Representation and boundary extraction of a 3D digital image using the EVM model

J. Rodríguez and D. Ayala
Universitat Politècnica de Catalunya, Barcelona (Spain)
A. Aguilera
Universidad de las Américas-Puebla, Puebla (México)

Abstract

In recent published papers we have presented the Extreme Vertices Model (EVM), a concise and complete model for representing orthogonal polyhedra. This model exploits the simplicity of its domain by allowing robust and simple algorithms for set-membership classification and Boolean operations that do not need to perform floating-point operations.

A digital image is geometrically analogue to an orthogonal polyhedron and can therefore be represented with the EVM.

In this paper we discuss the performance in terms of space occupancy and algorithm complexity of representing digital binary images by this model. We also present a boundary extraction algorithm of a 3D digital image using the EVM and discuss its performance.

1 Introduction

A digital image can be represented as a map $f : Z \otimes Z \otimes Z \to R$, in such a way that every point (voxel) is assigned a value representing its color. In a binary image the image set is $[0, 1]$.

The model used to represent digital 3D images is an important issue to study. The facility to compact the model for storage or transmission purposes and to expand for computing or visualization purposes is an advantage that we have to expect from a model.

There are many demanding operations on digital images. One of them, needed for visualization and other purposes, is boundary reconstruction. There are manipulating operations as dilation, erosion or thinning and MAT computation, and operations which involve analysis as computing measurements like distances, angles, surfaces and volumes or obtaining the connected component labeling.

An orthogonal polyhedron, OP, is a polyhedron with all of its edges and faces oriented in three orthogonal directions. An orthogonal pseudo-polyhedron, OPP, is a regular OP with a non-manifold boundary. Digital 3D images are mathematically modeled by OPP. Figure 3 shows an OPP.

The EVM has been presented and developed in [2], among other papers, as a concise and complete model for representing orthogonal polyhedra. A digital image is geometrically analogue to an orthogonal polyhedron and can therefore be represented by the EVM.

In this paper we discuss the performance in terms of space occupancy and algorithm complexity of representing digital binary images by this model. We also present a boundary extraction algorithm of a 3D digital image using the EVM and discuss its performance.

The paper is arranged as follows. Next section shows a review of representation schemes and boundary extraction methods. The two following sections introduce the EVM and present the boundary extraction algorithm using it. The two next sections discuss the performance and the obtained experimental results using the EVM. The last section concludes the paper and outlines the future work.

2 Review of representation schemes and boundary extraction methods

2.1 Efficient representation schemes for 3D digital images

The conventional boundary extraction algorithms generate a polygonal mesh directly from the voxelization without using an intermediate model. In this group of algorithms we find the Cuberille, which represents the extracted surface as a collection of boundary faces belonging to boundary voxels. This raw representation scheme implies an important drawback when we want to operate with the object in surgical simulation and measurement purposes.

The developing of efficient methods to extract and operate volumes and surfaces from 3D digital images requires flexible and compact representation schemes which allow manipulating high resolution data sets as quickly and easily as possible. When designing models for storing high resolution data a number of trade-offs, involving computation necessary to access the data, overhead storage needed, and visual quality of display generated, must be considered.

We present a brief review of models oriented to improve the computation necessary to access the data and, in a more detail, models, such as Octree, Semiboundary and Shell, oriented to improve the manipulation and operation with objects generated from 3D digital images.

Accelerating the data access

Srihari [7] describes some of the most common representation schemes used to improve the computation required to access the data. The first one, known as Dope Vectors, is an exhaustive representation scheme which consists in a linearization of multidimensional arrays. The data elements occupy consecutive locations and fixed offsets and coefficients are precalculated. Thus, accessing an element requires only two multiplications and three additions. A second representation scheme oriented to avoid calculations in accessing data is the Marginal Indexing. This model is based on the idea of using a linked set of tables to store the data. In order to find the element (X,Y,Z), X is used as an index into the first table to get a pointer to a second table. Then Y is used as an index into a second table to get a pointer to a third table. Each block in the third table contains the data for the given values of X and Y and varying values of Z. By using Z as the index, the proper element can be found. When all data elements of a slice are identical, data is stored at that level and no pointer proceeds from the first level to the second. Similarly, when a strip is homogeneous. Therefore, homogeneous regions at a given level of the structure can be expressed by a single value saving space requirement.

The octree representation

The octree representation, also known as Symmetric Recursive Indexing [7] is a very efficient scheme to represent and operate 3D digital images. The octree is able to manipulate homogeneous regions of the volume as single entities. This representation scheme divides the cubic space into eight sub-cubes (octants) of equal volume. Each of these octants will either be homogeneous or have some nonuniformity. The heterogenous octants are further divided into sub-octants recursively until certain predefined criterion is reached. In the 3D digital images, this criterion is associated with segmentation thresholds. The octree can be used to visualize 3D digital images [6] and to accelerate the process of surface extraction from 3D images [10]. The octree allows to achieve Boolean operations between objects, but we need a common cubic universe for both objects. Nevertheless, useful operations in the image processing as dilation, erosion and thinning have not been dealt with this scheme.

The Semi-boundary representation

The Semi-boundary representation [8] created to store a segmented 3D digital image has become a very useful method in fast visualization, manipulation and analysis of structures generated from a 3D data set. This representation scheme only stores the boundary voxels of the image keeping the information of the interior voxels in an implicit way. The model consists of two data structures. The first one is a pointer array P with $h2 \times h3$ elements where, $h2$ and $h3$ are the dimensions of the volumetric data set in Y and Z directions respectively. The second one is an arranged list M which contains all the boundary voxels of the 3D image. Each boundary voxel is represented by a triplet $d = (d1, \Delta(d), \mathbf{n})$, where $d1$ is the first coordinate of d, $\Delta(d)$ is a 6-bit code representing the 6-neighborhood around d and $\mathbf{n}$ is the normal vector estimated at the center of d using an appropriate digital gradient operator.

In this representation each cell (j, k) of P addresses the starting location in M of the boundary voxels belonging to the column j on the slice k of the 3D image. Figure 1 illustrates an intuitive idea of the SB.

Although there is no comparative analysis published in [8], the authors assure that other data structures such octrees and run-length codes will not match the SB's efficiency in visualization and manipulation. However Boolean operations and complex cuts are relatively expensive operations using this scheme.

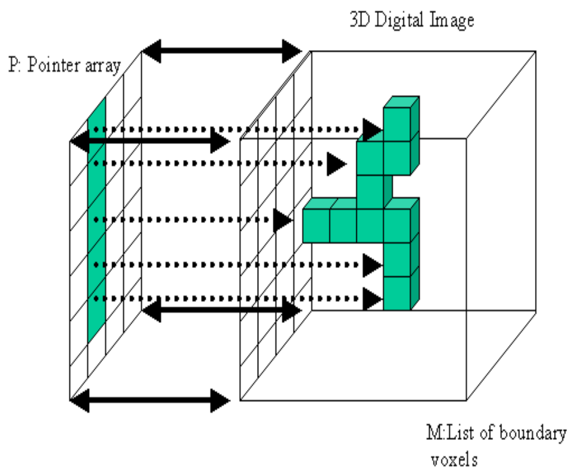


Figure 1: Illustration of the Semi-boundary

In order to represent objects with fuzzy boundaries Udupa and Odhner also proposed the Shell representation [9], an alternative scheme for 3D data sets which holds this property. This model has the same indexing scheme used in the semiboundary, but the set of voxels which belong to list M (Shell's voxels) was redefined. A Shell consists of those voxels whose opacity value exceeds the minimum opacity (hence it contributes to the rendered picture) and with, at least, one neighbor whose opacity value is less than the maximum opacity (hence the voxel is not completely surrounded by high-opacity voxels). The Shell is a representation scheme thought to improve the extraction of volumes using direct projection techniques and is not detailed in this paper.

2.2 Review of boundary extraction algorithms

There are many algorithms for constructing the boundary representation (B-Rep) of 3D digital images, but unfortunately we do not have an uniform criterion to classify them. However, grouping these algorithm according to the interpretation of the input data set becomes a suitable criterion.

First we can subdivide these algorithms into two distinct groups: 2D-based and 3D-based algorithms. The 2D-based approach treats the input data set as a stack of independent slices and the pixel is the basic element. On the other hand, the 3D-based techniques treat the input data set as a volumetric entity and the voxel becomes the basic element. These 3D-based algorithms can be further subdivided into two new categories. The first one holds the direct projection algorithms which treat the volumetric entity as a continue space. In this group we have Ray casting, Splatting and Shear-warp Factorization. The second one, consists of the algorithms which treat the volumetric entity as a discrete set.

This last category, known as surface rendering, data processing is based on the individual study of each voxel and its adjacency relationships around its neighborhood. In these techniques, voxels can be interpreted either as cuboids or grid's points on a 3D lattice and the basic boundary element can be represented either as a face or a vertex, depending on this interpretation. These two interpretations yield two different geometric representations of surfaces, and therefore two different kinds of B-Rep models: Block-form and Beveled-form. The algorithms which generate these representations are also called Block-form [4], [8] or Beveled-form [5] algorithms respectively. Figure 2 shows this classification.

The SB explained in 2.1 and the EVM presented in section 3 are both Block-form models.

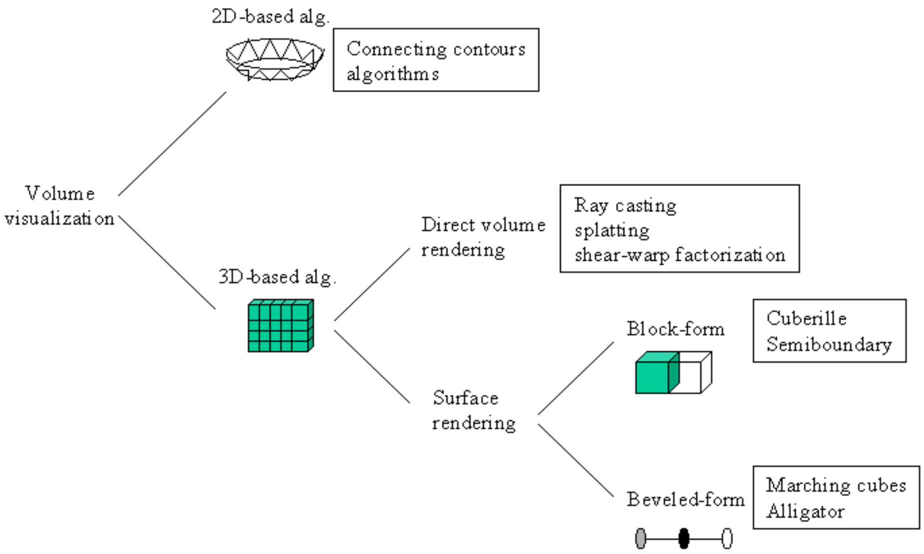


Figure 2: Classification of the volume rendering algorithms

3 Review of the EVM

The definitions and properties given in this section can be found more extensively explained and proved in [2].

3.1 Definitions

Let P be an orthogonal pseudo-polyhedron (OPP). A *brink* is the longest uninterrupted segment built out of a sequence of collinear and contiguous two-manifold edges of P.The ending vertices of a brink are called *extreme vertices, EV*. Figure 3 a) shows an OPP with a brink from vertex A to vertex E (both extreme vertices). In this brink, vertices B, C and D are non-extreme vertices.

The *Extreme Vertices Model, EVM*, is a representation scheme for OPP in which any OPP is represented by its (an only its) set of EV. In this model the set of EV can be ordered in six possible ways depending on the coordinate values: XYZ, XZY, YXZ, YZX, ZXY, and ZYX.

A *plane of vertices*, *plv*, is the set of vertices lying on a plane perpendicular to a main axis of P. A *slice* is the region between two consecutive planes of vertices. A *section* is the resulting polygon from the intersection between P and an orthogonal plane. Each slice has its representing section. Figure 3 a) shows an OPP with its planes of vertices and sections perpendicular to the X axis.

All these definitions can be extended to any dimension. In this paper we are concerned with dimension ≤ 3. In a plane of vertices (2D) we consider, then, its lines of vertices and its 2D slices with its corresponding 1D sections.

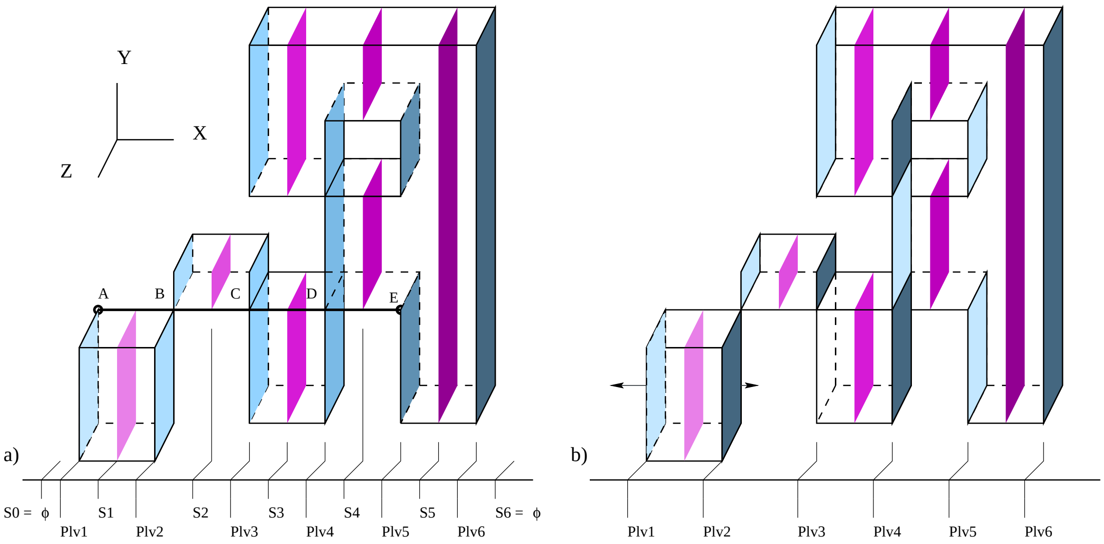


Figure 3: a) An OPP with a marked brink from vertex A to vertex E. Its planes of vertices and sections perpendicular to the X axis are showed in color blue and magenta respectively and color intensity increases from left to right. b) FD in dark blue (pointing to the right) and BD in light blue (pointing to the left).

3.2 Properties of the EVM

The first property concerning EVM is that coordinate values of non-extreme vertices may be obtained from EV coordinates.

The two next properties allow to compute sections from planes of vertices and vice-versa.

$$S_0(P) = \emptyset, \qquad \overline{S_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{plv_i(P)}, \forall i \in [1, np]$$
$$\overline{plv_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)}, \forall i \in [1, np]$$

$\overline{plv_i(P)}$ and $\overline{S_i(P)}$ denote the projections of $plv_i(P)$ and $S_i(P)$ onto a main plane parallel to P and $\otimes^*$ denotes the regularized XOR operation.

Applying the definition of the $\otimes$ operation, this last equation can be expressed as:

$$\overline{plv_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)} = (\overline{S_{i-1}(P)} -^* \overline{S_i(P)}) \cup (\overline{S_i(P)} -^* \overline{S_{i-1}(P)}), \forall i \in [1, np]$$

and, thus, we can decompose any plane of vertices into two terms that we will call *forward difference* or *FD*, for short $(\overline{FD_i(P)} = (\overline{S_{i-1}(P)} -^* \overline{S_i(P)})$ and *backward difference* or *BD* $(\overline{BD_i(P)} = (\overline{S_i(P)} -^* \overline{S_{i-1}(P)}))$.

The following property, based on these differences, guarantees that the correct orientation of all faces of P can be obtained from its EVM. $FD_i(P)$ is the set of faces on $plv_i(P)$ whose normal vector points to one side of the main axis perpendicular to $plv_i(P)$, while $BD_i(P)$ is the set of faces whose normal vector points to the other side. Figure 3 b) shows an OPP with its sections and with its FD and BD.

This property together with the first one allow to prove that the EVM is a complete model which means that all the geometry and connectivity information can be obtained from it.

The following property concerns to the XOR operation. Let P and Q be two d-D (d $\leq$ 3) OPP, having $EVM(P)$ and $EVM(Q)$ as their respective models, then $EVM(P \otimes^* Q) = EVM(P) \otimes EVM(Q)$. This property means that the XOR operations works in 0D, because is applied to the EV of the model.

General Boolean operations between two OPP can be carried out by applying recursively the same Boolean operation over the corresponding OPP sections. The corresponding algorithm is presented in [1] and consists in a geometric merger between the EVM of both operands which involve as a basic operation the XOR between EV.

4 Boundary extraction algorithm using EVM

The boundary extraction algorithm is a two-pass method. First it computes the EVM from the voxelization and then obtains the B-Rep from the EVM.

Given a 3D digital image, the EV are those vertices in the corresponding voxelization belonging to an odd number of black voxels. This property comes from the definition of extreme vertex and is proved by exhaustive analysis over the 256 cases in which 8 voxels (black or white) can surround a vertex [2]. The conversion algorithm from a voxelization to the EVM traverses all vertices of the voxelization and keeps those which fulfill this property. These vertices are traversed in a systematic way (by cuts and by rows into a cut) and an XOR operation is applied to the eight surrounding voxels in order to determine if the number of black surrounding voxels is odd. Bellow an outline of the algorithm is showed:

procedure $VoxelToEVM$(**Input** $image3D : voxelization$, **Ouput** $p : EVM$)
  var $ev : TVertex$ **endvar**
  for every cut k **in** image3D **do**
    for every row i **in** cut **do**
      for every vertex j **in** row **do**
        $odd := XOR(Voxel(j-1, i-1, k-1), Voxel(j-1, i-1, k),$
        $Voxel(j-1, i, k-1), Voxel(j-1, i, k), Voxel(j, i-1, k-1),$
        $Voxel(j, i-1, k), Voxel(j, i, k-1), Voxel(j, i, k));$
        if odd**then** $ev = (j, i, k)$; $AddVertexToEVM(ev, p)$
        endif
      endfor
    endfor
  endfor
endprocedure

Function $Voxel(i,j,k)$ returns the color 0 (black) or 1 (white) of the corresponding node and function XOR applies the exclusive OR to the operands.

The completeness of the EVM means that this model is actually a B-Rep model: a very concise and implicit one. But if we need to make it more explicit we are able to obtain all the geometry and connectivity information (also called topological relations) of vertices, edges and faces of the model. In [3] an algorithm that computes all vertices coordinates, all faces oriented normal vectors and the topological relations $face : edges$ and $edge : vertices$, i. e., the so called hierarchical B-Rep, is presented and explained. For the correct reading of this paper, we include a review of this algorithm.

This algorithm runs first in 3D and computes the set of faces with its corresponding normal vector, from the EVM of the object; then it runs recursively in 2D and, for each face, computes its set of edges correctly oriented with the associated vertices. The algorithm guarantees that non extreme vertices (which do not belong to the EVM) appear when they are needed. When the algorithm works in 3D, it traverses the set of EV in the three orderings that supply faces perpendicular to axis X, Y and Z respectively. The algorithm computes sections from planes of vertices which came directly from the EVM and then computes FD and BD from sections (see properties in section 3.2). The algorithm also works in 2D for the two possible orderings of each face.

The procedure for one of such orderings is:

procedure $ComputeBRep$(**Input** $p : EVM$, **Input** dim : **integer**, **I/O** $q : BRep$)
  var
    $Si, Sj : EVM$ {two consecutive sections}
    $plv : EVM$ {the current plane or line of vertices}
    $FD, BD : EVM$ {forward and backward differences}
  endvar
  $dim := dim - 1$; $Si := IniEVM()$; $plv := GetPlv(p, dim)$
  while $\neg EndEVM(p)$ **do**
    $Sj := ComputeSeccion(Si, plv, dim)$
    $FD := OpBool(Si, Sj, dim, \text{'-'})$
    $BD := OpBool(Sj, Si, dim, \text{'-'})$
    if $dim = 2$**then** $ComputeFaces(FD, BD, dim, q)$
    $\Box$ $dim = 1$**then** $AddEdgesBRep(FD, BD, q)$
      endif
      $Si := Sj$; $plv := GetPlv(p, dim)$
    endwhile
  endprocedure

This algorithm uses the basic operations with the EVM. $IniEVM$ creates an empty model, $GetPlv$ obtains a plane or a line of vertices from the model, $EndEVM()$ detects the end of the model and $ComputeSection$ computes a section from the previous one and the previous plane or line of vertices.

Procedure $ComputeFaces$ processes FD and BD in the two possible orderings and calls recursively to procedure $ComputeBRep$ and procedure $AddEdgesBRep$ adds the edges in the FD and BD to the B-Rep model.

5 The EVM performance

In this section we discuss the performance of the EVM. We can use it directly in many processes; although it is very concise, we can compress it more for storage or transmission purposes; and, as we have shown in the previous section, we can make it more explicit, by obtaining a hierarchical B-Rep, for visualization and other purposes.

EVM can be used directly using a data structure including all EV coordinates ordered in one of the six possible orderings. This representation is suitable for operations as Boolean operations [1] and computing measurements as perimeter, area and volume [2]. The operation of splitting by an orthogonal plane [3] needs the EV to be sorted in the three ways corresponding to the three main axis. And the algorithm that computes a hierarchical B-Rep, showed in the last section, requires a data structure with all six possible sorting ways accessible.

EVM can be compressed as we will explain in the next subsection.

Finally, EVM can be made explicit converting it to a hierarchical B-Rep. For simple visualizations that do not require to triangulate the model, the B-Rep obtained is very concise because it has less and bigger faces than methods that process voxels directly as the cuberile and the SB.

5.1 Storage

First of all we want to remark that in the EVM any connectivity information is required as it only is represented by the geometry, i.e., the coordinates of the extreme vertices.

The EVM consists on a set of vertices and then the storage requirements for an OPP, P is $O(n)$, $0 < n \leq nv$, n being the number of extreme vertices of P, $n = \#EV(P) = \#P$, and nv being the total number of vertices of P. Moreover n is always even. More precisely, the occupancy of an OPP with n extreme vertices is $3n$ float values, because any vertex has three coordinates (z, y, z).

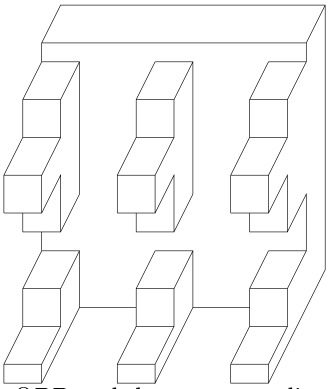
The EVM data structure allows different schemes for data compression using the fact that the EV are arranged in planes and lines of vertices.

S1. Data compression by brinks: The ending vertices of a brink share two coordinates and so a brink can be represented by only four floats (a, b, c_1, c_2). Then the occupancy of an OPP with n EV is $o_b = 2n$ values.

S2. Data compression by lines of vertices: Vertices belonging to a line of vertices share two coordinates, and for a line of k vertices only $k + 3$ floats are needed, the coordinates and a flag indicating the beginning or end of a new line $(k, a, b, c_1, c_2, \ldots, c_k)$. If nl is the number of lines of vertices and we suppose that line j has k_j vertices, then the occupancy is $o_l = \sum_{j=1}^{nl}(k_j + 3) = n + 3nl$ values. If, and only if, $3nl << n$ then it is worth to use lines of vertices instead of brinks

S3. Data compression by planes of vertices: All vertices in a plane of vertices share the same coordinate. Extending the reasoning of the previous item, the occupancy in this case is $o_p = n + 2nl + 2np$ values, being np the number of planes of vertices. And this compressing scheme will be better than using brinks if, and only if, $2nl + 2np << n$. This scheme is similar to the codification scheme implemented for the Semiboundary (see section 2.1).

As nl and np values depend on the ordering of EV, for the two last techniques we can choose the best ordering when compressing is our objective. Figure 4 shows an example with $n = 88$ EV. Then $3n = 264$ is the number of values needed without any compression scheme. Using brinks (scheme S1) we need $2n = 176$ values and we obtain a minimum of 130 values with the scheme S3 and the YZX ordering, yielding a compression rate of 50.7% with respect the first quantity and of 27.2% with respect the second one. However, as shows the table, using the two last schemes it is possible to produce more values than simply using brinks, but in our experience it is not very common on actual 3D digital images (see section 6.2).



sorting	nl	np	o_l (S2)	o_p (S3)
XYZ	20	6	148	140
XZY	44	6	220	188
YXZ	20	4	148	136
YZX	17	4	139	**130**
ZXY	44	8	220	192
ZYX	17	8	139	138

Figure 4: An OPP and the corresponding table comparing the compression techniques discussed.

EV ratio

We define the $EVratio$ as the ratio between the number of extreme vertices and the total number of vertices which has the property $n/nv \leq 1$. The EV ratio for the OPP in figure 3 is $34/41 = 0.83$ and for the OPP in figure 4 is 1 because all the 88 vertices are extreme. A kind of object in which this ratio is the best is showed in figure 5 and has been constructed using a 6×6 grid. In this case the EV ratio is 0.189. For a similar object constructed in a 20×20 grid the EV ratio would be 0.025.

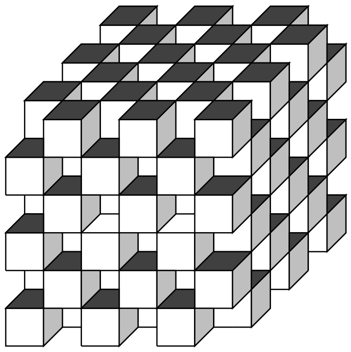


Figure 5: Special object that optimizes the EV ratio.

5.2 Robustness

Using EVM we never need to perform any arithmetic operation. All operations are performed by merging vertices coordinates. Coordinates of all new vertices that appear doing operations are combination of the coordinates of the input vertices. This property guarantees the stability in EVM algorithms allowing to achieve Boolean operations, collision detection and cuts between objects without the drawbacks of other representations. See figure 6 for an example on Boolean operations.

5.3 Complexity

The complexity of the algorithm that converts from a voxelization to EVM is $O(N)$, being N the number of voxels.

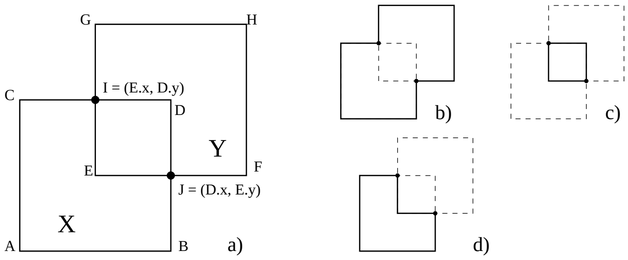


Figure 6: a) Input objects X and Y. b) $X \cup Y$. c) $X \cap Y$. d) $X - Y$

The bottle neck process in algorithms concerning EVM as Boolean operations algorithm and obtaining a hierarchical B-Rep algorithm is the computation of all the sections of the object from the EVM and this process gives the worst case complexity to the conversion algorithm. The worst case complexity of computing all sections is $O(n \times np)$, n being the number of extreme vertices and np being the number of planes of vertices. As np ranges from 2 to n, the worst case complexity is quadratic. However, experimental results show that the average experimental complexity is far less than quadratic but slightly greater than linear. Performing a numerical regression to the data used in these experimental results of the form $y = ax^b$ the coefficient b obtained was $b = 1.221$.

The algorithm that splits by an orthogonal plane is $O(n)$ and the algorithm that performs XOR between two OPP are also $O(n)$.

When performing operations, we need to work with not compressing EVM representations. We can access better to all EV coordinates if we have all brinks explicitly represented. We can get all brinks from a more compressed scheme with a process $O(n)$. If we need brinks (i.e., extreme vertices) sorted in other ways, then we need a preprocess of $O(nlgn)$ complexity.

6 Experimental results

In this section we present some experimental results to illustrate the potential of the EVM. We compare these results with those obtained using the Semiboundary representation because the Semiboundary is the Block-form model that best performs as we have concluded from the consulted literature.

We have processed three data sets: *boxes* of 18x18x10 voxels synthesized by computer, and two more generated by computer tomography, *skull* of 96x96x69 voxels corresponding to a human skull and *ctHead* of 256x256x94 voxels corresponding to a complete head of a child (see figures 7, 8 and 9).

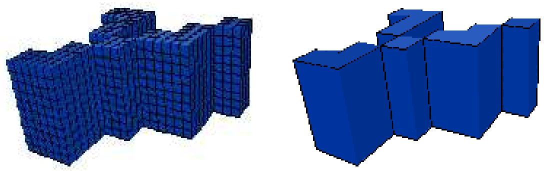


Figure 7: SB-BRep (left) and EVM-BRep (right) of *Boxes*

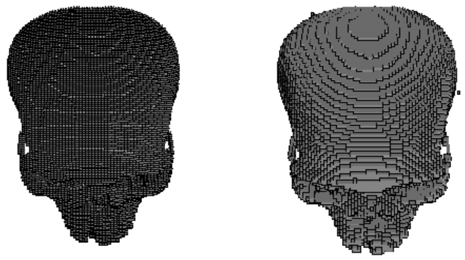


Figure 8: SB-BRep (left) and EVM-BRep (right) of *Skull*

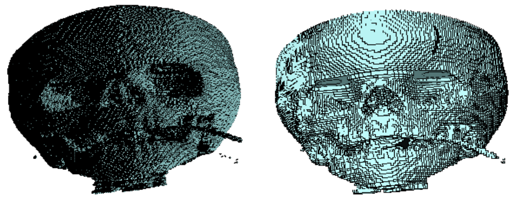


Figure 9: SB-BRep (left) and EVM-BRep (right) of *ctHead*

6.1 In terms of the compression

The following table shows the values of np and nl for each ordering and the amount of values stored for each compression scheme for the *skull* data set. The remarked element shows the optimum compression rate for this data set.

Ordering	np	nl	S1	S2	S3
XYZ	64	4031	46928	35557	31654
XZY	64	3202	46928	33070	**29996**
YXZ	91	4031	46928	35557	31708
YZX	91	3567	46928	34165	30780
ZXY	67	3202	46928	33070	30002
ZYX	67	3202	46928	34165	30732

The next table shows the occupation in bytes and the compression rate for each scheme against the conventional SB representation, for the *skull* and the *ctHead* data sets.

	skull		cthead	
scheme	bytes	rate	bytes	rate
SB	105979	-	453016	-
EVM(raw)	87990	16.9%	300120	33%
Scheme S1	58660	44.6%	200080	55.8%
Scheme S2	41337	60.9%	138042	69.5%
Scheme S3	37495	64.6%	83234	72.2%

6.2 In terms of the data structure

For each data set, we obtained the cuberille, SB and EVM representations. On the table we show the number of elements for each one. These elements are: boundary-faces for the cuberille, boundary-voxels for the Semiboundary and brinks (defined by pairs of extreme vertices) for the EVM.

scheme (elements)	boxes	skull	ctHead
cuberile (boundary-faces)	1236	49030	171446
SB (boundary-voxels)	848	25940	95391
EVM (brinks)	18	10623	40016

The next table shows the memory requirements (in bytes) needed for the SB and EVM (scheme 3) models. From the coding scheme presented in [8], SB requires $2 * (h2xh3)$ bytes for the indexing table plus 27 bits for each node (10 bits for the remaining coordinate, 6 bits for a neighborhood code and 11 bits for a discretized normal vector) [8]. We consider first the EVM without any information concerning vertices' normal vectors and using 10 bits for coordinate.

scheme	boxes	skull	ctHead
SB	3438	105979	453016
EVM	693	44989	231112

Now we consider a discretized normal vector associated to each extreme vertex. This normal vector is estimated by the gradient of gray-level. Our discretization method is slightly different of that of the SB. We will represent this normal vector with 12 bits, 4 for each direction. The next table shows the memory requirements (in bytes) for the SB and EVM (scheme 3, using normal vectors) models and the figure 10 shows the visual results obtained for the *Skull* and *CtHead* examples.

Memory(bytes)	boxes	skull	ctHead
SB	3438	105979	453016
EVM	720	60924	291136

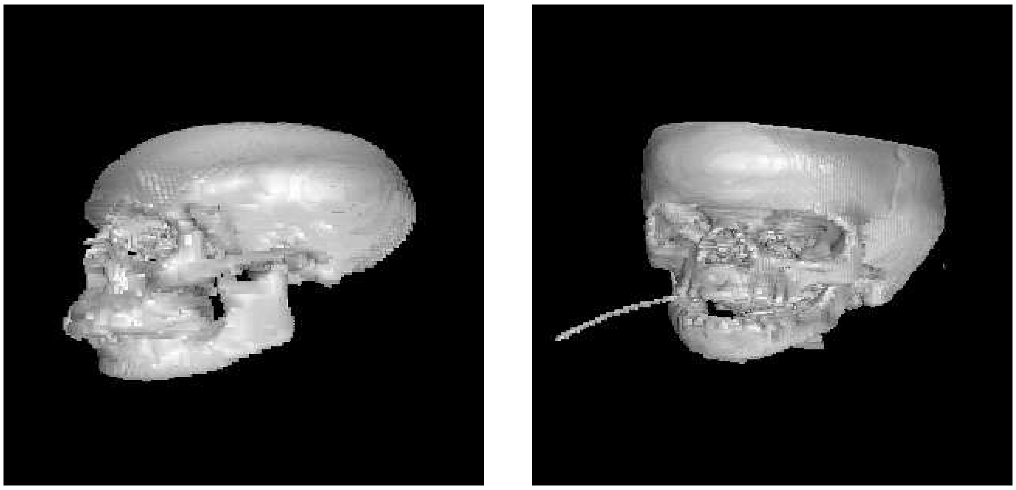


Figure 10: *Skull* and *CtHead*

The tables above show that the EVM's memory requirements in general are smaller than the other representation models.

6.3 In terms of the B-Rep

Continuing with this comparative study, the following table shows results in terms of the B-Rep constructed from three different models. The first B-Rep is obtained directly from the voxelization using the Marching Cubes algorithm (MC), the second one is obtained from the Semiboundary and, finally, the last one is generated from the EVM using the algorithm presented in section 4. The comparisons were done in terms of the number of faces generated in each case.

B-Rep(faces)	boxes	skull	ctHead
MC	1349	97467	336283
SB	1236	49030	180288
EVM(B-Rep)	36	16407	54570

Figures 7, 8 and 9 show the visualization of the SB-BRep and the EVM-BRep for the *boxes*, *skull* and *ctHead* data sets respectively. Whereas, the left figures present a lot of small square faces, the right figures group all the coplanar adjacent faces as a unique planar face just conserving the same geometry and avoiding redundant faces.

6.4 Examples of Boolean operations using the EVM

Figures 11 and 12 illustrate examples of cut and osteotomy achieved through Boolean operations between the EVM of the volume of interest and the EVM of another conveniently designed polyhedron. We want to remark that each object involved into the operation has its own EVM representation independent of the representation of the other object. On the contrary, the octree representation needs to build the octree of the smaller object in terms of the biggest one.

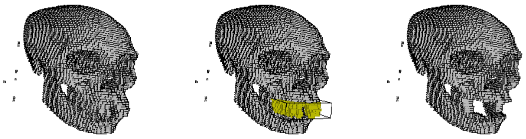


Figure 11: Selective osteotomy over *skull*

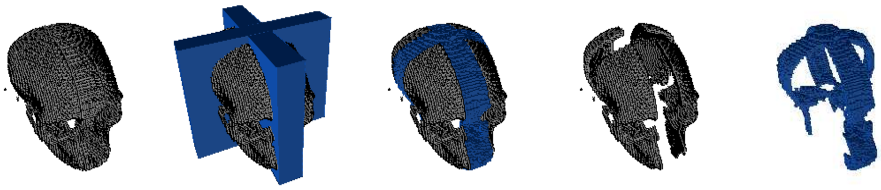


Figure 12: arbitrary cut of *skull*

7 Conclusion and future work

In this paper we have shown that the EVM is a suitable and powerful model for representing and operating digital images. The basic representation of this model includes all the extreme vertices ordered in one or more ways. This representation does not include any adjacency or connectivity information. We can do several operations as Boolean operations using this basic representation. We can compress this basic representation for storage and transmission purposes and we can expand it, obtaining a complete hierarchical B-Rep, for visualization purposes.

In this work we analyze this model and we present several experimental results. As a future work we will study the design of more operations in the digital images field by using the EVM: erosion, dilation, thinning, etc. We will also study if in these operations it is relevant or not the well-composedness of the image, i.e., if the corresponding orthogonal polyhedron is 2-manifold or not.

8 Acknowledgments

This work has been partially supported by a CICYT grant TIC99-1230-C02-02.

References

[1] A. Aguilera and D. Ayala. Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry. In C. Hoffmann and W. Bronsvort, editors, *ACM SM'97. Atlanta (USA)*, pages 56 – 67, 1997.

[2] A. Aguilera and D. Ayala. Domain extension for the extreme vertices model (EVM) and set-membership classification. In *CSG'98. Ammerdown (UK)*, pages 33 – 47. Information Geometers Ltd., 1998.

[3] A. Aguilera and D. Ayala. Converting Orthogonal Polyhedra from Extreme Vertice Model to B-Rep and to Alternative Sum of Volumes. *Computing. Springer-Verlag*, 2000. Still not published, now available in http://www.lsi.upc.es/dept/techreps/html/R00-66.html.

[4] E. Artzy, G. Frieder, and H. Gabor. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15:1 – 24, 1981.

[5] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surfaces construction algorithm. *Computer Graphics*, 21(4):163 – 169, 1987.

[6] H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE Transactions on Patern Analysis and Machine Inteligence*, 10(4):579 – 586, 1988.

[7] S. N. Srihari. Representation of three-dimensional digital images. *ACM Computing Surveys*, 13(4):399 – 424, 1981.

[8] J. Udupa and O. Odhner. Fast visualization, manipulation and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 4(1):53 – 62, 1991.

[9] J. K. Udupa and D. Odhner. Shell Rendering. *IEEE Computer Graphics and Applications*, 13(6):58 – 67, 1993.

[10] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201 – 227, 1992.