A Complete Solid Model for Surface Rendering

Jorge Rodríguez*[1], Dolors Ayala[1], and Antonio Aguilera[2]

[1] Universitat Politècnica de Catalunya, Diagonal 647, 08028 Barcelona, Spain
{*jrodri, dolorsa*}*@lsi.upc.es*
[2] Universidad de las Américas-Puebla, México
aguilera@mail.udlap.mx

Summary. The extreme vertices model (EVM) has been presented as a concise and complete model for orthogonal pseudo-polyhedra (OPP) in the solid modeling field. This model exploits the simplicity of its domain by allowing robust and simple implementations. In this paper we use the EVM to represent and process images and volume data sets. We will prove that the EVM works as an efficient scheme of representation for binary volume data sets as well as a powerful block-form surface renderer which avoids the redundancy of primitives on the extracted isosurface. In addition, to achieve more realism, the normal vectors at vertices, computed from the input data, can be added to the model. Furthermore, an efficient tessellator of non-convex orthogonal faces is presented. Also, useful operating and manipulating tools can be implemented over the EVM, like editing operations via Boolean operators, non voxel-based morphological operations and an improved connected component labeling algorithm. The well-composedness property of the volume can be detected easily as well as the critical zones where the non-manifold configuration occurs.

1 Introduction

A volume data set (or 3D digital image) can be represented as a map $f : Z \otimes Z \otimes Z \rightarrow R$, in such a way that every point is assigned a value representing its color. In a binary data the image set is $\{0, 1\}$. A volume can be defined as a union of voxels i.e. upright unit cubes whose vertices have integer coordinates [20]. In the field of digital image processing, the term well-composed has been used to name those digital images with a manifold boundary and have related the concepts of manifoldness and well-composedness [12].

Orthogonal polyhedra (OP) are polyhedra with all their faces oriented in three orthogonal directions. The term orthogonal pseudo-polyhedra (OPP) refers to regular and orthogonal polyhedra with a possible non-manifold boundary. The *extreme vertices model* (*EVM*) is a very concise model in which any OPP can be described using only a subset of its vertices. The EVM is actually a complete (unambiguous) solid model [2]. EVM is also suitable for 2D orthogonal polygons.

* This author has been supported by a grant from CONICIT (Venezuelan Council of Research in Science and Technology)

From all these concepts, we can say that the continuous analog of a general digital image is an OPP and the continuous analog of a manifold digital image is an OP and both can be represented and operated using the EVM.

In this paper we use the EVM to represent, visualize and operate volume data sets (or volumes). Concerning with representation and visualization purposes, we show that the EVM works as an efficient scheme of representation for segmented volumes and that it is a powerful block-form surface renderer which avoids the redundancy of primitives on the generated isosurface. We also present an efficient tessellator of non-convex faces to be used in case that only convex faces would be required for the visualization system. And we offer the possibility to enrich the EVM with the normal vectors at the vertices in order to obtain more realism.

Concerning with operations, we review the EVM useful operating and manipulating tools, which follow a non-voxel-based approach and work in 3D as well as in 2D. We have developed editing operations via Boolean operators with no arithmetic computing. Morphological operations such as erosion and dilation and its derivatives opening and closing have also been derived. Moreover, we have devised an improved connected component labeling algorithm which reduces the number of labels and equivalences produced. This method allows to detect easily the well-composedness property of the volume as well as the critical zones where non-manifold configurations occur.

From the literature revised, we have concluded that the EVM is closely related to the semiboundary representation (SB). Therefore, we will present experimental results obtained with the EVM but we will also compare them with results obtained using the SB.

2 Related work

There are two main families of models and methods for isosurface extraction. Digital (or block-form) methods represent the surface as a set of voxels or voxel faces [4], [20], [25], [8]. Polygonal (or beveled-form) methods represent the surface as a set of polygons (mostly triangular) [14], [9] [10].

Most of the developed techniques, mainly those of the digital family, are focussed on visualization and are not concerned on other operations, and there is a claim for a general model that can carry out the complete set of desirable tasks: visualization, manipulation and editing. Moreover, digital models exhibit formal properties such as closure, orientedness and connectedness whereas polygonal models still lack of a solid modeling foundation [8].

Another common drawback of the existing techniques is the redundancy of primitives in the isosurfaces obtained. Isosurfaces consist of lots of little quadrangular (in digital methods) or triangular (in polygonal methods) faces. Several adaptive attempts have been developed to reduce this redundancy in

[17], [16], [23], [22] but most of them actually make a post-processing and produce cracks which have to be solved by approximation.

EVM is a digital model, closely related to the octree and semiboundary (SB) models in our opinion and, for this reason, we include a little discussion on these models. Octree [6], [26], and SB [25], [24] models allow to store a volume in a proper data structure in order to carry out operations of visualization and editing efficiently. The octree model is a spatial partitioning which allows to manipulate homogeneous regions of the volume as single entities but which requires volumes with a size of a power of two. It can be used to visualize volumes and accelerate the process of surface extraction. It allows Boolean operations to be performed between objects, but it requires a common cubic universe for both operands. Consequently, operations between objects of very different sizes force an important overhead of memory for representing the smaller one. Nevertheless, useful image processing operations such as dilation, erosion and thinning have not been dealt in this scheme.

The SB representation[25] [8], is a useful model for fast visualization, manipulation and analysis of structures generated from volume data. It only stores the boundary voxels of the data, keeping the information of the interior voxels in an implicit way. SB allows to perform operations such as cut-away, osteotomy, reflections and interactive manipulation, and the authors assure that other data structures such as octrees and run-length codes will not match the SB efficiency in visualization and manipulation. However, Boolean operations are relatively expensive using this scheme because it is not easy to determine which part of one object lies inside the other one by merely checking the boundary voxels. Furthermore, like the octree model, the SB would require a common cubic universe for both operands involved.

With respect morphological operations like erosion and dilation, several approaches to improve them have been reported. In [27] a method is proposed for successive dilation operations of volumes by using a queue of contour voxels. But it demands an overhead of memory. In [24], the SB is used in a method for fast eroding and dilating. Both proposals traverse only the contour voxels. However, they follow a voxel-based approach, as most of other methods, and therefore the cost in terms of both memory requirements and algorithmic efficiency increases with the number of voxels. Lurig and Ertl [15] present a technique to improve the efficiency of erosion and dilation algorithms decomposing the multi-dimensional structuring elements into 1D elements using a pseudo tensor product approach.

Concerning with connected component labeling, several algorithms have been developed. In [24], Thurfjell et al. present a connected component labeling algorithm for volumes. Dillencourt, Samet and Tamminen [7] formalized a general approach for connected component labeling which can deal with d-dimensional images represented in several data structures such as arrays, quadtrees and bintrees. Finally, Oikarinem et al. [18] suggest an algorithm for volume rendering based on a 2D connected component search. Almost all these approaches are voxel-based and therefore their performance decreases as the number of voxels increases.

In [12] the term well-composed is introduced for a volume in which the configurations showed in Fig. 1 a), modulo reflections and rotations, are avoided. The author also relates the terms well-composed and 2-manifold. In most of the consulted bibliography, the authors tend to characterize and avoid non-well-composedness. In [13] the authors say that it is more important to report the non-manifold zones as zones corresponding with determined features of the volume than to deal with it. In [21] a method is presented that obtains a well-composed 2D image by doing simple deformations.

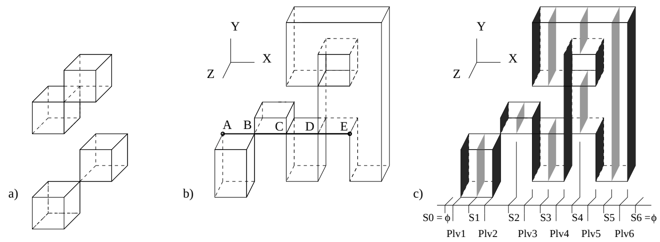

Fig. 1. a) Configurations not allowed in a well-composed volume. b) An OPP with a marked brink from vertex A to vertex E. c) Its planes of vertices and sections perpendicular to the X axis shown in dark and light grey respectively.

3 Review of EVM

Let P be an orthogonal pseudo-polyhedron (OPP). A *brink* is the maximal 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*. Fig. 1 b) shows an OPP with a brink from vertex A to vertex E (both extreme). In this brink, vertices B, C and D are non-extreme. The *EVM* is a representation scheme in which any OPP is represented by its (an only its) set of extreme vertices. This section summarizes some definitions and properties of the EVM and references [2] for more extensive explanations.

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* (S) is the resulting polygon from the intersection between P and an orthogonal plane. Each slice has its representing section. Fig. 1 c) shows an example. Computing planes of vertices from sections and vice-versa are the two basic operations with EVM, and they only imply doing XOR operations between vertices. Moreover, the EVM is a complete B-Rep model, i.e., all the geometry as coordinates of non-extreme vertices and normal vectors of faces can be computed from the extreme vertices.

4 Surface rendering using the EVM

In this section we present an isosurface extraction method using the EVM. It belongs to the digital category and reduces, without any post-processing, the excessive redundancy of primitives that present most isosurface extraction algorithms. It obtains a complete and closed solid model with an oriented boundary. We also present a way to improve the block-like appearance by using an enriched EVM and show a simple way to tessellate orthogonal faces if needed.

4.1 Isosurface extraction algorithm using EVM

To obtain an isosurface from a volume, for a given value of density, we first obtain the EVM of the volume and afterwards a hierarchical B-Rep is computed from the EVM. In this section we propose a volume data set to EVM conversion algorithm which complements the EVM to B-Rep conversion algorithm reported in [3].

Given a segmented volume data set, the extreme vertices are those belonging to an odd number of black voxels. This property can be proved by exhaustive analysis over the 256 cases in which 8 voxels (black or white) surround a vertex [2]. The algorithm that obtains the EVM from the volume traverses all its vertices and keeps those which fulfill this property.

procedure $VoxelToEVM$(**Input** $vol : tVvoxelization$, **Ouput** $p : tEVM$)
  var $ev : tVertex$ **endvar**
  for every cut k **in** vol **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 $odd \rightarrow ev = (j, i, k);\ AddVertex(ev, p)$ **endif**
      endfor
    endfor
  endfor
endprocedure

Function XOR applies exclusive-or to all the arguments. $Voxel_{j,i,k}$ is the value, 0 or 1, of the (j, i, k) voxel. $AddVertex$ adds a new vertex to the EVM.

4.2 Avoiding the redundancy of primitives

The hierarchical B-Rep obtained from the EVM is actually a polyhedral representation of the volume of interest in which the possible coplanar little adjacent faces are already merged in a large one (not necessarily convex).

Fig. 2 and Fig. 3 show the B-Rep obtained from the SB (SB-BRep) and the B-Rep computed from the EVM (EVM-BRep) for two data sets *Skull* and *ctHead*, obtained from CT data segmented by threshold. Table 1 compares the number of faces obtained in the corresponding B-Rep for both volumes. It also shows the number of triangular faces produced for the marching cubes algorithm (MC) [14].

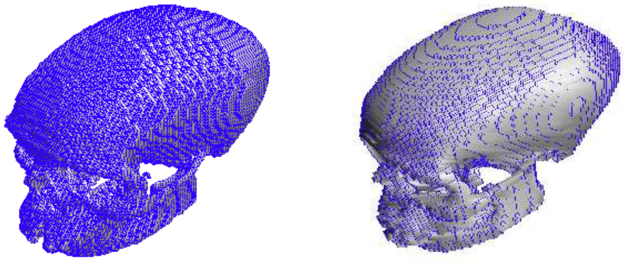

Fig. 2. SB-BRep (left) and EVM-BRep (right) of *Skull*

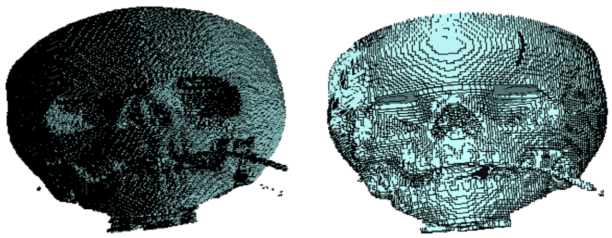

Fig. 3. SB-BRep (left) and EVM-BRep (right) of *ctHead*

BRep(faces)	MC-BRep	SB-BRep	EVM-BRep
skull	97467	49030	16407
ctHead	336283	180288	54570

Table 1. Number of faces

4.3 Efficient storing of the volume

In this and the following subsections we study the storage requirements of the EVM and compare them with the SB. We consider a volume data set of size $h_1 \times h_2 \times h_3$, assuming that 5% of voxels are boundary voxels and 3% of vertices are extreme vertices, which are reasonable upper bounds according to our experimental results. Also, we use the fact that a volume data set with $h_1 \times h_2 \times h_3$ voxels has $(h_1+1)\cdot(h_2+1)\cdot(h_3+1)$ vertices and we consider that a coordinate can be stored in 10 bits [25].

The SB of a volume C is a tuple (D, Δ), where D is the set of boundary voxels of the volume of interest and Δ is an *encoding function* which assigns to every voxel in D a 6-bit number, called the *neighbor code*. In terms of data structure, the SB is implemented as a pair {P,M}, where P is a pointer array with $h_2 \times h_3$ elements and M is a list of triplets $(d_1, \delta, \mathbf{n})$, where d_1 is the first coordinate of a voxel $d \in D$ (10 bits), $\delta = \Delta$ (6 bits) and $\mathbf{n}$ is a unit normal vector coded in 11 bits, [25].

From these definitions, we have estimated the memory requirements of SB as follow: $h_2 \cdot h_3 \cdot size(pointer)$ for P and $(h_1 \cdot h_2 \cdot h_3) \cdot 0.05$ triplets for M. Given that each triplet requires 27 bits, then M requires $(h_1 \cdot h_2 \cdot h_3) \cdot 0.05 \cdot 27$ bits. Therefore,

$$Mem(SB) = h_2 \cdot h_3 \cdot size(pointer) + (h_1 \cdot h_2 \cdot h_3) \cdot 1.35\ bits$$

Although for most operations, EVM is implemented as the set of ordered extreme vertices, it also can be implemented in a way similar to the SB, in order to compare them. Thus, we have a pointer array, P, with $h_2 \times h_3$ elements and a list, M, of brinks (pairs of extreme vertices). The memory requirements of this EVM implementation are $h_2 \cdot h_3 \cdot size(pointer)$ for P and $(h_1+1)\cdot(h_2+1)\cdot(h_3+1)\cdot 0.015$ brinks. Note that the number of brinks is estimated as 1.5% of the number of vertices (one half of the number of extreme vertices). Given that we require 10 bits per coordinate and that we have two coordinates for each brink (the third one for each extreme vertex of the brink), then,

$$Mem(EVM) = h_2 \cdot h_3 \cdot size(pointer) + (h_1+1)\cdot(h_2+1)\cdot(h_3+1)\cdot 0.015 \cdot 20\ bits$$

Comparing both expressions we get,

$$Mem(SB) > Mem(EVM) \Leftrightarrow (h_1 \cdot h_2 \cdot h_3)\cdot 1.35 > (h_1+1)\cdot(h_2+1)\cdot(h_3+1)\cdot 0.3$$

In order to facilitate the proof we suppose $h_1 = h_2 = h_3 = h$. Then,

$$h^3 \cdot 1.35 > (h+1)^3 \cdot 0.6$$

These cubic functions have an intersection point at $h = 1.54$. As a result, we can conclude that

$$Mem(SB) > Mem(EVM) \Leftrightarrow Min(h_1, h_2, h_3) \geq 2$$

Table 2 presents the estimated computed values, based on the previous discussion, for several data sets. The items on the table are the total number of voxels (#vox), the estimated number of boundary voxels (#bv), the estimated number of extreme vertices (#EV) and the memory requirements estimated for each model. The last two files correspond to the real data sets, *Skull* ($96 \times 96 \times 69$) and *ctHead* ($256 \times 256 \times 94$), shown in Figs. 2 and 3.

Memory	**#vox**	**#bv**	**#EV**	**Mem(SB)**	**Mem(EVM)**
256^3	16777216	838860	509237	2892Kb	750Kb
512^3	134217728	6710886	4050170	22630Kb	5458Kb
Skull	635904	31795	19758	122Kb	42Kb
ctHead	6160384	308019	188239	1143Kb	358Kb

Table 2. Estimated memory requirements

Table 3 shows real experimental values for the two data sets *Skull* and *ctHead*, that can be compared with the estimated ones. Now, the items #bv and #EV contain the real values of boundary voxels and extreme vertices respectively. The last column indicates the memory saving using EVM instead of SB. Both the previous estimated values and the experimental ones prove that EVM is a more concise representation scheme.

Memory	**#vox**	**#bv**	**#EV**	**M(SB)**	**M(EVM)**	Saving %
Skull	635904	36184 (5.6%)	23464 (3.5%)	137Kb	75Kb	45%
ctHead	6160348	153979 (2.4%)	80032 (1.2%)	635Kb	323Kb	49%

Table 3. Real memory requirements

4.4 Improving realism

Our surface renderer based on EVM lies in the block-form category and the main drawback of this group is a block-like appearance of the produced surfaces with a weak realism. To improve the appearance, we propose an *enriched EVM* with an associated normal vector to each extreme vertex. This normal vector is estimated by the gradient of grey level directly from the input volume using central differences around the extreme vertex. To avoid the over-growth of memory requirements, we have encoded the normal vector in 15 bits, 5 per direction, dividing each axis in 32 intervals, 16 per sign. Subsequently, when more realism is required, these normal vectors will be used to smooth the shading of the object. Fig. 4 shows how the enriched EVM improves the realism.

The memory occupancy of the enriched EVM grows in 30 bits per brink and therefore

$$Mem(EVM) = h_2 \cdot h_3 \cdot size(pointer) + (h_1+1) \cdot (h_2+1) \cdot (h_3+1) \cdot 0.015 \cdot 50 \, bits$$

Comparing again with the SB, now the cubic functions $1.35h^3$ and $0.75(h+1)^3$ have an intersection point at $h = 4.62$.

It is important to note that the extreme vertices are only a subset of the vertices of the object. The non-extreme vertices, which appear when the B-Rep is obtained, receive their normal vector computed by linear interpolation

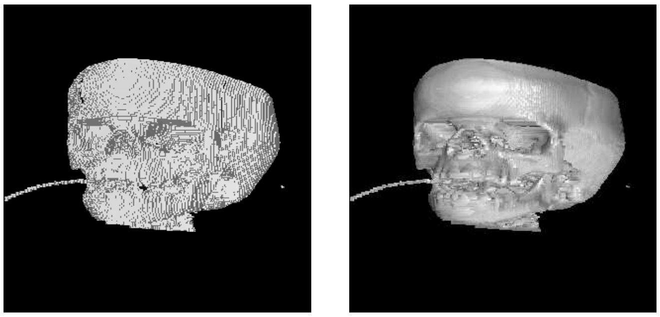

Fig. 4. Left: rendering from EVM. Right: rendering from Enriched EVM

Memory	**#voxels**	**#Bound**	**#EV**	**M(SB)**	**M(EVM)**	Saving %
skull	635904	36184	23464	137Kb	109Kb	20.4%
ctHead	6160348	153979	80032	635Kb	440Kb	30.7%

Table 4. Real memory requirements

between the extreme vertices of the brinks in which the non-extreme vertex lies. Fig. 5 illustrates the computation of the normal vector of a non-extreme vertex. Table 4 shows a comparative of experimental memory requirements between SB and enriched EVM. Finally, Fig. 6 shows several views of the *ctHead* data set.

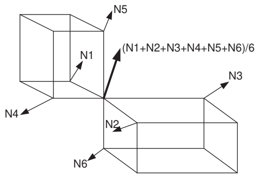

Fig. 5. Normal vector computing on an intermediate vertex

4.5 A specific EVM-tessellator

The B-Rep obtained from the EVM avoids the redundancy of primitives and obtains large orthogonal faces. In general, these faces are not convex and may contain holes. This feature is not a problem in most visualization systems because they have their own tessellation method. However, the tessellation strategies used in some of these systems follow general rules and split the face in an arbitrary way with no care about its orthogonal shape. Moreover, there are visualization systems that require convex faces. We present a tessellation strategy based on the EVM, which profits the knowledge about the geometry of faces to split them in an efficient way.

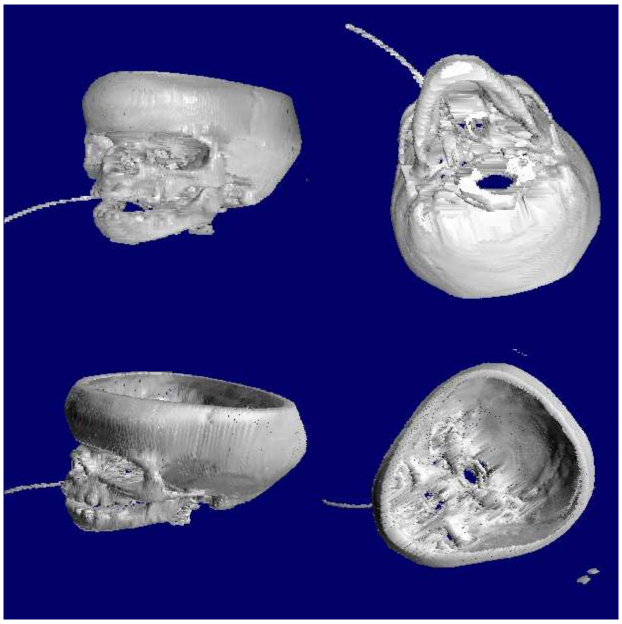

Fig. 6. Several views of *ctHead*

Let F be a face of an OPP, then F can be expressed as the union of all its slices (see section 3) in a certain orthogonal direction, $F = \cup_{k=1}^{n-1} slice_k(F)$. Each slice define one or more rectangles. The representative 1D sections of each slice are composed by one or more brinks (each brink corresponds to a rectangle) and can be computed doing an XOR operation between the extreme vertices of its previous and next planes of vertices. From these 1D sections, the rectangles of the slice are directly obtained. Furthermore, if triangles would be required for the visualization system, each rectangle could be split along its diagonal (see Fig. 7).

In contrast with general tessellation techniques this method does not check neither collinearity among vertices nor intersection among edges. Moreover, the orthogonal nature of faces allows a very simple test based on a trivial rule to detect non-convexity: an orthogonal polygon is non-convex if its number of vertices is greater than four.

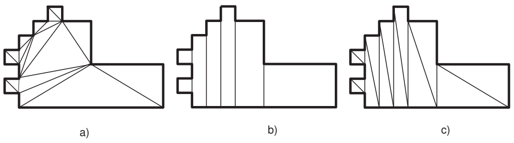

Fig. 7. a) General tessellation (18 triangles). b) EVM-based tessellation (7 rectangles) c) Further triangulation (14 triangles)

5 Volume operations using the EVM

In this section we review the EVM developed operations for volumes. All these operations are non-voxel oriented. Editing and morphological operations are based on the sections of the model (see section 3). Connected component labeling is based on a decomposition of the model into boxes. None of these operations require floating point arithmetic.

5.1 Editing operations via EVM-Boolean operators

Editing operations over volumes can be achieved via Boolean operators. General Boolean operations between OPP can be carried out by applying recursively the same Boolean operation over the corresponding sections. At the end, the operation is actually performed over 1D sections, which consist of one or several collinear brinks. So, the method is recursive over the dimension. The algorithm has been presented in [1] for solid objects and consists in a geometric merge between the EVM of both operands. In this work we have experimented this algorithm over volumes, obtaining the same good results. Fig. 8 illustrates a simulation of an interactive osteotomy performed via Boolean operations.

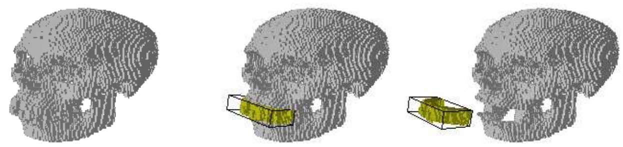

Fig. 8. Osteotomy of an human skull simulated via Boolean operations

5.2 Morphological operations

The approach for morphological operations (erosion and dilation), based on the EVM [19] is also recursive over the dimension. The algorithm identifies the inner sections of the object, then it operates (shrinks or elongates) them and, afterwards, computes the planes of vertices of the operated sections, obtaining the EVM representation of the transformed object.

This method is suitable for volumes as well as for 2D images, due to its recursive nature. It decomposes the volume into 2D sections which are themselves considered 2D images and, then, decomposes these 2D images into 1D sections. This is the basic case of the recursion and the algorithm actually performs 1D transformations. We can express the erosion of an EVM represented OPP P, $Shrink(P)$, as:

$$\begin{cases} Shrink(S_{i-1}) \otimes^* Shrink(S_i), \ i = 1 \dots n & dim > 1 \\ Shrink1D(P) & dim = 1 \end{cases}$$

S_i is the ith section of P, $i = 0, 1, ...n$, dim is the dimension of P (initially, $dim = 3$). The XOR operation applied to the shrunken sections gives the plane of vertices between them (see section 3. *Shrink1D (P)* consists in shrinking each brink of the 1D resulting object. An analogous recursive process can be defined for *Elongate(P)*.

The cost of these operations does not depend directly of the amount of voxels of the input data set, instead, it is associated with the complexity of the orthogonal shape of it, so we are able to operate large sets of adjacent voxels in a single step. Furthermore, our approach admits the use of box-shaped structuring elements of arbitrary size. Fig. 9 shows a 2D example (*Magallanes*) and Fig. 10 a 3D example of a human skull.

Fig. 9. 2D example. Left: Original image. Right: eroded once.

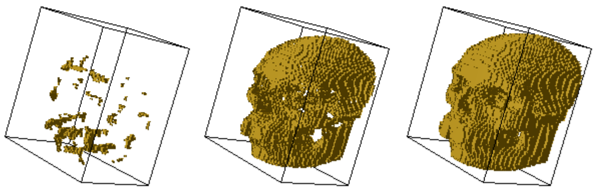

Fig. 10. Left to right: eroded once (*pieces*), original skull and dilated once

The algorithms have been implemented in C++ on a Sun Microsystems Ultra SPARC 60 machine. Table 5 presents, for these data sets, its resolution, the number of boundary voxels, the number of brinks within its EVM representation and the processing time (in seconds) for erosion and dilation operations. We can note that the processing time increases with the number of brinks. The right hand side of the table shows the results obtained when the size of the former data sets was scaled 1:2 maintaining the same shape. Note that the number of boundary voxels has increased accordingly but the number of brinks and the processing time remains almost the same. Therefore, bigger images have no greater time necessarily.

Data sets	Magall. (400x159)	skull (96^2x69)	Double1 (767x318)	Double2 (192^2x138)
#bv	9692	36184	37124	187592
#brinks	1173	11732	1173	11732
Erosion	0.19s	4.22s	0.28s	4.7s
Dilation	0.19s	6.38s	0.28s	6.78s

Table 5. Morphological processing time (in seconds).

5.3 Connected component labeling

The method for connected component labeling using EVM [5] is based on a particular decomposition of the model and follows the approach presented in [24]. It works in identical way for 2D images and volumes, is size-independent, improves the performance with respect to voxel-based approaches and deals indistinctly with manifold and non-manifold data.

Decomposing EVM into an OUDB

We first obtain a particular partition straightforward from the EVM which consists in an ordered union of disjoint boxes (OUDB). This partition can be thought as a special kind of cell decomposition which is axis aligned as octrees and bintrees but is performed along the object geometry like BSP.

We obtain the OUDB from EVM splitting first the data at every plane of vertices perpendicular to a main axis, say X for instance, obtaining a set of slices. Then, we split each slice at every plane of vertices perpendicular to another main axis, say Y, obtaining a set of boxes for each slice. All the obtained boxes constitute the OUDB partition (see Fig. 11). This method can be understood as the 3D extension of the method used in section 4.5 to tessellate non-convex orthogonal polygons.

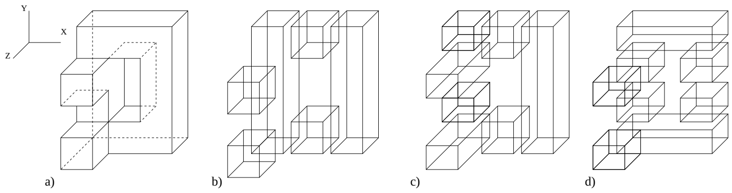

Fig. 11. a) An OPP. b) XZ decomposition (6 boxes). c) XY decomposition (7 boxes). d) YZ decomposition (8 boxes).

Connected component labeling method

Once the OUDB partition has been achieved the connected component labeling (CCL) process takes place. Previous algorithms use a sequential two-pass strategy which works at voxel-level [11], [24]. In the first pass (labeling) all voxels are labeled but there remain some unsolved label equivalences which are sorted out in a second pass (renumbering). In [24] this technique is improved by using the SB and, therefore, visiting only boundary voxels. Our method follows this classical two-pass approach, but it traverses the set of disjoint boxes(OUDB) instead of a set of boundary voxels, further diminishing the number of elements to label and the number of labels and equivalences generated.

Fig. 12 shows the three steps carried out to in a 2D example. First, the EVM is extracted from the input data and its OUDB partition is obtained, then the labeling of boxes is performed and finally the equivalences are solved and the connected components stored as separated objects.

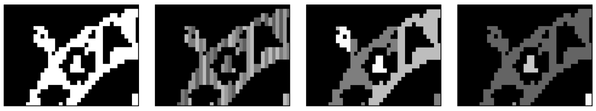

Fig. 12. Left to right: Original image, OUDB decomposition, OUDB labeled before renumbering pass and four connected components as separated objects.

Now we present some results for a 2D example (*magallanes*, shown in Fig. 13) and for a 3D example (*pieces*, shown in the left hand side of Fig. 10). Table 6 presents several results for each data set. It shows at the left hand side, the number of elements, boundary voxels (#bv) and disjoint boxes (#box). In the middle there is a comparison between the execution time (in seconds) using a voxel-based algorithm and with our approach. At the right hand side there is the number of connected components.

Fig. 13. CCL applied to a 2D example maintaining (right) or breaking (left) non-manifold zones.

Data sets	elements		voxel-based			EVM-CCL				#cc	
	#bv	#box	Label.	Renum.	Total	OUDB	Label.	Renum.	Total	#M	#B
Magall	4662	1113	0.06	0.03	0.1	0.05	0.02	0.0001	0.07	18	29
Pieces	1160	524	1.45	0.37	1.96	0.08	0.03	0.0001	0.11	56	78

Table 6. CCL results.

The performance of our algorithm is related with the number of boxes and this number is normally smaller than the number of boundary voxels. This implies a significant reduction on the labels and equivalences generated in the labeling pass and consequently a reduction of both execution time and memory requirements.

We say that our algorithm is image-size independent because if we scale a data set, the number of boundary voxels increases significantly while the number of boxes remains the same. However this is only true for images with the same shape. In general, increasing the size is associated with increasing the shape complexity and, then, the number of boxes also increases.

Two-manifold property

This algorithm allows to detect the well-composedness property of the volume as well as the critical zones where non-manifold configurations occur. This can be done in the labeling step when comparing boxes. If two boxes share a face or part of a face, then they will have the same label. If they are disjoint, then the new box will have a new label. But if they share an edge, a part of an edge or a vertex, then they can have the same label or not. In the former case, the algorithm will maintain the non-manifold zones. In the latter case, it will break the object at these zones and the obtained connected components will be 2-manifold. Fig. 14 and Fig. 13 show this method applied to a simple 3D example and to the *magallanes* example, respectively. Table 6 shows at the right hand side the number of connected components obtained for the two mentioned examples, maintaining (#M) or breaking (#B) non-manifold zones.

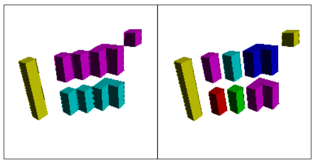

Fig. 14. CCL algorithm applied to a 3D data set without non-manifold detection(left) and with non-manifold detection(right)

6 Conclusions and future work

The aim of this work has been to use a concise and complete solid model, the EVM, to represent and process 2D images and volumes. EVM can represent binary data by storing implicitly the boundary of its continuous analog (an orthogonal polyhedron). Although EVM only stores some of the vertices of the object, it is a complete solid model and enjoys of formal properties like closure, orientedness, and connectedness.

We have studied and experimented the behavior of the EVM representing large data sets. We also have developed a method to obtain an isosurface from the EVM that avoids the excessive redundancy of primitives of most methods. Although our claim is that EVM behaves better for manipulation than for visualization, we have proposed an enriched EVM with normal vectors that give more realism to the rendered images.

We also have reported several operations developed for the EVM. Boolean operations, developed and experimented initially with solid models, behave also well with large volume data sets. Non-voxel based approaches have been developed for morphological operations and connected component labeling. Moreover, this last algorithm allows to detect non-manifold zones and to deal with them in the user desired way. We have presented results obtained with the EVM and we have also compared the EVM with the SB representation.

As a future work, we want to study the suitability of EVM, or some model devised from it, to represent and operate multi-modal volumes and also large time-varying data sets.

Acknowledgements

This work has been 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 Vertices Model to B-Rep and to Alternative Sum of Volumes. *Computing Suppl. Springer-Verlag*, (14):1 – 28, 2001.
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. D. Ayala, J. Rodríguez, and A. Aguilera. Connected component labeling based on the EVM model. In *Spring Conference on Computer Graphics 2002*, pages 55 – 63. Comenius University, 2002.
6. J. L. Coatrieux and C. Baillot. A survey of 3D display techniques to render medical data. In *3D imaging in Medicine*, volume 60 of *F: Computer and Systems Sciences*, pages 175 – 195. NATO ASI Series, 1990.
7. M.B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253 – 280, 1992.

8. G. J.Grevera, J. K. Udupa, and D. Odhner. An Order of Magnitude Faster Isosurface Rendering in Software on a PC than Using Dedicated, General Purpose Rendering Hardware. *IEEE Transactions on Visualization and Computer Graphics*, 6(4):335 – 345, 2000.
9. A. Kalvin, C. Cutting, B. Haddad, and M. Noz. Constructing topologically connected surfaces for comprehensive analysis of 3D medical structures. *SPIE. Image processing*, 1445:247 – 258, 1991.
10. Y. Kenmochi, A. Imiya, and A. Ichikawa. Boundary extraction of discrete objects. *Computer vision and image understanding*, 71(3):281 – 293, 1998.
11. T. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing*, 48:357 – 393, 1989.
12. L. Latecki. 3D Well-Composed Pictures. *Graphical Models and Image Processing*, 59(3):164 – 172, 1997.
13. C. Lee, T. Poston, and A. Rosenfeld. Holes and Genus of 2D and 3D Digital Images. *CVGIP: Graphical Models and Image Processing*, 55(1):20 – 47, 1993.
14. W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surfaces construction algorithm. *Computer Graphics*, 21(4):163 – 169, 1987.
15. C. Lürig and T. Ertl. Hierarchical volume analysis and visualization based on morphological operators. In *IEEE Visualization'98*, pages 335 – 341, 1998.
16. C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proceedings of visualization*, pages 281 – 287, 1994.
17. H. Muller and M. Stark. Adaptive generation of surfaces in volume data. *Visual computer*, 9:182 – 199, 1995.
18. J. Oikarinen, L. Jyrkinen, and R. Hietala. Volume rendering using seed filling acceleration: Supporting cut planes by fast re-seeding. *Computer-aided Surgery*, 4(4), 1999.
19. J. Rodríguez and D. Ayala. Erosion and dilation on 2D and 3D digital images: A new size-independent approach. In Ertl, Girod, Greiner, Niemann, and Seidel, editors, *Vision Modeling and Visualization 2001*, pages 143 – 150. Infix, 2001.
20. A. Rosenfeld, T. Kong, and A. Wu. Digital surfaces. *CVGIP: Graphical Models and Image Processing*, 53(4):305 – 312, 1991.
21. A. Rosenfeld, T. Y. Kong, and A. Nakamura. Topology-preserving deformations of two valued digital pictures. *Graphical Models and Image Processing*, 60(1):24 – 34, 1998.
22. R. Shekhar, E. Fayyad, R. Yagel, and J.F. Cornhill. Octree-based decimation of marching cubes. *IEEE Computer Graphics and Applications*, pages 335 – 342, 1996.
23. R. Shu, C. Zhou, and M. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11:65 – 70, 1995.
24. L. Thurfjell, E. Bengtsson, and B. Nordin. A boundary approach to fast neighborhood operations on three-dimensional binary data. *CVGIP: Graphical Models and Image Processing*, 57(1):13 – 19, 1995.
25. J. Udupa and O. Odhner. Fast visualization, manipulation and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 4(1):53 – 62, 1991.
26. J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201 – 227, 1992.
27. K. J. Zuidervelt. *Visualization of multimodality medical volume data using object oriented methods*. PhD thesis, U. Utrech. Deutch, 1995.