Orthogonal Polyhedra in *3D Time-Color Space* as a Geometric Model for Representation of Video Sequences with Low or Inexistent Redundancy between Frames

Ricardo Pérez-Aguila; Universidad Tecnológica de la Mixteca (UTM), Carretera Huajuapan-Acatlima Km 2.5; Huajuapan de León, Oaxaca/México: 69000

Abstract

Several video compression methods have in common their conciseness depends of the degree of redundancy between the frames associated to the sequences they describe. Cartoon animations are a good example of animations with an elevated redundancy and from which an elevated compression is expected. However, scientific sequences provide us with a huge set of examples where the degree of redundancy is very low or inexistent. Because of the accuracy required when these sequences are analyzed, any kind of threshold, which could elevate the redundancy degree, is prohibited. In this work, a proposed solution to the addressed problem considers 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 a vector is obtained. In this way each pixel can be referenced by only one coordinate in the vector. Starting from these considerations a geometric representation is built: A video sequence will be associated to a unique 3D Orthogonal Pseudo-Polyhedron (3D-OPP). Such 3D-OPP is embedded in a 3D Time-Color Space where X_1-axis corresponds to the position of pixels in the linearization, the second spatial dimension (X_2-axis) is associated to the color value of a pixel, and finally, X_3-axis describes the displaying time of frames in the original animation. Such 3D-OPP can be compressed, manipulated, and displayed in screen by expressing it according to the Extreme Vertices Model in the n-Dimensional Space (nD-EVM). The nD-EVM shares the representation of n-Dimensional Orthogonal Pseudo-Polytopes (nD-OPPs) by considering only a subset of their vertices.

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-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, in [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 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-OPPs vertices, there is much more information about the polytope hidden within this subset of vertices. In the following sections there will be described basic procedures and algorithms in order to obtain some of this information.

It is well known that several video compression methods were invented to be able to effectively store video data on common digital media. Their conciseness depends of the degree of redundancy between the frames associated to the sequences they describe. As noted in [4], cartoon animations are a good example of animations with an elevated redundancy where these compression methods can be applied expecting almost an optimal performance. 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 [4]. Another example is given by the work of Kwatra and Rossignac [6]. In their approach each region in a frame 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. The outputs given by the methods provided by these authors ([4] & [6]) are *vectorized video sequences:* animations described in terms of geometrical elements (polygons, lines, volumes, etc.) whose visualization on screen depends on the temporal dimension. However, these authors did not address the problem of vectorization for complex color and gray scale image sequences. A contribution for addressing the last comment was given in [11]. In such case, each frame in a color video is initially represented as a set of orthogonal polygons whose displaying time depends on the temporal dimension. Hence, a vectorized 2D color video sequence is expressed as a 4D Orthogonal Pseudo-Polytope (4D-OPP) whose first two dimensions corresponds to pixels' original coordinates in a frame, the third dimension is associated to the color to apply to such pixels, and finally the fourth dimension describes time. The size and compression of such 4D-OPP depend on the redundancy degree between the frames. As can be seen, the above methods are sustained by the hypothesis the sequences represented contain a high level of redundancy. Obviously, it can be concluded that by more level of redundancy more conciseness from these procedures can be expected.

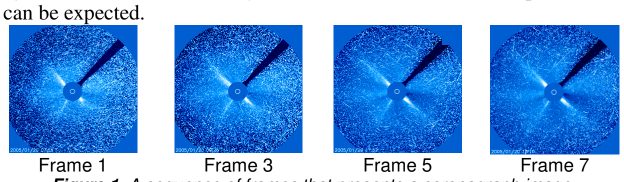

Figure 1. *A sequence of frames that presents a coronagraph image of a radiation storm (taken from [5]).*

An interesting situation comes from the fact that scientific sequences provide us with a huge set of examples where the degree of redundancy between frames is very low or inexistent. Consider for example the sequence presented in Figure 1. 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.

It is common to think of a frame by considering its 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 a vector is obtained. In this way each pixel can be referenced by only one coordinate in such vector. Hence, we deal with only one spatial dimension instead of the original two dimensions. Obviously, there is a 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, embedded in what we call *3D Time-Color Space*, can represent an animation: X_1-axis will correspond to the position of pixels in the linearization, X_2-axis will refer to their red-green-blue integrated components, and X_3-axis will be associated to time. The compression, manipulation, and displaying in screen of the represented sequences is performed through the 3D-EVM. The proposed methodologies will be described in the third main section.

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

Preliminary Background: n-Dimensional Orthogonal Pseudo-Polytopes

Definition 2.1: *A Singular n-Dimensional Hyper-Box in $\mathbb{R}^n$ is the continuous function* $I^n : [0,1]^n \rightarrow [0,1]^n$

$$x \sim I^n(x) = x$$

Definition 2.2: *For all i, 1 ≤ i ≤ 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*

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

Definition 2.3: *In a general singular nD hyper-box c the (i,α)-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 ≤ k ≤ n, for a closed set A is called a k-chain.*

Definition 2.7 [14]: *Given a singular nD hyper-box I^n the (n-1)-chain, called the boundary of I^n, is given 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 [14]: *Given a singular general nD hyper-box c the (n-1)-chain, called the boundary of c, is defined 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 [14]: *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*

$$\left[\bigcap_{\alpha=1}^{k} c_\alpha([0,1]^n) = \underbrace{(0,...,0)}_{n}\right] \wedge$$

$$\left[(\forall i,j,\ i \ne j,\ 1 \le i,j \le k)\left(c_i([0,1]^n) \ne c_j([0,1]^n)\right)\right]$$

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: *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.*

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: *The Extreme Vertices of an nD-OPP p are the ending vertices of all the brinks in p. EV(p) will denote to the set of Extreme Vertices of 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 it should be assumed that the coordinates of extreme vertices in the Extreme Vertices Model of an nD-OPP p, $EVM_n(p)$ are sorted according to one fixed ordering taken from the possible n! permutations.

Definition 2.15: *Let p be an nD-OPP. The Extreme Vertices Model of p, denoted by $EVM_n(p)$, is defined as the model as only stores to all the extreme vertices of p.*

Sections and Slices of nD-OPPs

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 ∂(p).*

Definition 2.17: *The Projection Operator for (n-1)D cells, points, and set of points is respectively defined 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* $v = (x_1,...,x_n)$ *a point in* $\mathbb{R}^n$. *The projection of v in the (n-1)D space, denoted by* $\underline{\pi_j(v)}$, *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* $\underline{\pi_j(Q)}$, *is defined as the set of points in* $\mathbb{R}^{n-1}$ *such that*

$$\pi_j(Q) = \left\{p \in \mathbb{R}^{n-1} : p = \pi_j(x),\ x \in Q \subset \mathbb{R}^n\right\}$$

Where $\hat{x}_j$ *is the coordinate corresponding to X_j-axis to be suppressed.*

Definition 2.18: *Consider an nD-OPP p:*

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

Definition 2.19: *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 X_i -axis, 1 ≤ i ≤ 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.* $\underline{S^i_k(p)}$ *will refer to the k-th section of p between* $\Phi^i_k(p)$ *and* $\Phi^i_{k+1}(p)$, *1 ≤ k < np_i.*

Computing Couplets and Sections

Theorem 2.1 [9]: *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 2.2 [9]: *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* $\pi_i\left(\Phi^i_k(p)\right)$.

The Regularized XOR operation on the nD-EVM

Theorem 2.3 [2]: *Let p and q be two nD-OPPs having* $EVM_n(p)$ *and* $EVM_n(q)$ *as their respective EVMs 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-OPPs sections from couplets and vice-versa, by means of their corresponding Extreme Vertices Models. These formulae are obtained by combining **Theorem 2.3** with **Theorem 2.1**; and **Theorem 2.3** with **Theorem 2.2**, 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]:

$$EVM_{n-1}\left(\pi_i(S^i_k(p))\right) = EVM_{n-1}\left(\pi_i(S^i_{k-1}(p))\right) \otimes EVM_{n-1}\left(\pi_i(\Phi^i_k(p))\right)$$

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-OPPs when that specific situation is met.

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

Table 1. Primitive operations in the nD-EVM.

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 X_1-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 X_1-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 X_1-axis, to 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 X_1-coordinate to coord on every vertex of the (n-1)D couplet p. For coord=0 it performs the projection $\pi_1(p)$. }
Input: Two nD-EVMs 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. }
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 X_1-coordinate of the next (n-1)D couplet to be extracted from p. }
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:** An integer **Procedure** GetN(EVM p) { Returns the number of dimensions of the space where p is embedded. }

Basic Algorithms for the nD-EVM

According to previous sections the primitive operations shown in Table 1 can be defined based in the functions originally presented in [2]. Function MergeXor, according to **Theorem 2.3**, performs an XOR between two nD-EVMs, 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].

```
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
```

Algorithm 1. Computing sections from couplets

```
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 2. *Computing couplets from sections*

From the above primitive operations, **Algorithms 1** and **2** are derived. The **Algorithm 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 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 nD-EVM p.
Procedure EVM_to_SectionSequence(EVM p)
    EVM hvl    // Current couplet.
    EVM $S_i$, $S_j$  // Previous and next sections about hvl.
    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
```

Algorithm 3. *Computing the sequence of sections from an nD-OPP p represented through the nD-EVM.*

Representing Color 2D Videos through 3D-OPPs and the EVM

We start the presentation of our methodologies by establishing some conventions:

- The **linearization** of a matrix Δ of size n × m converts the matrix into a vector of size n·m by stacking the columns of the matrix Δ on top of one another. Formally:

$$Lin(\Delta) = Lin\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}$$
$$= \left(a_{1,1}, a_{2,1}, \ldots, a_{n,1}, \ldots, a_{1,m}, a_{2,m}, \ldots, a_{n,m}\right)^T$$

- Each frame in the animation is labeled as f_k and *m* will be the number of such frames.
- A frame will be seen as a matrix of size xSize × ySize where the red-green-blue components of each pixel are integrated into a single value [12]. 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 red value and bits 24-31 to the *alpha* (transparency) value (Figure 2.a).

A color animation can be represented by a 3D-OPP in the following way:

a) Lin(f_k) is computed. It is a vector of size xSize · ySize (Figure 2.b).

b) Each pixel is associated to a segment whose coordinates along X_1–axis correspond to its respective coordinates in Lin(f_k). Such segment is extruded towards the second dimension where the value integrating its red-green-blue components now will provide its coordinates along X_2–axis [12].

Let $xf_{k,Lin}$ be the set composed by the rectangles (the extruded pixels) of each extruded linearized frame Lin(f_k) (see Figure 2.c). It is very important to avoid the zero value in the X_2 coordinate because a pixel could not be extruded and therefore its associated rectangle (a 2D-OPP) will not be obtained [12].

c) Let $rectangle_i$ be a rectangle in the set $xf_{k,Lin}$ and *npr* the number of rectangles in that set. Due to all the rectangles in $xf_{k,Lin}$ are quasi disjoint 2D-OPPs, the final 2D-OPP can be easily obtained. The respective 2D-EVM of the whole 2D linearized frame is obtained by computing the regularized union of all the rectangles in $xf_{k,Lin}$. Then, according to **Corollary 2.3**, we have to apply (all the vertices in a $rectangle_i$ are extreme):

$$EVM_2(F_{k,Lin}) = \bigotimes_{i=1}^{npr} EVM_2(\text{rectangle}_i \in xf_{k,Lin})$$

where $F_{k,Lin}$ is the 2D linearized frame (a 2D-OPP) that represents the union of all the rectangles in $xf_{k,Lin}$ (see Figure 2.d).

d) Let us extrude $F_{k,Lin}$ into the third dimension, and thus obtain a 3D prism $prism_k$ whose bases are $F_{k,\text{Lin}}$ and its length is proportional to the time the original frame f_k is to be displayed. The new third dimension will measure and represent the time (see Figure 3).

e) Let $p = \bigcup_{k=1}^{m} prism_k$, then *p* is a 3D-OPP that represents the given color 2D-animation by linearizing its frames. Due to all the *m* prisms are quasi disjoint 3D-OPPs, then the 3D-EVM for *p* can be obtained by: $EVM_3(p) = \bigotimes_{k=1}^{m} EVM_3(prism_k)$

```
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 3D-EVM of the polyhedron that codifies frames, in
          linearized mode, in the input animation.
Procedure
GenerateLinearized-3D-EVM-movie(Movie animation, xSize, ySize)
  /* The EVM that will store and codify the input animation. */
  EVM evmMovie
  EVM hvl              // A 2D-couplet to be inserted in the output EVM.
  // Previous and current linearized frames in the animation.
  EVM Fcurr, Fprev
  real t  // The amount of time that the processed frame is displayed.
  /* Position that a pixel in an original frame will occupy in the
     linearized frame.*/
  int linPosition
  Fprev = InitEVM( )
  for each frame in animation do
    Frame f = animation.nextFrame( )
    t = animation.getDisplayingTime( )
    /* Frame t is linearized and extruded towards the 2D space and
       its 2D-EVM is computed. */
    linPosition = 0
    Fcurr = InitEVM( )
    for x = 0 until xSize - 1 do
      for y = 0 until ySize - 1 do
        rgb = getColor(x, y, f)
        /* It is obtained the EVM of the 2D rectangle associated to
           point (x1position, rgb) */.
        EVM rectangle = GetRectangleEVM(linPosition, rgb)
        Fcurr = MergeXor(Fcurr, rectangle)
        linPosition++
      end-of-for
    end-of-for
    //The Xor, between current and previous 2D frames, is performed.
    hvl = MergeXor(Fcurr, Fprev)
    /* Amount of time t associated to linearized frame Fcurr is
       attached to current 2D couplet. */
    SetCoord(hvl, t)
    /* A new 2D couplet is attached to the 3D-OPP that codifies, in
       linearized mode, the input animation. */
    PutHvl(hvl, evmMovie)
    Fprev = Fcurr
  end-of-for
  return evmMovie
end-of-procedure
```

Algorithm 4. *Codifying a Color 2D-animation, in linearized way, through a 3D-OPP and the EVM.*

The **Algorithm 4** shows the procedure for converting a set of frames in an animation to a 3D-OPP that codifies them in linearized fashion. Such OPP is represented through a 3D-EVM.

By expressing a given color 2D-animation using linearization of frames and its 3D-EVM we have the following characteristics:

- The sequence of the projections of sections in p corresponds to the sequence of 2D linearized frames, i.e., $\pi_3\left(S_k^3(p)\right) = F_{k,Lin}$.
- Computation of 2D linearized frames: Because p is expressed through the EVM then by **Corollary 2.2** the 2D-EVM of the linearized frame $F_{k,Lin}$ is computed by

$$EVM_2\left(F_{k,Lin}\right) = EVM_2\left(F_{k-1,Lin}\right) \otimes EVM_2\left(\pi_3\left(\Phi_k^3(p)\right)\right)$$

The **Algorithm 5** applies the above ideas in order to recover animation colored 2D frames from a 3D-OPP and displays them. It extracts the 2D couplets perpendicular to X_3-axis and computes the sections that correspond to the extrusion to 2D space of the animation's linearized frames. When the extrusion of a linearized frame is obtained then its 1D couplets perpendicular to X_2-axis are extracted. Such 1D couplets are the segments to draw and their color is assigned according to their common X_2 coordinate in the 2D frame. A 1D couplet is drawn in its correct position through the procedure *DisplaySegments*.

The **Algorithm 6** corresponds to procedure *DisplaySegments*. It works only for 1D-OPPs. It proceeds to extract the initial and final coordinates along X_1-axis of each one of the segments in the input 1D-OPP. Such coordinates are labeled as S_1 and S_2. In fact, the values of S_1 and S_2 bound a set of adjacent pixels with the same color in the linearization of the original frame. The length of the segment is computed through $S_2 - S_1$. Each one of these pixels needs to be located and drawn in their original positions in the frame. There will be recovered the 2D coordinates of the first point of the segment to be painted. Its coordinates along X_1 and X_2 axes, in pixels space, are respectively inferred through:

$$x_1 = \lfloor s_1 / ySize \rfloor \qquad x_2 = s_1 \bmod ySize$$

where ySize is the height of the original animation. Next, we start to move along X_2-axis iterating S_2–S_1 times. In each one of these iterations a pixel with coordinates (x_1, x_2) is drawn and x_2 is updated by adding 1 to it. It could be the case a segment has to be broken because a set of contiguous pixels in the linearized frame is in fact two sets of pixels in the original frame: the first set occupy the last positions of a column while the second is occupying the first positions of the next column (it is possible, in fact, each set could be completely occupying their corresponding columns). In Figure 2 an instance of the situation is shown. For example, columns 8 and 9 from Figure 2.a are joined, in Figure 2.b, as a set of 12 contiguous pixels with the same color. In this situation, during the drawing of pixels, if the value of x_2 is equal to ySize then it is updated to $x_2 = 0$ and x_1 is updated by adding 1 to it in order to relocate the drawing towards the next column starting from its first position.

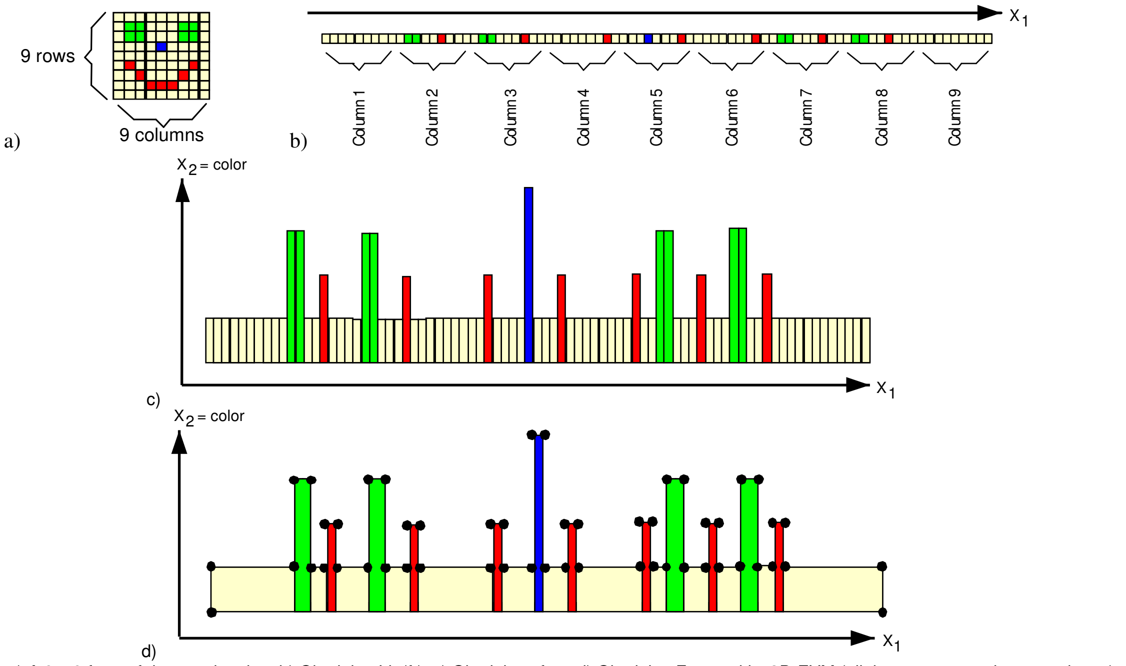

Figure 2. *a) A 9 × 9 frame f_k in an animation. b) Obtaining Lin(f_k). c) Obtaining $xf_{k,Lin}$. d) Obtaining $F_{k,Lin}$ and its 2D-EVM (all the extreme vertices are shown).*

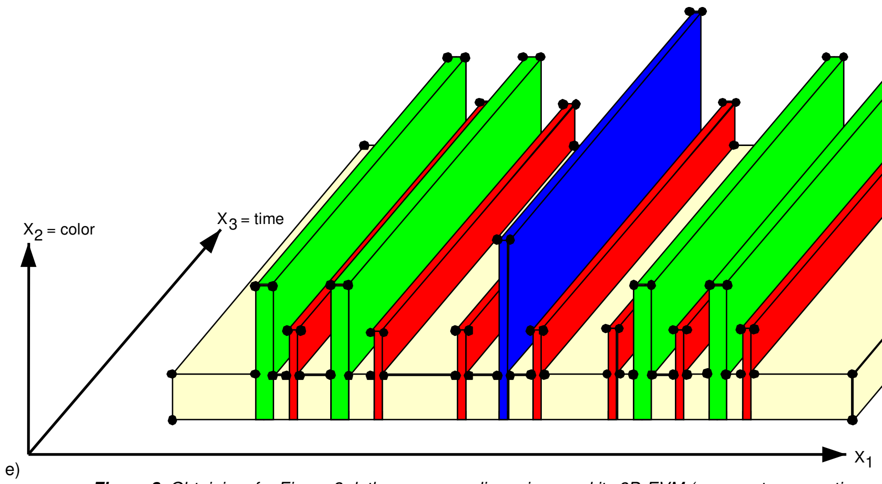

Figure 3. *Obtaining, for Figure 2.d, the corresponding prism$_k$ and its 3D-EVM (some extreme vertices are shown).*

```
Input:     A 3D-EVM p that represents, in linearized mode, a color
           2D-animation.
           The values xSize and ySize corresponding to the resolution
           of the animation to be visualized.
           The graphics context g where the animation is going to be
           displayed.
Procedure PlayLinearized-3D-EVMmovie(EVM p, xSize, ySize, g)
    EVM hvl    // Current 2D couplet in p.
    // Previous and current 2D linearized frames in the animation.
    EVM Fcurr, Fprev
    Fprev = InitEVM( )
    hvl = ReadHvl(p)
    while(Not(EndEVM(p)))
        // The next 2D linearized frame is obtained.
        Fcurr = GetSection(Fprev, hvl)
        if (Not(IsEmpty(Fcurr))) then
            int color = 0
            /* hvl_fcurr will be a 1D-OPP. This 1D-couplet will
               contain the segments to be painted. */
            EVM hvl_fcurr
            while(Not(EndEVM(Fcurr)))
                /* Get the common coordinate of the vertices in
                   the next 1D couplet to be extracted.*/
                color = getCurrentCoord(Fcurr)
                hvl_fcurr = ReadHvl(Fcurr)
                // Couplet hvl_fcurr is an 1D-OPP.
                g.setColor(color)
                /* Segments in hvl_fcurr are painted with the
                   current color.*/
                DisplaySegments(hvl_fcurr, xSize, ySize, g)
            end-of-while
        end-of-if
        Fprev = Fcurr
        hvl = ReadHvl(p)
    end-of-while
end-of-procedure
```

Algorithm 5. *Displaying a color 2D-animation represented in linearized mode through a 3D-OPP and the EVM.*

Experimental Results

The described procedures were evaluated 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). Both sequences were converted, for the experiment, to videos with resolution of 320 × 240 pixels (standard TV) and 64 colors. The first sequence was composed by 146 frames. The 3D-OPP that represented such set of selected frames has 535,382 extreme vertices. In another experimented case, a second movie sequence was considered. Its time length was 100 frames. The size of the 3D-EVM corresponding to its codification required 1,183,728 extreme vertices.

As can be noted, in the first referenced sequence there were required 535,382 extreme vertices for representing 146 animation frames while in the second sequence 1,183,728 extreme vertices were required for representing 100 frames through 3D-OPPs. The reason behind this behavior was yet identified in [2] and [11]: $EVM_2(F_{k,Lin}) = EVM_2(F_{k-1,Lin}) \otimes EVM_2(\pi_3(\Phi_k^3(p)))$, i.e., the regions at couplets $\Phi_k^3(p)$ represent the regions of a previous frame $F_{k-1,Lin}$ that need to be modified in order to update it to the following frame $F_{k,Lin}$. In other words, a couplet perpendicular to X_3-axis $\Phi_k^3(p)$ only stores the differences between consecutive 3D frames $F_{k-1,Lin}$ and $F_{k,Lin}$. The way the frames change through time has impact over the number of extreme vertices in the couplets associated to the 3D-OPP that represents the animation. The first animation contains a girl who is sat and working with a computer. As seen in Figure 4, the girl, along time, is practically immobile. Hence, there is a lot of redundancy between all frames in the animation. Therefore, only minimal differences are stored in the OPPs 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 5). In this case There is a level of redundancy that is minor than the one found in the first animation because there are more noticeable changes between consecutive frames.

```
Input:     A 1D-EVM p that contains a set of segments which were
           codified through a linearized 2D frame.
           The values xSize and ySize corresponding to the resolution
           of the original animation.
           The graphics context g where the rectangles in p are going
           to be displayed.
Output:    True if and only if the number of dimensions of p is 1.
           False if and only if the number of dimensions of p is not 1,
           hence, elements of p were not displayed.
Procedure DisplaySegments(EVM p, xSize, ySize, g)
    if (p.getN( ) ≠ 1) then return False
    int s_1, s_2   // Initial and final points of a segment in p.
    int sLength    // The length of a segment in p.
    int x1, x2     // Coordinates along $X_1$-axis of a pixel to be painted.
    while(Not(EndEVM(p)))
        s_1 = getCurrentCoord(p)
        ReadHvl(p)
        s_2 = getCurrentCoord(p)
        ReadHvl(p)
        sLength = s_2 - s_1
        x1 = floor(s_1 / ySize)
        x2 = s_1 mod ySize
        for i = 0 until sLength - 1 do
            g.fillRect(x, y, 1, 1)
            y++
            if (y == ySize) then
                y = 0
                x++
            end-of-if
        end-of-for
    end-of-while
    return True
end-of-procedure
```

Algorithm 6. *Displaying the segments that compose a 1D-OPP expressed through the EVM. Such segments are associated to a linearized frame.*

According to this experiment it can be concluded that the EVM's conciseness, respect to the representation of animations in linearized way, depends of the degree of redundancy between the frames associated to such animations. This valuable property, identified previously in [2] and [11] where the EVM was also used for managing video sequences, is preserved in the present methodologies.

Consider the animation presented in Figure 6. Such video sequence corresponds to the visualization of blood flow through laser speckle flowgraphy [7]. It is composed by eleven frames whose resolution is 601 × 545. The animation is given in grayscale with 256 levels. In Figure 1 there were presented some frames of a sequence that presents a coronagraph image of a radiation storm [5]. Such animation contained seven frames with resolution 256 × 256 and 256 colors. Both animations share the characteristic that the value of a pixel in a frame is distinct from the value of that same pixel in the next frame. Hence, the level of redundancy to expect is very low or possibly inexistent. As commented previously, because of the accuracy required when these sequences are analyzed, any kind of threshold, which could elevate the redundancy degree, is prohibited. For example, in the animation from Figure 6 it could be required to identify micro-vessels in a given region of a patient's tissue. This kind of vessels could be detected in regions whose width is just one pixel. Hence, some threshold, in order to reduce the complexity of the images and for increasing the level of redundancy between them, could be not appropriate because vital information could be omitted. The sequence from Figure 6 required, according to our method, 6,708,552 extreme vertices for its representation through the 3D-EVM. The EVM associated to the 3D-OPP that describes the animation from Figure 1 required 1,089,040 extreme vertices. At this point is important to mention that the obtained representations, for videos from Figures 1, 4, 5 and 6 are unique and accurate because methods for elevating redundancy were not applied.

Finally, the conciseness of the final representations can be increased by taking in account some of the known methods for file compression. In our case we compressed the final files using GZIP standard [13]. Animations from Figures 1, 4, 5 and 6 have associated final files of sizes 2.8 MB (256 × 256, 256 colors), 1.43 MB (320 × 240, 64 colors), 3.26 MB (320 × 240, 64 colors), and 20.6 MB (601 × 545, 256 gray levels) respectively.

Conclusions and Future Work

This work has been possible because of 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*. The description given here for the nD-EVM is in fact a very brief description of the capabilities of the model because there have been 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, set membership classification operations, and measure 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.

This work has presented some results obtained from a proposed method for representing video sequences by considering 3D-OPPs embedded in *3D Time-Color Space* and finally expressing such OPPs through the 3D-EVM. Hence, the next logical step considers the application of the algorithms in the nD-EVM in order to extract useful information from the represented animations. We will study how a geometrical and/or topological interrogation to an EVM can share information and knowledge about the sequence that it represents.

Figure 4. *Three main frames taken from the first animation used for conversion to the 3D-EVM: There were required 535,382 extreme vertices for encoding 146 frames (original sequence taken from [3]).*

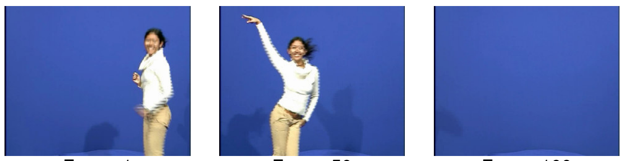

Figure 5. *Three main frames taken from the second animation used for conversion to the 3D-EVM: There were required 1,183,728 extreme vertices for encoding 100 frames (original sequence taken from [3]).*

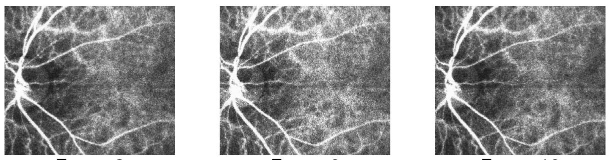

Figure 6. *Three main frames taken from an 11 frames animation obtained through laser speckle flowgraphy. This animation shares the visualization of blood flow in a given region (original sequence taken from [7]).*

References

[1] Aguilera, A. & Ayala, D. Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry. 4th ACM Siggraph Symposium on Solid Modeling and Applications SM'97, pp. 56-67. USA, 1997.
[2] Aguilera, A. 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, M. & Zára, J. Coding of vectorized cartoon video data. Proceedings of Spring Conference on Computer Graphics 2006, pp. 177-183. Comenius University, Bratislava, 2006.
[5] Koppeschaar, C. Astronet's Web Site: http://www.xs4all.nl/~carlkop/auralert.html
[6] Kwatra, V. & Rossignac, J. Space-Time surface simplification and Edgebreaker compression for 2D cel animations. International Journal of Shape Modeling, vol. 8, No. 2, December 2002.
[7] Laser Speckle Flowgraphy User Forum. Web site: http://leo10.cse.kyutech.ac.jp/lsfg/
[8] 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. México, May 2003.
[9] Pérez-Aguila, R. Orthogonal Polytopes: Study and Application. PhD Thesis. Universidad de las Américas-Puebla. México, 2006.
[10] Pérez-Aguila, R. Modeling and manipulating 3D Datasets through the Extreme Vertices Model in the n-Dimensional Space (nD-EVM). Journal Research in Computer Science, Special Issue: Industrial Informatics. Volume 31, 2007, pp. 15-31.
[11] Pérez-Aguila, R. Representing and Visualizing Vectorized Videos through the Extreme Vertices Model in the n-Dimensional Space (nD-EVM). Journal Research in Computer Science, Special Issue: Advances in Computer Science and Engineering. Volume 29, 2007, pp. 65-80.
[12] Pérez-Aguila, R.; Aguilera, A. & Lázzeri Menéndez, S. G. A Procedure for Comparing Color 2-Dimensional Images through their Extrusions to the 5-Dimensional Colorspace. Proceedings of the 15th International Conference on Electronics, Communications, and Computers CONIELECOMP 2005, pp. 300-305. Published by the IEEE Computer Society. ISBN: 0-7695-2283-1. México, 2005.
[13] Salomon, D. Data Compression: The Complete Reference. Springer, 1997.
[14] Spivak, M. Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus. HarperCollins Publishers, 1965.

Author Biography

Ricardo Pérez-Aguila received his BSc (2001), MSc (2003) and PhD (2006) degrees in Computer Science from the Universidad de las Américas-Puebla (UDLAP). In 2003-2006 he worked with the Actuarial Sciences, Physics and Mathematics Department at the same institution. In 2007 he incorporated as a full time professor/researcher at the Universidad Tecnológica de la Mixteca (UTM). His research interests consider the study of n-Dimensional Polytopes by analyzing their Visualization, Geometry, Topology, Representation, and Applications.