A fast voxelization algorithm for trilinearly interpolated isosurfaces

Rachid Namane, Serge Miguet, Fatima Boumghar Oulebsir

Noname manuscript No.
(will be inserted by the editor)

A Fast Voxelization Algorithm For Trilinearly Interpolated Isosurfaces

Rachid Namane · Serge Miguet · Fatima Boumghar Oulebsir

Received: date / Accepted: date

Abstract In this work we propose a new method for a fast incremental voxelization of isosurfaces obtained by the trilinear interpolation of 3D data. Our objective consists in the fast generation of subvoxelized isosurfaces extracted by a point-based technique similar to the Dividing Cubes algorithm. Our technique involves neither an exhaustive scan search process nor a graph-based search approach when generating isosurface points. Instead an optimized incremental approach is adopted here for a rapid isosurface extraction. With a sufficient sampling subdivision criteria around critical points, the extracted isosurface is both correct and topologically consistent with respect to the piecewise trilinear interpolant. Furthermore, the discretization scheme used in our method ensures obtaining thin - one voxel width - isosurfaces as compared to the given by the Dividing Cubes algorithm. The resultant subvoxelized isosurfaces are efficiently tested against all possible configurations of the trilinear interpolant and real-world datasets.

Keywords Voxelized Isosurface · Dividing Cubes · Trilinear Interpolant · Incremental Algorithm

R. NAMANE
LRPE Laboratory, ParIMed team, Computer Sciences and Electronics Faculty, USTHB university, Algiers, Algeria
E-mail: rdnamane@gmail.com

S. MIGUET
LIRIS Laboratory, Lyon 2 University, Lyon, France
E-mail: serge.miguet@univ-lyon2.fr

F. BOUMGHAR
LRPE Laboratory, ParIMed team, Computer Sciences and Electronics Faculty, USTHB university, Algiers, Algeria
E-mail: fboumghar@usthb.dz

1 Introduction

The accurate isosurface representation and its fast extraction have attracted the attention of researchers for several decades. The Marching Cubes (MC) algorithm [5] is arguably the most popular isosurface extraction technique available. In a nutshell, an isosurface is discretized using a triangular mesh by combining all triangular patches produced from the grid cells, with the intention to make it both correct and topologically consistent. An isosurface is correct if it accurately matches the behavior of a known function (or some assumed interpolant) that describes the phenomenon sampled in the data set [9]. If each component of an isosurface is continuous (i.e, there are no holes), then it is topologically consistent. It is possible for an isosurface to have a consistent topology but to not be correct [9].

When dealing with isosurface extraction from 3D data grid, applied algorithms only operate on the discrete data at the vertices of the grid because no surface function is known a priori. Several works were developed to improve the topological correctness and accuracy of the isosurface representation and many authors assume that the ideal isosurface is the one homeomorphic to the surface induced by the trilinear interpolant [1–4]. This motivates alternatives to the original MC algorithm that produce triangular meshes that are geometrically closer and having the same topology as the ideal surface.

An alternative to the isosurface polygonal representation is the use points and normals to represent small isosurface patches. The use of points in iso-surface visualization was first proposed in the Dividing Cubes (DC) algorithm [6]. Unlike in MC, where the surface within an intersected 3D cell is approximated by a polygonal mesh guided by a local configuration, the DC algorithm

approximates the surface by a set of sufficiently dense discrete points. The advantage of using points as compared to polygons is their structural simplicity which is easier to manage, parallelize and no pre-built lookup tables required for local topological guidance. The main drawbacks of point-based approaches, as in the DC algorithm, is the high complexity of the exhaustive scan performed in the extraction process with the large number of points generated.

The main contribution of our work is therefore to develop a fast and efficient incremental point-based method for the discrete approximation and generation of isosurfaces described by trilinear interpolation. In addition, we attempt to get a voxelized isosurfaces which is thin, correct and topologically consistent with respect to the trilinear interpolant without look-up tables. Our incremental generation approach is based on a Bresenham-like algorithm [10]. Using normal information, the surface is divided into different zones. For not missing any portion of the surface, the traversal direction in generating the points of each zone is chosen appropriately. The extraction process is repeated recursively until all surface components are correctly retrieved.

The structure of our paper is as follows: after a short presentation of related work in section 2, we introduce in section 3 the necessary terms traditionally used in digital topology. Then in section 4, both the exhaustive scan search and the graph-based search strategies are reviewed before giving a detailed explanation about our incremental approach. The theoretical analysis about the correctness and consistency of the obtained voxelized isosurfaces is discussed in section 5. A further optimization in the running time of the proposed incremental approach is presented in section 6. The efficiency of our approach will be illustrated in section 7 with the obtained experimental results in terms of both computational complexity and surface accuracy.

2 Related Work

Since the original Marching Cubes algorithm proposed in [5], several variants of that algorithm were developed to resolve topological and geometrical problems related to the triangle mesh representation. Most of these works tend to resolve all possible face and internal ambiguities of every grid cell, and at the same time trying to achieve the best possible polygonal approximation of the trilinear interpolant [1–3].

The polygonal mesh refinement relies mostly on supplementary points introduced on the faces and in the interior of the cube. Based on this principle, A. Lopes and K. Brodie proposed in [3] an extension version of the MC algorithm. A similar work has been proposed by M. G . Nielson in [1] but with less triangles since extra points are added just in the interior of the cuboid and not on the faces. Both works lead to topological correct surfaces and their supplementary points are determined based on the analytical behavioural analysis of the trilinear interpolant implicit function. In [4], H.Carr and M. G. Nielson have demonstrated and proved the correctness and completeness of the full possible configurations, classified in three hierarchical levels, already proposed in [1]. A more recent similar work of [8] presented by L.Custodio and all, in which the authors address some issues related to case ambiguity in the MC 33 algorithm [7] and proposed a disambiguation procedure to solve the raised issues and obtain topological correct surfaces.

We need to mention that in polygon-based approaches there is always a compromise between a smoother representation of the surface and a minimization of supplementary triangles. This means that a good approximation can be achieved with a finer mesh having a large number of triangles. However, rendering such meshes might lead to triangles whose projected area is less than one pixel, which results in wasting time in their scanline rendering.

Another trend in isosurface visualization investigates the use of points as rendering primitives instead of triangles when multiple facets are projected to a single pixel. Due to their structural simplicity, less computational effort has to be spent per primitive which reduces significantly computations. In addition, no connectivity or topological information is required which makes the geometric processing of highly complex 3D-models more flexible. The Dividing Cubes algorithm [6] is among the proposed point-based alternatives to solving problems related to MC-like methods. However, it has some drawbacks related mainly to its time complexity and the excessive number of points used in surface representation. Other uses of point-based primitives for isosurface representation and fast rendering appeared in [17–20]. In our paper, we are interested by this family of isosurface point-based representation approaches, in which we propose a much more efficient discretization scheme for fast generation of most classically used point-based primitives. Our fast incremental point generation technique is then based on a Bresenham-like algorithm [10] and its 3D extension [11], and leads to correct and topologically consistent isosurfaces while avoid using excessive number of points in their representation.

3 Notions and Problem Definition

3.1 Notations and Preliminaries

The set $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$) of points with integer (resp. real) coordinates is called the discrete (resp. continuous) space and its elements are called discrete (resp. Euclidean) points. Throughout we will assume that $n \leq 3$, and we denote by upper-case letters e.g, A,B,C... (resp. calligraphic upper-case letters e.g, $\mathcal{A}$,$\mathcal{B}$,$\mathcal{C}$...) the points of the discrete space $\mathbb{Z}^n$ (resp. the Euclidean space $\mathbb{R}^n$). By $[a..b]$, we denote the closed discrete interval in $\mathbb{Z}$ between a and b and by $[a, b]$ the continuous closed interval in $\mathbb{R}$ between a and b. For $n = 3$, a discrete (resp. continuous) domain D (resp. $\mathcal{D}$) is a subset of $\mathbb{Z}^3$ (resp. $\mathbb{R}^3$) given by $D = [1..X] \times [1..Y] \times [1..Z]$ (resp. $\mathcal{D} = [1, X] \times [1, Y] \times [1, Z]$). D can be seen as a $3D$ regular grid of size $X \times Y \times Z$. Each element in the grid is called a *vertex*. An *edge* is a line joining two neighboring vertices, i.e. vertices whose coordinates differ only by one unity, along one of the three coordinate axes. A *3D cell* is the axes-aligned unit cube in the grid bounded by eight vertices, and which can be defined by $C = [i, i + 1] \times [j, j + 1] \times [k, k + 1]$ with $i, j, k \in \mathbb{Z}^3$. When the *3D cell* is subdivided - along the three coordinate axes - by $a \times b \times c$ with $a, b, c \in \mathbb{Z}^3$, we then talk about a sub-grid of size $a \times b \times c$. A *3D sub-cell* is the resulting axes-aligned unit cube in the sub-grid bounded by eight vertices.

We assume that our input data consists of a $3D$ regular grid dataset. This grid can be represented with a domain D given by $D = [1..X] \times [1..Y] \times [1..Z]$. The *3D image* denoted by I can be represented as a function from D to $[0, l]$, where $[0, l]$ is the depth of the image, and which can be defined for every point P in D by :

$$I : D \to [0, l]$$
$$P \mapsto I(P)$$

3.2 Introduction to Object Digitization

To obtain more details about the following notions, the reader is referred to [21]. Below we consider a discrete (resp. Euclidean) point as an element of $\mathbb{Z}^3$ (resp. $\mathbb{R}^3$), and a discrete (resp. Euclidean) object as a set of discrete (resp. Euclidean) points. Let P be a point in a discrete domain D. The coordinates of a point P are denoted by the tuple (x, y, z). A *voxel* is a unit cube the center of which is P. This voxel is denoted by $\mathcal{V}(P)$.

The voxels $\mathcal{V}(P_1)$ and $\mathcal{V}(P_2)$, having their center points at $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ respectively, are said to be $26 - neighbors$ if $max(|x_2 - x_1|, |y_2 - y_1|, |z_2 - z_1|) = 1$. if $\mathcal{V}(P_1)$ and $\mathcal{V}(P_2)$ are $26-neighbors$, they are also said to be $18-neighbors$ if $|x_2-x_1|+|y_2-y_1|+|z_2-z_1| \leq 2$. Similarly, if $\mathcal{V}(P_1)$ and $\mathcal{V}(P_2)$ are $18-neighbors$, they are also said to be $6-neighbors$ if $|x_2-x_1|+|y_2-y_1|+|z_2-z_1| = 1$. Intuitively, these three neighbouring schemes can be described as a set of voxels which share one face for $6-neighbors$, share at least one edge for $18 - neighbors$, and share at least one vertex for $26 - neighbors$. A $k - path$, with $k \in \{6, 18, 26\}$, in a discrete object (A) is a sequence of discrete points from (A) such that every two consecutive points are $k - neighbors$. (A) is called $k - connected$ if there is a $k - path$ connecting any two points of (A). A $k - component$ is a maximal $k - connected$ subset of (A).

The problem of discretization is to identify discrete points (mesh approach) or what cell (paving approach) belong to the discrete equivalent of an Euclidean object. Several discretization schemes have been proposed. One of the discretization schemes used for digitizing curves -open or closed- is the *Grid Intersect Quantization* model, denoted *GIQ*. Whenever an edge -in the working lattice- intersects the curve, then the *GIQ* model chooses the closest lattice point among the two ones connecting the intersected edge. In fact, this process is similar to the Bresenham algorithm used for line drawing. For closed curves, the scheme *Object Boundary Quantization*, denoted *OBQ* (respectively *Background Boundary Quantization*, denoted *BBQ*), in which all what is considered as a part of the discrete object are extreme points of each intersected edge and which are located in the interior (respectively exterior or background) of that object. Figure 1.(a), (b), and (c) illustrate these three quantization schemes for 2D curves. See [22] for a survey on digitization schemes.

In paving approaches, the discretization scheme - in which the resulting discrete object consists of every cell intersected by the continuous object - is called *supercover*. This scheme has the advantage of being simple. However its drawback is the possible presence of what are called "bubbles" when a continuous curve passes exactly through a point of the lattice. This is because all the cells sharing this point will be selected (see Figure 1.(d) for an example of a 2D line). Although the probability that a continuous curve passes through a point of the lattice is very low, some cares can be taken to resolve this problem. The resulting discrete scheme is called *standard* [23].

3.3 Basics of The Trilinear Interpolant and isosurfacing

The isosurfacing problem can be then rewritten as follows:
Given the scalar field values $T(x, y, z)$ of the trilinear

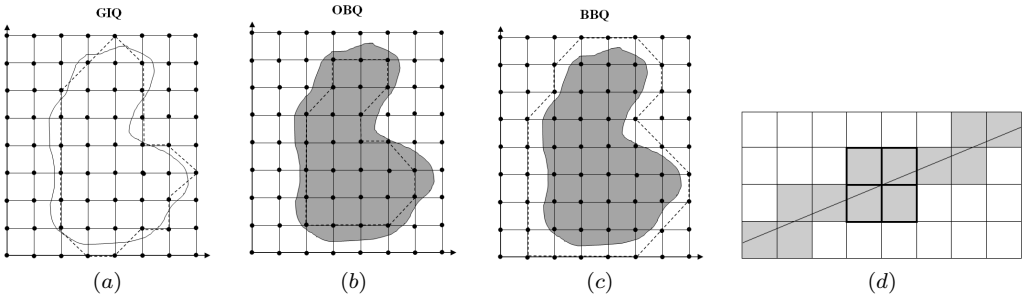


Fig. 1 (*a*), (*b*), and (*c*) are the GIQ, OBQ, and BBQ methods for 2D curve representation respectively. (*d*) A straight line segment and its super-cover (dashed), which contains a "bubble".

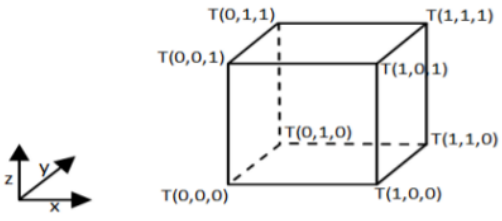


Fig. 2 Grid cell notation.

function $\mathcal{T}(x, y, z)$, at the vertices of axis-aligned cube $\mathcal{C} = [i, i+1] \times [j, j+1] \times [k, k+1]$ with $(i, j, k) \in \mathbb{Z}^3$, determine the set of points $\mathcal{P}$ of the isosurface corresponding to isovalue σ where

$$\mathcal{P}(x, y, z) = \{(x, y, z) \in \mathcal{C} | \mathcal{T}(x, y, z) = \sigma\}$$

Without loss of generality, we transform the cubic cell $\mathcal{C}$ into the unit cube $\mathcal{U} = [0, 1] \times [0, 1] \times [0, 1]$ for convenience, so that trilinear interpolant will be expressed as follows:
Given the grid cell of Figure 2 which is composed of eight grid vertices with the scalar values $T(0, 0, 0)$, $T(1, 0, 0)$, ..., $T(1, 1, 1)$. Based on the scalar values at these vertices, the trilinear scalar function $\mathcal{T}$ on $\mathbb{R}^3$ is given by:

$$\mathcal{T}(x, y, z) = a + bx + cy + dz + exy + fxz + gyz + hxyz \quad (1)$$

Where

$$\begin{aligned}
a &= T(0,0,0) \\
b &= T(1,0,0) - T(0,0,0) \\
c &= T(0,1,0) - T(0,0,0) \\
d &= T(0,0,1) - T(0,0,0) \\
e &= T(0,0,0) - T(1,0,0) - T(0,1,0) + T(1,1,0) \\
f &= T(0,0,0) - T(1,0,0) - T(0,0,1) + T(1,0,1) \\
g &= T(0,0,0) - T(0,1,0) - T(0,0,1) + T(0,1,1) \\
h &= T(1,0,0) - T(0,0,0) + T(0,1,0) - T(1,1,0) \\
&\quad + T(0,0,1) - T(1,0,1) - T(0,1,1) + T(1,1,1)
\end{aligned}$$

The approximated continuous surface $\mathcal{S}_{(\mathcal{T},\sigma)}$ which is consistent to $\mathcal{T}$ for a given isovalue σ within a unit cube in $\mathbb{R}^3$ can be defined by the zero set of a geometrically represented implicit function $\mathcal{F}$ given by:

$$\mathcal{F}(x, y, z) = \mathcal{T}(x, y, z) - \sigma = 0 \quad (2)$$

Recall that by definition a cubic cell is being *active* if it is intersecting the surface. In a discrete domain $\mathcal{D}$, this may occur if σ falls between the minimum and the maximum of all scalar values at the eight vertices of this cubic cell.

3.4 Problem Statement

Let I be our 3D input image defined on a discrete grid domain D, and $\mathcal{F}$ the implicit trilinear interpolation function defined on a continuous domain $\mathcal{D}$. $\mathcal{F}$ coincides with I on the points of D. For all other points $\mathcal{P}$ of $\mathcal{D}$, $\mathcal{F}(\mathcal{P})$ is given by trilinear interpolating values of I at the eight vertices of the 3D cell containing $\mathcal{P}$. This allows to define $\mathcal{F}$ as a continuous surface, denoted $\mathcal{S}$, on all points of $\mathcal{D}$. The main problem of our work consists in development of a rapid voxelization algorithm for extracting an isosurface $\mathcal{S}$, defined by an isovalue σ, from the $3D$ input dataset. The voxelized isosurface generated by our algorithm, denoted S, consists of a set of discrete points rather than polygons. However, the point-based representation used in our approach should guarantee producing correct and topologically consistent voxelized isosurfaces with respect to $\mathcal{S}$.

4 Searching Strategies in Point-Based Isosurface Extraction

Several approaches were developed for accelerating and optimizing isosurface extraction, when dealing with polygonal-based techniques. To obtain more details about these approaches, the reader is referred to [16]. We need to

mention here that accelerating the search of active cells in the input 3D image is beyond the scope of our work. In fact, we are looking to optimize and accelerate the searching process of isosurface points in the interior of active cells. Before proceeding to the presentation of our incremental approach, we are going first to present briefly the two main different possible strategies that can be used for isosurface points extraction; the exhaustive and the graph-based scan strategies.

4.1 Exhaustive Scan Search

This is the simple and basic strategy applied in the Dividing Cubes. The dividing cubes subdivides each cell intersected by the isosurface into $o \times p \times q$ sub-cells along the X, Y, and Z directions respectively, such that the sub-cells project onto a single pixel on the image plane. The scalar field values at the vertices of every sub-cell are obtained off-line by a trilinear interpolation of the values located at the eight vertices of the cell being subdivided. The same process is applied for interpolating the gradient values, that can be used as approximations of the surface normals. Thereafter, the searching of the active sub-cells is performed by applying an exhaustive search; i.e scan the $o \times p \times q$ new 3D volume along the three axis. The computational time is however costly as the sampling resolution gets finer. The algorithm's complexity it exhibits is of $O(n^3)$, when o, p, and q are equal to n. A statistical study on the number of points extracted as a function of the subdivision parameters is given in [12]. An optimization of the execution time of the "dividing-cubes" algorithm was proposed in [13] by subdividing the input 3D image into blocks of voxels in order to reduce the time complexity by eliminating the blocks that do not contain active sub-cells.

4.2 Graph-based Search

The previous exhaustive strategy can be improved by applying a graph-based searching technique. In this way, the exhaustive scan process is avoided by propagating in the neighbourhood of every active sub-cell. However, in this approach, a starting point must be found for every connected component of the isosurface. Since every connected component must at least intersect one edge of the cell, an initial preprocessing test is made through the edges to identify those points. The isosurface is then tracked through the resulting 3D lattice in the interior of every active cell. The tracking process is performed by an iterative examination of the $6-connected$ neighborhood of every active sub-cell, noting the active ones which have already been traversed. Flag labels are associated with every sub-cell, which is useful to check if it is already traversed. This process continues until all the active sub-cells have been traversed. We note here that the average complexity it exhibits is of $O(n^2)$, except for the $O(n^3)$ flags initialization. We will show in *section*4.3.6 an optimization that allows to reach an $O(n^2)$ complexity even for this step.

4.3 Incremental Search

To optimize more the extraction process, we developed an incremental search method. It is based on the incremental point generation of the discrete implicit surface in the trilinear interpolant. This approach is based on a Bresenham-like algorithm adapted to 3D hyperbolic surface generation. The main advantages of this method are fast surface generation, thinner surface discretization very closer to GIQ, and much less operation spent per primitive. In the following, we give a detailed description of this approach. Before proceeding we need to define our discrete surface and how it is structured into different zones. First, the generated discrete surface S can be considered as the union of the three subsets S_x, S_y, and S_z ($S = S_x \cup S_y \cup S_z$). Each of these subsets might be itself composed of one or several surface components belonging to the same zone type. The number of connected components in each zone depends on the shape of the trilinear function in the active $3D$ cell. We define S_x as the zone type which consists of the set of points $P(x, y, z)$ such that $argmax_{(x,y,z)}(|g_x|, |g_y|, |g_z|) = |g_x|$. We define S_y and S_z in a similar way, i.e $argmax_{(x,y,z)}(|g_x|, |g_y|, |g_z|) = |g_y|$ for S_y and $argmax_{(x,y,z)}(|g_x|, |g_y|, |g_z|) = |g_z|$ for S_z. Based on the previous definition and for efficient surface discretization, the following three possible cases considered during the incremental traversal:

- Determine the closest x for every (y, z) in zone S_x.
- Determine the closest y for every (x, z) in zone S_y.
- Determine the closest z for every (x, y) in zone S_z.

4.3.1 Discretization Algorithm

First we present the main steps of our voxel traversing algorithm, then more details will be given in the next subsections. As we will show later, the algorithm generates discrete points of the trilinear interpolant by selecting one of the two possible points after a one-step increment along the axis to which the part of surface is most parallel. This axis is changed adaptively as the orientation of the surface is changed based on the normal

associated with each point. In the following, every discrete point generated corresponds to the discrete point more closest to the exact surface.

The main steps of the discretization algorithm can be summarized as follows: First, with linear interpolation along edges we locate a starting point on one of the edges of an active cell or on the neighbourhood of a previously extracted surface part. Then, we select one of the three axis which is the most parallel to the surface's normal; the other two axes belong to the plane which is the most parallel to the surface. Once these axes are specified, we incrementally vary the two variables corresponding to the detected plane and calculate the remaining one (Bresenham-like method), followed by updating incrementally the surface's normal. After each increment step, we test whether we are still in the same zone of the surface; i.e check whether the normal is still most parallel to the same axis. The increment process is repeated until all points of that zone are extracted. Finally, we switch to another zone and repeat the whole extraction process until all parts of the surface are fully extracted.

4.3.2 Algorithm Illustration

For more clear presentation of our method, first we show its behaviour in generating points for the three different zone types separately, and then we give its overall process for generating all surface points. Assuming that the surface's implicit function is $\mathcal{F}(x,y,z)=0$, then we refer by Δ_x, Δ_y, and Δ_z to the step amounts by which the three variables x, y, and z are incremented/decremented respectively.

The algorithm used for the generation of each surface zone is similar, however the role played by control variables in the main loops are interchanged. For example, the generation of one connected component of S_x requires two nested loops on y and z coordinates, and generating at each step the corresponding x coordinate.

In the following, we will give our algorithm's illustration for the generation of S_x, and it will be the same thing for S_y and S_z by just changing their axis variables accordingly. In addition, we refer by E_i^x -in S_x- as an indicator of how much a point $P_i(x_i,y_i,z_i)$ -generated at iteration i- is far away from the targeted implicit surface $\mathcal{F}(x,y,z)=0$. E_i^y and E_i^z are defined in a similar way in S_y and S_z respectively. In the following we show that this indicator value, which is related to scalar values, is proportional to the amount of error related to distances.

Recall that the scalar value of the trilinear interpolant T is equal to σ on the isosurface, less than σ below the isosurface, and greater than σ above the isosurface.

Now, consider $\mathcal{F}(x,y,z)=\mathcal{T}(x,y,z)-\sigma=0$ to be the implicit function of the targeted isosurface. Based on equation (1), and if we let $\mathcal{T}(x',y,z)=\sigma'$, then

$$\mathcal{T}(x',y,z)-\mathcal{T}(x,y,z)=\mathcal{T}_x(x'-x,y,z)=\sigma-\sigma'=\Delta\sigma=E^x$$

Where

$$\mathcal{T}_x(x,y,z)=bx+exy+fxz+hxyz$$

Finally

$$x'-x=\frac{E^x}{\mathcal{T}_x(x'-x,y,z)}$$

Similarly, when either variable y or z is changed we obtain $y'-y=\frac{E^y}{\mathcal{T}_y(x,y'-y,z)}$ and $z'-z=\frac{E^z}{\mathcal{T}_z(x,y,z'-z)}$ respectively. Since the change in the trilinear interpolant's scalar value is proportional to the change in point's position, therefore the absolute value of that error will indicate the proportion of how much a point is far way from the exact surface. We will be based mainly on this error to choose the point more closest to the targeted isosurface during the incremental traversal.

Now let's illustrate our algorithm for generating points in S_x. Remember that in S_x, variables z and y are being changed in two nested loops and x is computed incrementally for each step. The algorithm stops in either direction if the tracking process either switches to another zone or reaches the boundaries of bounding cubic volume. Consider the starting point P_0, along one of the edges of an active cell, of coordinates (x_0,y_0,z_0). Based on equation (2), the initial amount of indicating error made at that point from being on the exact surface is given by

$$E_0^x=a+bx_0+cy_0+dz_0+ex_0y_0+fx_0z_0+gy_0z_0+hx_0y_0z_0-\sigma \tag{3}$$

And using partial derivative formula, the normal components at that point are

$$g_{x0}=b+ey_0+fz_0+hy_0z_0 \tag{4}$$

$$g_{y0}=c+ex_0+gz_0+hx_0z_0 \tag{5}$$

$$g_{z0}=d+fx_0+gy_0+hx_0y_0 \tag{6}$$

From P_0, a one step increment along O_yz is performed iteratively in two nested loops. After every increment step, the point more closest to the exact surface along the O_x is chosen among the two possible ones (x_i, or $x_{(i+\Delta_x)}$), where x_i is the $x-component$ of the point extracted in the previous iteration and Δ_x is the step increment which is equal to ±1.

Now assume that incrementing along O_z corresponds to the inner most loop of the algorithm. Therefore, z variable is incremented with Δ_z; $z_{(i+\Delta_z)}$ equals to

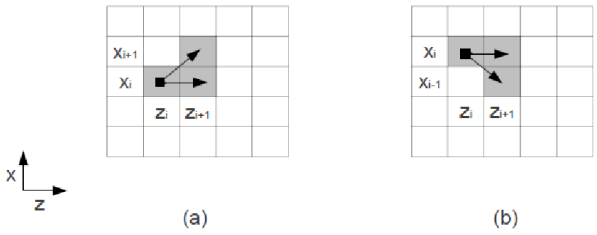


Fig. 3 The two possible cases that may happen along O_x after one step increment of z. (a) case when $\frac{\partial x}{\partial z} > 0$. ($b$) case when $\frac{\partial x}{\partial z} < 0$. In both cases $\frac{\partial x}{\partial z} = -\frac{g_z}{g_x}$.

$z_i + \Delta_z$, where Δ_z equals ± 1 depending on the direction of the traversal along (O_z); upward or downward. The traversal direction can be determined based on the sign of the normal components at the point. If the point extracted in the previous iteration was $P(x_i, y_i, z_i)$, then in the current iteration we look for the nearest discrete point to the exact surface among the two possible ones which are $P(x_i, y_i, z_{(i+\Delta_z)})$ or $P(x_{(i+\Delta_x)}, y_i, z_{(i+\Delta_z)})$. Δ_x equals ± 1, and it can be determined by just testing the sign of the partial derivative of x with respect to z based on the implicit function theorem. Figure 3 shows the two possible cases that may happen for the z incrementation.

The recurrence formulas for the errors of the two possible points at the current step are

$$E^x_{(i+\Delta_z)} = E^x_i + \delta E_z \tag{7}$$

$$E^x_{(i+\Delta_z)+\Delta_x} = E^x_{(i+\Delta_z)} + \delta E_x \tag{8}$$

where

$$\delta E_z = K + L \times \Delta_z \tag{9}$$

$$\delta E_x = M + N \times \Delta_x \tag{10}$$

Where K, L, M, and N are coefficients that depend only on y. Since y is unchanged, these coefficients are calculated one time as function of y_0 just before running the loop process. Since $E^x_{(i+\Delta_z)} = \mathcal{F}(x_i, y_0, z_{i+\Delta_z})$ and $E^x_{(i+\Delta_z)} = \mathcal{F}(x_{i+\Delta_x}, y_0, z_{i+\Delta_z})$, the previous coefficients can be calculated as follows:

$$K = (d \times \Delta_z) + (g \times y_0 \times \Delta_z) \tag{11}$$

$$L = (f \times \Delta_z) + (h \times y_0 \times \Delta_z) \tag{12}$$

$$M = (b \times \Delta_x) + (e \times y_0 \times \Delta_x) \tag{13}$$

$$N = (f \times \Delta_x) + (h \times y_0 \times \Delta_x) \tag{14}$$

Either $P(x_i, y_0, z_{i+\Delta_z})$ or $P(x_{i+\Delta_x}, y_0, z_{i+\Delta_z})$ is chosen according to $argmin(|E^x_{i+\Delta_z}|, |E^x_{(i+\Delta_z)+\Delta_x}|)$. Once this point is deduced, the normal components are updated accordingly to check whether we are still in the same zone. This is performed using the following equations

$$g_{x(i+1)} = g_{x(i)} + \alpha \times \Delta_z \tag{15}$$

$$g_{z(i+1)} = \begin{cases} g_{z(i)} & \text{if } x_{i+1} = x_i \\ g_{z(i)} + \alpha \times \Delta_x & \text{if } x_{i+1} = x_i + \Delta_x \end{cases} \tag{16}$$

$$g_{y(i+1)} = \beta + \gamma \times \Delta_z \tag{17}$$

where

$$\alpha = f + h \times y_0 \tag{18}$$

$$\beta = \begin{cases} unchanged & \text{if } x_{i+1} = x_i \\ \beta + e \times \Delta_x & \text{if } x_{i+1} = x_i + \Delta_x \end{cases} \tag{19}$$

$$\gamma = \begin{cases} unchanged & \text{if } x_{i+1} = x_i \\ \gamma + h \times \Delta_x & \text{if } x_{i+1} = x_i + \Delta_x \end{cases} \tag{20}$$

Where α is a constant calculated just before starting the iterative process. Once the normal components are updated and their new values still satisfy the current zone requirements, we pass to the next iteration. Otherwise, $P(x_{i+1}, y_{i+1}, z_{i+1})$ generated at the current iteration $(i+1)$ is added to S_x, and the same process is repeated for one step increment along (O_y) in the outer loop. The generation of discrete points in S_x is terminated when that zone boundaries are reached along both (O_y) and (O_z).

Algorithm 1 summarizes the iterative process for generating points of S_x along O_z.

Algorithm 1 points of S_x generation along O_z

Input: The current active $3D$ cell $\mathcal{C}$
Output: The set of points $P(x, y, z)$ added to the isosurface S
Let $P_0(x_0, y_0, z_0)$ be a starting point in S_x along one of the edges in $\mathcal{C}$
Let $i = i_0$
while $P_i \in (S_x \cap C)$ **do**
  $z_{i+1} = z_i + \Delta_z$
  $E^x_{(i+\Delta_z)} = E^x_i + \delta E_z$
  $E^x_{(i+\Delta_z)+\Delta_x} = E^x_{(i+\Delta_z)} + \delta E_x$
  if $(|E^x_{(i+\Delta_z)+\Delta_x}| < |E^x_{(i+\Delta_z)}|)$ **then**
    $x_{i+1} = x_i + \Delta_x$
  end if
  Incrementally update normal components
  if $(|g_x| > |g_y| > |g_z|)$ **then**
    if $P_{i+1}(x_{i+1}, y_{i+1}, z_{i+1}) \notin S$ **then**
      $S = S \cup P_{i+1}(x_{i+1}, y_{i+1}, z_{i+1})$
    end if
  end if
  $i = i + 1$
end while

4.3.3 Overall Algorithm Behaviour

From the starting points on the edges of the active 3D cell, all the corresponding portions of the discrete surface are extracted. This is considered as the first stage of the extraction algorithm in which its iterative process was explained previously. The second stage of the algorithm is performed in the same manner as the first one, except that the starting points are now the end boundaries of the different zones extracted in the previous stage which were already stored in memory. This is to guarantee that no hole appears in the discrete surface. The process is repeated until all points are extracted; i.e all zones meet to form the full different discrete connected components of the desired discrete isosurface in the interior of the cell.

Figure 4 shows the resulting subvoxelized isosurfaces for two different configuration examples of the trilinear interpolant. For both examples, The scalar values at the eight vertices of the 3D cell are given and the isovalue is chosen to be $\sigma = 90$. Points in zones S_x, S_y, and S_z are shown in red, green, and blue colors respectively. Starting points are shown in yellow color. In the first example, and starting from the cell's edge, all surface components are generated during one stage. In the second example, surface components are generated by applying the incremental process for two stages. Notice that not all starting point candidates for this second stage will be used. In fact, if any one of these points is generated from another one, then this later is omitted for being a starting point candidate.

4.3.4 Speed Up Hashing Functions

In equation (1), if any two variables are fixed and the third one is computed then this latter, if it exists, is always unique. In other words, if all points of the surface are projected on any of the three planes (O_{xy}, O_xz, or O_yz) then no more than one point can be projected on the same location. This is true if we consider the euclidean space. In discrete space, this may not be the case especially for points of the surface that are close to an asymptotic plane.

To avoid processing every point more than one time, and therefore speed up the extraction process, three simple hashing functions were used which are $h_x(i,j,k) = (j,k)$, $h_y(i,j,k) = (i,k)$, and $h_z(i,j,k) = (i,j)$. Their corresponding Hash tables contain a linked list of the 3^{rd} component for points having the same projection on their corresponding plane. Each point is hashed to only one table according to its greatest normal component. When two or all normal components are equal, an arbitrary choice of any table is taken in order to avoid any collision that may happen. Experimental studies showed us that the length of the linked list in every hash table is almost one element, except for special configurations where this length is two, but not more, for just few surface points.

The advantage of these hashing functions is to use just $(3 \times n^2)$ Booleans instead of (n^3) that help to check points traceability during the incremental search. This reduces the complexity of their initialization process from $O(n^3)$ to $O(n^2)$.

5 Voxelized Isosurface Correctness and Consistency Theoretical Analysis

5.1 Correctness and Consistency

As explained before, topological correctness implies that the discrete approximation should be homeomorphic to the implicit surface of the trilinear interpolant (i.e, it accurately matches its behavior). Consistency if satisfied if no holes are generated in the voxelized approximation.

To verify the accuracy of the extracted voxelized isosurface, we are going to prove that all points generated incrementally belong to the super-cover of the continuous surface $\mathcal{S}$. In other words, we want to prove that the points generated incrementally are a subset of the points generated by the DC algorithm. Based on a recursive approach, we are going to show that if a subvoxel extracted at iteration "i" is being intersected with the trilinear interpolant implicit function, then the subvoxel extracted at the next iteration "$i+1$" would be intersected too. Furthermore and without loss of generality, we limit our proof for one slice across a certain surface zone which can be extended to the whole surface by just using appropriate variables across every volume slice within any surface zone.

Before proceeding further, let's first consider that we are in the deepest loop of S_x extraction process. Meaning that y variable is kept unchanged, z variable is incremented and the closest x variable, denoted X, is determined. Given $\mathcal{F}(x,y,z) = 0$ as an implicit surface function, this function can be rewritten in the form $x = \mathcal{F}_y(z)$. Then we refer by $\mathcal{E}_i = X_i - \mathcal{F}_y(z_i)$ which represents the horizontal error along O_x between a generated point $P(X_i, y, z_i)$ and the point $P(x_i, y, z_i)$ on the exact surface. In addition, the sub-voxel containing P is intersected with the exact surface if and only if $|\mathcal{E}_i| \leq \frac{1}{2}$. In the following, $\mathcal{F}'_y(z)$ refers to the derivative of $\mathcal{F}_y(z)$ with respect to z. The curve is most parallel to the O_z axis if $|\mathcal{F}'_y(z)| \leq 1$

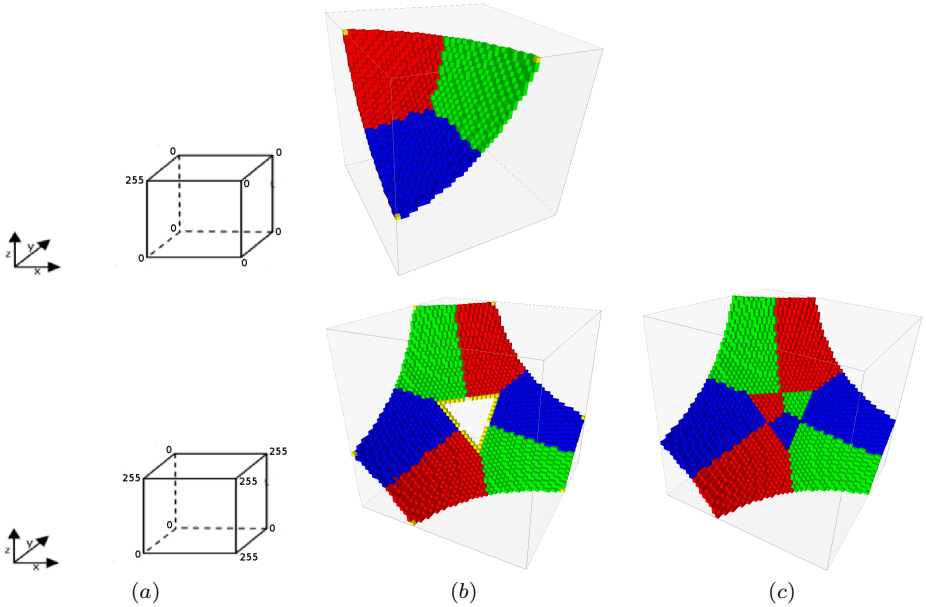


Fig. 4 (*a*) One grid cell synthetic dataset. (*b*) First stage of extraction. (*c*) Second stage of extraction.

Lemma 1 *If* $|\mathcal{E}_i| \leq \frac{1}{2}$, *and* $|\mathcal{F}'_y(z_i + t)| \leq 1$ *for all* $t \in [0, 1]$, *then* $|\mathcal{E}_{i+1}| \leq \frac{1}{2}$.

Proof Without loss of generality, assume that $P(x_i, y, z_i)$ is the point generated at iteration i and $\Delta_z = 1$. Δ_z is the step amount by which z_i is incremented, meaning $z_{i+1} = z_i + \Delta_z$.
Since $\mathcal{F}_y(z)$ is continuous and differentiable over $[z_i, z_{i+1}]$, then applying intermediate values theorem we get

$$\mathcal{F}_y(z_{i+1}) - \mathcal{F}_y(z_i) = \left[\mathcal{F}_y(z_i + t)\right]_0^1 \tag{21}$$
$$= \int_0^1 \mathcal{F}'_y(z_i + t)\mathrm{d}t \tag{22}$$
$$\leq \int_0^1 |\mathcal{F}'_y(z_i + t)|\mathrm{d}t \tag{23}$$
$$\leq \int_0^1 1\mathrm{d}t \tag{24}$$
$$\leq 1 \tag{25}$$
$$\mathcal{F}_y(z_{i+1}) \leq \mathcal{F}_y(z_i) + 1 \tag{26}$$

Now let $P(X_i, y, z_i)$ and $P(X_{i+1}, y, z_{i+1})$ be the discrete points generated at iterations i and $i+1$ respectively. In other side, let $P(x_i, y, z_i)$ and $P(x_{i+1}, y, z_{i+1})$ be their corresponding points laying on the surface S.
According to the horizontal error defined above, we have the following two statements:

$$\mathcal{E}_i = X_i - x_i = X_i - \mathcal{F}_y(z_i)$$

$$\mathcal{E}_{i+1} = X_{i+1} - x_{i+1} = X_{i+1} - \mathcal{F}_y(z_{i+1})$$

Based on these two statements and using equation (26) we can write:

$$\mathcal{E}_{i+1} \leq X_{i+1} - (\mathcal{F}_y(z_i) + 1) \tag{27}$$
$$\mathcal{E}_{i+1} \leq X_i + \Delta_x - (\mathcal{F}_y(z_i) + 1) \tag{28}$$
$$\mathcal{E}_{i+1} \leq (X_i - \mathcal{F}_y(z_i) + (\Delta_x - 1)) \tag{29}$$
$$|\mathcal{E}_{i+1}| \leq |X_i - \mathcal{F}_y(z_i)| + |\Delta_x - 1| \tag{30}$$
$$|\mathcal{E}_{i+1}| \leq |\mathcal{E}_i| + |\Delta_x - 1| \tag{31}$$
$$|\mathcal{E}_{i+1}| \leq \frac{1}{2} + |\Delta_x - 1|; \quad as \;\; |\mathcal{E}_i| \leq \frac{1}{2} \tag{32}$$

Since $|\Delta_x - 1| \leq 1$ then, equation (32) implies that $|\mathcal{E}_{i+1}| \leq \frac{3}{2}$.
$|\mathcal{E}_{i+1}| \leq \frac{3}{2}$ means that we have two possible cases for x_{i+1} which is either x_i when kept unchanged or $x_i + 1$ when incremented by one. Since the smallest one which is chosen every iteration, this yields that $|\mathcal{E}_{i+1}| \leq \frac{1}{2}$ and this completes our proof.

Clearly, this proof shows that the set of incrementally extracted points are a subset of the *supercover* of the surface $\mathcal{S}$. Furthermore, the way in which surface points are tracked along the three coordinate axes makes every resulting discrete surface component free from holes.

5.2 Sampling Issues For Topology Preserving

In the full trilinear function $\mathcal{T}$ over $\mathbb{R}^3$, saddle points are approached by contour surfaces from all directions

[4]. If the analysis range is limited to a bounding cuboid in $\mathbb{R}^3$, there exist two kinds of saddles, face and body. Face saddle occurs on a face having two diagonally opposed vertices with different signs. Body saddles in the interior of the cuboid are points where the gradient of $\mathcal{T}$ vanishes. There are exactly 0, 1, or 2 body saddle points with distinct isovalues [1]. One of the properties of both face and body saddles is that the axis-parallel lines through them have constant isovalue [4], and isosurfaces only change overall topology at these saddle points. There are five possible cases for the number of face and body saddles that may exist, and the most complicated case occurs when there are six face saddles [14].

The voxelized isosurface is correct and topologically consistent if it accurately matches the behaviour of the trilinear interpolant function without missing any point. This means that for topology preserving purposes, our discretization scheme should ensure that both surfaces must be having always the same number of continuous connected components and the same number of tunnels. Which implies that the accuracy of our approximate representation depends on the rate of sampling (i.e, the resolution of subdivision). A sufficient discretization sampling resolution should be specified in order to guarantee that all separated components should not meet, and tunnels should not be filled. With respect to this, and discarding the degenerate cases, a sufficient condition for topology preservation is to make sure that voxels containing saddle points and some of their neighbours are not active. For face saddles, voxels at saddle points with two of their $6-$*connected* neighbors but along different axes, should not be active. This is to make sure separating different components on the face. The same thing for body saddle where voxels at the body saddles with their $6-$*connected* neighbours should not be active in order to keep all components that may exist separated and any tunnel not to be filled. A simple checker subroutine can be added to verify the previous conditions and to tell whether the chosen subdivision factors fulfil those conditions or we need to sample further.

6 Incremental Extraction Time Optimization

For further extraction time reduction in the incremental generation method, our algorithm was further optimized by generating first all zones boundaries then filling rapidly the interior of every zone. The prior knowledge of all zones limits makes the generation of points at the interior of the zones more straightforward. This is because the normal change checking is no more necessary which reduces much more the extraction time.

6.1 Zone Boundaries Extraction

The boundaries of all zones are first extracted on the faces of the cube then in its interior. On the faces, limit points are extracted incrementally starting from a point of intersection between the isosurface and a given edge of the cube. Starting from the end points of the discrete curves extracted previously on the faces, limit points separating surface zones in the interior of the cube are extracted in a zig-zag sequence by searching in the 26-neighbour of already extracted point belonging to one zone in order to find the next limit point belonging to the adjacent zone. Already traversed branches are skipped.

6.2 Zone Interior Filling

Once all zone boundaries are extracted, the remaining parts of the isosurface, i.e zone interior filling, is processed based on a scan-line polygon fill like algorithm. The scan extrema points are the limit points found in the previous stage. The type, either lower or upper, of every extrema point is also deduced during their extraction by checking the sign of the normal components at that point with the type of its adjacent zone. All surface portions of the same zone type are filled by first sorting extrema points, laying on the same row, in the plane to which the surface is most aligned. Then incrementally generating the corresponding hyperbolic arc that falls between its corresponding lower and upper endpoints. The sorting process is not a bottleneck while trying to optimize our algorithm. This due to the limited number of extrema points on each plane's row. In most of the time there are either two or four extrema points. The main advantage of our optimization is to avoid calculating the normal components as described in the proposed incremental search method discussed previously.

6.3 Discussion

As seen from equations (7) to (10), our incremental algorithm uses only six arithmetic operations for the computation of each point. This has to be compared to the $twenty-three$ arithmetic operations required by a direct implementation of the implicit surface equation (2). The computation of the surface's normal in our method requires *one* test and $four$ arithmetic operations when the coordinate of the generated point, along the axis most perpendicular to the surface, doesn't change. However it requires *one* test and $eight$ arithmetic operations when this coordinate is incremented/decremented

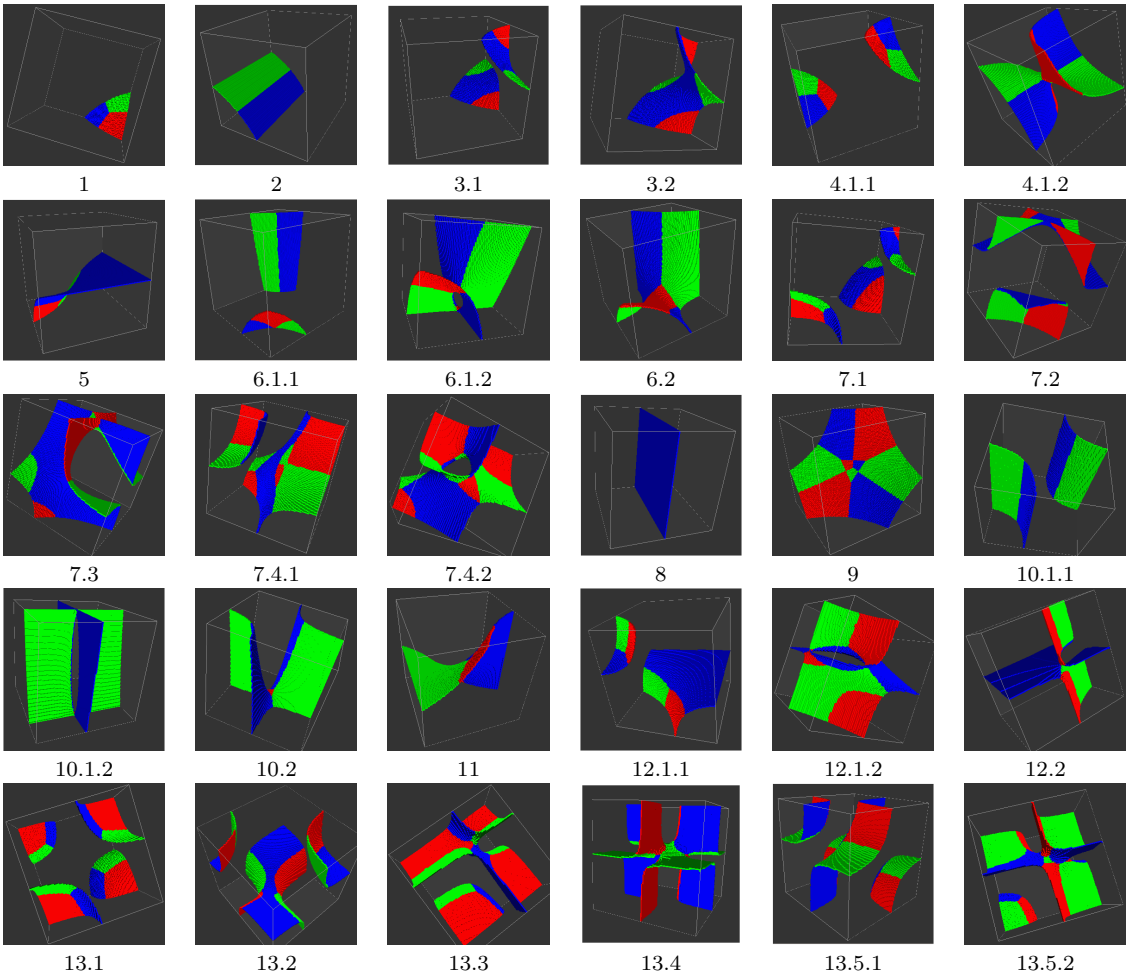


Fig. 5 The discrete approximation of all possible topological configurations of the trilinear interpolant using the same notation as in [3] for their labelling.

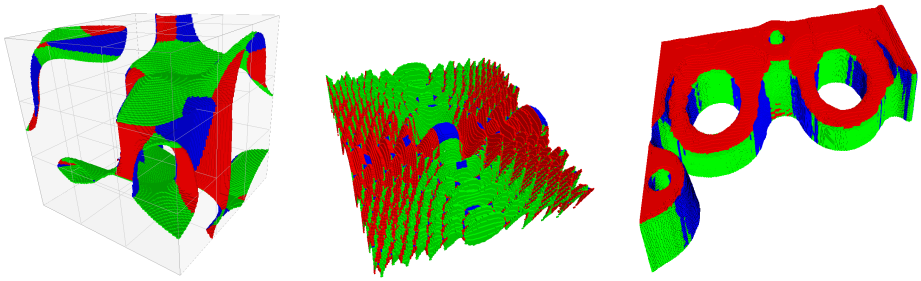
(equations (15) to (20)). In case the surface's normal is required for further processing, as in surface shading, this later is already obtained by our method with less number of operations as compared to the $twenty - one$ additional operations required when computing it directly as in equations (4) to (6).

With the optimization performed (zone boundaries extracted first), the need for the normal information is no more required. In this case, the extraction process consists of just three steps. The first step is for extracting zone boundaries as shown above. The second step is for sorting extrema points on the plane to which the surface is most aligned. The last step is for the incremental scan hyperbolic curves filling between lower and upper endpoints (zone interior filling). Since the normal at each point is no more computed here, the algorithm complexity is further reduced as compared to algorithm 1.

7 Experimental Results and Analysis

In Figure 5, we present the obtained results for all possible configurations of the trilinear interpolant function including tunnel cases and multiple isosurface components per cell. Notice that the same notation as in [3] for labelling the cases was adopted here. For visual clarity, the different zones in every connected component are visualized with different colors by involving the free open source C++ *DGtal* Library [24] in our design environment. The three colors correspond to the three surface zones S_x, S_y and S_z.

Figure 6.(a) represents a synthetic grid dataset of size $3 \times 3 \times 3$, with random scalar values between 0 and 255, extracted with isovalue value 150 and a sampling resolution of $50 \times 50 \times 50$. Figure 6.(b) is $(8 - bit, 41 \times 41 \times 41)$ marschnerlobb dataset extracted with isovalue 192 and sampling resolution of $5 \times 5 \times 5$. Figure



(a) (b) (c)

Fig. 6 (a) $3 \times 3 \times 3$ synthetic random scalar values grid dataset. (b) marschnerlobb Dataset. (c) A part of the CT scan of an engine block.

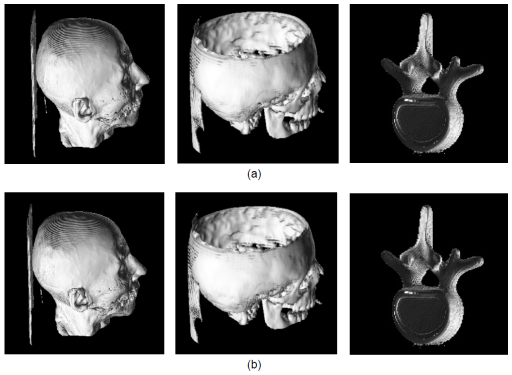
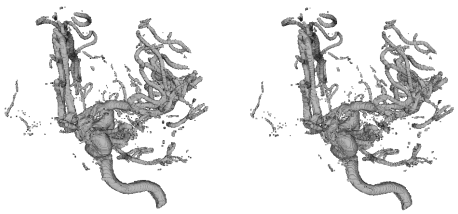


Fig. 7 Examples of a real CT medical datasets. (a) Extracted with DC algorithm. (b)Extracted with our incremental algorithm.



(a) (b)

Fig. 8 Example of the Aneurism Dataset. (a) Extracted with DC algorithm. (b) Extracted with our incremental algorithm.

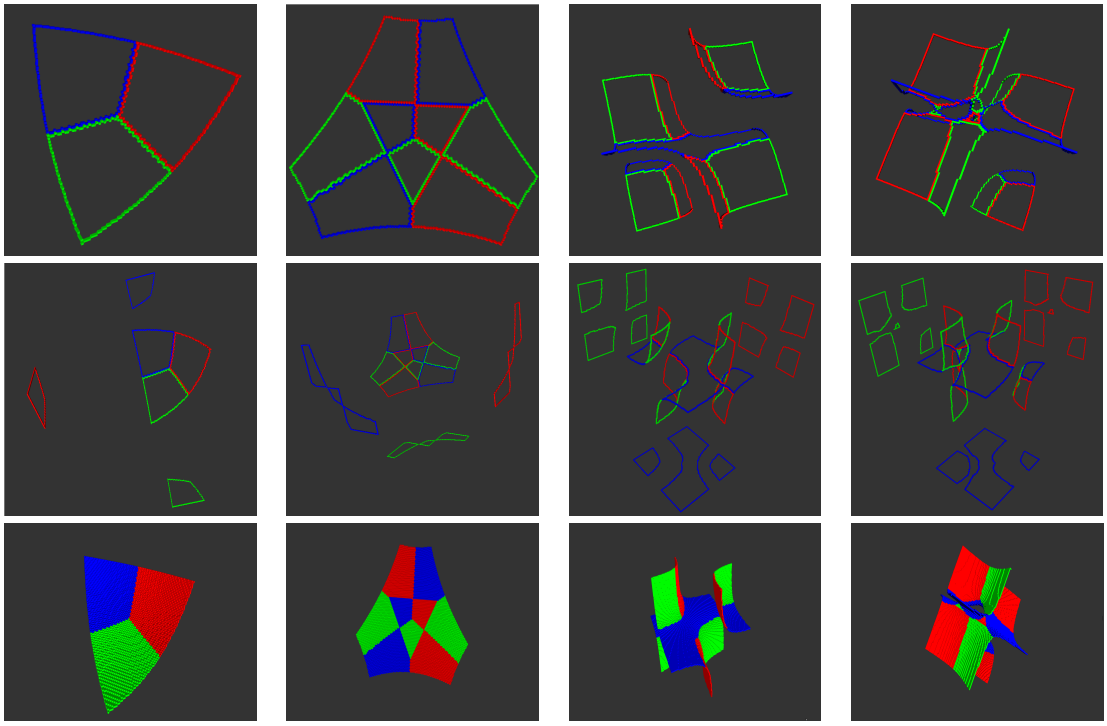
6.(c) is a part of the voxelized isosurface of the CT scan of two cylinders engine block extracted with isovalue 80 and sampling resolution of $5 \times 5 \times 5$. Both the marschnerlobb and the engine datasets were downloaded from volvis.org.

In Figure 7, we show some isosurface examples of real CT medical datasets extracted with an isovalue of 80 and a sampling resolution of $5 \times 5 \times 5$. The DC algorithm was applied on the first row and with our incremental approach on the second row. Isosurfaces of isovalue 80 and sampling resolution of $5 \times 5 \times 5$ extracted from the $(8 - bit,\ 256 \times 256 \times 256)$ Aneurism Dataset, downloaded from volvis.org, are shown in Figure 8. As we can see from these figures, there is no visual difference between images extracted by DC and our incremental approach, whereas our representation requires less points and shorter time for surface extraction as will be seen in the next figures and tables.

In Figure 9 the results of optimizing the incremental method are presented. Four different configurations have been chosen and shown in every column. Each row represents one step of the optimization process. The first row represents the results of the zones boundaries extraction stage for the four different configurations. In the second row we show all zone limits with their projection in their corresponding most aligned plane for every zone type. The last row shows the results of the surface filling process.

In Figure 11, a comparative study for the running time of the three isosurfacing methods (DC, graph-based, and our incremental method with normal checking) is given. As examples, we have chosen the three most complex configurations of figure 5 for our tests; i.e configs. 13.4, 13.5.1, and 13.5.2. All computation results were performed on a Mobile DualCore Intel Core i5-460M, 2.533 GHZ provided with an NVIDIA GeForce GT 425M. In this figure, we plotted curves for the running time (T) for the different methods as a function of the input sampling resolution N along each axis. Therefore, the resulting sub-voxelized resolution in the input active $3D$ cell is of $(N \times N \times N)$. Notice that given running time consists of the time required for points generation with normal checking. The running time required for flag initialization is not considered.

To trace the complexities of these running times, mainly for graph-based and our incremental approach, we plotted in figure 10 the curves for $\frac{T}{N^2}$ versus the input sampling resolution along each axis N; where T is



(a) (b) (c) (d)

Fig. 9 Zones boundaries extraction and surface filling (*a*) Config. 1 (*b*) Config. 9 (*c*) Config. 13.5.1 (*d*) Config. 13.5.2

the computational time already shown in figure 11. We can see clearly that, both the graph-based and our incremental methods are changing quadratically but our incremental approach is faster; i.e ($\frac{T}{N^2} \approx K_1$) for graph search and ($\frac{T}{N^2} \approx K_2$) for incremental search where K_1, and K_2 are positive constants with $K_2 < K_1$.

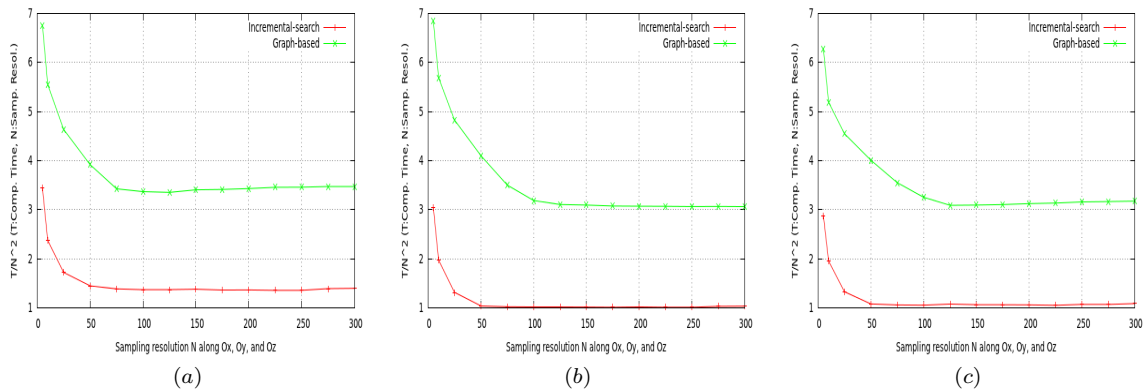
Table 1, 2, and 3 show a comparison between the three methods - DC, graph-based, and our incremental approach- in terms of the total number of points generated within the previous three test configurations. We can see that the number of points generated by our incremental algorithm is significantly reduced as compared to the other two methods especially for high sampling resolutions. In fact the resulting discrete surface, produced by both DC and graph-based, corresponds to the super-cover of the continuous trilinear interpolant surface. Therefore the obtained discrete surfaces are 6-connected, in this case. In counter part, the discrete surface produced by our incremental approach corresponds to the GID discretization scheme described previously. Therefore it is an 18-connected discrete surface. Since the number of points in a 6-connected surface is higher than the number of points in an 18-connected one, the surface generated by our method is thinner than the one generated by both DC and graph-based approaches. The results show that the number of points extracted by our algorithm is almost half the number of points produced by DC and graph-based techniques.

Table 1 Comparison of the three extraction methods in terms of the number of extracted points for config. 13.5.2

Reso.($N \times N \times N$)	DC/Graph.(♯pts)	Our Algo.(♯pts)
$50 \times 50 \times 50$	(6538pts)	(4409pts)
$100 \times 100 \times 100$	(26123pts)	(17287pts)
$150 \times 150 \times 150$	(58785pts)	(39391pts)
$200 \times 200 \times 200$	(104499pts)	(70008pts)
$250 \times 250 \times 250$	(163268pts)	(109329pts)
$300 \times 300 \times 300$	(235109pts)	(157581pts)

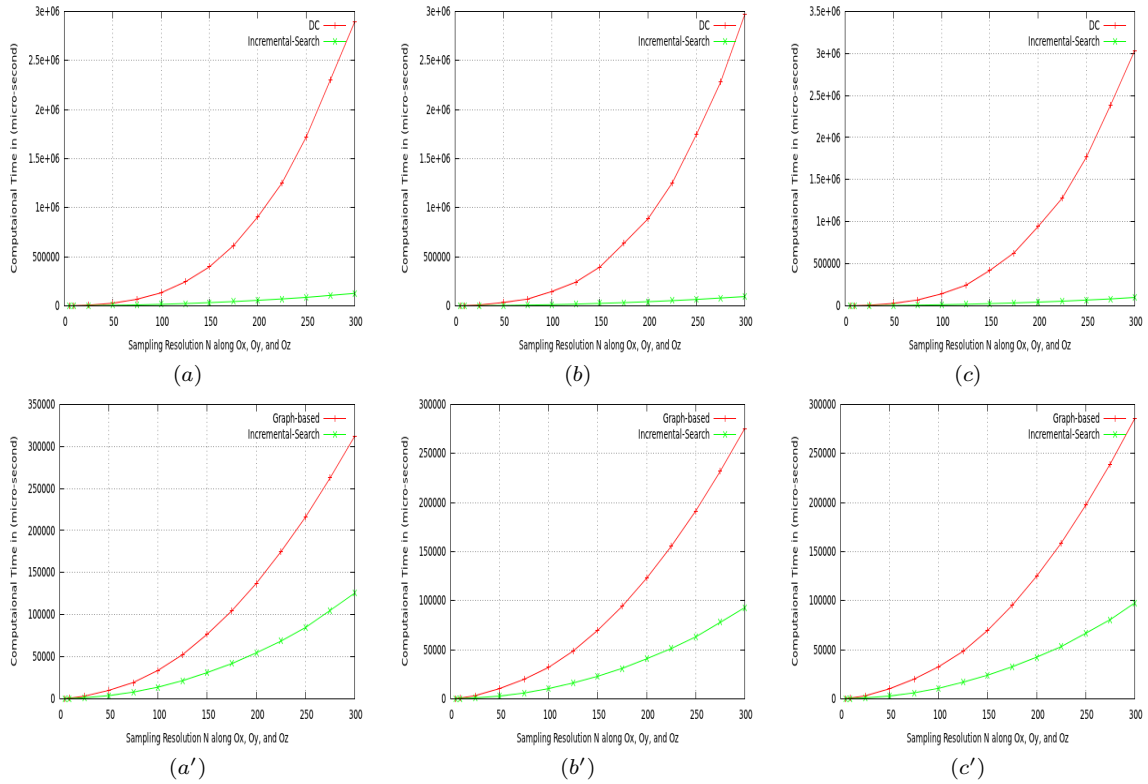
Table 2 Comparison of the three extraction methods in terms of the number of extracted points for config. 13.5.1

Reso.($N \times N \times N$)	DC/Graph.(♯pts)	Our Algo.(♯pts)
$50 \times 50 \times 50$	(6505pts)	(4186pts)
$100 \times 100 \times 100$	(26047pts)	(16743pts)
$150 \times 150 \times 150$	(58623pts)	(37626pts)
$200 \times 200 \times 200$	(104229pts)	(66689pts)
$250 \times 250 \times 250$	(162839pts)	(104067pts)
$300 \times 300 \times 300$	(234472pts)	(149580pts)



(a) (b) (c)

Fig. 10 Time Complexity Analysis. (a) Config.13.4. (b) Config.13.5.1. (c) Config.13.5.2.



(a) (b) (c)

(a') (b') (c')

Fig. 11 Computational Time Vs Sampling Resolution. $(a) - (a')$ Config.13.4. $(b) - (b')$ Config.13.5.1. $(c) - (c')$ Config.13.5.2.

Table 3 Comparaison of the three extraction methods in terms of the number of extracted points for config. 13.4

Reso.($N \times N \times N$)	DC/Graph.(♯pts)	Our Algo.(♯pts)
$50 \times 50 \times 50$	$(7177pts)$	$(5813pts)$
$100 \times 100 \times 100$	$(28709pts)$	$(22947pts)$
$150 \times 150 \times 150$	$(64526pts)$	$(51579pts)$
$200 \times 200 \times 200$	$(114761pts)$	$(90907pts)$
$250 \times 250 \times 250$	$(179372pts)$	$(142095pts)$
$300 \times 300 \times 300$	$(258142pts)$	$(204500pts)$

In Figure 12, we give some configuration examples in which we show how the discrete topology changes as a function of the discretization sampling resolution. As an example, the first column of this figure represents the case where the topology is not preserved due to the low sampling resolution (separated components are touching and tunnels are filled). For more illustration about what we have discussed in section 5.2, inactive sub-cells covering surface saddle points are shown in

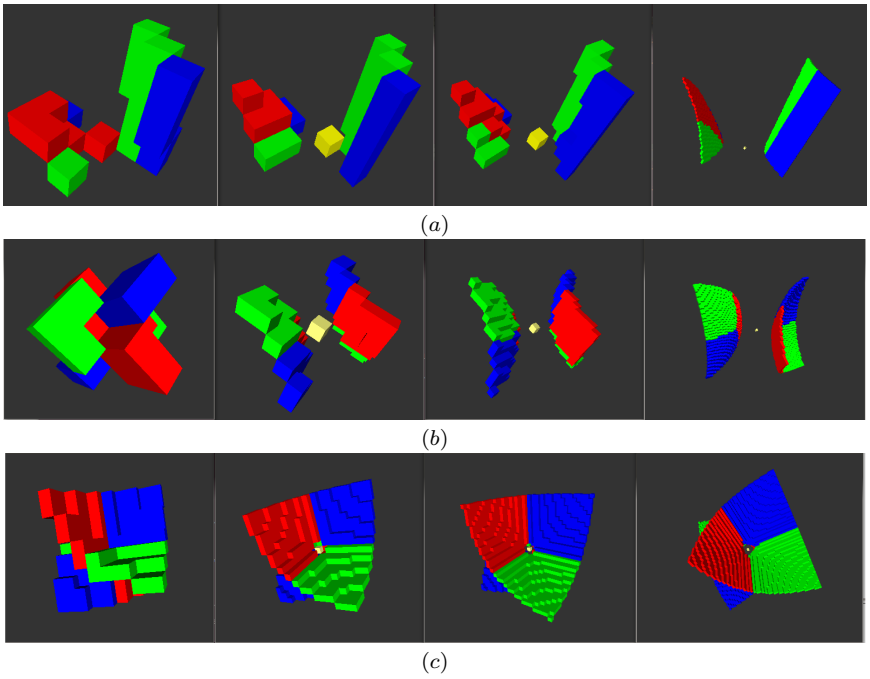


Fig. 12 The effect of the discretization sampling resolution on the correcteness of the discrete surface topology.(a) Config. 6.1.1 (b) Config. 4.1.1 (c) Config. 4.1.2

yellow. Here we can see that the state, active or inactive, of these sub-cells covering saddle points and their $6-$*connected* neighbours have an effect on the correctness of the resulting voxelized surface. In terms of implementation, the topological correctness of the resulting discrete isosurface can be checked with a simple iterative algorithm.

8 Conclusion and Future Work

In this work, we have presented a point-based isosurface extraction method based on an incremental voxelization approach. As compared to the DC and graph-based approaches, our method produces thinner discrete isosurfaces in a fast and optimized way. Furthermore, our voxelization technique guarantees obtaining correct and topologically consistent isosurfaces. Our method has been successfully tested on all different topological configurations of the trilinear interpolant. The complexity of our incremental algorithm is significantly reduced as compared to either an exhaustive scan or to a graph-based approach.

We have proved that the discrete surface generated by our incremental algorithm is a subset of the surface generated by DC algorithm and which is consistent with the trilinear interpolant. In addition, we have discussed the problem about the choice of the sufficient discretization sampling resolution which ensures the topological correctness of the voxelized surface around critical points.

The use of the trilinear interpolant for isosurface approximation is mainly due to its simplicity. However this type of interpolant generates G^0 continuous surfaces. For that reason, we plan to smoothly blend the generated discrete patches to make them globally G^1 continuous without changing their topologies. One way to solve this problem is by applying an appropriate re-parametrization of the domain of the scalar field as done in [15] followed by a discretization scheme similar to ours. Further direction includes developing a similar strategy for higher order interpolant, such as tricubic spline, in order to obtain G^1 continuous surfaces with low computational time. Finally, a more important computational time reduction can be achieved via a *GPU* implementation. One possible way to achieve this is by finding starting points laying on surface face curves and then extracting different parts of the surface in parallel.

9 Acknowledgement

These results were obtained during an eighteen months internship in LIRIS Laboratory, Lyon2 University of France. Therefore we would like to express our gratitude

to all members of the LIRIS-M2DisCo team for their valuable feedback and guidance that helped us significantly throughout this work. The work was funded by Algerian Ministry of Higher Education and Research.

References

1. Nielson M.G., On Marching Cubes, IEEE Trans. Visualization and Computer Graphics, vol. 9, pp 283-297 (2003).
2. Sreevalsan-Nair J., Linsen L., Hamann B., Using Ray Intersection for Dual Isosurfacing, Int. Conf. Computer Graphics Theory and Applications, (2006).
3. Lopes A., Brodlie K., Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing, IEEE Trans. Visualization and Computer Graphics, vol. 9, pp 19-29 (2003).
4. Carr H., Max N., Subdivision Analysis of the Trilinear Interpolant, IEEE Trans. Visualization and Computer Graphics, vol. 16, pp 533-547 (2010).
5. Lorensen W.E., Cline H.E., Marching Cubes: A high resolution 3D surface construction algorithm, In: Computer Graphics, Vol. 21, pp 163-169 (1987).
6. Cline H.E., Lorensen W.E., Ludke S., Crawford C.R., Teeter B.C., Two algorithms for three-dimensional reconstruction of tomograms, Medical Physics, Vol. 15, pp 320-327 (1988).
7. Chernyaev E.V., Marching Cubes 33: construction of topologically correct isosurfaces. Technical Report CERN CN 9517 (1995).
8. Custodio L., Etiene T., Pesco S., Silva C., Practical considerations on Marching Cubes 33 topological correctness, Computer & Graphics, Vol. 37, pp 840-850 (2013).
9. Newman T. S., Yi H., A survey of the marching cubes algorithm, Computers And Graphics, Vol. 30, pp 854-879 (2006).
10. Bresenham J. E., Algorithm for computer control of a digital plotter, IBM Systems Journal, Vol. 4, pp 25-30 (1965).
11. Liu X-W., Cheng K., Three-dimentional extension of Bresenhams algorithm and its application in straight-line interpolation, Proc. Instn Mech Engrs, Vol. 216, pp 459-463 (2002).
12. Boumghar F.O., Miguet S., Nicod J.M., Complexity of discrete surfaces in the dividing-cubes algorithm. DGCI 1996, LNCS Springer Verlag, pp 269-280 (1996).
13. Kebaili A., Boumghar F., Optimal blocs subdivision in the "Dividing-cubes" algorithm. Application to the 3D medical imagery, In Proc. MGV vol. 9, pp 281-288 (2000).
14. Bong-Soo S., Topology Preserving Tetrahedral Decomposition Applied To Trilinear Interval Volume Tetrahedrization, in KSII Transactions on Internet Information Systems, vol. 3, pp 667-681 (2009).
15. Theisel H., Exact Isosurfaces for Marching Cubes, Computer Graphics forum, vol. 21, pp 19-31 (2002).
16. Newman T.S., YiA H., A survey of the marching cubes algorithm, Computers Graphics, vol. 30, pp 854-879 (2006)
17. Hamann Co.B., Joy K. I., Iso-splatting: A Point-based Alternative to Isosurface Visualization, In Proc. Computer Graphics and Applications, pp 325-334 (2003).
18. Livnat Y. , Tricoche X., Interactive point-based isosurface extraction, In: Proc. of visualization 04, Austin, pp 457-464 (2004).
19. Zhang H., Kaufman A., Interactive point-based isosurface exploration and high-quality rendering, IEEE Trans Vis Comput Graph., vol. 12, pp 1267-1274 (2006).
20. Guanfeng J., Han-Wei S., Jinzhu g., Interactive Exploration of Remote Isosurfaces with Point-Based Non-Photorealistic Rendering, In Proc. IEEE Pacific Visualization Symposium, pp 25-32 (2008).
21. Kong T.Y., Rosenfeld A., Digital topology: introduction and survey, In Journal of Computer Vision, Graphics, and Image Processing, Vol. 48, pp 357-393 (1989).
22. Jonas A., Kiryati N., Digital representation schemes for 3D curves, In Pattern Recognition, Vol. 30, pp 1803-1816 (1997).
23. Andres E., Discrete linear objects in dimension n: the standard model, In Graphical Models, vol. 65, pp 92-111 (2003).
24. DGtal: Discrete Geometry Tools and Algorithms, http://liris.cnrs.fr/dgtal/.

Author Biographies

Rachid NAMANE obtained, in 2001, an engineering degree in computer engineering from the Institute of Electrical and Electronics Engineering of Boumerdes University. He then obtained a Magister degree in computer engineering from Polytechnic Military School of Algiers in 2003. Since 2004, he is an assistant professor at the Institute of Electrical and Electronics Engineering of Boumerdes University, and an associate researcher in the Laboratory of Robotics, Parallelism and and Embedded systems - Parallelism and Medical Imaging Research Team - at the University of Sciences and Technology of Algiers where he is preparing his PhD degree in computer engineering.

Serge MIGUET graduated from the ENSIMAG (Grenoble, France) in 1988. He obtained a PhD from the INPG in 1990. He was an Assistant Professor at the ENS de Lyon, and a member of the LIP laboratory from 1991 to 1996. He received his Habilitation Diriger des Recherches from the Universit Claude Bernard Lyon 1 in 1995. Since 1996, he is a full Professor in Computer Science at the Universit Lumire Lyon 2, and a member of the LIRIS laboratory, UMR CNRS 5205. His main research activities are devoted to models and tools for image processing, image analysis, shape recognition.

Fatima OULEBSIR-BOUMGHAR is Professor in Electronic and Computer Sciences at the Algiers University of Sciences and Technology (USTHB) since 1980. She was the Head of the Laboratory of Robotics, Parallelism and Embedded systems (LRPE) and she managed the ParIMed research team. She was responsible for Control Process and Interactive Rendering (CPRI) PhD program. She obtained her Engineer Degree from E.S.E. (Supelec, Paris, 1980) and her PhD in Signal and Image parallel processing from USTHB in 1998, with the collaboration with LIP-ENS (Lyon). Her main research activities are devoted to Medical Imaging, Shape and Surface Modeling, Point-based Graphics, Computational Brain Imaging and Parallelism.