Representing and Visualizing Vectorized Videos 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. Several video compression methods were invented to be able to effectively store video data on common digital media. One method of compression we will explore in this work is oriented to vectorized video sequences. Each frame in a color video is initially represented as a set of orthogonal polygons whose displaying time depends on the temporal dimension. Moreover, one spatial dimension will be assigned to the color to apply to such polygons. Hence, a vectorized 2D color video sequence can be expressed as a Four-Dimensional Orthogonal Pseudo-Polytope which will be represented under the Extreme Vertices Model in the n-Dimensional Space (nD-EVM). The nD-EVM shares the representation of n-Dimensional Orthogonal Pseudo-Polytopes (nD-OPP's) by considering only a subset of their vertices: the Extreme Vertices. This work will describe how the source sequences can be converted in a vectorized video and then compressed, expressed, manipulated, and displayed in screen through the 4D-EVM. The results obtained from the conversion of two video sequences motivate some observations and properties of the proposed methodology.

Keywords: n-Dimensional Orthogonal Polytopes Modeling, Geometrical and Topological Representations, Color 2D-Videos Compression, Computational Geometry.

1 Introduction and Problem Statement

The Extreme Vertices Model (3D-EVM) was originally presented, and widely described in [1], for modeling 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 ope-rations 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 [8], where the validity of the model was assumed true in order to represent 4D and

A. Gelbukh, S. Suárez, H. Calvo (Eds.)
Advances in Computer Science and Engineering
Research in Computing Science 29, 2007, pp. 65-80

Received 07/09/07
Accepted 19/10/07
Final version 24/10/07

5D-OPPs. Finally, in [9] was formally proved that the nD-EVM is a complete scheme for the repre-sentation of nD-OPPs. The meaning of complete scheme was based in Requicha's set of formal crite-rions that every scheme must have rigorously defined: Domain, Completeness, Uniqueness and Vali-dity. 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. In **Sections 2.5** and **2.6** we will describe basic procedures and algorithms in order to obtain some of this information.

It is well known that classical storage techniques like celluloid films or analogue video tapes carry various mechanical and physical degradations that significantly reduce their visual quality along time [4]. The sampling of analog signals and expressing them in digital form is used today to guarantee the quality of the information and make them media independent. As pointed out in [4], in order to achieve good fidelity it is often necessary to produce a large amount of data. Cartoon animations usually provide source video sequences to be vectorized. For example, Koloros & Zára [4] separate in first place the original animation frame into a set of regions using unsupervised image segmentation techniques. Then they use motion estimation in order to register parts of the background to stitch and store background layer as one big image. Shapes of homogeneous color regions in the foreground layer are converted from raster to vector representation and encoded separately. To search for frame duplicities and to store new frames they use a pool of already stored frames. During the playback standard graphics hardware is used to render the background layer as a textured rectangle and in front of it foreground layer as a set of flat colored polygons [4]. Another example of vectorization of cartoon animations is given by the work of Kwatra and Rossignac [6]. In their approach each region is first represented as a 3D volume by sweeping its 2D shape through the time. Then their Edgebreaker compression scheme is used to encode volume geometry. However, these authors ([4] & [6]) did not address the problem of vectorization for complex color and gray scale image sequences.

In this work, each frame in a color video is initially represented as a set of orthogonal polygons whose displaying time depends on the temporal dimension. Moreover, we will assign one spatial dimension to the color to apply to such polygons. Hence, we will express a vectorized 2D color video sequence as a Four-Dimensional Orthogonal Pseudo Polytope (4D-OPP) which will be represented under the 4D-EVM. In **Section 3** we will describe how the source sequences can be converted in a vectorized video and then compressed, manipulated, and displayed in screen through the 4D-EVM.

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* ¡ ^{n} *is the continuous function*

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

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 $\underline{I^n_{(i,0)}}$ and $\underline{I^n_{(i,1)}}$ are defined as follows: If $x \in [0,1]^{n-1}$ then* $I^n_{(i,0)}(x) = I^n(x_1,...,x_{i-1},0,x_i,...,x_{n-1}) = (x_1,...,x_{i-1},0,x_i,...,x_{n-1})$ and $I^n_{(i,1)}(x) = I^n(x_1,...,x_{i-1},1,x_i,...,x_{n-1}) = (x_1,...,x_{i-1},1,x_i,...,x_{n-1})$

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 in precise way what we consider as the orientation of a (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 [11]: *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 [11]: *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 [11]: *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 $c_1, c_2, ..., c_k$, $1 \le k \le 2^n$, of general singular nD hyper-boxes is a combination of nD hyper-boxes if and only if*

$$\left[\mathop{\mathrm{I}}_{\alpha=1}^{k} c_\alpha([0,1]^n) = (0,\underbrace{...}_{n},0)\right] \wedge \left[\left(\forall i,j,\ i \ne j,\ 1 \le i,j \le k\right)\left(c_i([0,1]^n) \ne c_j([0,1]^n)\right)\right]$$

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 [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.*
Definition 2.16: *Let p be a 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 a (n-1)D cell present in $\partial(p)$.*

Let Q be a finite set of points in ¡ ^{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.*

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 $c(I^n_{(i,\alpha)}(x)) = (x_1,..,x_n)$ be an (n-1)D cell embedded in the nD space. $\underline{\pi_j\left(c(I^n_{(i,\alpha)}(x))\right)}$ will denote the projection of the cell $c(I^n_{(i,\alpha)}(x))$ onto an (n-1)D space embedded in nD space whose supporting hyperplane is perpendicular to X_j-axis: $\pi_j\left(c(I^n_{(i,\alpha)}(x))\right) = (x_1,...,\hat{x}_j,...,x_n)$*

- *Let* $v=(x_1,...,x_n)$ *be 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* $\pi_j(Q)=\{p\in\mathbb{R}^{n-1}: p=\pi_j(x),\ x\in Q\subset\mathbb{R}^n\}$

In all the cases $\hat{x}_j$ *is the coordinate corresponding to* 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* $\Phi_k^i(p)$ *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 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* X_i*-axis,* $1\le i\le n$, *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.* $S_k^i(p)$ *will refer to the k-th section of p between* $\Phi_k^i(p)$ *and* $\Phi_{k+1}^i(p)$, $1\le k<np_i$.

2.4 Computing Couplets and Sections

Theorem 2.2 [9]: *The projection of the set of (n-1)D-couplets,* $\pi_i\left(\Phi_k^i(P)\right)$, *of an nD-OPP P, can be obtained by computing the regularized XOR (⊗) between the projections of its previous* $\pi_i\left(S_{k-1}^i(P)\right)$ *and next* $\pi_i\left(S_k^i(P)\right)$ *sections, i.e.,* $\pi_i\left(\Phi_k^i(P)\right)=\pi_i\left(S_{k-1}^i(P)\right)\ \otimes^*\ \pi_i\left(S_k^i(P)\right),\ \forall k\in[1,np_i]$

Theorem 2.3 [9]: *The projection of any section,* $\pi_i\left(S_k^i(p)\right)$, *of an nD-OPP p, can be obtained by computing the regularized XOR between the projection of its previous section,* $\pi_i\left(S_{k-1}^i(p)\right)$, *and the projection of its previous couplet* $\pi_i\left(\Phi_k^i(p)\right)$.

2.5 The Regularized XOR operation on the nD-EVM

Theorem 2.4 [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 2.4** with **Theorem 2.2**; and **Theorem 2.4** with **Theorem 2.3**, respectively:

Corollary 2.1 [2]: $EVM_{n-1}\left(\pi_i(\Phi_k^i(p))\right)=EVM_{n-1}\left(\pi_i(S_{k-1}^i(p))\right)\otimes EVM_{n-1}\left(\pi_i(S_k^i(p))\right)$

Corollary 2.2 [2]: $EVM_{n-1}\left(\pi_i(S_k^i(p))\right)=EVM_{n-1}\left(\pi_i(S_{k-1}^i(p))\right)\otimes EVM_{n-1}\left(\pi_i(\Phi_k^i(p))\right)$

Finally, the following corollary can be stated, which correspond to a specific situation of the XOR operands. It allows computing the union of two nD-OPP's when that specific situation is met.

Corollary 2.3 [2]: *Let p and q be two disjoint or quasi disjoint nD-OPP's having $EVM_n(p)$ and $EVM_n(q)$ as their respective Extreme Vertices Models, then* $EVM_n(p \cup q) = EVM_n(p) \otimes EVM_n(q)$.

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

```
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:  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:  An nD-EVM p
Output: An integer
Procedure GetN(EVM p)
{ Returns the number n of dimensions of
  the space where p is embedded.        }
```

```
Input:  An nD-EVM p
Output: A Boolean.
Procedure IsEmpty(EVM p)
{ Returns true if p is an empty set.    }
```

```
Input:  An nD-EVM p
Output: A coordinate of type CoordType
(the chosen type for the vertex
coordinates: Integer or Real)
Procedure GetCurrentCoord(EVM p)
{ Returns the common X1-coordinate
  of the next (n-1)D couplet to be
  extracted from p.                     }
```

```
Input/Output: An (n-1)D-EVM p embedded
in (n-1)D space.
Input:  A coordinate coord of type
CoordType (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:  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 2.4**, 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, plv)
end-of-procedure
```

```
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 2.1. Computing $EVM_{n-1}\left(\pi_1(S_k^i(p))\right)$ as $EVM_{n-1}\left(\pi_1(S_{k-1}^i(p))\right) \otimes EVM_{n-1}\left(\pi_1(\Phi_k^i(p))\right)$

Algorithm 2.2. Computing $EVM_{n-1}\left(\pi_1(\Phi_k^i(p))\right)$ as $EVM_{n-1}\left(\pi_1(S_{k-1}^i(p))\right) \otimes EVM_{n-1}\left(\pi_1(S_k^i(p))\right)$

```
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.

3 Representing Color 2D Videos through 4D-OPP's and the EVM

The procedure described in [2] for processing black & white 2D animations can be directly extended to control colored frames through a 4D-OPP represented through the EVM. In the **Figure 1** an example of a simple color 2D-animation composed by four frames whose resolution is 9 × 9 pixels is shown. In each frame can be identified yellow, red, green and blue regions. We will use this simple animation to exemplify our procedure. We will label each colored frame in the animation as f_k and m will be the number of such frames.

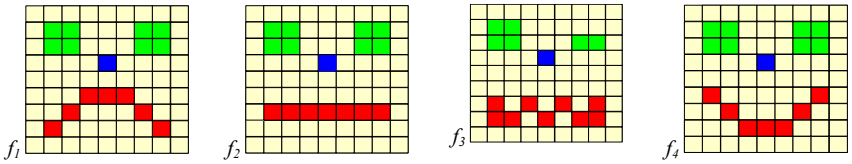


Figure 1. Example of a simple color 2D-animation.

A color animation can be handled as a 4D-OPP in the following way [9]:

a) The red-green-blue components of each pixel will be integrated into a single value. Such value represents the red-green-blue components as an integer with 32 bits. Bits 0-7 correspond to the blue value, bits 8-15 correspond to the green value, bits

16-23 correspond to the red value and bits 24-31 to the *alpha* (transparency) value. Each pixel will now be extruded towards the third dimension where the value integrating its red-green-blue components will now be considered as its X_3 coordinate (coordinates X_1 and X_2 correspond to the original pixels' coordinates). See **Figure 2**.

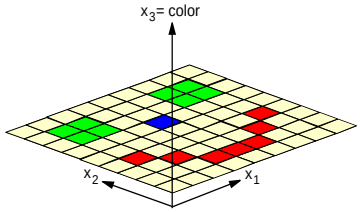


Figure 2. The 3D space defined for the extrusion of color 2D-pixels.

Let us call xf_k to the set composed by the rectangular prisms (the extruded pixels) of each extru-ded frame f_k. It is very important to avoid the zero value in the X_3 coordinate because a pixel could not be extruded and therefore its associated prism (a 3D-OPP) won't be obtained. See in **Figure 3** the sets of prisms xf_k which are the result of the extrusion of frames f_k of the animation from **Figure 1**.

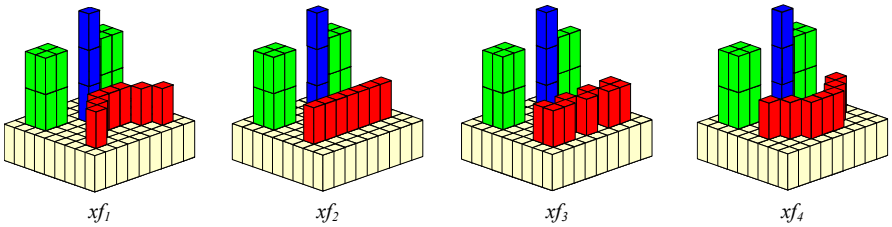


Figure 3. The sets of prisms which are the result of the extrusion of the frames of an animation (presented in Figure 1).

b) Let $prism_i$ be a prism in xf_k and npr the number of prisms in that set. Due to all the prisms in xf_k are quasi disjoint 3D-OPP's, we can easily obtain the final 3D-OPP and its respective 3D-EVM of the whole 3D frame by computing the regularized union of all the prisms in xf_k. Then, according to **Corollary 2.3**, we have to apply (all the vertices in a $prism_i$ are extreme):

$$EVM_3(F_k) = \bigotimes_{i=1}^{npr} EVM_3(prism_i \in xf_k)$$

where F_k is the 3D frame (a 3D-OPP) that represents the union of all the prisms in xf_k.

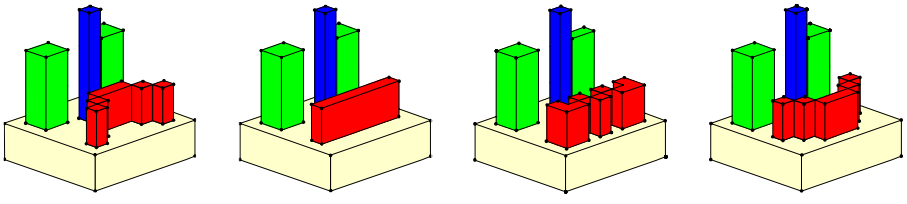


Figure 4. The 3D frames that represent a 2D colored animation (presented in Figure 1. Some of their extreme vertices are shown).

In the **Figure 4** are shown the 3D frames F_k from the animation presented in **Figure 1**.

c) Let us extrude F_k into the fourth dimension, and thus obtain a 4D hyperprism $hyperprism_k$ whose bases are F_k and its length is proportional to the time f_k is to be displayed. The new fourth dimension will measure and represent the time. See **Figure 5**.

d) Let $p = \bigcup_{k=1}^{m} hyperprism_k$, then p is a 4D-OPP that represents the given color 2D-animation. Due to all the m hyperprisms are quasi disjoint 4D-OPP's, then the 4D-EVM for p can be obtained by:

$$EVM_4(p) = \bigotimes_{k=1}^{m} EVM_4(hyperprism_k)$$

In the **Figure 6** are shown the couplets perpendicular to the axis that represent the time, of the 4D-OPP p that represents the animation from **Figure 1**. The **Algorithm 3.1** shows the procedure for converting a set of frames in an animation to a 4D-OPP that codifies it. Such 4D-OPP is represented through a 4D-EVM.

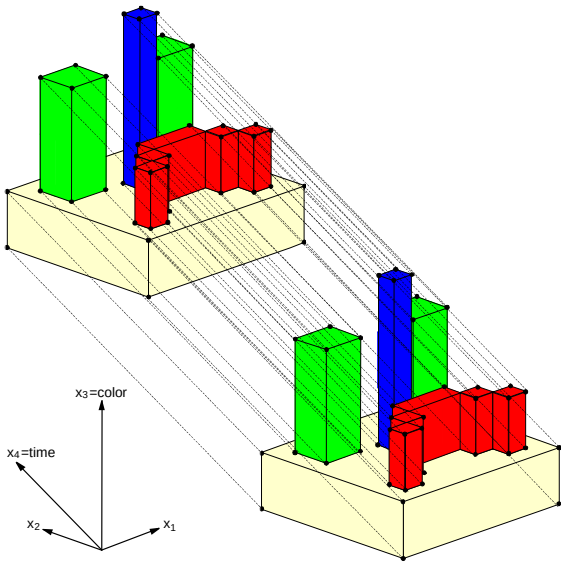
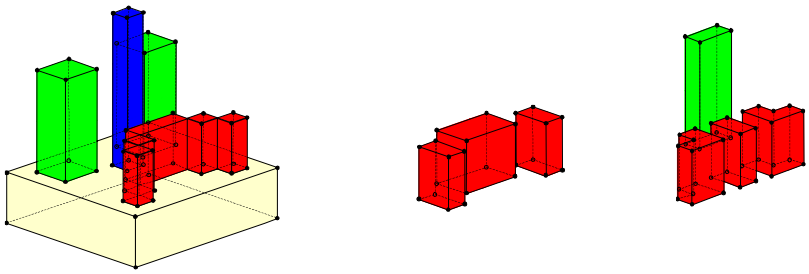


Figure 5. The process of extrusion of a 3D frame in order to obtain a *hyperprism* (some of its extreme vertices are shown).



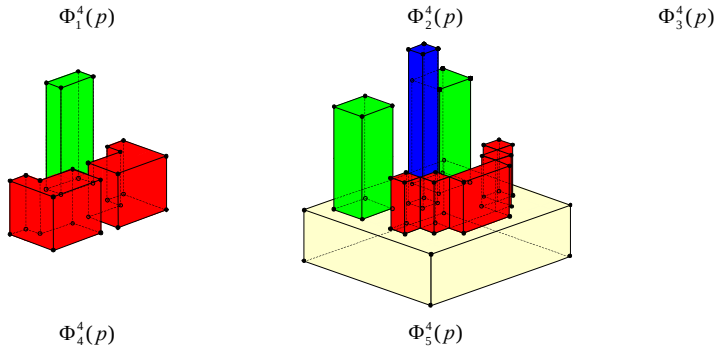


Figure 6. The 3D couplets of the 4D-OPP p that represents a color 2D-animation (from Figure 1. Their extreme vertices are shown).

```
Input:  A sequence of frames associated to a color 2D animation.
        The values xSize and ySize corresponding to the resolution of the input
        animation.
Output: The 4D-EVM corresponding to the polytope that codifies frames in the
        input animation.
Procedure GenerateEVM-movie(Movie animation, xSize, ySize)
   EVM evmMovie // The EVM that will store and codify the input animation.
   EVM hvl
   EVM Fcurr, Fprev      // Current and previous 3D frames being processed.
   real t                // The amount of time that current processed frame is
                         // displayed.
   Fprev = InitEVM( )
   for each frame in animation do
        Fcurr = InitEVM( )
        Frame f = animation.nextFrame( )
        t = animation.getDisplayingTime( )
        // Frame f is extruded towards 3rd dimension and its 3D-EVM is computed.
        for x = 0 until xSize – 1 do
            for y = 0 until ySize – 1 do
                rgb = getRGBComponents(x, y, f)
                // We obtain the EVM of the prism associated to (x, y, rgb).
                EVM prism = GetPrismEVM(x, y, rgb)
                Fcurr = MergeXor(prism, Fcurr)
            end-of-for
        end-of-for
        // We perform the Xor between the current and previous 3D frames.
        hvl = MergeXor(Fcurr, Fprev)
        // Amount of time t associated to frame Fcurr is attached to the current
        // 3D couplet.
        SetCoord(hvl, t)
        // A new 3D couplet is attached to the 4D polytope that codifies the
        // input animation.
        PutHvl(hvl, evmMovie)
        Fprev = Fcurr
    end-of-for
    return evmMovie
end-of-procedure
```

Algorithm 3.1. Codifying a Color 2D-animation through a 4D-OPP and the EVM.

```
Input:  A 4D-EVM p that represents a color 2D-animation.
        The graphics context g where the animation is going to be displayed.
Procedure playEVM-movie(EVM p, g)
    EVM hvl               // Current 3D couplet in p.
    EVM Fprev, Fcurr      // Previous and current 3D frames in the animation.
    EVM hvlF              // Current 2D couplet in Fcurr. It contains polygons
                          // to display.
    int color             // The color to apply to the polygons to be displayed.
    Fprev = InitEVM( )
```

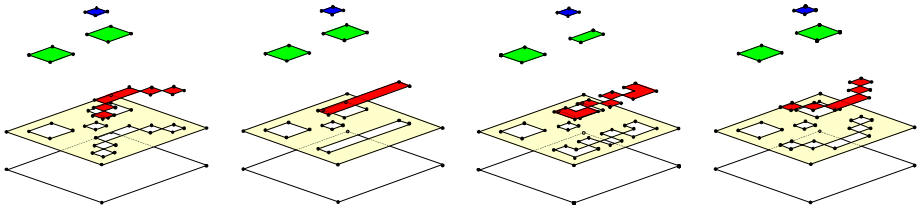
```
    hvl = ReadHvl(p)
    while(Not(EndEVM(p)))
         Fcurr = GetSection(Fprev, hvl)    // We get the next 3D frame.
         // We proceed to display the current frame in the animation.
         while(Not(EndEVM(Fcurr)))
                 // Get the common coordinate of the vertices in the next 2D
                 // couplet to be extracted from Fcurr.
                 color = GetCurrentCoord(Fcurr)
                 g.setColor(color)
                 hvlF = ReadHvl(Fcurr)
                 // Rectangles in the 2D couplet are displayed.
                 DisplayPolygons(hvlF, g)
         end-of-while
         Fprev = Fcurr
         hvl = ReadHvl(p)                  // Read next 3D couplet.
    end-of-while
end-of-procedure
```

Algorithm 3.2. Displaying a color 2D-animation represented through a 4D-OPP and the EVM.

By representing a given color 2D-animation using a 4D-OPP p and its 4D-EVM we have the following characteristics [9]:

- The sequence of the projections of sections in p corresponds to the sequence of 3D frames, i.e., $\pi_4\left(S_k^4(p)\right) = F_k$.
- Computation of 3D frames: Because p is expressed through the EVM then by **Corollary 2.2** the 3D-EVM of the frame F_k is computed by $EVM_3(F_k) = EVM_3(F_{k-1}) \otimes EVM_3\left(\pi_4\left(\Phi_k^4(p)\right)\right)$.
- Displaying the 2D colored animation: Each couplet perpendicular to the X_3 axis in each 3D frame F_k contains the polygons to display. The colors to apply to those polygons are referred through the X_3 coordinate that contains the integrated red-green-blue components.

In the **Figure 7** are presented the sequences of couplets of the 3D frames F_k for the 2D animation presented in **Figure 1**.



F_1's 2D couplets F_2's 2D couplets F_3's 2D couplets F_4's 2D couplets

Figure 7. The sequences of couplets of the 3D frames that represent a color 2D-animation.

The **Algorithm 3.2** applies the above ideas in order to extract animation colored 2D frames from a 4D-OPP and display them. Basically it extracts the 3D couplets perpendicular to X_4-axis and computes the sections that correspond to the extrusion to 3D space of the animation's 2D frames. When the extrusion of a frame is obtained then its 2D couplets perpendicular to X_3-axis are extracted. Such 2D couplets are the polygons to draw and their filling color is assigned according to their common X_3 coordinate in the 3D frame. A 2D couplet is processed through the procedure *DisplayPolygons* in the algorithm.

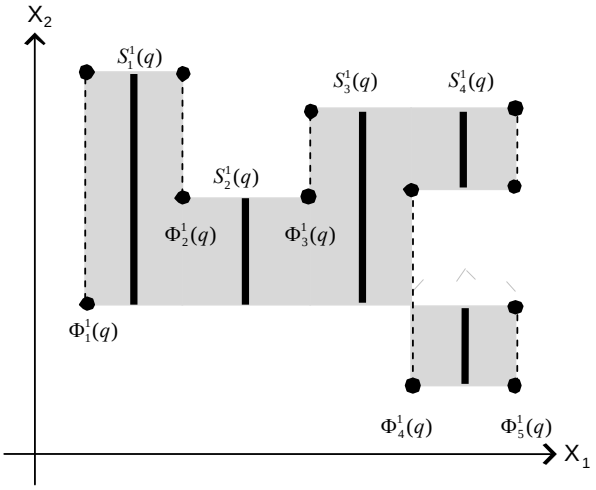


Figure 8. A 2D-OPP q whose composing rectangles are being computed. The coordinates of a rectangle in $Slice_i^1(q)$ are given by the coordinates of the projection of $S_i^1(q)$ and common coordinates of its bounding couplets $\Phi_i^1(q)$, Φ_{i+1}^1.

DisplayPolygons is implemented in **Algorithm 3.3**. In order to draw the rectangles that compose an input 2D-OPP we will consider the partition induced by its Slices. A Slice from a 2D-OPP can be seen as a set of one or more disjoint rectangles whose 1D base is the slice's section. The coordinates that define an specific rectangle in $Slice_k^1(p)$ can be determined through its respective section $S_k^1(p)$ (the 1D base of $Slice_k^1(p)$) and the common coordinates of $\Phi_k^1(p)$ and $\Phi_{k+1}^1(p)$, i.e., the common coordinates of the 1D-couplets that bound section $S_k^1(p)$. See **Figure 8**.

```
Input:  A 2D-EVM p and the graphics context g where p is going to be displayed.
Output: True if and only if the number of dimensions of p is 2.
        False if the number of dimensions of p is not 2, hence, no elements of p
        were displayed.
Procedure DisplayPolygons(EVM p, g)
    if (GetN(p) ≠ 2) then
        return False
    EVM hvl           // Current 1D couplet in p.
    EVM S_i, S_j      // Previous and next sections about hvl.
    S_i = InitEVM( )
    int rectangleX[4] // Coordinates along X_1-axis of a rectangle to be displayed.
    int rectangleY[4] // Coordinates along X_2-axis of a rectangle to be displayed.
    int point1, point2   // Two consecutive points in 1D section S_j.
    if (Not(IsEmpty(p))) then
        double prevCoord = GetCurrentCoord(p)
        hvl = ReadHvl(p)
        while(Not(EndEVM(p)))
           S_j = GetSection(S_i, hvl)       // Current section is an 1D-OPP.
           // We extract the ordered sequence of points in 1D section S_j.
           int points[ ] = GetEVM(S_j)
           k = 0
           while(k < points.size)
                // Each segment in the 1D current section is extruded and
                // displayed.
                point1 = points[k]
                point2 = points[k+1]
                // prevCoord and GetCurrentCoord(p) are the X_1-coordinates about
                // section S_j.
                rectangleX[0] = prevCoord
                rectangleY[0] = point1
                rectangleX[1] = prevCoord
```

```
                rectangleY[1] = point2
                rectangleX[2] = GetCurrentCoord(p)
                rectangleY[2] = point2
                rectangleX[3] = GetCurrentCoord(p)
                rectangleY[3] = point1
                // We display the rectangle.
                g.fillPolygon(rectangleX, rectangleY, 4)
                k = k + 2
            end-of-while
            prevCoord = GetCurrentCoord(p)
            Si = Sj
            hvl = ReadHvl(p)
        end-of-while
    end-of-if
    return True
end-of-procedure
```

Algorithm 3.3. Displaying the rectangles that compose a 2D-OPP expressed through the EVM.

3.1 Experimental Results

We evaluated our procedure through two blue screen video sequences which were produced originally at a TV studio of the University of Arts in Bremen [3]. Such sequences are AVI XVID codified videos (720 × 576, 24 bits color). We converted such sequences, for our experiment, to videos with resolution of 320 × 240 pixels (standard TV) and 64 colors.

The first sequence was composed by 146 frames. The 4D-OPP that represented such set of selected frames has 848,598 extreme vertices. In another experimented case we considered a second movie sequence whose time length was 100 frames. The size of the 4D-EVM corresponding to its codification as a 4D-OPP required 1,472,174 extreme vertices.

As can be noted, in the first referenced sequence we required 848,598 extreme vertices for representing 146 animation frames, while in the second sequence we required 1,472,174 extreme vertices for representing 100 frames. The reason behind this behavior was yet identified in [2]: $EVM_3(F_k) = EVM_3(F_{k-1}) \otimes EVM_3\left(\pi_4\left(\Phi_k^4(p)\right)\right)$, i.e., the regions at couplets $\Phi_k^4(p)$ represent the regions of a previous frame F_{k-1} that need to be modified in order to update it to the following frame F_k. In other words, a couplet perpendicular to X_4-axis $\Phi_k^4(p)$ only stores the differences between consecutive 3D frames F_{k-1} and F_k. The way the frames change through time has impact over the number of extreme vertices in the couplets associated to the 4D-OPP that represents the animation. The first animation contains a girl who is sat and working with a computer. As seen in **Figure 9**, the girl, along time, is practically quiet. Hence, we have a lot of redundancy between all frames in the animation. Therefore, only minimal differences are stored in the OPP's couplets, except the first and last couplets, whose visualization coincide with the first and last frames in the original animation. On the other hand, the second animation is a sequence where the girl is jumping and dancing along the screen from right to left (See **Figure 10**). In this case we have a level of redundancy that is minor than the one found in the first animation because we have more noticeable changes between consecutive frames.

According to this experiment we can conclude that the EVM's conciseness, respect to the representation of animations, depends of the degree of redundancy between the frames associated to the animations. As noted in [4], cartoon animations are a good example of animations with an elevated redundancy, but we are one step further by considering sequences with complex color and gray scale frames.

Figure 9. Five main frames taken from the first animation used for conversion to the 4D-EVM: 848,598 extreme vertices were required for encoding 146 frames (original sequence taken from [3]).

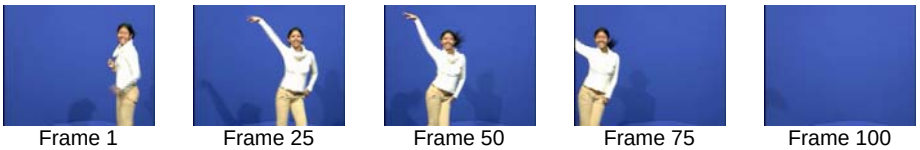


Figure 10. Five main frames taken from the second animation used for conversion to the 4D-EVM: There were required 1,472,174 extreme vertices for encoding 100 frames (original sequence taken from [3]).

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 [9] for more details). In this aspect we mention the development of other "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 nD-EVM, consider: (1) a method for comparing images oriented to the evaluation of volcanoes' activity; (2) the way the nD-EVM enhances Image Based Reasoning; (3) the manipulation and extraction of information from 3D datasets (see also [10]), and finally, (4) an application to collision detection between 3D objects through the nD-EVM.

There are many aspects related to the procedure we have described in this work that can be improved. An idea to consider is concerned to the non-supervised detection of polygons in the 2D video sequences to be vectorized. As commented in **Section 1**, Koloros & Zára [4] and Kwatra & Rossignac [6] consider the detection of polygons as a core step in their respective methodologies. In the case of [4] detected polygons in one frame are evaluated with the remaining frames in order to identify repeated or similar polygons with the objective to compress the final representation of

the vectorized animation. In [6] detected polygons are considered 3D volumes whose third dimension is given by time. Finally, through their Edgebreaker compression the evolution across time of the polygons are encoded as volume geometry. We are open to consider the ideas given in [4] and [6] in order to provide much more compression

We commented in **Section 3.1** the results obtained from the vectorization of two video sequences. It is well known that Telecine process is used to transform motion picture film into digital video format [4]. For reproduction PAL or NTSC standard under the respective resolution of 720×576 or 720×480 pixels is used and the final data is commonly stored at betacam tapes or di-rectly in the computer. The use of MPEG-1 or MPEG-2 commonly results in lower quality with arti-facts that make the vectorization difficult [4], and in our case, impacts the cardinality of the obtained EVM's. According to the use of MPEG-4 codification we can expect better quality at reasonable data-rate. For our intention to vectorize video data we need the best possible quality. We will test our procedure with video sequences codified in MPEG-4 (yet available in the next generation DVD formats such as HD-DVD [7]) in order to prove that by obtaining better quality in our video sources we obtain better compression of the final vectorized video represented under the EVM.

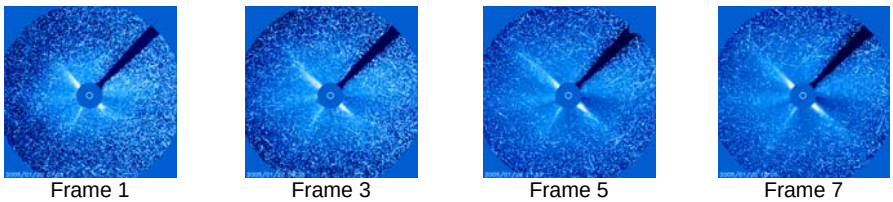


Figure 11. A sequence of frames that presents a coronagraph image of a radiation storm (taken from [5]).

We will finalize by establishing a question to be addressed. According to our experiments and by the fact that a couplet perpendicular to X_4-axis $\Phi_k^4(p)$ only stores the differences between consecutive 3D frames F_{k-1} and F_k. we conclude that by more level of redundancy between consecutive frames we can expect more conciseness from the EVM. However, scientific sequences provide us with a huge set of examples where the degree of redundancy is very low or inexistent. Consider for example the sequence presented in **Figure 11**. Such sequence describes a coronagraph image of a radiation storm which took place in January 20, 2005 [5]. The main characteristic in this sequence shows that the value of a pixel (inside the main circle) in a frame is distinct from the value of that same pixel in the next frame. Because of the accuracy required when these sequences are analyzed, any kind of threshold, which could elevate the redundancy degree, is prohibited. Hence, the cardinality of the EVM's that represent these sequences is expected to be high. The procedures we have described consider the representation of a frame by considering the original two spatial dimensions for each one of its pixels. A possible solution to the addressed problem could consider the linearization of each frame. That is, a frame can be considered as a matrix but by stacking its columns on top of one another we obtain a vector. In this way we have that each pixel can be referenced by only one coordinate in the vector. Hence, we deal with only one spatial dimension instead of the original two dimensions. Obviously we have the way to recover the original position of a pixel

given the original width and height of its frame. It can be observed that a 3D-OPP can represent our animation: X_1-axis will correspond to the position in the linearization, X_2-axis will refer to its red-green-blue integrated components, and X_3-axis will be associated to time. **Algorithms 3.1** to **3.3** should be modified to consider an OPP that represents a set of linearized frames. In the future we will discuss these described ideas and the obtained results.

References

1. Aguilera, Antonio. & Ayala, Dolors. 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, Antonio. Orthogonal Polyhedra: Study and Application. PhD Thesis. Universitat Politècnica de Catalunya, 1998.
3. Center for Computing Technologies, Digital Media/Image Processing, University of Bremen. Web site: http://www.tzi.de/tzikeyer/index.html
4. Koloros, Martin & Zára, Jirí. Coding of vectorized cartoon video data. Proceedings of Spring Conference on Computer Graphics 2006, pp. 177-183. Comenius University, Bratislava, 2006.
5. Koppeschaar, Carl. Astronet's Web Site: http://www.xs4all.nl/~carlkop/auralert.html
6. Kwatra, Vivek & Rossignac, Jarek. Space-Time surface simplification and Edgebreaker compression for 2D cel animations. International Journal of Shape Modeling, vol. 8, No. 2, December 2002.
7. Moeritz, S. & Diepold, K. Understanding MPEG 4: Technology and Business Insights. Focal Press, 2004.
8. Pérez-Aguila, Ricardo. 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.
9. Pérez-Aguila, Ricardo. Orthogonal Polytopes: Study and Application. PhD Thesis. Universidad de las Américas - Puebla. Cholula, Puebla, México, November 13, 2006.
10. Pérez-Aguila, Ricardo. Modeling and manipulating 3D Datasets through the Extreme Vertices Model in the n-Dimensional Space (nD-EVM). Accepted and to appear in the First International Conference on Industrial Informatics, CICINDIN 2007. To be held in México City, México, November 5 to 9, 2007.
11. Spivak, M. Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus. HarperCollins Publishers, 1965.