the projection of the
    // next section of an nD-OPP
    // whose previous section is S.
    return MergeXor(S, hvl)
end-of-procedure
```

Algorithm 2.1. Computing $EVM_{n-1}(\pi_1(S_k^i(p)))$ as

$$EVM_{n-1}(\pi_1(S_{k-1}^i(p))) \otimes EVM_{n-1}(\pi_1(\Phi_k^i(p)))$$

```
Input: An (n-1)D-EVM corresponding to
section Si. An (n-1)D-EVM corresponding
to section Sj.
Output: An (n-1)D-EVM.
Procedure GetHvl(EVM Si, EVM Sj)
    // Returns the projection of the
    // couplet between consecutive
    // sections Si and Sj.
    return MergeXor(Si, Sj)
end-of-procedure
```

Algorithm 2.2. Computing $EVM_{n-1}(\pi_1(\Phi_k^i(p)))$ as

$$EVM_{n-1}(\pi_1(S_{k-1}^i(p))) \otimes EVM_{n-1}(\pi_1(S_k^i(p)))$$

```
Input:  An nD-EVM p.
Procedure EVM_to_SectionSequence(EVM p)
        EVM hvl          // Current couplet.
        EVM Si, Sj       // Previous and next sections about hvl.
        hvl = InitEVM( )
        Si = InitEVM( )
        Sj = InitEVM( )
        hvl = ReadHvl(p)
        while(Not(EndEVM(p)))
                Sj = GetSection(Si, hvl)
                Process(Si, Sj)
                Si = Sj
                hvl = ReadHvl(p) // Read next couplet.
        end-of-while
end-of-procedure
```

Algorithm 2.3. Computing the sequence of sections from an nD-OPP p represented through the nD-EVM.

```
Input: An nD-EVM p and the number n of
dimensions.
Output: The content of nD space
enclosed by p.
Procedure Content(EVM p, int n)
  // Variable cont stores the content
  // of nD space enclosed by p.
  real cont = 0.0
  // Couplets between a slice of p.
  EVM hvl1, hvl2
  // Current section of p.
  EVM s
  if(n=1) then
    // Compute the length of the input
    // 1D-OPP expressed under the EVM.
    return Length(p)
  else
    n = n - 1
    hvl1 = InitEVM( )
    hvl2 = InitEVM( )
    s = InitEVM( )
    hvl1 = ReadHvl(p)
    while(Not(EndEVM(p)))
        hvl2 = ReadHvl(p)
        s = GetSection(s, hvl1)
        // Recursive Call.
        cont = cont + Content(s, n) *
                      dist(hvl1, hvl2)
        hvl1 = hvl2
    end-of-while
    return cont
  end-of-else
end-of-procedure
```

Algorithm 2.4. Computing the content of nD space enclosed by p.

```
Input: An nD-EVM p and the number n of
dimensions.
Output: The content of (n-1)D space
enclosed by the boundary of p.
Procedure BoundaryContent(EVM p, int n)
   // cont stores the content of
   // (n-1)D space enclosed by the
   // boundary of p.
   real cont = 0.0
   // Couplets between a slice of p.
   EVM hvl1, hvl2
   EVM s     // Current section of p.
   if(n=2) then
     // Compute the perimeter of
     // the input 2D-OPP expressed
     // under the 2D-EVM.
     return x1Sum(p) + x2Sum(p)
   else
     n = n - 1
     hvl1 = InitEVM( )
     hvl2 = InitEVM( )
     s = InitEVM( )
     hvl1 = ReadHvl(p)
     while(Not(EndEVM(p)))
         hvl2 = ReadHvl(p)
         s = GetSection(s, hvl1)
         // Call to algorithm Content
         // and recursive call.
         cont=cont + Content(hvl1, n) +
               BoundaryContent(s, n) *
                     dist(hvl1, hvl2)
         hvl1 = hvl2
     end-of-while
     cont = cont + Content(hvl1, n)
     // hvl1 contains the last
     // couplet of p.
     return cont
   end-of-else
end-of-procedure
```

Algorithm 2.5. Computing the content of (n-1)D space enclosed by the boundary of p.

2.7 Computing the Content of an nD-OPP

Now, we will describe a procedure in order to compute the content of nD space enclosed by an nD-OPP (length of a 1D-OPP, area of a 2D-OPP, volume of a 3D-OPP, and so on). In this case we will consider the partition induced by its Slices. Let p be an nD-OPP. The nD space enclosed by p, denoted by $Content_{(n)}(p)$, can be computed as the sum of the contents of its nD slices (**Equation 2.1**):

$$Content_{(n)}(p) = \begin{cases} Length(p) & n = 1 \\ \sum_{k=1}^{np_i - 1} Content_{(n-1)}\left(S_k^i(p)\right) \cdot dist\left(\Phi_k^i(p), \Phi_{k+1}^i(p)\right) & n > 1 \end{cases}$$

Where np_i is the number of couplets of p perpendicular to X_i-axis; $S_k^i(p)$ is the k-th section of the nD-OPP p which is perpendicular to X_i-axis and it is between couplets $\Phi_k^i(p), \Phi_{k+1}^i(p)$ whose distance is given by $dist\left(\Phi_k^i(p), \Phi_{k+1}^i(p)\right)$. The **Algorithm 2.4** implements **Equation 2.1** in order to compute the content of nD space enclosed by an nD-OPP p expressed through the nD-EVM.

2.8 Computing the Content of the Boundary of an nD-OPP

Now, we will describe the way to compute the content of (n-1)D space enclosed by the boundary of an nD-OPP (perimeter of a 2D-OPP, area of the boundary of a 3D-OPP, volume of the boundary of a 4D-OPP, and so on). Let p be an nD-OPP. The (n-1)D space enclosed by p, denoted by $BoundaryContent_{(n-1)}(p)$, can be computed as follows (**Equation 2.2**):

$$BoundaryContent_{(n-1)}(p) =$$

$$\begin{cases} x_1 Sum(p) + x_2 Sum(p) & n = 2 \\ \sum_{k=1}^{np_i} Content_{(n-1)}\left(\Phi_k^i(p)\right) + \sum_{k=1}^{np_i-1} BoundaryContent_{(n-1)}\left(S_k^i(p)\right) \cdot dist\left(\Phi_k^i(p), \Phi_{k+1}^i(p)\right) & n > 2 \end{cases}$$

According to previous equation we reach the basic case when n = 2. In this situation, the perimeter of a 2D-OPP p is computed as $x_1 Sum(p) + x_2 Sum(p)$ where $x_i Sum(p)$ is the sum of the lengths of all brinks parallel to X_i-axis. The **Algorithm 2.5** implements **Equation 2.2** in order to compute the content of (n-1)D space enclosed by the boundary of p expressed through the nD-EVM.

3 Manipulating "Real World" 3D Datasets with the nD-EVM

In this section we will describe some results related to the conversion from voxelizations to our specific implementation of nD-EVM when n = 3. Such voxelizations correspond to "real world" datasets taken from the MoViBio Research Group [5], and the University of Tübingen's Project VolRen [9]. As commented in the **Section 1**, a 3D voxelization is a set of black and white cells where each cell is a convex orthogonal polyhedron. The set of black cells represents an nD-OPP p whose vertices coincide with some of the black cells' vertices. Each of these vertices may be common to (surrounded by) up to 8 black cells. So, according to **Theorem 2.1**, if a vertex is surrounded by an odd number of black cells then it is an Extreme Vertex. Thus, a 3D voxelization to the 3D-EVM conversion algorithm is as simple as collecting every vertex that belongs to and odd number of cells, and discarding the remaining vertices. The **Tables 1** and **2** show the measures we obtained when we converted 3D voxelizations, taken from the mentioned research groups, to our implementation in the Java Language, of the nD-EVM when n = 3. We report the following execution times:

- Time for computing the conversion from 3D voxelization to 3D-EVM.
- Time for computing the content of the 3D-OPP (**Section 2.7**) expressed under the 3D-EVM.
- Time for computing the boundary content of the 3D-OPP (**Section 2.8**) represented by the EVM.
- Time for computing the 2D sections, starting from the 2D couplets (**Section 2.6**) perpendicular to X_1-axis, of the resulting 3D-OPP.

The descriptions corresponding to the set of objects being modeled in each voxelization are given in **Tables 1** and **2**, as well as the total number of voxels in each representation. The conversions and tasks were performed in a computer with Intel Celeron Processor at 900 Mhz and 256 megabytes in RAM memory.

From **Tables 1** and **2** can be observed, in first place, that the processing time for the conversion from the 3D voxelizations to the 3D-EVM was the largest from all the considered tasks. In this situation model *Skull* (**Table 2**) required 2,626.757 seconds (almost 45 minutes) for its conversion while, on the other hand, the shortest time for conversion corresponds to the model *Marschner/Lobb* (**Table 1**) with 1.332 seconds. It is interesting to observe that the model *Skull* does not contain the maximum number of extreme vertices in its corresponding EVM. The 3D model *Leg of Statue* (**Table 1**) was represented with 1,604,538 extreme vertices while *Skull* was represented with 1,302,134 extreme vertices, however, *Leg of Statue* required 480.902 seconds for its conversion to the 3D-EVM (approximately 8 minutes). The model *Marschner/Lobb* has both the lowest number of extreme vertices (872) and the lowest processing time for its conversion (1.332 seconds).

Now, let p be a 3D-OPP expressed under a voxelization with size ($x_1Size \times x_2Size \times x_3Size$) and with $EVM_3(p)$ as its corresponding EVM. Consider the ratio

$$\frac{x_1Size \cdot x_2Size \cdot x_3Size}{Card(EVM_3(p))}$$

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 3D-EVM. For example, consider model *CSF* (**Table 2**). Its source voxelization has size (256 × 256 × 124) which implies that we require to store 8,126,464 voxels. The 3D-EVM associated to *CSF* has 86,570 extreme vertices (see **Table 2**). Hence, our proposed ratio gives us the value 93.87 which implies that the number of stored voxels that belong to the original representation of the object is precisely 93.87 times greater than the number of obtained

extreme vertices. The **Table 3** shows the ratio Number-of-voxels/Number-of-Extreme-Vertices for the models described in **Tables 1** and **2**. According to the results we obtained, the number of voxels in the model *Lobster* is 131.38 times greater than the cardinality of its corresponding 3D-EVM. In fact, model *Lobster* has the largest ratio from our seven tested models. On the other hand, the model *Leg of Statue* (**Table 1**) has a number of voxels which is 6.73 times greater than the number of extreme vertices required for representing it. 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.

Table 1. Results from the conversion of 3D voxelizations (*Bonsai*, *Leg of statue*, *Lobster* and *Marschner/Lobb*) to the 3D-EVM.

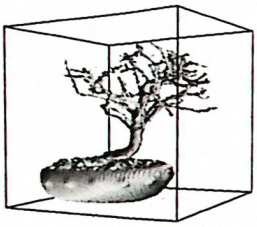
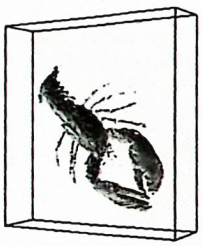
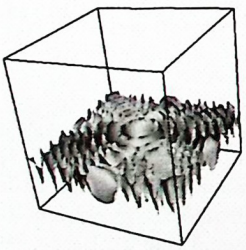
	Voxelization size:	$(256 \times 256 \times 256) \equiv 16{,}777{,}216$ voxels
	Name and Description:	**Bonsai.** Computed tomography of a bonsai tree.
	EVM size:	**641,462**
	Time for conversion:	**858.905 s**
	Content:	3,412,818 u^3
	Time for computing 3D content:	**33.709 s**
	Boundary content:	2,098,246 u^2
	Time for computing boundary content:	**71.322 s**
	Time for computing 2D sections:	**36.282 s**
	Voxelization size:	$(341 \times 341 \times 93) \equiv 10{,}814{,}133$ voxels
	Name and Description:	**Leg of statue.** Computed tomography of a leg of a bronze statue.
	EVM size:	**1,604,538**
	Time for conversion:	**480.902 s**
	Content:	2,499,595 u^3
	Time for computing 3D content:	**136.847 s**
	Boundary content:	3,244,280 u^2
	Time for computing boundary content:	**367.579 s**
	Time for computing 2D sections:	**130.257 s**
	Voxelization size:	$(301 \times 324 \times 56) \equiv 5{,}461{,}344$ voxels
	Name and Description:	**Lobster.** Computed tomography of a lobster contained in a block of resin.
	EVM size:	**41,566**
	Time for conversion:	**335.722 s**
	Content:	3,831,352 u^3
	Time for computing 3D content:	**0.921 s**
	Boundary content:	263,910 u^2
	Time for computing boundary content:	**1.582 s**
	Time for computing 2D sections:	**0.751 s**
	Voxelization size:	$(41 \times 41 \times 41) \equiv 68{,}921$ voxels
	Name and Description:	**Marschner/Lobb.** Simulation of high frequencies where 99% of the sinusoids are right below the Nyquist frequency.
	EVM size:	**872**
	Time for conversion:	**1.332 s**
	Content:	68,637 u^3
	Time for computing 3D content:	**0.010 s**
	Boundary content:	11,070 u^2
	Time for computing boundary content:	**0.020 s**
	Time for computing 2D sections:	**0.001 s**

Table 2. Results from the conversion of 3D voxelizations (*Silicium, Skull* and *CSF*) to the 3D-EVM.

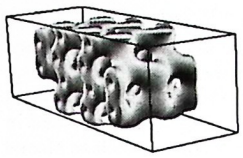
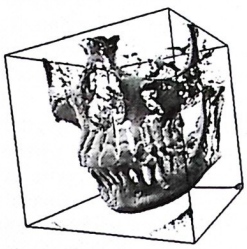
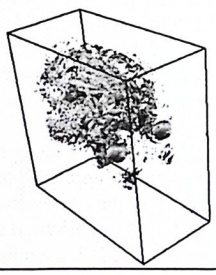
	Voxelization size:	$(98 \times 34 \times 34) \equiv 113{,}288$ voxels
	Name and Description:	**Silicium.** Simulation of a silicium grid.
	EVM size:	**1,164**
	Time for conversion:	**1.412 s**
	Content:	66,163 u^3
	Time for computing 3D content:	**0.010 s**
	Boundary content:	11,274 u^2
	Time for computing boundary content:	**0.020 s**
	Time for computing 2D sections:	**0.001 s**
	Voxelization size:	$(256 \times 256 \times 256) \equiv 16{,}777{,}216$ voxels
	Name and Description:	**Skull.** Rotational computer arm x-ray scan of a human skull.
	EVM size:	**1,302,134**
	Time for conversion:	**2,626.757 s**
	Content:	14,834,427 u^3
	Time for computing 3D content:	**87.907 s**
	Boundary content:	2,656,000 u^2
	Time for computing boundary content:	**245.873 s**
	Time for computing 2D sections:	**78.723 s**
	Voxelization size:	$(256 \times 256 \times 124) \equiv 8{,}126{,}464$ voxels
	Name and Description:	**CSF.** Dataset corresponding to the Cerebro-Spinal-Fluid in a human head.
	EVM size:	**86,570**
	Time for conversion:	**798.458 s**
	Content:	8,115,182 u^3
	Time for computing 3D content:	**2.293 s**
	Boundary content:	323,810 u^2
	Time for computing boundary content:	**3.395 s**
	Time for computing 2D sections:	**2.283 s**

Table 3. The ratio Number-of-voxels/Number-of-Extreme-Vertices for the 3D voxelizations shown in **Tables 1** and **2**.

Object p	Voxelization Size (Number of voxels)	Card(EVM_3(p))	$\frac{x_1Size \cdot x_2Size \cdot x_3Size}{Card(EVM_3(p))}$
Bonsai	16,777,216	641,462	26.15
Leg of Statue	10,814,133	1,604,538	6.73
Lobster	5,461,344	41,566	131.38
Marschner/Lobb	68,921	872	79.03
Silicium	113,288	1,164	97.32
Skull	16,777,216	1,302,134	12.88
CSF	8,126,464	86,570	93.87

The importance behind a "real world" 3D dataset is the information can be obtained about it. As commented in the introduction of this section, we computed for each model shown in **Tables 1** and **2**, through its corresponding 3D-EVM, the volume (u^3), the area of its boundary (u^2), and its sections. The last task refers to the case if our 3D models are represented through the 3D-EVM then, according to the procedures mentioned in **Section 2.6**, their sections will describe to us the interior of the modeled objects with the objective to perform the appropriate analyses according to the application. The 3D model *Leg of Statue* (that with the highest number of extreme vertices) required the maximum processing times for computing both its volume as the area of its boundary: 136.847 seconds and 367.579 seconds respectively. Conversely, the 3D model *Marschner/Lobb* (that with the lowest number of extreme vertices) required the minimum processing times for computing both its volume as the area of its boundary: 0.01 seconds and 0.02 seconds respectively. In the case related to

the computing of sections, we found again that *Leg of Statue* required the maximum processing time: 130.257 seconds. A "tie" was found in the minimum processing time for sections corresponding to 3D models *Marschner/Lobb* and *Silicium* (**Tables 1** and **2** respectively): their sections were computed in only one millisecond.

4 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-OPP's by means of a single subset of their vertices: the *Extreme Vertices*. **Section 2** is 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 closed and regularized Boolean operations, boundary extraction, and set membership classification operations (see [2] and [7] for more details). These procedures and algorithms provide us a way to obtain much more information than the extracted from the examples presented in **Section 3**. Moreover, **Table 3** shows the conciseness of the nD-EVM respect to voxelizations because we have obtained in all our described examples that the ratio Number-of-voxels/Number-of-Extreme-Vertices is between 6.73 and 131.38.

In this work we have concentrated about the representation of binary voxelizations through the EVM. However, it is well known that some devices express their outputs through 3D datasets whose voxels contain more information besides the empty/occupied property. In this sense, we propose as future work the representation of these datasets through nD-OPP's by considering their voxels' additional properties as independent spatial dimensions. Our proposal implies that these datasets can be expressed and manipulated as polytopes whose number of dimensions in greater than three. Furthermore, we will consider the representation of time varying 3D and 4D datasets through the nD-EVM. Because of the intensive use of the XOR operation in the nD-EVM (**Sections 2.4**, **2.5** and **2.6**), we exploit spatial and temporal redundancies between 2D and 3D frames, in these 3D and 4D datasets respectively, without compromising the quality of the information for the required purposes. We can guarantee, according to previous experiences, the model will represent these datasets in a very concise way.

Finally, we mention the development of other "real world" practical applications under the context of the nD-EVM, which are widely discussed and modeled in [7]. These practical applications, through we have showed the versatility of application of the nD-EVM, 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 through the nD-EVM. As previously commented, details and results about these four practical applications can be found in [7].

References

1. Aguilera, A. & Ayala, D. Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry. Fourth ACM Siggraph Symposium on Solid Modeling and Applications SM'97, pp. 56-67. Atlanta, USA, 1997.
2. Aguilera, A. Orthogonal Polyhedra: Study and Application. PhD Thesis. Universitat Politècnica de Catalunya, 1998.
3. Coxeter, H.S.M. Regular Polytopes. Dover Publications, Inc., New York, 1963.
4. Jonas, A. & Kiryati, N. Digital Representation Schemes for 3-D Curves. Technical Report CC PUB #114, The Technion - Israel Institute of Technology, Haifa, Israel, July 1995.
5. MoViBio Research Group (Modeling and Visualization of Biomedical data). Web Site: http://truja.lsi.upc.edu/movibio (Site visited in May, 2007).
6. Pérez-Aguila, R. The Extreme Vertices Model in the 4D space and its Applications in the Visualization and Analysis of Multidimensional Data Under the Context of a Geographical Information System. MSc Thesis. Universidad de las Américas, Puebla. Puebla, México, May 2003.
7. Pérez-Aguila, R. Orthogonal Polytopes: Study and Application. PhD Thesis. Universidad de las Américas - Puebla. Cholula, Puebla, México, November 13th, 2006.
8. Spivak, M. Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus. HarperCollins Publishers, 1965.
9. VolRen: Project Volume Rendering with Graphics HW. Web Site: http://www.gris.uni-tuebingen.de/areas/volren (Site visited in May, 2007).