Volume Octree with an Implicitly-defined Dual Grid

Alejandro León Juan Carlos Torres Francisco Velasco

E.T.S. de Ingenierías Informática y de Telecomunicación, Dpto. Lenguajes y Sistemas Informáticos, C/ Periodista Daniel Saucedo Aranda s/n, 18071, Granada, (Spain)

Abstract

Volume data are usually represented as a uniform structured grid. Nevertheless, this representation imposes large memory requirements when large volumes are stored.

Hierarchical representations, can be used to reduce either memory requirements or rendering time. Their use to store the volume model make the surface extraction process more complex, as cracks may appear on the boundary of cells with different size. When they are used only as an index to the cells of the grid the memory requirements are increased. Dual cells method has been proposed to solve cracks problems, but it require to explicitly generate a dual grid, which increases the memory requirements.

This paper presents a new, octree based, multirresolution representation for volumes that implicitly stores the dual grid using just one byte per leaf node. Isosurfaces are built from these implicitly stored dual cells on the fly when the voxel octree is traversed.

Our proposal permits to have a volume represented by a hierchical structure without storing both structures, the grid of samples and the index tree; just the tree stores all the information. So, we can have a storage requeriment similar to a grid and a visualization time similar to a structure that uses an index tree.

Key words: Volume Modelling, Adaptive Representations, Isosurface Extraction, Contouring

1 Introduction

The standard structured grid representation scheme is not suitable for representing high-resolution volumes. In order to represent accurately fine details, the grid must store dense samples. This leads to large memory requirements because the entire volume is sampled at the same rate, even for areas where such a resolution is unnecessary. Moreover, the isosurface extraction process is slow, as require to traverse the whole grid to look for active cells.

Email address: aleon@ugr.es (Alejandro León).

Several solutions have been proposed to reduce the memory requirements and/or processing time, most of them using some kind of hierarchical structure[1]. These structures make possible to represent the volume by different sampling rates at different locations. Thus, the representation contains cells of different size. The main drawback of these approaches, when the volume is visualized by isosurface contouring [2,3], is the *crack problem*—holes in the isosurface placed at the boundary between cells of different size. Cracks arise on these boundaries because the triangle vertices on each side of the boundary are computed using different resolutions. So, they do not coincide and this way the crack is produced. Hence, the visualization method for these multirresolution schemes is normally accomplished by one of two different approaches: either avoiding the cracks [4,5,6,7] or by filling them with triangles [8,9,10].

A different strategy is to generate dual isosurfaces [11,12]. The vertices of each triangle in a dual surface are located inside the cells instead of being placed on the cell edges as in the classical approaches. In the former case (dual surfaces), every triangle is crossed by a cell edge. Every cell has a unique associated vertex and so there are no cracks on the surface [13].

Several proposals have been made to generate a non-uniformly sampled representation. Frisken and Perry use distance fields which are adaptively sampled (ADF) [4,14], thus obtaining a higher sampling rate in regions where many details are found, and a lower one in homogeneous regions. This information is stored in an octree which allow the samples to be processed in an efficient way.

Ju et al. [5] proposed an octree representation whose leaf nodes, which are actually representing cells, have signs at their corners with exact intersections and normals tagging whatever edges exhibiting sign changes. To contour the octree, they used a dual method similar to Kobbelt's [13] which had not any need for checking sharp features to calculate the minimizer of the quadratic error function (QEF) inside a cell. Instead, for each cell that exhibits a sign change, the method generates a vertex positioned at the minimizer of the quadratic function. Before contouring and to simplify the octree, they constructed a QEF associated with each heterogeneous leaf and collapse leaf nodes whose sum of QEFs is smaller than a predefined tolerance.

Schaefer proposed to use an octree considering the volume elements (voxels), which represent the object, in a traditional exhaustive spatial enumeration view [15]. To extract the isosurface he use a dual grid, whose vertices are located at the centre of a cubic voxel (this ensures that the generated isosurfaces are crack free). This approach is iso-value independent. That is, it is not necessary to create a new octree when the iso-value is changed. This is not the case for methods based on adaptive sampling [4,5], as these methods use the vertex labelling information (for the Ju method) or the extracted isosurface (for the Frisken approach). The main drawback of Schaefer approach is that the dual grid is explicitly stored, thus increasing the required space. Moreover, this inhibit its use for interactive applications (e.g., vir-

tual sculpting), as the dual grid would have to be recomputed when the octree is modified.

Proposed solutions either solve the problem by reducing the storage space required, thus increasing the execution time, or reduce the procesing time increasing the storage space used. The goal of this work is to propose a new volume representation that reduces memory requirements for the representation of large volumes, while permits us a reasonably fast generation of isosurfaces with the same quality that those extracted following a classical marching approach on uniform grids. Our approach is similar to Schaefer's, but we use the octree to store the information of the dual grid topology, using one Byte per octree node. The proposed algorithm extracts a classical Marching Cubes surface using a grid dual to the voxels octree. So, we do not generate a dual surface as in dual contouring methods but a "primal surface" [15].

The following section presents the key ideas for the representation. Section 3 presents the isosurface extraction process. Finally, section 4 evaluates the representation method and the extraction algorithm.

2 Proposed representation scheme

Our goal is to use an octree to store all the volume information, in such a way that a marching algorithm can be used to render such information by traversing the octree, and to obtain a representation to be iso-value independent. Using a *marching* [2] approach implies to define a *cell tessellation* of the volume bounding box, a *traversing method* that visit every cell, and an *interpolation* function to compute isosurface vertices in the cells. Next we briefly explain how we solve these three issues in order to justify our representation.

First, the space is *tessellated* by a dual grid. The cells that make up this grid are dual to the voxels which are represented by the octree's leaf nodes. The vertices of the dual grid are located inside the voxels, and the property value of each vertex is the value of the voxel that contains it. The precise location of the cell vertices is at the centre of voxels. These vertices are joined to build the cells by taking into account the neighbouring relationships among voxels as figure 1 shows in a 2D example. There is a cell wrapping the corner of every voxel, and each vertex of a cell is at the centre of a voxel. The voxel corners lying at the boundary of the volume are an special case.

We associate each cell in the dual grid with an only voxel. This voxel is chosen among the set of voxels sharing the vertex wrapped by the cell, that is the voxels that contain a dual cell's vertices. We call this designated voxel the *minimal voxel* of the cell. A voxel may be a minimal voxel for more than one cell. As will be

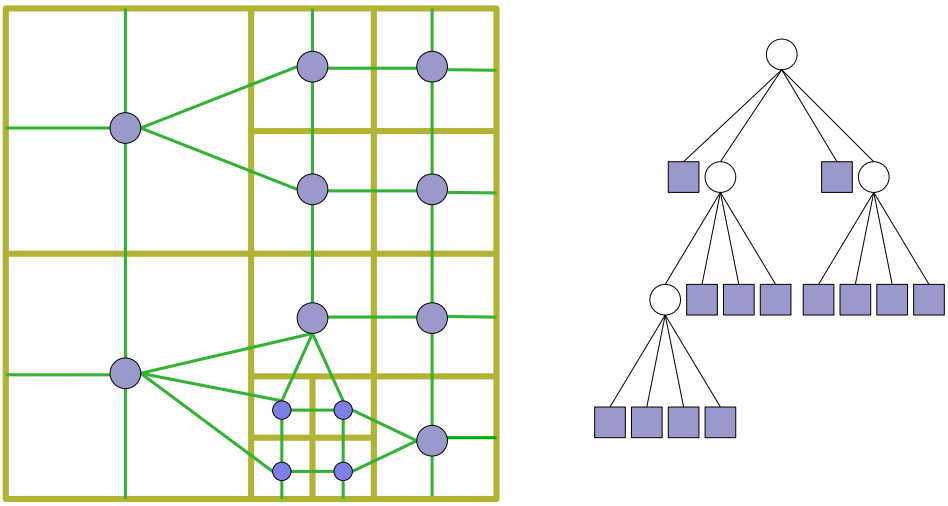

Figure 1. Picture on the left shows a grid of dual cells (green) of a quadtree (yellow) with variable size “pixels”. On the right, a classical tree representation of the same quadtree is shown.

explained below in section 2.2, the criteria used to choose these minimal voxels ensure that any voxel may be at most the minimal voxel of eight distinct cells, thereby it has at most eight associated cells. Therefore, it is possible to store the set of cell references, for which a voxel is a minimal voxel, in just one byte. The set of cell references is an ordered set, in such a way that each bit position “points to” a voxel corner. This order is fixed *a priori*.

Once the minimal voxels are selected and its adequate bits are set, the geometry of any dual cell is easily generated from its orientation with respect to its minimal voxel and the octree topology. In this way, it is possible to *traverse* the cells by traversing the octree, and for each leaf node (voxel) to visit only the cells for which the current voxel is a minimal voxel. Our approach generates the cells of the dual grid on the fly, during the rendering process, avoiding the need to store the dual grid explicitly.

Finally, the patch of geometry inside each dual cell through which the isosurface passes is generated. A modified linear interpolation scheme that takes into account the different resolution of the voxels covered by the cell is used (see section 3).

This representation permits to store regular grid datasets using octrees in which homogeneous regions are represented by large voxels. Figure 2 illustrates the representation with a simple example. On the left, it presents voxels with different sizes and two minimal voxel highlighted in red and blue. Their respective red and blue circles reflect the corners wrapped by the dual cells (not shown by purpose of clarity) for which they are minimal. On the right, our representation is shown. The leaf nodes corresponding with the minimal voxels have the adequate bits set. The example only shows the minimal voxels for voxels’ corners lying in the interior of the volume. In section 2.2 will be explained the minimal voxels assignment for

voxels' corners lying at its boundary.

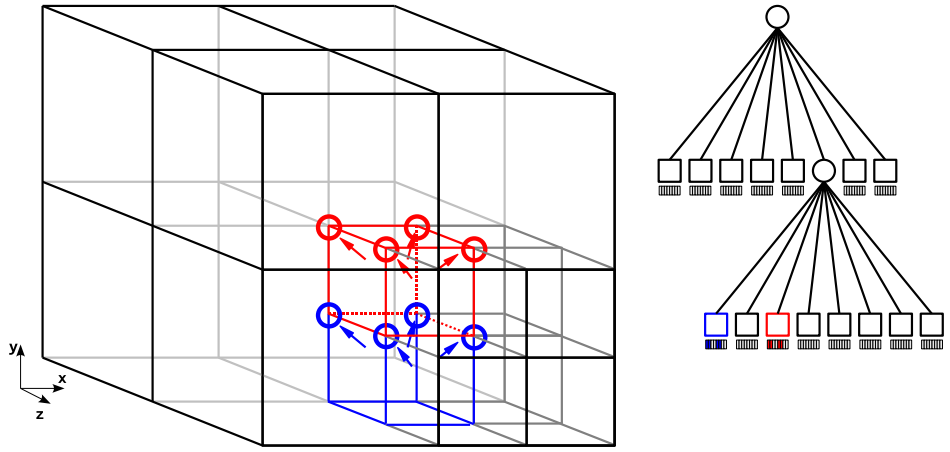

Figure 2. Representation with a minimal voxels assignment (minimal voxels highlighted in red and blue). For each minimal voxel, the octree implicitly stores in a Byte the dual cells that wrap the corners highlighted by red and blue circles. Arrows represent the particular responsibilities assigned to each minimal voxel.

2.1 Building the representation

The proposed volume representation can be easily generated from a regular grid. First of all, a complete octree is generated in which the leaf nodes correspond to a grid value (minimum size voxel). Then, the octree is compressed following a *bottom-up* approach. Whenever eight child leaf nodes satisfy a similarity criterion (*homogeneity criterion*) established by the user, these eight leaf nodes are combined into their parent node which becomes a new leaf node at the next higher level. The simplified octree represents a volume dataset in which every voxel has a single property value, the property value of the *volume inside the voxel*, which is constant. The property value in the border of each voxel (*boundary band*), whose size is one half of the voxels' side in the original grid, changes linearly.

Figure 3 shows the partition of the voxel domain into *inside volume* (green) and *boundary band* (cyan) for a quadtree. The dark red rectangles represent the property value in the inside volume of voxels, whilst the dark and light red lines represent the linear change of property value in the boundary bands. Of course, a voxel of highest resolution does not has inside volume, i.e. all its volume constitutes the boundary band.

In order to complete the representation by adding the dual cells implicitly defined, minimal voxels' bits must be correctly established. We call this process *responsibility assignment* because each bit set in a voxel means that when the octree is traversed this voxel has *the responsibility of processing the corresponding dual cell*.

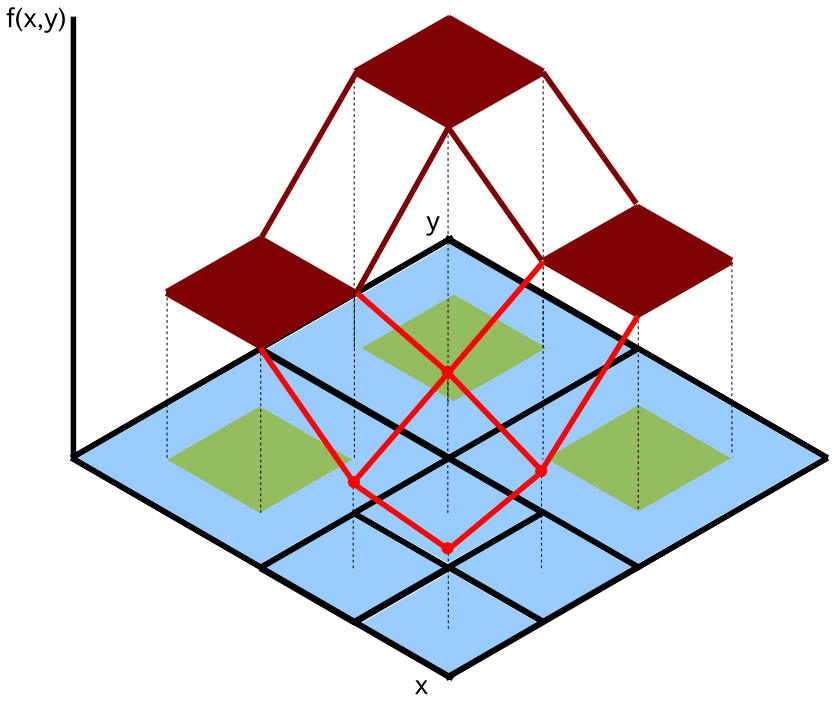

Figure 3. A partition in the voxel domain is established, which divides it into two type of property value assignments: *inside volume*, with constant property value, and *boundary band*, with a trilinear interpolation function for assigning values.

2.2 Responsibility assignment method

The responsibility of processing each dual cell is assigned to the smallest voxel among those sharing the vertex wrapped by the cell. This ensure that no voxel can be a minimal voxel for more than eight cells. The process of computing responsibilities is done only once, when the octree is build. Once the process of responsibilities assignment has finished, each minimal voxel in the octree has set the adequate bits in its associated one Byte field—we refer it as *responsibilities field*. Each bit references the location of one of the eight corners of the voxel. Setting a bit means that the cell wrapping the corner in such a location must be visited (processed) from this voxel. The criteria to set all these bits must ensure that:

(1) The only voxel, among the set of voxels that share a voxel corner, that may have the responsibility of processing the corner's associated cell is the minimal voxel of such a cell.
(2) There are no two voxels that are responsible of visiting the same cell. This prevents a dual cell from being processed twice.
(3) Every cell making up the dual grid must have an associated minimal voxel, so that all of them can be processed.

We achieve this requirements by carring out the process that follows. Firstly, we assume that the octree is placed in the first octant of euclidean space. This permits us to determine the adequate corner that initially will be checked for each voxel with the aim of trying to set its associated bit. This corner is the one placed in South East Top (SET) direction. Figure 4 shows this selected location along with the edges and faces that must be considered to search for neighbours. We choose such conditions because, in this way, our method guarantees that all voxels' corners interior to the volume will have its minimal voxel at the end by checking only this direction for each voxel, whenever the voxels to be visited in Morton order. Therefore, the redundant search for neighbouring voxels in the rest of voxel's corner is avoided, so saving execution time.

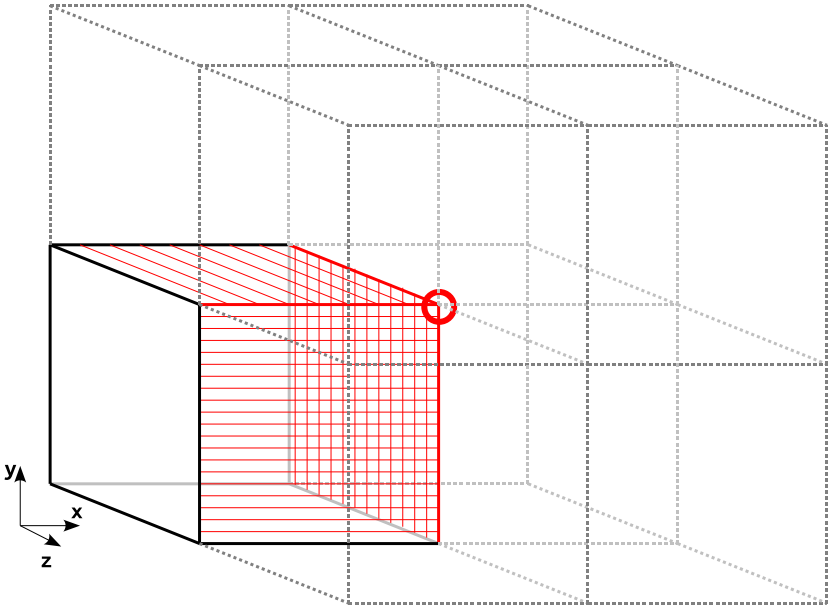

Figure 4. Each voxel is checked for the corner placed in the South East Top direction to try to set its bit in the responsibilities field. The size of the neighbours to the vertex, edges and faces highlighted in red is not known a priori. An octree search must be done.

Next, we traverse the octree following a Morton order and visit each leaf node (voxel). For each voxel, we find all the neighbouring voxels that are necessary for the method to determine the dual cell of the corner in SET location. These are the voxels that share a vertex, edge or face with the highlighted vertex, edges and faces in figure 4. If all of them are greater than or equal to the current voxel size (see top picture in figure 5), then the bit which corresponds with such a corner is set in the currently visited voxel. This is represented in the figure with a red arrow. If at least one of the neighbouring voxels is subdivided, then for each recursively subdivided voxel we set the adequate bit in each descendant that is a neighbouring voxel, taking into account the relative location of such a voxel with respect to the visited voxel. This responsibility assignment is illustrated at the bottom picture in figure 5. Red arrows represent the responsibility asignment for the neighbours at the top face, front top edge and SET vertex of the currently visited voxel. Cyan

and blue arrows are responsibility asignments from another neighbourhoods. In this case, the algorithm must entirely check the list of neighbouring voxels to guarantee requirements 1 and 2. It clear the previously set bits in voxels sharing vertices for which more than one responsibility has been assigned. For example, in figure 5 the current voxel's vertex in the SET location has its processing responsibility assigned in three distinc voxels. In this case, the algoritm choose one among them because all of them have equal size and clear the corresponding bit in the others. This subprocess is called *responsibilities move*.

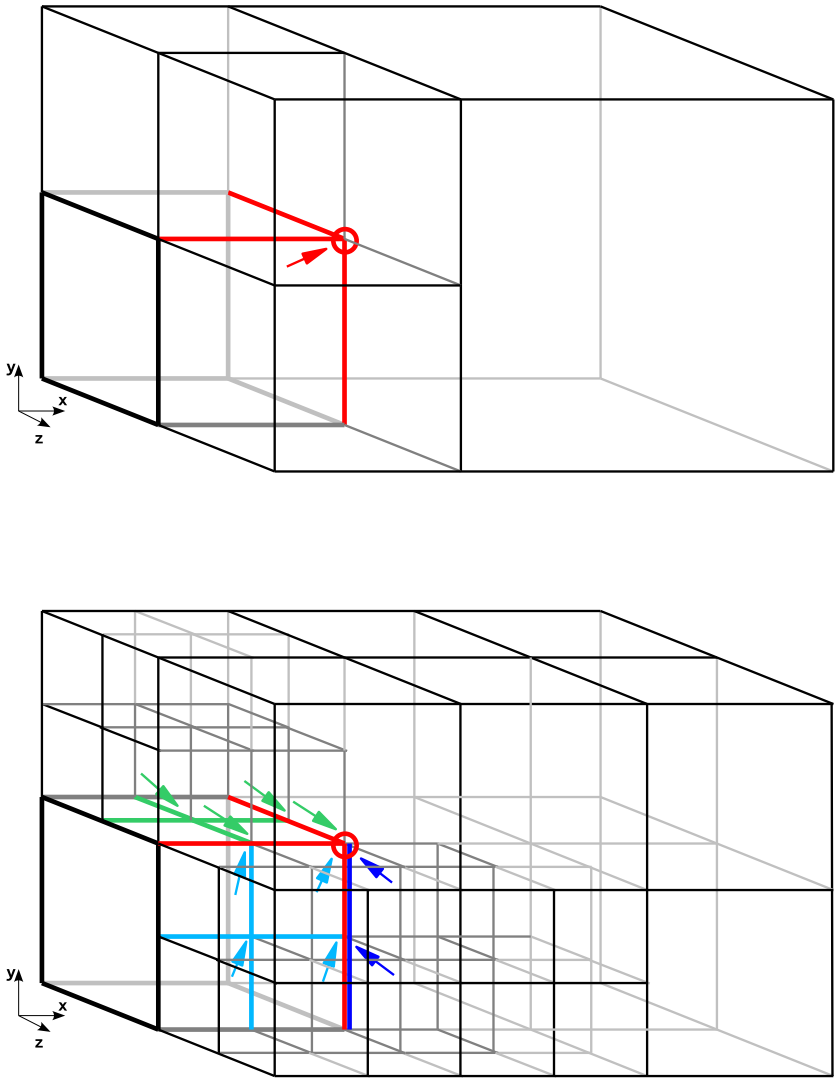

Figure 5. Top picture shows the responsibility assignment whenever the size of the neighbours is greater than or equal than the visited voxel. Bottom picture shows the case in which at least one of the neighbouring voxels is subdivided.

Our traversal method guarantees, once the bit associated to the SET corner has been set or the responsibilities move subprocess has finished, there is no necessity of check the rest of the voxel corners. Only will be necessary to set other bits if this voxel has some corner on the boundary of the volume. Therefore, once the octree

traversal is finished it is only necessary to assign responsibilities to the voxels' coners lying at the boundary of the volume, and the third requirement is fulfilled.

Figure 6 shows a 2D example that permits to observe more clearly than in the 3D example the situations in which the responsibilities move subprocess clear the adequate bits (remove previously assigned responsibilities). The figure simulates a step applied on a currently visited voxel (located in the top left), considering the corner in the South East (SE) direction as the prefixed one. The picture on the left shows the initial state. The picture in the middle shows the responsibilities assignment and two situations, (a) and (b), where first and second requirements are not fulfilled respectively. The corner (a) has its processing responsibility assigned to a voxel that is not its minimal voxel. The corner (b) has assigned two processing responsibilities from two distinct voxels. The picture on the right shows the correct responsibility assignment and the dual cells associated to each corner.

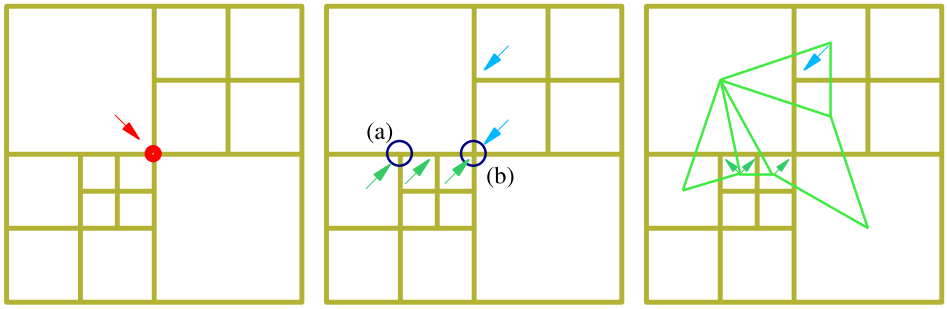

Figure 6. A 2D example that shows the process of responsibility assignment and the situations in which it is necessary to apply the responsigilities move subprocess.

The method fulfils the three requirements previously stated. Thus our representation keep an implicitly defined dual grid that may be traversed in an efficient way thanks to the concept of minimal voxel. Every cell has an associated minimal voxel and can be processed by traversing the octree. The next section explain the way we extract an isosurface from this representation.

3 Isosurface extraction from the dual grid

In order to extract an isosurface, the algorithm traverses the octree and, for every leaf (i.e., for every voxel), examines the responsibilities of processing cells stored in it. For each recorded responsibility, we check the associated dual cell—whose centre is at the voxel corner pointed by the responsibility. The voxels that form the cell topology are retrieved, and it is decided whether the isosurface crosses the cell taking into account the isosurface threshold. The property values assigned to the vertices of such a cell are queried from the voxels. It is not necessary to generate the geometry of the dual cell if the test fails. Dual cells are not necessarily cubic, but it is possible to process them as *virtual cubic cells*, in which more than one adjacent vertex may correspond to the same voxel.

Figure 7 shows an example. Black squares in dashed lines represent the topology of the dual cell and polygons in light green represent the final geometry of the dual cell. As can be seen, the dual cell (a) has two “topological” vertices that coincide at the same geometrical vertex, and yet the topological vertices of cell (b) present a one to one mapping to the geometrical ones. From now on, the ordering (and coincidences) of the vertices of a dual cell will be referred as *topological cell*, meaning that this information reflects the topological relationships among them. This relationships are imposed by the particular responsibility assigned to the minimal voxel of the cell and the different size of each neighbouring voxel that constitutes it.

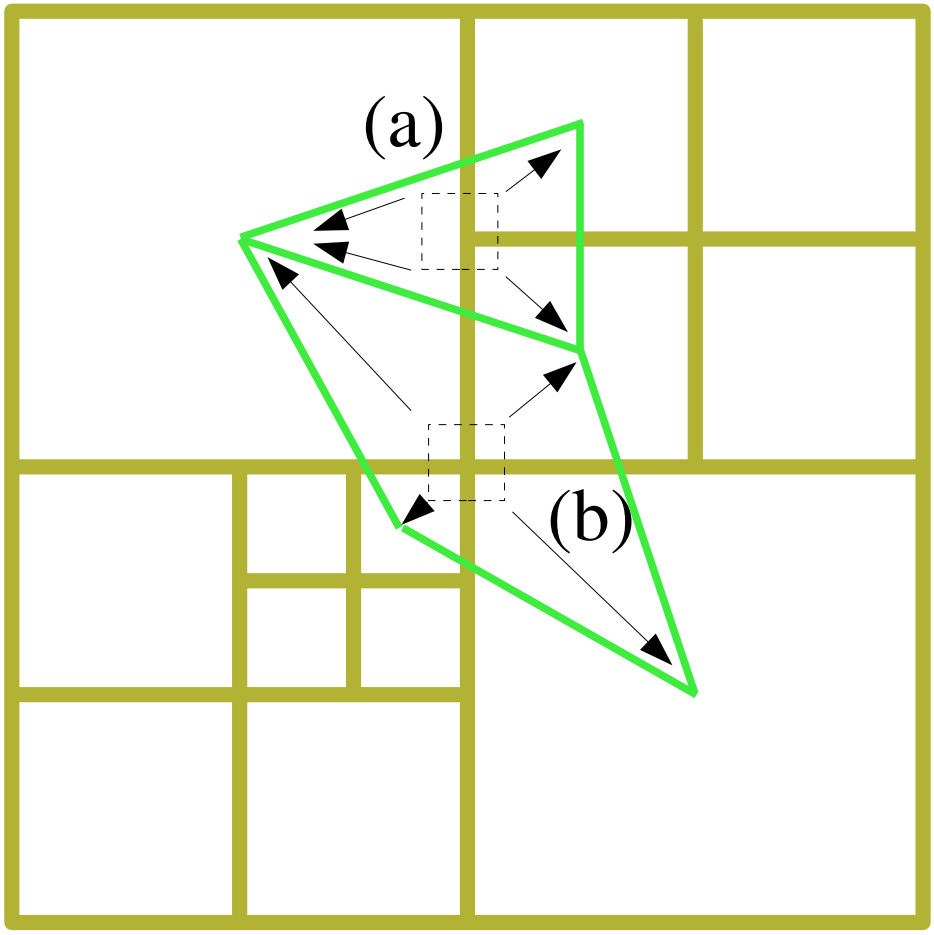

Figure 7. Mapping from topological cells to geometrical cells. Two vertices of the topological cell (a) maps to a single geometric vertex. All the vertices of the topological cell (b) maps to a different geometric vertex each.

Whenever the isosurface passes through a dual cell, the geometry of this cell is generated by positioning its vertices at the centre of the voxels that contain them. Depending on the relative size of these voxels, the dual cell can have different shapes. Nevertheless, it is possible to work on a *virtual cubic cell*, on which property values for vertices corresponding to the same voxel are duplicated. That is, edges joining virtual cell’s vertices located in the same voxel will constitute a degenerated edge because its vertices will have the same value. However this is not

a problem to use whatever version of Marching Cubes. The results are the same if the marching cubes method process a cubic cell and the resulting mesh is adapted to the corresponding dual cell. Obviously, the process can be carried out directly in the dual cell. Moreover, any degenerated edge will not be crossed by the isosurface unless the threshold value coincides with the vertices value, because its vertices have got the same property value.

Once the surface topology of the active cell is determined it is necessary to generate the geometry of the mesh by interpolating in zero crossing edges. This is usually done using linear interpolation. However, if a simple linear interpolation between the end-points of the zero crossing edges is performed, the resulting isosurface greatly differs from the one which would have been extracted from the initial regular grid if marching cubes algorithm had been applied directly. The problem lies in the process of voxels aggregation. This process increases the distance between the end-points of the edges of the dual grid. Figure 8 shows this problem in a 2D example. Blue and red circles represent voxels with equal property value respectively. Voxels are represented in light brown and dual cells in light green. The picture (a) shows that intersection points between the isosurface and the cell edges do not lie in a single straight line as the black circles joined by line segments reflect. Although the property values at the end of each zero crossing edge are the same, the length of each one is different and therefore the resulting cross-point locations varies among edges. The isosurface that would have been extracted using linear interpolation through the original regular grid of voxels and dual cells is showed in picture (b). Brown and green dashed lines represent the regular grid's voxels and its dual cells respectively. All the crossing edges have the same length, thus all the cross-point locations lie on the same straight line. As we want the isosurface extracted from our representation to be as similar as possible to the one extracted from the initial regular grid, we solve this problem by forcing the cross-point to be located in its corresponding edge but only in the segment that overlap the border area delimited by the dual cells of highest resolution. This border area is showed as a cyan band in picture (c) of figure 8. This way the difference between the isosurfaces extracted by both methods is greatly reduced.

Moreover, whether simple linear interpolation is done the extracted surface could be entirely located at the interior of a voxel. This result is incorrect as we consider a voxel as an homogeneous region (see section 2.1). Picture (a) in figure 8 shows this effect that depends on the property values of the end-points of the edge. Our solution guarantees that the isosurface always lie in the correct area. In fact, the area in which the isosurface must lie because is there where the property value actually changes from one neighbouring voxel to another.

Figure 9 shows that the simple interpolation along an edge of a dual cell produces an erroneous result when the two vertices of the edge are located at the centres of coalesced voxels. In the far left of the picture, it can be seen that the cross-points do not lie on the same plane, as would have happened in the regular grid case, because

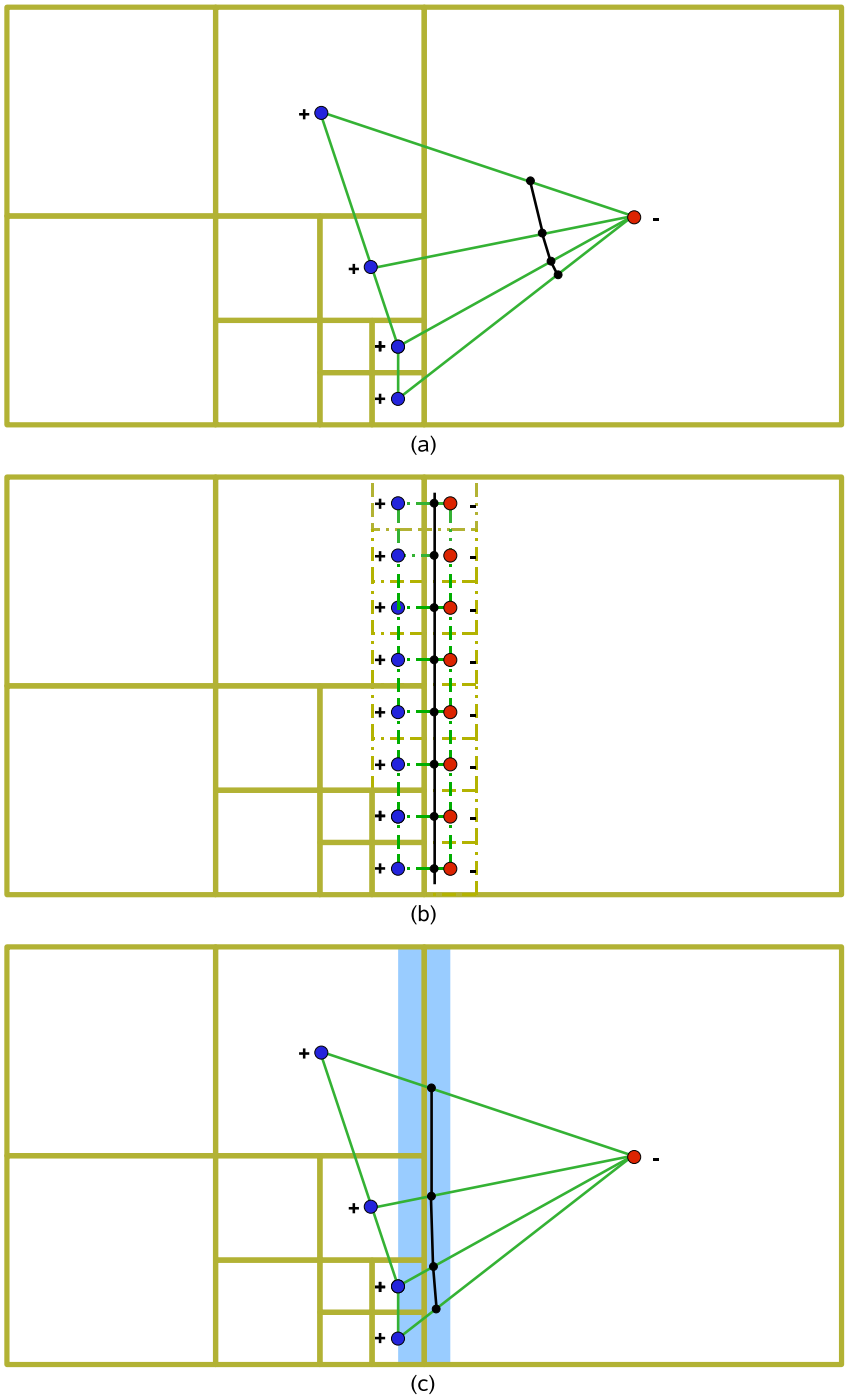

Figure 8. Effect of applying simple linear interpolation along the edges of the dual cells (a). Isocurve generated from original grid (b). Isocurve generated using only the segment of each dual cell that overlaps the border band.

each smaller region (voxel) has the same property value. The effect can be clearly seen in the middle of the picture. If we change the threshold value, results like those presented on the far right of the picture can be obtained, where the surface patch lies entirely inside the biggest voxel.

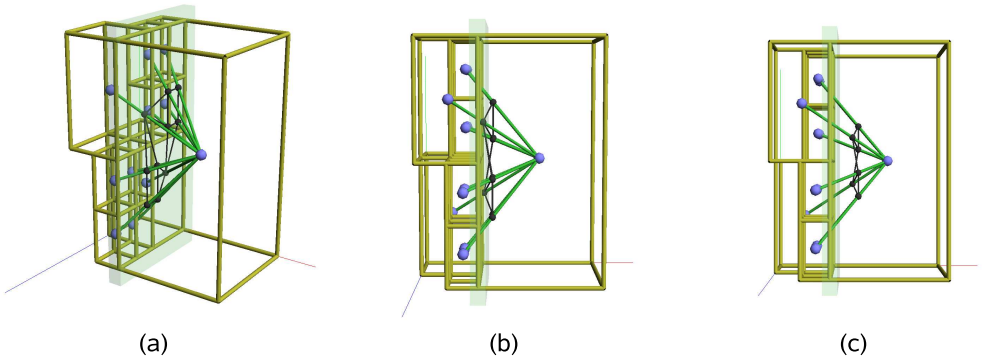

Figure 9. Images (a) and (b) show two different views of the set of cross-points, which do not lie on the same plane. Image (c) shows a situation in which all the cross-points lie inside the biggest region. The cyan transparent region is the border band where isosurface extracted from the regular grid would lie. Black lines represent the linearly interpolated mesh obtained from the dual cells.

To solve this problem, our interpolation function takes into account for each cell vertex the size of its corresponding voxel. The basic idea on our interpolation method is to do not interpolate voxel centres. Instead, we intersect the dual cell with the border band and interpolate inside the resulting polyhedra. In order to compute the intersection points in a feasible way we first interpolate along the edge to get the points, P', closest to the centre of the voxels in the initial regular grid (i.e., close to surface of the border band). Then we interpolate between these points to compute the isosurface intersection point. Being points P_1 and P_2 the centre points of each voxel, P_0 the point located at the contact surface between the two voxels; and P_1' and P_2' the points located along the edge. The following equation system is deduced from Euclidean geometry as can be seen in figure 10:

$$\frac{2^{l_1} \cdot d}{P_1 - P_0} = \frac{d}{P_1' - P_0} \tag{1}$$

$$\frac{2^{l_2} \cdot d}{P_2 - P_0} = \frac{d}{P_2' - P_0} \tag{2}$$

$$P_0 = \frac{P_1' + P_2'}{2} \tag{3}$$

Points P_1' and P_2' bound the edge segment where the isosurface must actually cross. In order to obtain the intersection point, we only have to interpolate linearly between them using the scalar property values of the two voxels.

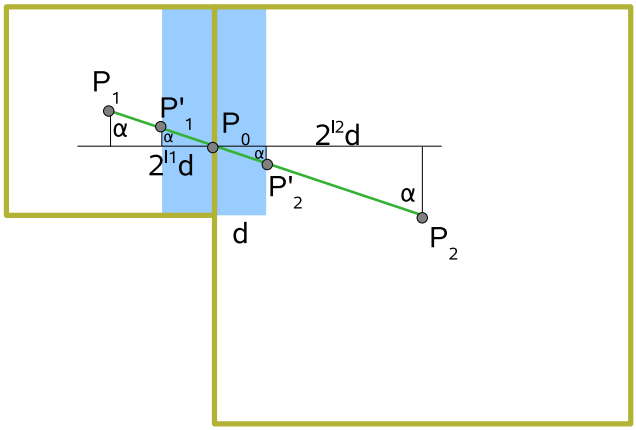

Figure 10. Geometrical relationships establishing the equation system used for calculating the points P_1' and P_2'.

4 Results

The more important characteristics of a volume representation are its memory usage and its rendering time. The proposed representation use a hierarchical structure that can be immediately used to index the volume (for instance storing the property interval in the internal nodes. This type of indices have been used to accelerate marching algorithms, at the cost of doubling the memory requirement [16]. So, we do not want to test the use of the representation as volume index (as it is obviously faster), but to check it directly against the uniform grid with marching cubes. That means, we do not make use of the octree as spatial index. Our conclusion is that the proposed representation has a behaviour similar to that of uniform grid both in space used and isosurface extraction. This implies that it can work as well as BONO [1] for isosurface extraction, without a space penalty.

We have tested the method taking as input grids that have been generated from synthetic models and various volumetric datasets. For the synthetic models, we used two different sampling approaches to determine the simplification ratio obtained by our octree. The first approach generates samples with equal property values within the object, while the second one produces property values which decrease as the distance from the position where the sample is taken to the centre of the object increase. All tests were carried out on a standard PC with an AMD 64 (2 MHz) processor and 1 GB of RAM memory.

In order to compare the sensibility of the representation against the size of the dataset, several volumetric representations with different sampling frequencies were generated using both methods. For the approach that generates samples in a binary domain, we chose a homogeneity criterion based on an equal property value. In the case of sampling with a discrete domain of property values, a tolerance error value was used for checking equality among samples.

The following tables show the internal and leaf nodes generated for each kind of sampling, together with different sampling resolutions of the grid. A comparison between the grid and the simplified octree is presented, including the number of elements of each structure required to represent the object, the number of bytes required by each structure, and the percentage of space occupied by our representation respect to the regular grid.

The number of internal and leaf nodes of the octree are shown separately because the space occupied by each type of node is different. The property value in the regular grid uses 2 *bytes* while in the octree, each internal requires 4 *bytes* and each leaf node requires: 2 *bytes* for the property value and 1 *byte* for storing responsibilities (one *bit* for each possible one).

Table 1 shows the sampling method to produce different property values in the range 0 to 255, and for the homogeneity criterion, the selected tolerance error, ϵ, is equal to 2. This is a worst case example as very few compression can be achieved.

Res.	Nodes		Space		
	Int.	Leaf	Grid	Octree	Oct./ Grid
25^3	1465	10256	31250 B	36628 B	117%
32^3	2881	20168	65536 B	72028 B	109%
64^3	21121	147848	512 KB	515 KB	100%
100^3	77388	541717	1953 KB	1889 KB	96%
128^3	158039	1106274	4096 KB	3858 KB	94%

Table 1
Storage space required by the grid and the octree for different sampling rates of a sphere. Property domain in the range 0 to 255.

Obviously, when the first sampling approach is applied, a higher coalescing level is obtained and there is no need to consider a similarity ϵ among property values, because we only have two values which identify the interior and exterior part of the object. The results can be seen in table 2.

In the latter case, the memory space requirements of the different sampling resolutions are considerably lower than in the first approach, where the data has different property values in the inner part that represents the object. Obviously, with our homogeneity criterion, we obtain greater savings in memory space as there are more regions with similar property values and the size of such regions increases.

Tables 3 and 4 show the results for the time spent in constructing the octree with the implicitly defined dual grid, and the time spent on the isosurface extraction from our representation and from a regular grid. Table 3 shows the results obtained from

Res.	Nodes		Space		
	Int.	Leaf	Grid	Octree	Oct./ Grid
25^3	573	4012	31250 B	14328 B	45%
32^3	887	6210	65536 B	22178 B	33%
64^3	3613	25292	512 KB	88 KB	17%
100^3	8991	62938	1953 KB	219 KB	11%
128^3	14675	102726	4096 KB	358 KB	8%

Table 2
Storage space required by the grid and the octree for different sampling rates of a sphere. Binary property domain.

a sphere sampled with values in the range 0 to 255, whilst Table 4 shows those derived from a binary sampling which represents the inner and outer parts of a sphere.

	Octree			Grid
Res.	Simplif. Octree	Dual Grid	Extraction	Extraction
50^3	0.19	1.10	1.95	0.50
128^3	1.38	18.80	27.88	7.81
256^3	11.10	156.21	217.28	70.00

Table 3
Processing time, in seconds, for a sphere with a property domain in the range 0 to 255.

	Octree			Grid
Res.	Simplif. Octree	Dual Grid	Extraction	Extraction
50^3	0.20	0.43	1.61	0.58
128^3	1.56	3.64	12.31	8.36
256^3	12.60	16.36	**55.00**	63.55

Table 4
Processing time, in seconds, for a sphere with a binary property domain.

As shown in the previous tables, our representation spends extra time in generating on the fly the cells needed for the isosurface extraction compared to the case of contouring a regular grid. However, if the number of cells to be processed decreases in a similar way to the sphere represented by a binary dataset of 256^3 samples (the third row in table 4), our octree requires a shorter extraction time than the grid does.

As concerns applying the representation to volumetric datasets, the following tables present the space taken up by both representations and the time spent: by the construction of the simplified octree, by the process of responsibility assignment that implicitly defines the dual grid, and by the surface extraction from the octree and the regular grid.

Dataset	Space		
Name: Resolution	Grid	Octree	Oct./ Grid
aneurism: $256x256x256$	32768 KB	2044 KB	6%
bonsai: $256x256x256$	32768 KB	16788 KB	51%
lobster: $301x324x56$	5333 KB	13585 KB	250%
skull: $256x256x256$	32768 KB	55366 KB	169%

Table 5
Size taken up by the grid and the octree for different datasets.

Table 5 shows that the *aneurism* and *bonsai* datasets in our representation require a considerately smaller space than the regular grid. However, the others take up much more space. The reason for this is that the former do not contain much noise in the outer part of the represented object, and therefore, the homogeneity criterion produces very high aggregation rates. In the second case, there is a high frequency data, and so the octree cannot be simplified too much. In the latter case, a solution taking into account more information can be used to eliminate noise. If the relevant interval set of working thresholds is known, an initial filtering of the volume can be carried out.

With regard to the extraction time of the volumetric datasets, we see a similar situation to the case of synthetic models. Our representation spends extra time in the surface extraction process whenever it does not obtain a high simplification rate in the octree. The *aneurism* dataset shown in the first row of Table 6 is an example of a good simplification rate.

Dataset	Octree			Grid
Name: Resolution	Octree	Dual Grid	Isosurf.	Isosurf.
aneurism: $256x256x256$	12.60	21.59	39.23	60.65
bonsai: $256x256x256$	11.71	115.55	183.49	64.60
lobster: $301x324x56$	103.30	81.92	120.82	21.19
skull: $256x256x256$	9.82	289.94	468.37	71.63

Table 6
Processing time, in seconds, spent in constructing the representation and extracting the isosurface.

As can be seen in figure 11, the images generated by Marching Cubes and our method are not visually distinguishable, although the first ones present a jagged appearance because of the binary domain of the property values. When the interior of the synthetic object is discretely sampled with different property values, this effect is eliminated. In the figure 12 the images generated, on the one hand, by Marching Cubes applied in the regular grid which represents the volumetric datasets and, on the other, by our extraction method from the octree, can be compared.

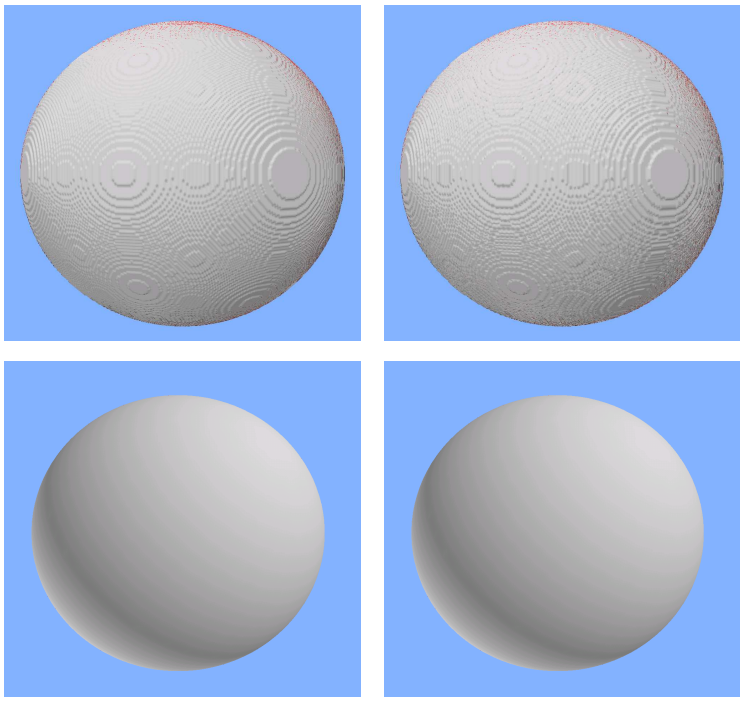

Figure 11. Surfaces extracted from the two sampling approaches. In the first column, images generated by Marching Cubes and, in the second one, images generated by our method.

5 Conclusions

A volume representation has been introduced that enable us to reduce the amount of memory required by a regular grid which represents a volume. To prevent cracks at the isosurface which is extracted from the representation, while maintaining the memory saving, a method for generating and implicitly storing a dual grid of cells from the voxels represented by the octree has been developed.

When it is necessary to interpolate along the zero crossing edges to determine the cross-points between the isosurface and the cells, the difference in size between the voxels where the vertices of the edge come from is taken into account. Thus, we prevent the location of the cross-point depending on the relative size of the voxels and/or on their property values.

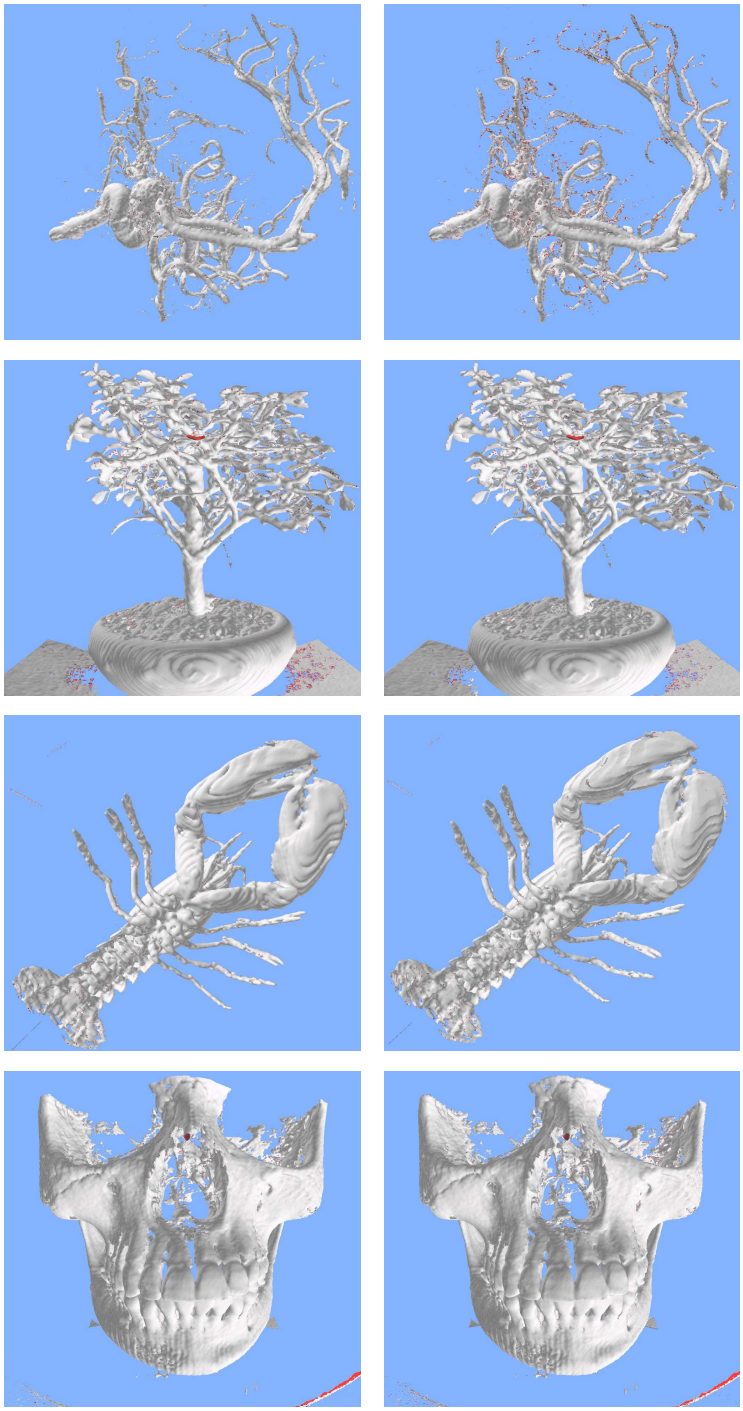

Figure 12. Surfaces extracted from volumetric datasets. In the first column, images generated by Marching Cubes and, in the second one, images generated by our method.

6 Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Technology and EU FEDER funds (projects TIN2004-06326-C03-02 and TIN2007-

67474-C03-02) and by the Andalusian Ministry of Innovation, Science and Enterprise (project PE-TIC-401).

References

[1] J. Wilhelms, A. van Gelder: Octrees for faster isosurface generation. ACM Trans. on Graphics 11 (3) (1992) 201–227.

[2] W. L. William, H. Cline: Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH'87 – ACM Computer Graphics 21 (4) (1987) 163–169.

[3] G. Wyvill, G. McPheeters, B. Wyvill: Data structures for soft objects. The Visual Computer 2 (4) (1986) 227–234.

[4] S. F. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones: Adaptively sampled distance fields: A general representation of shape for computer graphics. SIGGRAPH 2000 ACM Press/ACM SIGGRAPH, 2000, pp. 249–254.

[5] T. Ju, F. Losasso, S. Schaefer, J. Warren: Dual contouring of hermite data. ACM Transactions on Graphics 21 (3) (2002) 339–346.

[6] F. Velasco, J. Torres, P. Cano: Marching edges: A method for isosurface extraction. SIACG 2002, 2002, pp. 199–208.

[7] G. Varadhan, S. Krishnan, Y. Kim, D. Manocha: Feature-sensitive subdivision and iso-surface reconstruction. IEEE Visualization 2003, 2003, pp. 99–106.

[8] H. Müller, M. Stark: Adaptive generation of surfaces in volume data. The Visual Computer 9 (3) (1993) 182–199.

[9] R. Shu, C. Zhou, M. S. Kankanhalli: Adaptive marching cubes. The Visual Computer 11 (4) (1995) 202–217.

[10] R. Shekhar, E. Fayyad, R. Yagel, J. F. Cornhill: Octree-based decimation of marching cubes surfaces. IEEE Visualization '96, 1996, pp. 335–344.

[11] S. Gibson: Using distance maps for accurate surface representation in sampled volumes. IEEE Symposium on Volume Visualization '98, 1998, pp. 23–30.

[12] G. Nielson: Dual marching cubes. IEEE Visualization'04, 2004, pp. 489–496.

[13] L. P. Kobbelt, M. Botsch, U. Schwanecke, H. P. Seidel: Feature-sensitive surface extraction from volume data. SIGGRAPH 2001 ACM Press/ACM SIGGRAPH, 2001, pp. 57–66.

[14] R. N. Perry, S. F. Frisken, Kizamu: A system for sculpting digital characters. SIGGRAPH 2001 ACM Press/ACM SIGGRAPH, 2001, pp. 47–56.

[15] S. Schaefer, J. Warren: Dual marching cubes: Primal contouring of dual grids. Computer Graphics Forum 24 (2) (2005) 195–201.

[16] F. Velasco, J.C. Torres: Cells Octree: A New Data Structure for Volume Modelling and Visualisation. 6th International Fall Workshop on Vision, modeling and visualization 2001. IOS Press. pp. 151-158. Stuttgart, Germany November 2001.