Fast Neighborhood Operations for Images and Volume Datasets

J. Rodríguez *and D. Ayala
Universitat Politècnica de Catalunya, Barcelona (Spain)
Email: [jrodri,dolorsa]@lsi.upc.es

Abstract

This paper presents a new approach to carry out erosion, dilation and connected component labeling. We use the Extreme Vertices Model (EVM), an orthogonal polyhedra representation, to describe binary images and volume datasets in a very efficient way.

Our proposal does not use a voxel-based approach but deals with the inner sections of the object. It allows to treat images and volumes indistinctly using the same algorithm and data structure with no overhead of memory and can be applied to manifold as well as non-manifold data. The connected component labeling algorithm actually detects non-manifold zones and permits to break or not the objects at these zones by an user-specified parameter.

Keywords: Images and Volume datasets, morphological operations, connected component labeling, non-manifold zones

1 Introduction

A volume dataset (or 3D digital image) can be represented as a map $f : Z \otimes Z \otimes Z \to R$, in such a way that every point is assigned a value representing its color. In a binary volume the image set is $\{0, 1\}$. For Rosenfeld et al. [12] a 3D digital image is defined as an union of voxels i.e. upright unit cubes whose vertices have integer coordinates. Analogously, a 2D digital image is defined as an union of pixels. In order to generalize this term, we will call the elements in a n-dimensional digital image as voxels.

Morphological operations such as erosion and dilation and its derivatives opening and closing are used in several tasks such as elimination of small spurious objects, smoothing of boundaries, object separation and elimination of small holes. Connected component labeling (CCL) of an image is a process which consists in identifying and labeling the separate different regions of interest of the image [13] and has applications in the fields of image understanding, volume visualization, character recognition and computer vision.

In this work we propose a new approach to perform erosion, dilation and connected component labeling operations using the Extreme Vertices Model (EVM) as a representation scheme of the input data. This representation model allows us to describe images and volume datasets in a very concise way. The CCL algorithm generates, from the EVM-represented object, a particular kind of spatial partitioning consisting in an ordered union of disjoint boxes (OUDB) and then applies, onto this set of boxes, an algorithm which follows the basic two-pass strategy first introduced by Rosenfeld and Pfaltz [13].

Our proposal does not use a voxel-based approach but deals with the inner sections of the object. We admit the use of box-shaped structuring elements of arbitrary size for erosion and dilation. The EVM is suitable for manifold as well as non-manifold data and the CCL presented algorithm can detect non-manifold regions allowing to break or not the objects at these zones.

The paper is arranged as follows. Next section includes related work and section three reviews the EVM showing the definitions and properties necessary to understand this work. Section four describes our proposal for both erosion and dilation operations and its derivations opening and closing using the EVM. Section five presents the CCL al-

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

gorithm, explaining how to obtain the OUDB partition and the connected boxes algorithm. Section six shows the performance of all the proposed algorithms by discussing experimental results. Finally, the last section concludes the paper and outlines future work.

2 Related work

A number of different approaches to improve 2D erosion and dilation algorithms have been reported. All of them use a more suitable representation of the image data to improve the performance. Boomgaard and van Balen [20] use a bitmap representation of binary images because the CPU operates on 32 pixels in parallel. Verwer and van Vliet [21] use a queue of the contour pixels to avoid traversing all the image each time. Parker [10] uses maps of distance and Young et al. [22] use a run-length representation of the image.

There are also 3D approaches. Zuiderveld [23] proposes a method for successive dilation operations of volumes by using a queue of contour voxels, following the idea of Verwer and Vliet [21]. Nevertheless, this proposal demands an overhead of memory. Thurfjell et al. [17] propose a method for fast erode and dilate using the semiboundary representation (SB) [18]. These two proposals traverse only the contour voxels in the 3D data set. However, they can not use structuring elements of arbitrary size. Furthermore, the algorithms for 2D and 3D data sets vary radically, that is, they need different algorithms for each case. More recently, Cardoner and Thomas [3] published a method where erosion and dilation are described as a sequence of intersections between the image and its displaced copies. This strategy allows them to operate run-length coded images but the generalization for 3D objects use no encoding schemes and loses efficiency. Lurig and Ertl [7] presented a technique that decomposes the multi-dimensional structuring elements into effectively one-dimensional ones using a pseudo tensor product. In [5] this technique has been accelerated using graphics hardware.

In our proposal, the algorithm actually carries out one-dimensional transformations as we will show later. In all methods above, the cost both in terms of memory requirements and in terms of algorithmic efficiency, increases when the binary data size increases because all of them use a voxel-based approach. Finally, all of them have been to transform discrete objects, except the approach in [3] which is also capable to work on solid modeling objects which are defined in the Euclidean space R^n.

Concerning with CCL, a great diversity of strategies have been devised to solve this problem: from hardware based solutions [8] to methods using optimization techniques. There are also algorithms that simultaneously perform surface tracking and connected-component labeling [19]. However the strategy followed by most developed approaches is based on using a more suitable representation instead of the raw data, in order to improve the corresponding algorithms. In 2D, representations such as run-length and quad-trees [14] among others have been proposed.

In 3D there has also been an effort to develop new representations for the data. The SB, created to store a segmented volume, has become a very useful model in fast visualization, manipulation and analysis of structures generated from volume data and in Thurfjell's work [17] a model based on the SB is used for performing erosion, dilation and connected component labeling for 3D volume datasets.

Dillencourt, Samet and Tamminen [4] formalized a general approach for connected component labeling which can deal with d-dimensional images represented in several data structures such as arrays, quad-trees and bintrees. Finally, Oikarinem et al. [9] present an algorithm for volume rendering using seed filling and supporting cut planes (restricted to orthogonal planes). The algorithm performs a 2D connected component search on the cut plane within the volume dataset in order to find seed points, which is based on that presented in [4].

Almost all of these approaches are voxel-based and therefore their performance decreases as the number of voxels of the image increases.

3 Review of the EVM

The *Extreme Vertices Model* (*EVM*) is a very concise representation scheme in which any orthogonal polyhedron (OP) or pseudo-polyhedron (OPP) can be described using only a subset of its vertices. A digital image is geometrically analogue to an OPP and can therefore be represented by the EVM.

Let P be an 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* (EV). Figure 1 shows an OPP with a brink from vertex A to vertex E (both extreme vertices). Vertices B, C and D are non-extreme vertices. The *EVM* is a representation scheme in which any OPP is described by its (an only its) set of EV.

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, that is, the interior of P. A *section* (S) is the resulting polygon from the intersection between a slice and an orthogonal plane. Each slice has its representing section. See Figure 1.

Planes of vertices and sections obtained from a 3D object are, then, 2D orthogonal polygons. From them, we can obtain their 1D lines of vertices and their 2D slices with their corresponding 1D sections. Finally, lines of vertices and 1D sections are 1D objects which are composed of one or several brinks.

In the EVM model the set of EV can be ordered in six possible ways depending on the coordinate values: XYZ, XZY, YXZ, YZX, ZXY, and ZYX. In an XYZ ordered EVM, planes of vertices perpendicular to the X axis appear ordered from low to high x values and, in each of them, lines of vertices parallel to the Y axis appear also ordered from low to high y values.

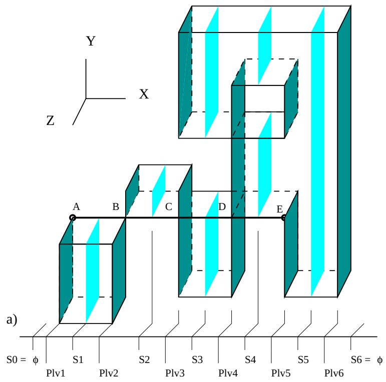

Figure 1: An OPP with a marked brink from vertex A to vertex E. Its planes of vertices and sections perpendicular to the X axis are shown in dark and light grey respectively.

The EVM has several interesting properties. The first one is that coordinate values of non-extreme vertices may be obtained from EV coordinates.

The second property allow sections to be computed from planes of vertices and vice-versa:

$$\begin{aligned} \overline{S_0(P)} &= \overline{S_n(P)} = \emptyset \\ \overline{S_i(P)} &= \overline{S_{i-1}(P)} \otimes^* \overline{plv_i(P)}, i = 1 \ldots n \\ \overline{plv_i(P)} &= \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)}, i = 1 \ldots n \end{aligned} \quad (1)$$

where $\overline{plv_i(P)}$ and $\overline{S_i(P)}$ denote the projections of $plv_i(P)$ and $S_i(P)$ onto a main plane parallel to P, n is the number of planes of vertices and $\otimes^*$ denotes the regularized XOR operation. Note that in order to operate with the projections we need not take into account the coordinate of the EV that corresponds to the projecting plane.

Finally, this last property concerns 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,

$$EVM(P \otimes^* Q) = EVM(P) \otimes EVM(Q) \quad (2)$$

This property means that the XOR operation works in 0D, because it is applied to the EV of the model. Therefore, sections are obtained from planes of vertices and vice-versa by simply applying the XOR operation to the extreme vertices.

The following property is a corollary of the previous one. If P and Q are quasi-disjoint, then

$$EVM(P \cup^* Q) = EVM(P) \otimes EVM(Q) \quad (3)$$

Several algorithms have been developed using the EVM. A general Boolean operations algorithm has been presented in [1], a conversion algorithm from a 3D dataset to the EVM in [11] and, as the EVM is a complete B-Rep model, in [2] a conversion algorithm from the EVM to a hierarchical B-Rep has been presented in [2].

4 Morphological operations using the EVM

Dilation and Erosion are the primary operations of mathematical morphology. Let A and B be sets in $Z^n, n > 0$.

The erosion of A by B, $A \ominus B$, is defined as

$$A \ominus B = \{x | \forall b \in B x + b \in A\} \quad (4)$$

which, in words, says that the erosion of A by B is the set of all points x such that B, translated by x, is contained in A.

Analogously, we can define the *dilation* of A by B, $A \oplus B$.

From the erosion and dilation operators we can define opening and closing operators as:

$$A \circ B = ((A \ominus B) \oplus B) \tag{5}$$

$$A \bullet B = ((A \oplus B) \ominus B) \tag{6}$$

Erosion and dilation can be described simply in terms of removing or adding voxels from the binary image, according to certain rules which depend on the pattern of the neighboring voxels and the size and shape of the structuring element.

But erosion and dilation can also be explained as processes where the interior of the object is shrunk or elongated as much as the size of the structuring element. The algorithms presented in this work identify the inner sections of an object directly from its EVM representation, then modify (shrink or elongate) these sections and, afterwards, compute the new EVM representation of the transformed object. Now we describe the *Erosion* algorithm. The *Dilation* algorithm can be described analogously.

Let $EVM(P)$ be the EVM representation of an OPP, P, and let plv_i and S_i be the ith plane of vertices and section of P respectively, $i = 0, 1, ...n$.

The erosion of P using the EVM is a process that, for each coordinate axis, traverses all the planes of vertices of $EVM(P)$ and computes its sections; then, it *shrinks* these sections and recomputes the new planes of vertices obtaining the EVM of the eroded P. This process uses the properties described by equations 1 and 2 (see section 3). The object can be seen as a sequence of slices and each slice is represented by its section, then shrinking a section causes the same effect to the corresponding slice and to the object.

This process is recursive over the dimension. It decomposes the 3D object into 2D sections which are themselves considered 2D images and decomposed again into 1D sections. This is the base case of the recursion and the algorithm shrinks these resulting 1D sections which are lists of collinear brinks.

The shrink operation, *Shrink(P)*, is defined by:

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

where *dim* is the dimension of P and *Shrink-1D (P)* is the process that shrinks each brink of the one-dimensional P. Figure 2 illustrates the recursion for the erosion of a single polyhedron for one direction.

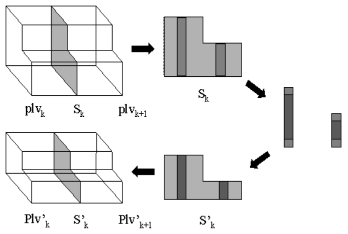

Figure 2: Single-direction erosion of a polyhedron

The shrinking depends on the size of the structuring element at the current shrink direction. The structuring element is a single measure (which can be different for each dimension) to shorten the brinks. It is not a mask as in the convolution like approaches. Thus, we can define structuring elements of arbitrary size. Nevertheless, in our approach, the structuring elements are always box-shaped.

The following algorithm shows this recursive process for the erosion operation along one of the main axes (the C axis).

```
procedure C-shrink (Input p: EVM, Input dim, ratio:
integer, Output q: EVM)
var
  plv, S0, S1, ShrunkS0, ShrunkS1, New-plv: EVM; nc:
integer
endvar
  if (dim = 1) then
    Shrink-1D (p, ratio, ShrunkS1);
  else
    dim:= dim - 1;
    S0:= ∅; ShrunkS0:= S0;
    plv:= ReadPlv (p, dim);
    while ¬ EndEVM (p) do
      S1:= S0 ⊗* plv;
      nc:= GetCoord (plv, dim);
      C-shrink (S1, dim, ratio, ShrunkS1);
      New-plv:= ShrunkS0 ⊗* ShrunkS1;
      SetCoord (New-plv, nc);
      PutPlv (New-plv, q, dim);
      S0:= S1; ShrunkS0:= ShrunkS1;
```

```
        plv:= ReadPlv (p, dim);
      endwhile
   endif
endprocedure
```

In this algorithm p is the initial object and q is the eroded one (in the C direction). It uses the basic operations of the EVM: *ReadPlv (p, dim)* returns the current plane of vertices of p. *PutPlv (p, q, dim)* adds the plane or line of vertices p to the EVM q. *EndEVM (p)* detects the end of the model. The $\otimes^*$ operation refers the properties described in section 3. Finally, *GetCoord (plv, dim)* gets the plane coordinate ($dim = 2$), or the line coordinate ($dim = 1$) of *plv* and *SetCoord (plv, nc)* sets the corresponding coordinate of *plv* as *nc*.

In order to apply this algorithm to the remaining directions, the EVM representation of the object (the set of extreme vertices) needs to be ordered accordingly. So the *Erosion* algorithm uses the operation *ABCSort (p, A)*, $A \in \{X, Y, Z\}$ in order to perform the complete erosion. *ABCSort (p, Z)* sorts the extreme vertices first by coordinate X, then by Y and then by Z.

```
function Erosion (Input p: EVM, Input dim, rx, ry, rz: integer) return EVM
var q1, q2, q3: EVM endvar
   ABCSort(p, X)
   C-shrink(p, dim, rx, q1)
   ABCSort(q1, Y)
   C-shrink(q1, dim, ry, q2)
   if dim=3 then
      ABCSort(q2, Z)
      C-shrink(q2, dim, rz, q3)
      return q3
   else
      return q2
   endif
endfunction
```

Procedure *Shrink-1D* shrinks a 1D-EVM which consists of one or more brinks. All brinks are shortened in both extreme vertices by the specified ratio and some of them may become null (see Figure 3).

Dilation only differs in the implementation of the 1D-transformation, changing the call to *Shrink-1D* for *Elongate-1D*, which elongates all the brinks. In this case, some of them can be merged (see also Figure 3).

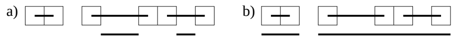

Figure 3: Procedures (a) Shrink-1D and (b) Elongate-1D applied to a 1D section composed by three brinks. In (a) the first brink becomes null and in (b) the two last brinks are merged. □ is the structuring element.

Opening and closing operators can be obtained from erosion and dilation:

```
function Opening (Input p: EVM, Input dim, rx, ry, rz: integer) return EVM
   return Dilation (Erosion (p, dim, rx, ry, rz), dim, rx, ry, rz)
endfunction
function Closing (Input p: EVM, Input dim, rx, ry, rz: integer) return EVM
   return Erosion (Dilation (p, dim, rx, ry, rz), dim, rx, ry, rz)
endfunction
```

5 EVM-based connected component labeling algorithm

The connected component labeling algorithm (EVM-CCL) presented in this section is based in the approach of [17] but using the EVM model and its particular partitioning.

5.1 Decomposing EVM into an OUDB

We first obtain a particular partitioning straightforward from the EVM which consists in an ordered union of disjoint boxes (OUDB). This partitioning can be thought as a special kind of cell decomposition like octrees, bintrees [4] and BSP [15] which have been used for representing volume datasets. The OUDB is axis aligned as octrees and bintrees but the partition is done along the object geometry like BSP.

As most of the algorithms dealing with EVM, the algorithm performing the OUDB decomposition from EVM is also recursive over the dimension. When $dim = 3$, the object is split at every plane of vertices perpendicular to a main axis, say X for instance, obtaining a set of slices. Then the

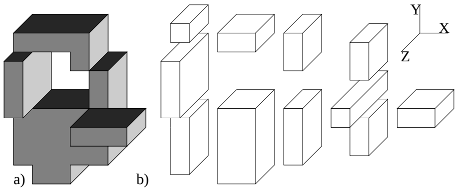

Figure 4: a) An OPP. b) XY decomposition.

algorithm applies recursively to the 2D section representing each slice, which is split at every plane of vertices perpendicular to another main axis, say Y, obtaining a set of 2D slices represented by their 1D sections which are composed by a set of collinear brinks. Each of these brinks gives a box. All the obtained boxes constitute the OUDB partitioning. An object can be decomposed into six different sets of boxes depending on the axes we choose to split the data: XY, XZ, YX, YZ, ZX, ZY. Figure 4 shows an object and the XY decomposition. Procedure EVMtoOUDB is as follows:

```
procedure EVMtoOUDB (Input p: EVM Input dim:
integer Input PCoord, LCoord: integer I/O q: OUDB)
  var br: Brink; S1, S2, plv: EVM endvar
  if dim = 1 then ReadBrink (p, br)
    while ¬ EndEVM (p) do
      AppendBox (q, br, PCoord, LCoord);
      ReadBrink (p, br);
    endwhile
  else { dim ≠ 1 }
    dim:= dim - 1
    S1:= ∅;
    plv:= ReadPlv (p, dim)
    while ¬ EndEVM (p) do
      S2:= S1 ⊗* plv;
      if dim = 3 then Pcoord:= GetCoord (plv, dim)
      else            Lcoord:= GetCoord (plv, dim)
      endif
      EVMtoOUDB (S2, dim, PCoord, LCoord, q)
      S1:= S2; plv:= ReadPlv (p, dim)
    endwhile
  endif
endprocedure
```

This procedure uses the basic operations of EVM, described in the previous section and function *ReadBrink*, that obtains a brink from a 1D EVM (which consists of a set of collinear brinks). It also uses procedure *AppendBox* that builds a box using the brink *br* and the values $PCoord$ and $LCoord$ and appends it to the OUDB q.

The inverse conversion algorithm, i. e., obtaining the EVM from the OUDB, is very simple. We have to perform the union of all boxes and as they are quasi-disjoint, these unions consists in doing the XOR operation among the extreme vertices of all the boxes (see property 3).

```
procedure OUDBtoEVM (Input q: OUDB I/O p: EVM)
  var evq: EVM endvar
  p:= IniEVM()
  for i in [1..NumberOfBoxes (q)] do
    evq:= BoxToEVM(q_i); p:= p ⊗ evq
  endfor
endprocedure
```

BoxToEVM converts a box to its EVM representation by putting its 8 vertices in an ordered way.

5.2 Connected boxes labeling algorithm

Once the OUDB partition has been achieved the connected boxes labeling process (CBL) takes place. Previous works use a sequential two-pass strategy which works at voxel-level [6], [17]. In the first pass all voxels are labeled but there remain some unsolved label equivalences which are sorted out in a second pass. Thurfjell et al.'s algorithm [17] improved this technique by working with the SB representation of the data, so they label only voxels belonging to the boundary of the object. The method we propose follows the classical two-pass approach (labeling and renumbering pass), but it traverses the set of disjoint boxes(OUDB) instead of a set of boundary voxels, further diminishing the number of elements to label.

5.2.1 The labeling pass

Suppose an OUDB partition obtained by splitting first at the X coordinate (X-slices) and then each X slice at the Y coordinate (XY slices). First, a label is set to the first box. Then, the remaining boxes are traversed and each of them is labeled according to a subset of the previous boxes: its neighbor boxes. This neighborhood includes those boxes in the immediate previous XY-slice and those boxes in the immediate previous X-slice. The following algorithm shows this traversal.

```
procedure CBL (I/O obj: OUDB)
  var i, j: integer; mark: boolean; EqT: table endvar
  FirstLabel (obj, 1)
  for i in [2..NumberOfBoxes (obj)] do
    j:= i - 1; mark:= false
    {Skip boxes in the same slice XY}
    while j ≥ 1 ∧ SameSliceXY (obj, i, j) do
      j:= j - 1
    endwhile
    {Compare with boxes in the previous slice XY}
    while j ≥ 1 ∧ SliceXYadjacent (obj, i, j) do
      if NeighborsXY (obj, i, j) then
      UpdateLabel (obj, EqT, mark, i, Label (obj, j))
      endif
      j:= j - 1
    endwhile
    {Skip boxes in the same slice X and in slices Y
    not adjacents}
    while j ≥ 1 ∧ SameSliceX(obj, i, j) do
      j:= j - 1
    endwhile
    {Compare with boxes in the previous slice X}
    while j ≥ 1 ∧ SlicesXadjacent (obj, i, j) do
      if NeighborsX (obj, i, j) then
      UpdateLabel (obj, EqT, mark, i, Label (obj, j))
      endif
      j:= j - 1
    endwhile
    if ¬mark then NewLabel (obj, i, EqTable) endif
  endfor
endprocedure
procedure UpdateLabel (I/O obj: OUDB, EqT: table,
mark: boolean, label1: integer, label2: integer)
  if ¬mark then
    AssignLabel (obj, i, Label (obj, j))
    mark:=true;
  else
    if label1 < label2 then
      SetEquivalence (EqT, label1, label2)
    else {label1>label2}
      SetEquivalence (EqT, label2, label1)
    endif
  endif
endprocedure
```

NewLabel assigns a new label to box i. *AssignLabel* sets box i with the label of box j. *SetEquivalence* saves label equivalences in the equivalence table EqT.

Figure 5 illustrates the labeling process with a 2D example. The OUDB partition has been computed giving a top to down and left to right ordered list of boxes. At the beginning all these boxes are unlabeled and the first box is set with label 1. Then, the labeled neighborhood of the second box is checked. As any neighbor box is found, the current box is set with a new label (label 2). Next the third box is checked with the set of previous and adjacent boxes (only box 1) and therefore box 3 is labeled with label of box 1. This process continues until all the boxes are labeled.

5.2.2 Setting equivalences

In the labeling pass, label equivalences can be detected. When a box i has two or more neighbors labeled with different labels, there is an equivalence between two or more regions. These label equivalences are saved in an *equivalence table*. Following the same labeling strategy proposed by Thurfjell et al. [16], this table has an entry for each label. The index corresponds to the region number and the content to its label. When a new label is set, an element is added to the table with the same value for the region number and label. In the Figure 5 box 15 has two neighbors with different labels (box 12 with label 5 and box 11 with label 4) and therefore a new label equivalence is set. Procedure *SetEquivalence* updates this table in the way shown below:

```
procedure SetEquivalence (I/O EqTable: Table,Input
label1, label2: integer)
  if EqT[label2]=label2∨EqT[label2]=label1 then
    EqT[label2]=label1
  else
    if EqT[label2] < label1 then
    SetEquivalence (EqT, EqT[label2], label1)
    else {EqT[label2]>label1}
      SetEquivalence (EqT, label1, EqT[label2])
    endif
  endif
endprocedure
```

Each time an equivalence is detected the equivalence table is updated to guarantee an ordered storing of nested equivalences.

5.2.3 The renumbering pass

The renumbering sorts out all the equivalences saved in the labeling pass. Setting equivalences in

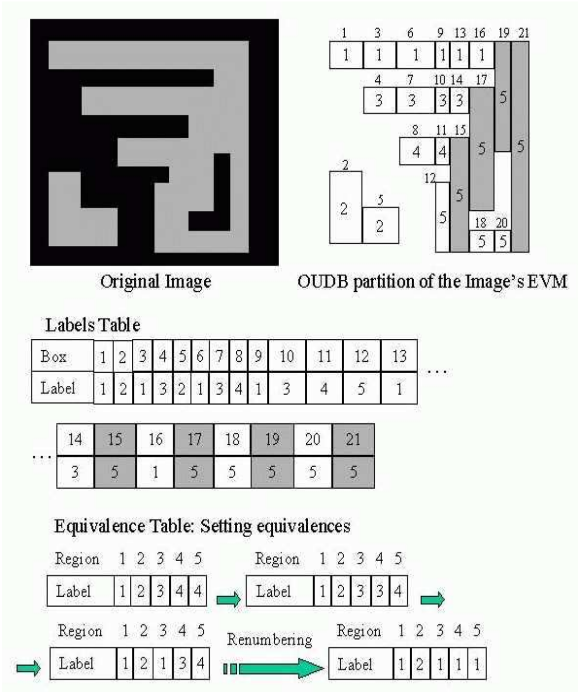

Figure 5: Up: Original image and its labeled OUDB. Equivalences detection in dark grey boxes. Bottom: Successive equivalences tables. Each dark grey cell in the Labels Table produces an equivalence to be saved in the Equivalence Table.

an ordered way, as explained above, has the advantage that it makes the renumbering easier. As the equivalences have been saved in increasing order, the nested equivalences are linked in increasing order too. So, the renumbering pass is just a traversal of the table in such a way that for each i: if $EqT[i] \neq i$ then $EqT[i] := EqT[EqT[i]]$, which has been processed previously. At the end, the nested equivalences will be propagated correctly and all the regions with equivalent labels will have the same label. The main drawback of this strategy deals with the size of the equivalence table. However, our proposal reduces considerably the number of needed labels as we will prove later. Figure 5 shows the final state of the equivalence table after the renumbering pass.

Finally Figure 6 shows with a 2D example the three steps carried out to achieve the EVM-based connected component labeling algorithm. First, the EVM is computed from the input data and its OUDB partition is obtained (each grey level is a box), then the labeling of boxes is performed and finally the equivalences are solved and the connected components stored as separated objects (each one with its own EVM).

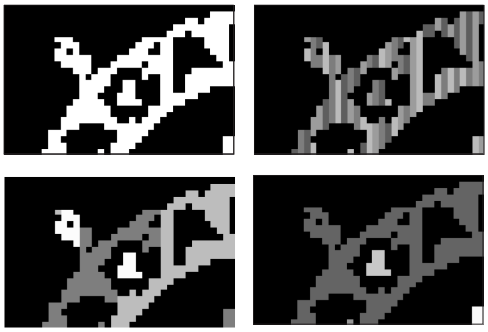

Figure 6: up-left: Original image, up-right: OUDB decomposition, bottom-left: OUDB labeled before renumbering pass and bottom-right: four connected components as separated objects.

5.3 Two-manifold property and other applications

When comparing two boxes, we can maintain or not the non-manifold zones. 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, 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, the algorithm will break the object at these zones and the obtained connected components will be 2-manifold. These cases are related to 6 and 26-adjacency respectively using the terminology of voxel based approaches.

A box is represented by the two ending vertices of its main diagonal and the $NeighborsXY$ and $NeighborsX$ functions perform comparisons between the corresponding coordinates. When these comparisons include the equality, non-manifold zones are maintained, otherwise the object is broken at these zones and the two-manifold property for each component is guaranteed. This choice can be defined by an user-specified parameter.

Figure 7 shows a simple 3D dataset with its components as 2-manifold or non-manifold objects. Each component with a different color.

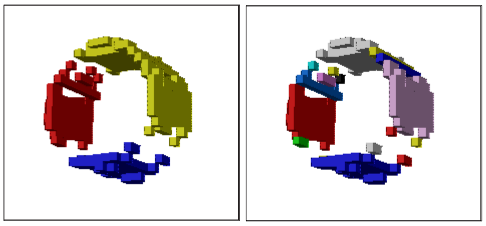

Figure 7: EVM-CCL algorithm applied to a 3D data set, without non-manifold detection(left) and with non-manifold detection(right)

Once we have each component we can, for instance, compute its volume and reject it if it is smaller than a specified threshold.

5.4 Analysis of the algorithm

The obtaining of the OUDB partition from the EVM of the input data is a needed preprocess for the method. As most of the algorithms concerned with EVM it needs to compute sections. This computation has a quadratic worst case complexity, but the average experimental complexity is slightly greater than linear, $nv^{1.2}$, nv being the number of extreme vertices.

The complexity of the EVM-CCL algorithm is $O(nb)$, nb being the number of boxes of the OUDB representation. This number depends on the decomposition but we cannot know it a priori from the EVM model. The renumbering pass (as in Thurfjell's algorithm) is $O(nl)$, nl being the number of labels generated in the labeling pass. So our method will have better performance in images (volumes) where the boxes decomposition produces fewer boxes than boundary pixels (voxels). This is true in general except in extremely sparse datasets where almost each pixel corresponds to a box.

6 Experimental Results

All the algorithms presented have been implemented in C++ on a Sun Microsystem ultrasparc60 machine. In this section we present several experimental results. As our approach is based on the extreme vertices and other approaches are voxel based, we first present a comparative table showing the growth of the quantity of elements to be treated when the resolution of the image grows. Table 1 shows the number of brinks (#brinks), boundary and interior voxels (#biv), boundary voxels (#bv) and interior (black) nodes for an octree representation (#Obn), for several resolutions of the truck shown in Figures 12 (up).

Datasets	#brinks	#biv	#bv	#Obn
truck016	36	146	142	146
truck032	118	666	567	659
truck064	452	3588	2633	3336
truck128	1238	20220	11457	15614
truck256	4009	127280	49557	71147
growth	3.271	5.469	4.328	4.703

Table 1: Growing of elements

The growth factor of the whole number of voxels when the resolution grows from $(2^n)^3$ to $(2^{n+1})^3$ is 8 which is greater than the average growth factor of any of the columns of table 1, shown in the last row. The greater factor corresponds to the representation using boundary and interior voxels. The least corresponds to the extreme vertices. As the growth factor corresponding to boundary voxels is lesser than that corresponding to extreme vertices, we can expect that our approach will have better processing times that the approach based on the semiboundary representation which deals with boundary voxels.

6.1 Morphological algorithms

Figure 8 presents an image with its corresponding eroded images using a conventional voxel-based approach and using our algorithm. The final result (white regions) in both cases are, of course, the same. The result obtained with the voxel-based approach shows in grey, pixels that would be eliminated.

Figure 9 shows with a very simple image how erosion and dilation work thinning and widening the object. Note that the erosion operation has eliminated a thin part on the twice eroded image.

Figure 10 presents a non-uniform dilation, applied only in the X-direction. Note the unchanging white line at the bottom and note also that the im-

Figure 8: Original image (top), voxel-based eroded image (middle) and EVM-eroded one (bottom).

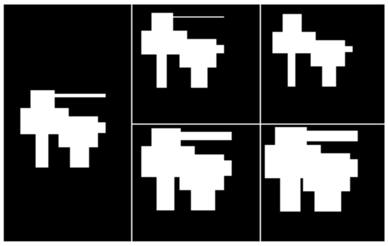

Figure 9: Left: original image. Middle- and right-up: eroded once and eroded twice. Middle- and right-bottom: dilated once and dilated twice.

age was initially a set of separated letters and after dilation became a single object.

Figure 10: Non-uniform dilation. Top-Down: original image, dilated once, dilated twice.

Figures 11 and 12 show a 2D and a 3D example, respectively.

Table 2 presents, for data sets of Figures 11 (up) and 12 (up) its resolution, the number of boundary voxels, the number of brinks and the processing time (in seconds) for erosion, dilation, opening and closing operations.

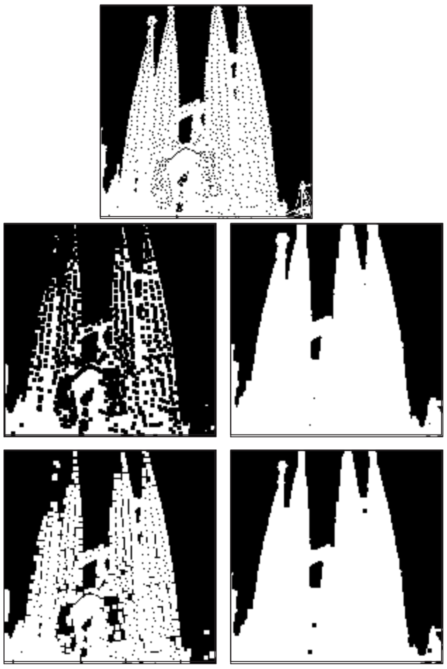

Figure 11: Up: Original image (church); middle-left and right: eroded and dilated once; bottom-left and right: Opening and Closing.

6.2 CCL algorithm

For the CCL method, we present the results obtained with image *logoUPC* (Figure 13), image *Church2* (Figure 14 or Figure 11, bottom-left) and a volume dataset *Truck2* (Figure 15 or Figure 12 middle-right).

Table 3 shows the time taken for the OUDB partition (in seconds). The other rows on the table are the number of components observed with and without non-manifold detection (on the table 4{6}-

Data set	Church (150x150)	Truck (256^3)
#bound	1471	49557
#brinks	1667	4009
Erosion	0.26	3.09
Dilation	0.1	3.66
Opening	0.49	3.16
Closing	0.15	6.01

Table 2: Results for morphological algorithms.

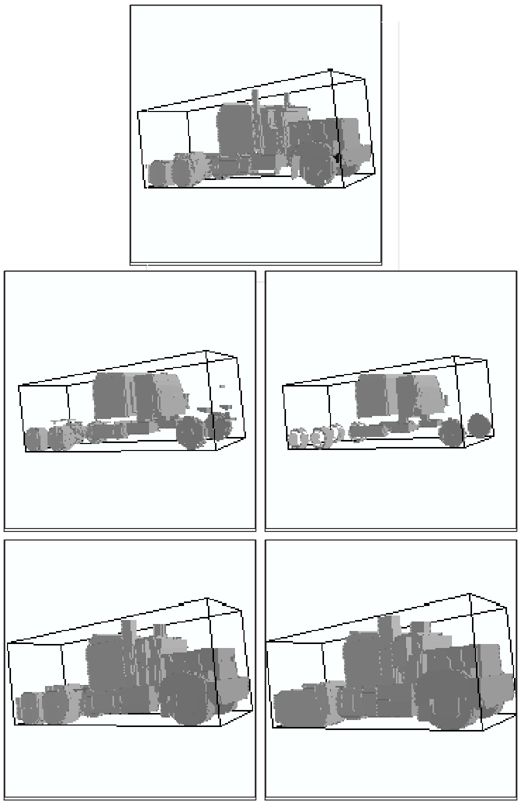

Figure 12: Up: Original object (truck); middle-left and right: eroded once and twice; bottom-left and right: dilated once and twice.

adj and 8{26}-adj respectively), and the number of boundary pixels or voxels and the number of boxes resulting by the OUDB decomposition. These are the elements considered in voxel-based and EVM approaches respectively

The next table 4 presents, per each dataset described in the previous table, a comparison between the average execution time (in seconds) using a voxel-based algorithm and our EVM-CCL algorithm.

Finally, the EVM-CCL algorithm allows significant reduction on the labels and equivalences generated through the labeling pass, with a consequent reduction on time and memory requirements. Figure 16 illustrates the conventional labeling pass (left) and the boxes labeling pass (right). Both images have been processed in the same order, top to down and left to right. Each level of grey represents a different label and it can be noted that the

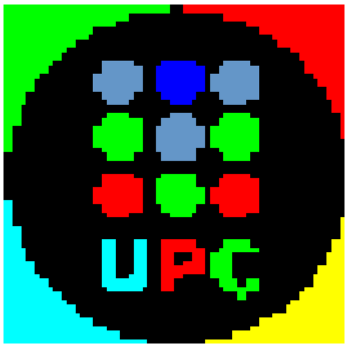

Figure 13: Connected components of *logoUPC*.

Datasets	logoUPC (324x332)	Church2 (150x150)	Truck2 (256^3)
OUDB time	0.01	0.06	0.14
4(6)-adj	16	18	81
8(26)-adj	16	17	15
#bound	4324	4636	14249
#boxes	177	797	1413

Table 3: Description of datasets for the CCL algorithm.

left image has more labels than the right one. This is due to the OUDB partition that allows to check adjacency along many pixels (voxels) at the same time.

Table 5 presents the number of labels and equivalences generated by the voxel-based method and the EVM-CCL algorithm for several datasets.

Datasets	logoUPC	Church2	Truck2
voxel-based:			
Labeling	0.15	0.04	8.77
Renumbering	0.06	0.01	8.23
Total	0.24	0.19	21.45
EVM-CCL:			
Labeling	0.0001	0.02	0.26
Renumbering	0	0.0001	0.0001
Total	0.01	0.08	0.4

Table 4: Time results for the CCL algorithm.

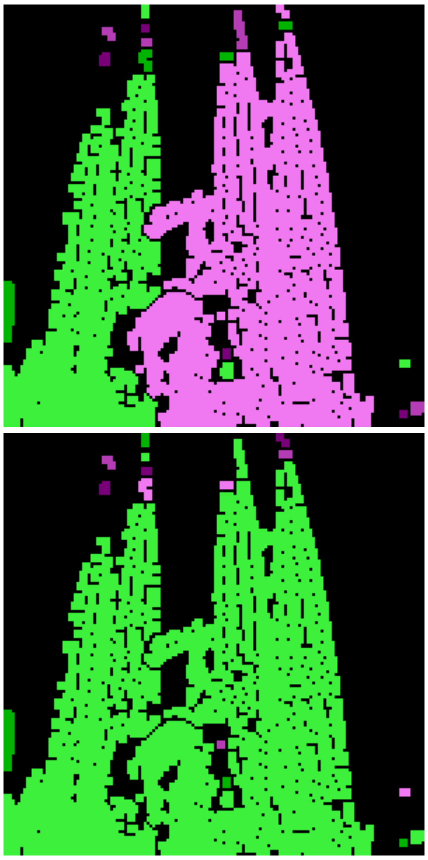

Figure 14: *Church2* with and without non-manifold detection.

7 Conclusions and future work

The algorithms presented in this work treat images and volumes in both real space and its discrete equivalent. They don't follow a voxel based approach, instead they work with the sections of the model EVM used to represent the objects.

The erosion and dilation algorithms admit the use of box-shaped structuring elements of arbitrary size.

The CCL algorithm decomposes the object in a particular set of disjoint boxes(OUDB). Each box contains many voxels and then the algorithm produces less labels and equivalences and is faster than the voxel-based ones. Moreover, this algorithm can also detect non-manifold zones and separate the object at them.

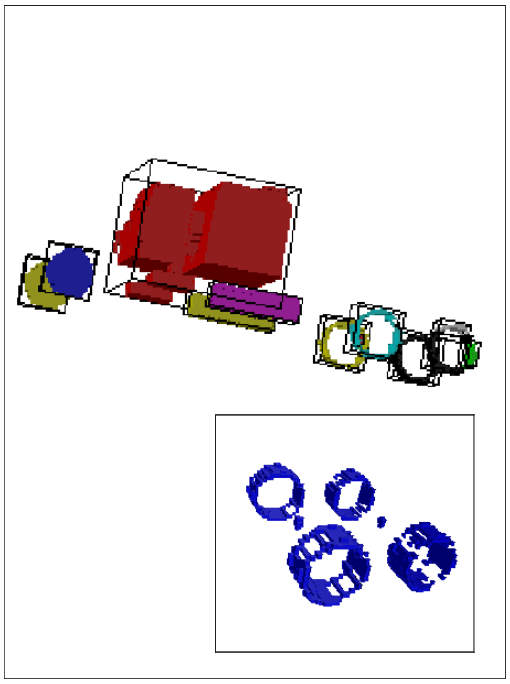

Figure 15: *Truck2* with a detail of its wheels.

As a future work, we are studying thinning methods, the detection of holes and computation of the genus of 2D and 3D images and other operations using the EVM. Furthermore we are trying to extend it in order to represent and operate with multimodal images.

Acknowledgements

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

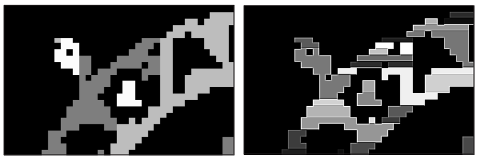

Figure 16: Generation of Equivalences EVM-based method (left) and pixel-based method (right). Each change of color on the same component indicates an equivalence

Datasets	voxel-based		EVM-CCL	
	Labels	Equival.	Labels	Equival.
Detail	29	24	8	3
logoUPC	50	34	16	0
Church2	99	82	47	30
Truck2	320	305	33	18

Table 5: Labels and equivalences generated in the labeling pass

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. Converting Orthogonal Polyhedra from Extreme Vertices Model to B-Rep and to Alternative Sum of Volumes. *Computing Suppl. Springer-Verlag*, 14:1 – 28, 2001.

[3] R. Cardoner and F. Thomas. Residual + directional gaps = skeletons. *Pattern recognition letters*, 18:343 – 353, 1997.

[4] 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.

[5] M. Hopf and T. Ertl. Accelerating morphological analysis with grapgics hardware. In *Vision Modeling and Visualization VMV'2000*, pages 337 – 345. Infix, 2001.

[6] T. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing*, 48:357 – 393, 1989.

[7] C. Lürig and T. Ertl. Hierarchical volume analysis and visualization based on morphological operators. In *IEEE Visualization'98*, pages 335 – 341, 1998.

[8] C. J. Nicol. A systolic approach for real time connected component labeling. *Computer Vision and Image Understanding*, 61(1):17 – 31, 1995.

[9] 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.

[10] J. Parker. A system for fast erosion and dilation of bi-level images. *Science computer*, 5(3):187 –198, 1990.

[11] J. Rodríguez, D. Ayala, and A. Aguilera. *Geometric Modeling for Scientific Visualization*, chapter A Complete Solid Model for Surface Rendering. Springer-Verlag, 2003. submitted.

[12] A. Rosenfeld, T. Kong, and A. Wu. Digital surfaces. *CVGIP: Graphical Models and Image Processing*, 53(4):305 – 312, 1991.

[13] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471 – 494, 1966.

[14] H. Samet. Connected component labeling using quadtrees. *Journal of the ACM*, 28(3):487 – 501, 1981.

[15] K. R. Subramanian and B. F. Naylor. Converting Discrete Images to Partitioning Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(3), 1997.

[16] L. Thurfjell, E. Bengtsson, and B. Nordin. A new three-dimensional connected component labeling algorithm with simultaneous object feature extraction capability. *CVGIP:Graphical Models and Image Processing*, 54(4):357 – 364, 1992.

[17] 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.

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

[19] J. K. Udupa and V. G. Ajjanagadde. Boundary and object labeling in three-dimensional images. *Computer Vision, Graphics and Image Processing*, 51(3):355 – 369, 1991.

[20] R. van Boomgaard and R. van Balen. Methods for fast morphological image transforms using bitmapped binary images. *CVGIP: Graphical Models and Image Processing*, 54(3):252 – 258, 1992.

[21] L. van Vliet and B. Verwer. A contour processing method for fast binary neighborhood operations. *Pattern Rec. Letters*, 1(1):27 – 36, 1988.

[22] I. Young, R. Peverini, R. Verbeek, and P. van Otterloo. A new implementation for the minkowski operators. *Computer Graphics and Image Processing*, 17:189 – 210, 1981.

[23] K. J. Zuiderveld. *Visualization of multimodality medical volume data using object oriented methods*. PhD thesis, U. Utrech. Deutch, 1995.