Efficient processing of Minkowski functionals on a 3D binary image using binary decision diagrams

I. BLASQUEZ
LICN–IUT, Université de LIMOGES
Allée André Maurois
87065 LIMOGES CEDEX, FRANCE
isabelle.blasquez@unilim.fr

J.-F. POIRAUDEAU
LICN–IUT, Université de LIMOGES
Allée André Maurois
87065 LIMOGES CEDEX, FRANCE
poiraudeau@unilim.fr

ABSTRACT

The Morphological Image Analysis characterizes binary digitized 3D images in terms of shape (geometry) and connectivity (topology) by means of the Minkowski functionals known from integral geometry. In three dimensions, these functionals correspond to the enclosed volume, surface area, mean breadth and connectivity (Euler characteristic). To compute these functionals, it is necessary to count the number of open cubes, open faces, open edges and open vertices of the discretized object in the 3D image.

In this paper we propose a new method to count the number of these geometric elements in a discretized binary image. We focus on the local configuration around a voxel and we report a fast algorithm for computing discrete Minkowski functionals with related topological conditions using binary decision diagrams. These diagrams could be applied to several binary image processing algorithms which evaluate a discrete function for small parts of this image. We also choose to create and implement a reduced and ordered triple-ADD adapted to our problem. We show that this algorithm is 17 times faster than the algorithm proposed recently in the literature by Michielsen. Moreover, large volumes of data, which become increasingly accessible and current,can be treated thanks to this algorithm.

Keywords
Minkowski functionals, image processing, binary decision diagram, mathematical morphology.

1 Introduction

Many fields of science are very data intensive and often these data are obtained as images. The interpretation of images involves some kind of image processing. A key problem in low level vision is to reduce the raw image data and to construct a more descriptive representation in terms of relevant features, which can be used more effectively by other level processing. Hence, the main purpose of image analysis is to provide a quantitative characterization of the shape, structure and connectivity of the constituents. Therefore, how to form a fast algorithm is of extreme importance in applications [1].

Journal of WSCG, Vol.11, No1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

The purpose of this paper is to describe an efficient and versatile method to compute the morphological properties of images.

The mathematical morphology aims to characterize 3D shapes thanks to four measures [2]. These measures, also called Minkowski functionals (or intrinsic volumes, quermass integrals), are respectively the volume, the area, the mean breadth and the Euler-Poincaré characteristic. They were initially defined for convex objects in the field of integral geometry. In the late fifties, Hadwiger showed that every measure on the finite union of compact convex sets can be written as a linear combination of the four Minkowski functionals. These functionals have a property in common: they are additive. The functional $\mathcal{M}_\nu$ of the union $A \cup B$ of two convex sets A and B is the sum of the functional of the single convex sets subtracted by the intersection $\mathcal{M}_\nu(A \cup B) = \mathcal{M}_\nu(A) + \mathcal{M}_\nu(B) - \mathcal{M}_\nu(A \cap B)$. This relation generalizes the common rule for the addition of the volume of two convex sets to the case of a general morphological measure, i.e. the measure of the double-counted intersection has to be subtracted.

Over the last few years, these functionals have been widely used in a number of fields such as de-

termination of the large scale structures of the universe [3] and modelling of porous media [4]. However, they are not yet much used by the image processing scientists. In the case of discrete images, the property of additivity simplifies the processing of the functionals.

2 Problem presentation

We define an $n-$dimensional discrete space as a lattice grid of $\mathbb{Z}^n$. The nodes of this grid are generally represented by hypercubes centered on the nodes.

A 3D binary image is usually defined as a cubic grid of $\mathbb{Z}^3$ and represented by a lattice of voxels, where each voxel may have only one of the two values, say 0 or 1. The set of all voxels with a 1-value is called the object of the image: they are also called black voxels or full voxels or active voxels. The set of all voxels with a 0-value is called the background of the image. They are also called white voxels or empty voxels or non-active voxels.

By considering each voxel as the union of the disjoint collection of its interior, open faces, open edges and open vertices, it is possible to determine the four Minkowski functionals for a 3D discretized object: the enclosed volume ($V = n_3$), the surface area ($S = -6n_3 + 2n_2$), the mean breadth ($2B = 3n_3 - 2n_2 + n_1$) and the connectivity or Euler characteristic ($\chi = -n_3 + n_2 - n_1 + n_0$) where n_3 is the number of open cubes (or open voxels), n_2 the number of open faces, n_1 the number of open edges and n_0 the number of open vertices [5].

The morphological characterization of a 3D object is thus reduced to the enumeration elementary geometrical objects (open cubes, open edges, open faces, open vertices) which constitute this object. For example, figure 1 shows a 3D simple object with 3 black voxels.

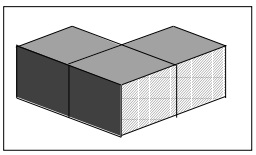

Figure 1: Three black voxels: $n_3 = 3$ open cubes, $n_2 = 16$ open faces , $n_1 = 28$ open edges and $n_0 = 16$ open vertices.

The enumeration of the number of open faces, open edges, open vertices for a 3D image first requires to examine all the 26 neighbors of each voxel and then to calculate a Boolean function with these values. In the literature, only the implementation proposed by Michielsen and De Raedt in [6] seems to be used for this problem. In this article, they describe equations which determine how the number of open faces, open edges and open vertices changes when one adds one black voxel to a given 3D pattern to the position $x = (i, j, k)$. By using these equations, it is easy to compute the Minkowski functionals for a given pattern, simply by adding the black voxel one-by-one to an initially complete white background. However, we find that the method suggested in [6] has two drawbacks. The first one relates to the algorithm itself, especially the number of neighbors to be explored: all the 26 neighbors of the current voxel must be examined to be able to solve the equations. The second one relates to the implementation of the algorithm. Two arrays are used: the first one to store the whole 3D analyzed image, the other to treat the 3D image being analyzed (it is filled voxel by voxel), this is expensive in memory and execution time. To our knowledge, this algorithm was implemented in FORTRAN 90 on a single processor. To limit the number of accesses to the neighbors of a current voxel V and to avoid the use of two arrays, we decided to tackle the problem differently.

First, we focused on the algorithmic considerations, based on geometry which are used to reduce the problem into smaller sub-problems, for instance by taking into account the symmetries of the problem. The sub-problems are then usually solved by using lookup-tables or quadtrees [7]. But an approach with a such large lookup-table is undesirable because of memory occupation. Indeed, all the input variables are examined once in order to compute the address of the entry in the lookup-table : if n voxels are examined, the number of entries of lookup-tables is 2^n. Sometimes a trade-off is chosen between space and time complexity. In [8] another approach was proposed. It relies on the use of binary diagrams of decision (BDDs) to generate automatically a very efficient code for image processing algorithms (such as the test of simplicity or the thinning in 3D). The authors also produce functions must faster than the previous implementations, reducing the execution time by a factor up to 20.

Then, we decided to consider an approach by BDDs as a convenient representation for the discrete functions of booleans variables used in our problem of counting the number of open faces, open vertices and open edges. We will implement our algorithm in C/C++ on a single processor.

3 Towards a fast enumeration of the morphological characteristics of a 3D discrete image

We consider the 3D image as a pure volume: it is decomposed by a number of planes for the algorithm. We also consider that the lattice is traversed from the

left column to the right, in line upwards, and from the front plane of the image to the backwards plane. In figure 2, we represent the local configuration around the current voxel V by a sub-lattice. The 26 neighbor voxels are noted Nip where i indicates the position of the voxel in the plane ($0 \leq i \leq 8$) and p indicates the plane in which the neighbor voxels are located (1: front plane, 2: current plane, 3: back plane). For each new black current voxel V, we always try to determine the number of open faces (Δn_2), open edges (Δn_1), and open vertices (Δn_0) introduced by the insertion of this voxel into the lattice.

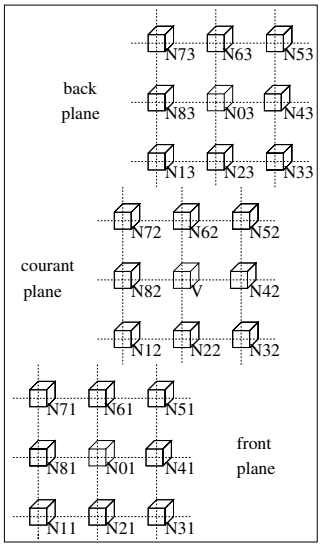

Figure 2: 26 neighbor voxels of the current voxel V

First, we reduce the problem by examining only the 13 neighbor voxels preceding the current black voxel ($Ni1$ for $i = 0..8$, $Nj2$ for $j = 1, 2, 3, 8$). The order of study of the value of these voxels depends on the exploration sense of the lattice. The black current voxel V will always add at least: 3 open faces, 3 open edges and 1 open vertex to the 3D enumeration, whatever the value of the 13 other neighbor voxels (black or white). These neighbor voxels will be examined one by one when the lattice goes on to be explored.

For each geometric element (open face, open edge, open vertex) of the current voxel, it is necessary to determine if this element has already been counted. All the preceding neighbor voxels likely to share this element with the current voxel must be examined: if only one neighbor voxel is black, the considered element should not be taken into account, because it has already been counted in the enumeration. The maximum number of open faces which can be added is 3, the maximum number of open edges is 9, the maximum number of open vertices is 7. We also define the following discrete equations which depend on the 13 preceding neighbor voxels:

Number of open faces : $\Delta n_2 =$

$$3 + Q01 + Q22 + Q82 \quad (1)$$

Number of open edges : $\Delta n_1 =$

$$3 + Q01.Q22.Q21 + Q01.Q41 + Q01.Q61 + Q01.Q82.Q21 + Q22.Q32 + Q82 + Q12.Q22.Q82 + Q22 + Q82 \quad (2)$$

Number of open vertices : $\Delta n_0 =$

$$1 + Q82 + Q22.Q32 + Q12.Q22.Q82 + Q01.Q41.Q51.Q61 + Q81.Q01.Q61.Q71.Q82 + Q21.Q31.Q41.Q01.Q22.Q32 + Q11.Q21.Q01.Q81.Q12.Q22.Q82 \quad (3)$$

with $Qij = 1 - Nij$,

where $Nij = 1$ for a black voxel (object) and $Nij = 0$ for a white voxel (background).

The experimental evaluation will show that the execution time being much faster than the algorithm of Michelsen is already significant by only using these equations.

Then, we are still looking for an optimization for our problem. We propose an algorithm which only examines the neighbor voxels whose values change the result. This algorithm uses the same technique as that used in the image processing which consists in evaluating a Boolean function by transforming these equations in Binary Decision Diagrams (presented in the next section). This algorithm is also composed of series of branching tests judiciously ordered to examine the lowest number of neighbor voxels for each current voxel: the tests will be stopped as soon as we ascertain that a remaining geometric element has already been shared by a neighbor voxel, as be studied in section 5.

Several authors, as [9, 10], noted the benefit of representing images as a finite cell complex. A 3D image can be decomposed into black or white voxels. But it is also possible to give a complete description of an area of a 3D image simply by using the borders which separate it from the close areas. This space decomposition is called intervoxel decomposition. A cell of dimension 3 (3-cell) is an open cube. A 2-cell is a face shared by two adjacent 3-cells. A 1-cell is an edge shared by two adjacent 2-cells and a 0-cell is a vertex shared by two adjacent 1-cells. Each single voxel consists of one 3-cell, six 2-cells, twelve 1-cells, eight 0-cells. In such a representation, our problem is limited to the enumeration of cells. When we examine a new current voxel, the intervoxel decomposition can be included in equations (1)-(3). For example, equation (2) becomes:

Number of open edges: $\Delta n_1 = 12 -$ Number of 1-cells

$$\Rightarrow \text{Number of 1-cells} = 9 - (Q01.Q22.Q21 + Q01.Q41 + Q01.Q61 + Q01.Q82.Q21 + Q22 + Q82 + Q12.Q22.Q82 + Q22.Q32 + Q82) \quad (4)$$

4 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) are compact and efficient representations of the symbolic manipulation of boolean functions. Their concept was introduced by Lee et Akers [11]. Over the last few years, BDDs have been used efficiently in many fields for many tasks such as digital-system design, combinatorial optimization, mathematical logic, artificial intelligence [12], as well as image processing [13] and image encoding [14].

4.1 Boolean functions with Boolean variables

The values of boolean variables are in $\mathbb{B} = \{0, 1\}$. A binary decision diagram (BDD) represents a boolean function $f(x_1, x_2, ..., x_n)$: $\mathbb{B}^n \rightarrow \mathbb{B}$ as a directed acyclic graph, each node corresponds to a test of a boolean variable x_i (Shannon representation) :

$$f = \bar{x_i} f_{x_i=0} + x_i f_{x_i=1} \ (1 \leq i \leq n)$$

Each node has two children which are also BDDs. At each node, a child is chosen according to the value of the variable associated to this node. The function value is determined by tracing a path from the root to a terminal node following the appropriate branch from each node. Terminal nodes of the graph are the function values of $\mathbb{B} = \{0, 1\}$.

A BDD is called *ordered* if each variable is counted at most once on each path from the root to the terminal node and if all the variables appear in the same order along all paths from the root to terminal nodes. So the co-factoring variables (splitting variables) always follow the same order:

$$(x_1 < x_2 < ... < x_n)$$

A BDD is called *reduced* if it respects the following reduction rules : any node with two identical children is removed and two nodes with isomorphic BDDs are merged.

An ordered and reduced Binary Decision Diagram is unique. The Binary Decision Diagrams have really been developed after an article by Bryant [12] which defined the ordered BDD (OBDDs) like a subset of BDDs. The OBDDs are one of BDDs where the variables are strictly ordered from the root to the terminal nodes. One of the advantages of OBDDs is canonicity: if variable ordering is fixed and reduction is applied, two equivalent boolean functions are guaranteed to have the same BDD. An OBDD in its canonical form is also called Reduced Ordered Binary Decision Diagram (ROBDD).

Algorithms for the handling of graphs can be applied to BDDs. Their complexity is calculated in polynomial time in the size of the BDDs and they generate canonical graphs. But one of the disadvantages of ROBDDs is that their efficiency depends essentially on the variable ordering. The determination of the most effective order is a difficult NP-problem. Moreover, the more appropriate order can evolve during the construction of a ROBDD or during its use. Thus, most ROBDDs are based on a dynamic variable ordering implementation [15].

4.2 Discrete functions

The size of the BDD can be exponential in the number of variables. A solution to the problem of this combinatorial explosion of some representations with ROBDDs is to extend the concept and to represent numeric-valued functions over Boolean variables, with non-Boolean ranges, such as integers. Thus one representation allows to represent all the range of terminal values instead of using one BDD per value.

The Algebraic Decision Diagrams (ADDs) as the Multi-Terminal BDDs (MTBDDs) [16] are derived from BDDs where terminal nodes represent arbitrary integer values in $\mathbb{Z}$, not restricted to $\mathbb{B}$.

An ADD represents a numeric-valued function $f(x_1, x_2, ..., x_n)$: $\mathbb{B}^n \rightarrow \mathbb{B}$ where each node is submitted to a test of a boolean variable x_i. Keeping the boolean variables allows the use of branching structure similar to BDDs. The discrete functions having numeric range can efficiently be represented with ADDs. For example, the function : $f(x_0, x_1, x_2) = x_0 + 2 * x_1 + 4 * x_2$ corresponds to the unsigned integer value of the bits vector "x_0, x_1, x_2" .

5 Which BDD should be used to count the number of geometric elements?

The efficiency of a BDD is strongly related to its size and its cost of construction. These parameters depend essentially on three points: the function to be represented, the ordering of the boolean variables, and the strategy used for the construction of the diagram.

In this section, we present our approach to implement the discrete functions corresponding to equations (1)-(3) step by step. First, we choose the BDD adapted to our problem, then, we propose an optimal variable ordering.

5.1 Choice of a BDD

In [8], Robert et Malandain proposed to use ROBDDs for classical image processing techniques. All of these techniques rely on only one boolean function with boolean variables and for a given pixel the value of its neighbors must be analyzed.

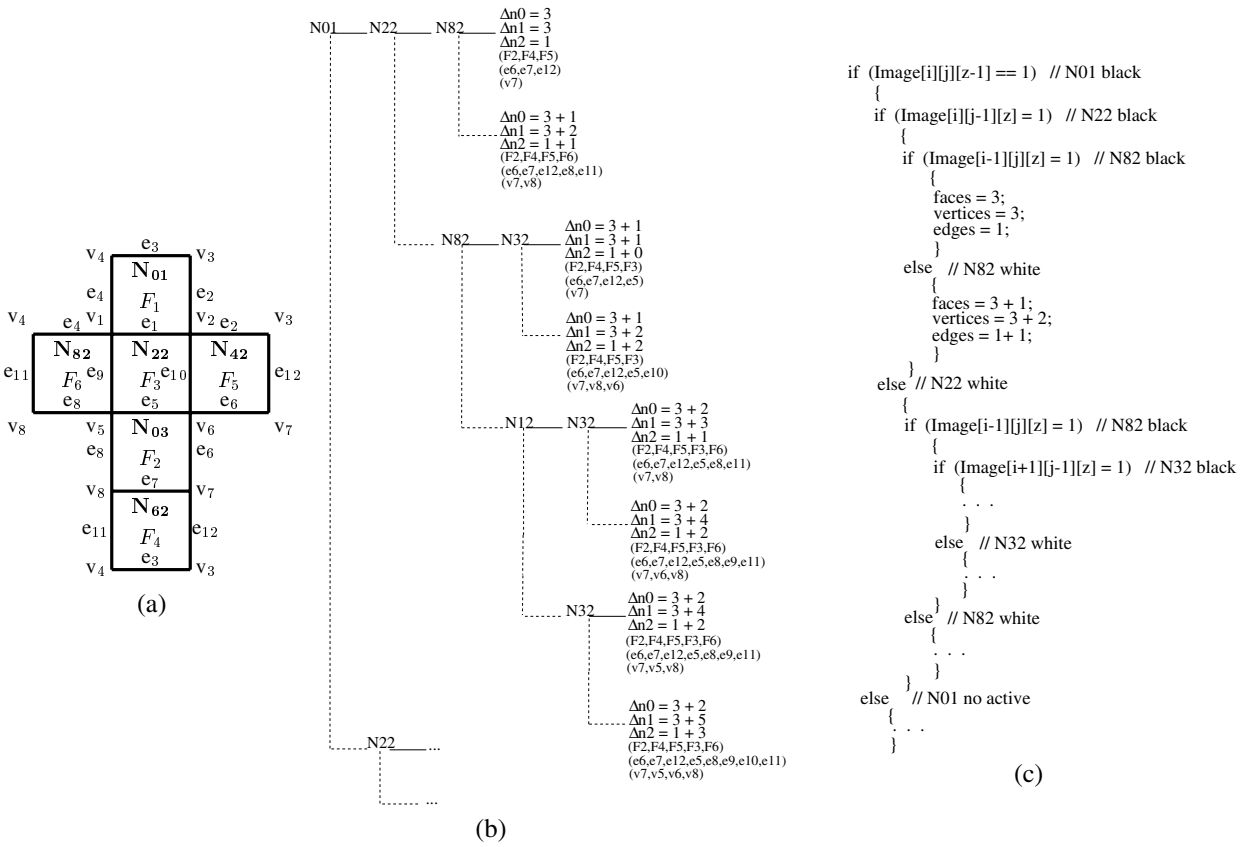

Figure 3: Triple-ADD: the solid lines correspond to the branchings "then" (black voxels), the dotted lines correspond to the branchings "else" (white voxel).

Our problem consists in implementing in the same BDDs the three functions (Δn_2, Δn_1, Δn_0) presented in section 3. There are discrete functions with boolean variables. We also choose to use an ADD. Although these discrete functions depend on the value of the 13 neighbor voxels, they are independent. It could thus be possible to associate each equation with one ADD: we would then have three ADDs to implement. However, we want to develop a fast algorithm. Thus, we focus on the following criterion: we try to minimize the number of accesses to image data. To respect it, we decide to implement only one ADD for all the three functions. We call it: *triple-ADD*. Each terminal node of this triple-ADD does increment not one equation, but three discrete equations. That is why it is impossible to use an existing package to create this particular triple-ADD.

On the one hand, using only one triple-ADD for three discrete functions is much faster than using three simple ADDs. On the other hand, the choice of the variables ordering for a simple ADD is easier to implement because it corresponds to the canonical form of the studied function. For a triple-ADD, we must define an order according to the booleean variables used in the three functions by considering their occurrence in these equations. We also decide to use the 3D topology, especially the connectivity to determine a variable ordering for our triple ADD.

5.2 Choice of variables ordering

5.2.1 Notion of 3D discrete topology

The topological study of the binary images requires the use of the discrete connectivity. The connectivity consists in defining relations of adjacency between the nodes of the grid. For a 3D image, there are three ways to define the notion of neighborhood between voxels. The 6-connectivity is defined when a face is shared by two voxels (6 because there are 6 faces per voxel). The 18-connectivity is defined when a face or an edge is shared by two voxels (18 because there are 6 faces and 12 edges per voxel). The 26-connectivity is defined when a face or an edge or a vertice is shared by two voxels (26 because there are 6 faces, 12 edges and 8 vertices per voxel). We can also express these 3D relations of adjacency for the cell complex. A 2-cell corresponds to a strict 6-adjacency between two voxels. A 1-cell corresponds to a strict 18-adjacency between two voxels. A 0-cell corresponds to a strict 26-adjacency between two voxels. By using the nota-

tion in figure 2, we propose a classification for the 26 neighbor voxels of the current voxel V according to their connectivity:

- 6-connectivity: N01, N02, N22, N42, N62, N82
- 18-connectivity: N21, N41, N61, N81, N12, N32, N52, N72, N23, N43, N63, N83
- 26-connectivity: N11, N31, N51, N71, N31, N33, N53, N73

5.2.2 Determination of a topological order

As our criterion consists in minimizing the number of accesses to the voxels, we must choose a variable ordering so that the greatest number of geometric elements is eliminated for each test. A neighborhood by one face eliminates immediately one face, four edges and four vertices. This case corresponds to a 6-connectivity (voxels: N01 or N22 or N82). The occurrence of such voxels in our three discrete equations is one time for Δn_2 (open faces), four times for Δn_1 (open edges), and four times for Δn_0 (open vertices). A neighborhood by one edge eliminates one edge and two vertices. This case corresponds to a 18-connectivity (voxels: N21 or N81 or N41 or N61 or N12 or N32). A neighborhood by one vertex eliminates only one vertex. This case corresponds to a 26-connectivity (voxels: N11 or N31 or N71 or N51).

We also choose the connectivity as a strategy to define an order between our 13 boolean variables. The first branching tests are carried out on the 6-connectivity voxels, then on the 18-connectivity voxels and at last on the 26-connectivity voxels. The voxels having the same connectivity are then ordered according to the exploration sense of the lattice. The order for the thirteen boolean variables of our triple-ADD is the following:
N01 $<$ N22 $<$ N82 $<$ N21 $<$ N81 $<$ N41 $<$ N61 $<$ N12 $<$ N32 $<$ N11 $<$ N31 $<$ N71 $<$ N51

In figure 3.a, we propose a classification of the open faces, open edges and open vertices of the current voxel. In figure 3.b, we represent the beginning of our ordered and reduced triple-ADD. For example, if the three voxels having 6-connectivity are black, the branching tests are stopped after analyzing the value of these voxels. The only three 1-values of these voxels eliminate any possibility of counting new geometric elements because all the terms of the three functions are cancelled : Δn_2 is still equal to 3 (for the 3 open faces: F2, F3 and F4 which might be shared by neighbor voxels following the current voxel), Δn_1 is still equal to 3 (open edges: e6, e7 and e12) and Δn_0 is still equal to 1 (open vertice v7). The complete final diagram is composed of 181 terminal nodes which are distributed in the following way:

Length of branching test	Number of terminal nodes	Length × Number
3	2	6
4	2	8
5	7	35
6	7	42
7	15	105
8	14	112
9	34	306
10	32	320
11	36	396
12	16	192
13	16	208
Total	181	1730

Table 1: Distribution of the length of the branching test in our reduced and ordered triple-ADD

Then we implement the triple-ADD in a C/C++ source code and we choose to keep the nested structure of the BDD in the generated code as shown in figure 3.c. The diagram is also transformed into a series of branching tests (`if...else...`). Even if the C/C++ source code is a little long, it is also very effective: at each stage, the function is guaranteed to examine only the pertinent input data, i.e. the values which affect the result. For each value, at least three tests and branchings are performed : the minimal length of the branching tests is three when the three voxels of 6-connectivity (N01, N22 and N82) are black because only these three neighbor voxels values are examined. The maximal length of the branching tests is 13. In [8], on average, at each voxel 8.7 neighbor voxels values are examined to set one boolean function. In our algorithm, only 9.5 $(1730/181)$ neighbor voxel values are examined to set up all the three boolean functions.

In fact, during the counting of the number of geometric elements in a 3D image, we can estimate that the length of branching tests most frequently performed for a black voxel is three. This case corresponds to a black voxel located inside the 3D object (N01, N81 and N22 are black voxels), i.e. all the three remaining faces of the current voxel are shared by the preceding neighbor voxels.

6 Experimental evaluation

In this section, we show the efficiency of using a reduced and ordered triple-ADD to enumerate the number of open faces, open edges, open vertices in a 3D binary image. To evaluate the algorithms, a 1 GHz Duron PC is used with a level-one 128 Ko cache memory and a 256 Mo RAM, and we compile source code with Visual C++ 6.0.

To count the number of geometric elements, we implement and compare three algorithms. The first

one is the "Michielsen and De Raedt algorithm" as it can be viewed in [5]. The second one implements the three equations: Δn_2, Δn_1, Δn_0 without optimization; it is also necessary to determine for each new black voxel all the thirteen neighbor voxel values. We call it "algorithm of equations". The last one is the algorithm presented previously and named "algorithm of the triple-ADD". It is the longest source code.

Many systems observed in nature may be modeled by point patterns [17]. For example, a system of particles may be viewed as a system of points generated by the centres of the particles. Points systems may be considered as black-and-white pictures. In order to study the characteristics (degree of randomness, clustering, periodic ordering,...) of the point system on a cubic lattice, we attach cubes to each point. As in the article by Michielsen and De Raedt, we choose to use the germs model to study the execution time of our algorithm. We consider a collection of N voxels V_i ($i = 1..N$) in a cubic domain. These voxels are called the germs of the model and their positions are generated from a uniformly uncorrelated random distribution (Poisson law). We consider the germs to be cubes of length $r = 1$ (voxel) and the grains as enlarged cubes of edge $2r + 1$, $r \geq 0$. The study of the coverage of the image by the grains gives information about the system under investigation. We will now study the counting of open cubes, open faces, open edges and open vertices for sets of points which are randomly positioned in a cube of length L_x. By making use of the graining procedure described above, we transform the point pattern into a pattern of cubic grains of edge length $a = 2r + 1$, $r \geq 0$ and study the processing time as function of r. The first tests simulate a germ model with $N = 1,024$ grains for a $128 \times 128 \times 128$ image ($L_x = 128$). The maximal number of black voxels in such a 3D image will be $126 \times 126 \times 126 = 2,000,376$ because we let a border of one white voxel all around the lattice. The execution time of these three algorithms during the graining procedure are represented in figure 4. On average, the algorithm of the triple-ADD is 17 times faster than the Michielsen and De Raedt algorithm and twice as fast as the algorithm of equations. Then, we have simulated a new germs model with $N = 4,000$ grains for a $500 \times 500 \times 500$ image ($L_x = 500$). The execution times for the algorithms of equations and triple-ADD are represented in figure 5. The two curves can be divided in three parts. In the first part, r varies from 0 to 5: very tiny cubes are isolated from each other. In the second part, the radius r increases up to 25 and grains join. Finally, the whole lattice is filled. We note that for large volume data, the algorithm of triple-ADD is on average 2.2 times faster than the algorithm of equations.

The binary decision diagrams, used up to now for classical image processing techniques with one boolean function, seem to be well adapted to our problem. Indeed, the algorithm of triple-ADD is 17 times faster than the Michielsen and De Raedt algorithm (which is also memory expensive) and 2.2 times faster than the algorithm of equations, as volume data increase.

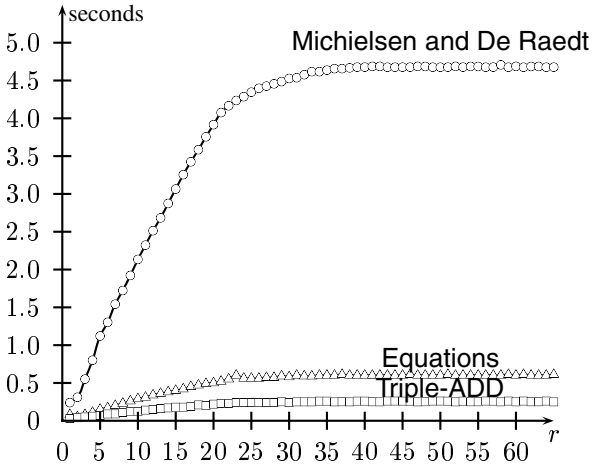

Figure 4: Execution time as a function of radius r during the graining procedure for $N = 1,024$ germs in an image of length $L_x = 128$.

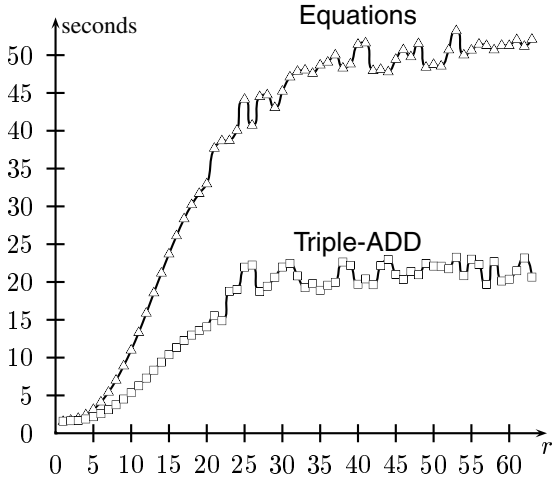

Figure 5: Execution time as a function of radius r during the graining procedure for $N = 4,000$ germs in an image of length $L_x = 500$.

In the following table, we represent the evolution of the number of black voxels during the beginning of the graining procedure for a $500 \times 500 \times 500$ image:

Nb of graining	Nb of blacks voxels	Nb of Voxels V_{t3}	Nb of Voxels V_{others}
0	4,000	0	4,000
1	107,514	47,712	59,802
2	496,075	317,052	179,023
3	1,356,936	1,008,254	348,682
4	2,817,760	2,244,454	573,306
5	5,052,381	4,199,777	852,604

We have only distinguished two types of voxels. The first ones are called V_{t3}. These voxels represent the voxels having performed exactly 3 tests, i.e. the minimum number of branching. For these V_{t3} voxels, the first two ordered neighbor voxels ($N01$ and $N22$) are black. The location of the V_{t3} voxels in the 3D image depends on the value of the following 6-connectivity voxel $N82$. If the neighbor voxel $N82$ is black, the current V_{t3} voxel is located inside the 3D object. If the neighbor voxel $N82$ is white, the current V_{t3} voxel is located on the left border of the 3D object: at least one face (F6) of the current V_{t3} voxel is not shared by preceding voxels. The other voxels are called V_{others}: they undergo up to 3 tests and they represent all the other black voxels of the 3D image. We can notice that the more the graining procedure is engaged, the higher the number of black voxels, but the shorter the branching because we have more and more V_{t3} voxels.

7 Conclusions and perspectives

In this paper, we propose a fast algorithm which relate topological conditions using binary decision diagrams. Counting the number of open faces, open edges, open vertices can be done time-efficiently thanks to the use of a reduced and ordered triple-ADD, even if the data volume is large. The number of open cubes corresponds to the number of black voxel in the 3D image. Once all these geometric elements have been enumerated, it is possible to calculate the Minkowki functionals.

In the future, we would like to use this method to represent 3D convex and non-convex bodies thanks to a mapping in $\mathbb{R}^2$ [18].

References

[1] P.J. FLYNN, A. HOOVER, and P.J. PHILLIPS. Special issue on empirical evaluation of computer vision algorithms. *Computer Vision and Image Understanding*, 84:1–4, 2001.

[2] J. SERRA. *Image analysis and mathematical morphology*. Academic Press Inc, London 1982.

[3] A.S. SZALAY, J. GRAY, and J. VANDENBERG. Petabyte scale data mining: dream or reality. *Proc. SPIE Conference on Advanced Telescope Technologies*, 4836, August 2002, Hawaii.

[4] D. JEULIN, P. MONNAIE, and F. PÉRONNET. Gypsum morphological analysis and modeling. *Cement & Concrete Composites*, 23:299–311, 2001.

[5] K. MICHIELSEN and H. DE RAEDT. Aspects of integral-geometry. *Advances in Imaging and Electron Physics*, 125, Academic Press, 2002.

[6] K. MICHIELSEN and H. DE RAEDT. Morphological image analysis. *Computer Physics Communications*, 132:94–103, 2000.

[7] R.W. HALL and C.-Y. HU. Time-efficient computation of 3d topological functions. *Pattern Recognition Letters*, 17:1017–1033, 1996.

[8] L. ROBERT and G. MALANDAIN. Fast binary image processing using binary decision diagrams. *Computer Vision and Image Understanding*, 72(1):1–9, October 1998.

[9] V.A. KOVALEVSKY. Finite topology as applied in image analysis. *Computer Vision, Graphics and Image processing*, 46:141–161, 1989.

[10] A. ROSENFELD, T.Y. KONG, and A.Y. WU. Digital surfaces. *CVGIP : Graphical Models and Image Processing*, 53(4):305–312, 1991.

[11] C.Y. LEE. Representing of switching functions by binary decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[12] R.E. BRYANT. Symbolic boolean operation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–317, September 1992.

[13] S. BISCHOFF and L. KOBBELT. Isosurface reconstruction with topology control. *Pacific Graphics 2002 Proceedings*, to appear October 2002.

[14] M. STARKEY and R. BRYANT. Using ordered binary-decision diagrams for compressing images and image sequences. Technical report, CMU-CS-95-105, January 1995.

[15] R. RUDELL. Dynamic variable ordering for ordered binary decision diagrams. *Proc. of the International Conference on Computer Aided Design*, pages 42–47, 1993.

[16] R.I. BAHAR, E.A. FROHM, C.M. GAONA, G.D. HACHTEL, E. MACII, A. PARDO, and F. SOMENZI. Algebraic Decision Diagrams and Their Applications. *Proc. ACM /IEEE International Conference on CAD*, pages 188–191, 1993.

[17] U. BRODATZKI and K. MECKE. Simulating stochastic geometries: morphology of overlapping grains. *Computer Physics Communications*, 147:218–221, 2002.

[18] J.-F. POIRAUDEAU and I. BLASQUEZ. Analysis of sets of convex bodies in 3d space with Minkowski functionals. *Proceedings of SPIE-Vision Geometry XI*, 4794, 7-8 July 2002, Seattle.