SIACG 2002 - 1st Ibero-American Symposium on Computer Graphics
1-5 July 2002, Guimarães - Portugal

Marching Edges: A Method for Isosurface Extraction

Francisco Velasco Juan Carlos Torres Pedro Cano
Computer Graphics Research Group
University of Granada, Spain
{fvelasco,jctorres,pcano}@ugr.es

Abstract

Volumetric data can be represented as a rectilinear structured grid and can be displayed by the rendering of an isosurface that is built from the grid and defining a threshold value. The know marching cubes method builds the isosurface cell by cell. However, from a set of cells that comply with some conditions, a bigger cell can be built. In these cases, the volume is represented by a set of cells of different size. Then, the isosurface built by marching cubes will have holes on the borders between cells of different size. In this paper, we show a new method, called marching edges, to build the isosurface edge by edge, that generates a hole-free isosurface.

Keywords

Volume visualization, Cell Octree, Marching Cubes, Isosurface extraction without cracks, Triangulation

1. INTRODUCTION

Volumetric data are usually represented as a set of property values v_i at a set of points $(x_i, y_i, z_i) \; : \; i = 1, 2, \ldots, N$; these values are samples from some unknown continuous function $f(x, y, z)$. In order to obtain pictures from that set of samples, a process of three step can be done [Haber90]:

1. *Data enrichment:* An estimation, $F(x, y, z)$, of the unknown function $f(x, y, z)$ is made with which new couples $(point, property\ value)$ can be estimated.
2. *Mapping:* Some geometric interpretation of the function $F(x, y, z)$ is chosen in order to understand its behaviour. This geometric interpretation will typically be a three dimensional object that can be rendered in the next step.
3. *Rendering:* The geometry obtained in the previous step is rendered using standard computer graphics techniques.

An usual way to render volumetric data is the marching cubes method [Lorensen87]. A rectilinear structured grid of the samples is made from the original volume data; an isosurface $F(x, y, z) = v_k$ is built and rendered for some threshold value v_k. Only a subset of the volume is rendered.

In order to build the surface, every cubic cell in the grid is processed: each cell is classified like one of 15 distinct cases by comparing v_k with the property values of the 8 vertices of the cell; every case has a triangulation that represents the isosurface inside the cell; the union of all piece of isosurfaces of all cells forms the isosurface that is rendered.

Since 1987, several papers have been published about the improvement of marching cubes method [Brodlie01].

Concretely, this method needs a long time, most of which is spent processing cells that do not have isosurface inside. In order to reduce this time, several methods have been proposed to search for cells intersecting the surface. These methods can be classified according to the search criteria used:

- *Range-based*, each cell is labelled with the interval that it spans in the range of the property field. This allows to search for intervals that contain a given threshold value. Cignoni et al. propose to use an interval tree structure [Edelsbrunner80, Preparata85] in order to retrieve these intervals, from which active cells are found [Cignoni97]. In particular, they use a method that combines range-based and surface-based search.

- *Surface-based*, only the cells that are adjacent to cells for which the surface has been previously rendered are processed [Shekhar96]. This solution does not display the complete surface when it has more than one component. It is necessary to have a seed set of cells from which all the components can be rendered. Bajaj et al. propose a method to generate seed sets that allow to get all components for all possible threshold values [Bajaj96].

- *Space-based*, domain spanned by the data set is partitioned in order to search active cells. These partitions can be hierarchical [Wilhelms92].

A tipical space-based searching is the bono structure. Wilhelms et al. proposed to build an octree [Meagher80] that indexes the grid, and which stores, in every octree node, the maximum and minimum property value of the subvolume covered by the node [Wilhelms92]. This information is used when traversing the volume and allows that those nodes for which the property interval does not contain the threshold to be skipped. This structure, called BONO (Branch On Need Octree), makes the rendering faster but increases the storage requirement as it must store both the grid and the octree.

Other advantage of the tree is the multiresolution capability. Volume can be rendered at different levels of detail by pruning the tree at a specific level when it is rendered. Furthermore, an adaptive isosurface can be built if different branches of the tree are pruned at different levels. Works like [Ohlberger97, Westermann99] use these issues, however, the prune condition is dependent on the threshold and, in the case of [Westermann99], also on the point of view.

It is also possible to use hierarchical structures and prunning of branches oriented to the visualization process. Livnat et al. use a bono in order to do a hierarchical front-to-back traversal of the data set with dynamic pruning of sections that are hidden from the point of view by previously extracted sections of the isosurface [Livnat98]. Only the isosurface that is visible from the point of view is built. When the viewpoint changes, the isosurface must be rebuilt.

However, when different branches of the tree are pruned at different levels, there will be joined cells of different size. Then, holes (or cracks) on the isosurface can arise on the border between those cells of different size. This is because the isosurface inside the large cell is built from less information than the isosurface inside the small ones (see figure 1).

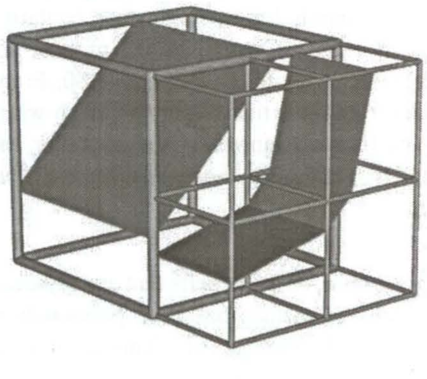

Figure 1. Example of crack

Several solutions have been proposed for this problem. Shu et al. propose covering the crack with a polygon [Shu95]; Shekhar et al. propose removing it by moving the vertices of the triangles in the small cells so that they coincide with the edges of the triangle in the big one [Shekhar96]; Westermann et al. propose to replace a triangle in the large cell with a fan of triangles, in order to adapt the isosurface inside the large cell to the isosurface inside the small ones [Westermann99]; this method with several improvementes is used by Engel et al. to implement a web-based volume visualization system [Engel99]. In these cases, the process can be performed after the triangle mesh has been built or when it is being built.

In a previous work we proposed a modification of bono, called *cell octree*, in which the branches that represent a set of cells whose property values verify some uniform gradient conditions are forever pruned [Velasco01]. The main difference between a cell octree and other approaches is that the prune condition is independent on the threshold. Note that this does not forbit to use an aditional view dependent prune condition.

In a cell octree, cells of different size can be adjacents. In order to avoid crashes on surfaces that cross over two adjacent cells of different size, the prune condition is very restrictive. That is, it allows to prune a branch only when it can be ensured that no crack will appear for any threshold value.

In this paper, we present a method to build the triangles of the isosurface, edge by edge instead of cell by cell. Isosurfaces built with this method can cross the border between cells of different size without any discontinuity. This is so because there are no triangle's vertices on the border between cells, then it is not necessary to make coincide any geometric data on both sides of that border. The cells will have up to one triangle's vertex that will be inside the cell. So, triangles will cross the border. The quality of no discontinuity allows us to use a less restrictive prune condition obtaining a cell octree with less storage requirements.

Next section summarizes the cell octree structure. In section three our proposal, called *marching edges*, is presented, where the concept of *isopoint* is explained. In section four we explain how to compute the isopoints, how to label the edges to avoid processing an edge more than once, and the rendering algorithm. Finally, section five shows data and pictures from real volumes.

2. CELL OCTREE

We use a bono as start point and, by a bottom-up process, making prunes where some uniform gradient conditions are observed. On every prune, eight *brothers* leaf node are forever pruned and their old *father* internal node is converted into a new leaf node. On every prune, the egiht cells represented by the pruned leaf nodes are no longer accessibles and a new bigger cell that is represented by the new leaf node is accessible now (see figure 2).

In the following subsections, some concepts are defined and the prunning criteria for cell octree, as presented in [Velasco01], is explained.

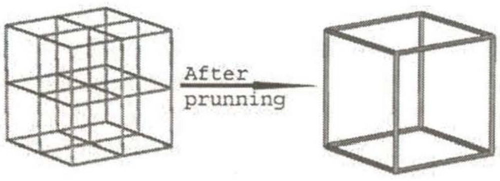

Figure 2. Pruned cells and the new cell

2.1. Definitions

A *face* is *monotonous* when for every possible threshold value, the isosurface crosses it no more than once.

A *cell* is *monotonous* if every of its faces is monotonous.

A *group of eight brother cells* (see figure 3) will be *monotonous* when:

1. Every cell of the group is monotonous.
2. Every face of the cell after prunning is monotonous.
3. For every B vertex (half edge vertex), its property value is between the property values of the end edge vertices A.
4. For every C vertex (center face vertex), its property value is between the property values of the corner face vertices A.
5. For the D vertex (center cube vertex), its property value is between the property values of the corner cube vertices A.

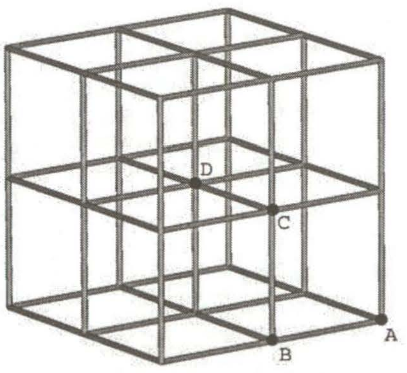

Figure 3. Group of cells to be prunned

The test of monotony for faces is done by checking the property values at its four corners; note that even when the faces of the cell after prunning can be monotonous by using the previous test, the group can be not monotonous if there is a maximun or minimun on vertices of kind B or C, as for some threshold value, the isosurface wild cross the face more than once. The last condition (on vertex D) is set in order to do not loss any important value, as for some threshold value an internal isusurface around the central vertex D will appear.

2.2. Prunning Criteria

The prunning criteria is defined so as to ensure that no cracks can arise on the isosurface built from a cell octree for any threshold value.

A group of eight brother cells will be prunned if the group is monotonous and if any vertex *center of face* is going to need two different property values to avoid the cracks. The remaining of this section justifies this pruning criteria.

A group of eight brother cells must be obviously monotonous to be prunned as if this is not the case there could be a maximum at the center of a face that could be lost in the prunned cells. In this case a crash similar to the one shown in figure 4 will arise.

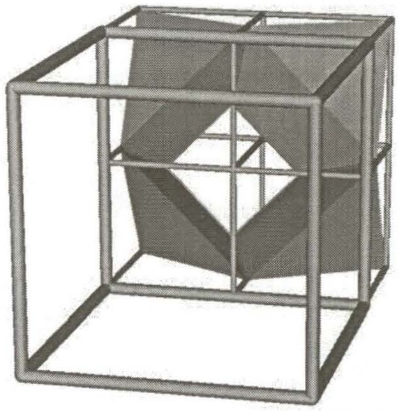

Figure 4. Case of hole

But, this is not sufficient for avoiding all the cracks. In order to avoid cracks like the one showed in the figure 1 more conditions are needed. After a prune is done, the vertices in the center of each new edge can be used by other cells that have not been prunned, so the property value of these vertices are modified in order to make the intesection isosurface-edge coincide in all the cells that share that edge.

Usually, the intersection isosurface-edge is computed by linear interpolation, so the vertices at the midle of the edges of a cell resulting of a prune are modified with the property value resulting of a linear interpolation between the end vertices of the edge. These vertices can be used in the non prunned joined cells.

However, the vertices in the center of each new face are not similar to the vertices at the center of the edges. If the center of a new face is modified with the property value resulting of a linear interpolation between the four corners of the face, there will be cracks on that face (figure 5). There is not a single property value for a vertex *center of face*: it depends on the threshold value and, more important, there are cases where two different property values are needed (figure 6). So if this case is possible for some face of the group of brother cells, the prune is not done.

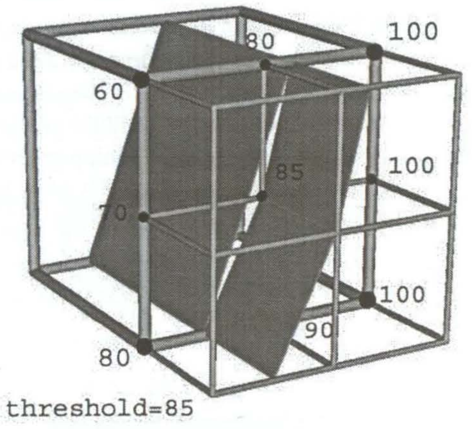

Figure 5. Particular example of crack even when the central vertex has a value resulting of linear interpolation

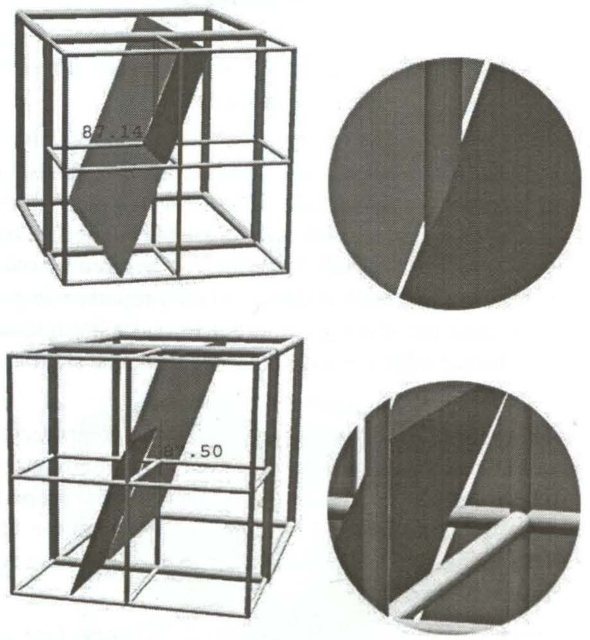

Figure 6. The value that makes coincide one part of both local isosurfaces does not make coincide the other part (see zoomed images). It would have to use both values to make coincide both parts of both local isosurfaces

3. MARCHING EDGES

Let c be a cell intersected by the isosurface. This cell, called *active cell*, has some edges that are crossed by the isosurface. These edges are called *active edges*. Let us consider the active edge (A, B) on the figure 7; the active edge is shared by four active cells. For every cell one point inside of it, called *isopoint*, is calculated and two triangles are built by the four isopoints.

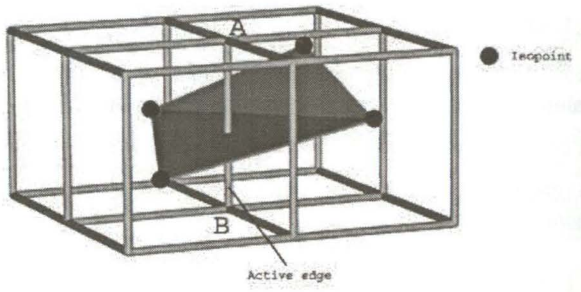

Figure 7. Triangulation for an active edge

Let us consider now cells of different size. Only two distinct cases are possible: first, the active edge (A, B) in the smallest cell is on a face of a larger one (figure 8); second, the active edge (A, B) in the smallest cell is on an edge (C, B) of a larger one and it is not on a face of another cell (figure 9).

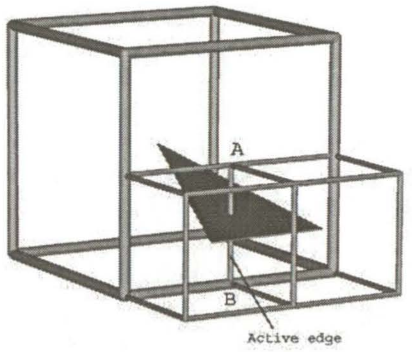

Figure 8. Triangulation by 1 triangle

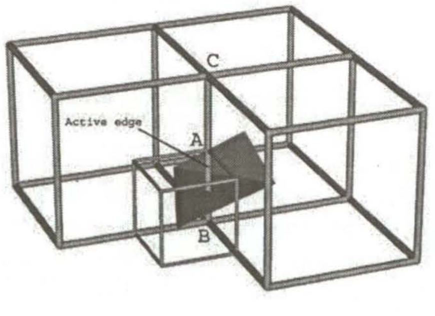

Figure 9. Triangulation by 2 triangles

Every cell that shares the active edge contributes one isopoint to the triangulation, so the triangulation of the first case is made using one triangle, whereas the triangulation of the second case is made using two triangles.

In both cases, if the small edge is active, then the large cell is active too, as the large cell is monotonous.

So, the number of distinct cases by marching edges is three, when the edge is not active, no one triangle is generated; when the edge is shared by three cells (figure 8), one triangle is generated; and when the edge is shared by four cells (figure 9), two triangles are generated.

This method has a drawback: some cases of active cells (the cases 1, 3, 4, 6, 7, and 13 of [Lorensen87]) generate more triangles by marching edges than by marching cubes. However, other cases (the cases 5, 9, 11 and 14 of [Lorensen87]) generate less triagles by marching edges than by marching cubes.

For example, in the figure 10 we can see the case number 1 of [Lorensen87] that generates one triangle using the marching cubes method. As shown in the figure 11 the same configuration has three active edges. For each active edge, up to two triangles are generated using marching edges. These triangles are shared by up to four cells; then, the active cell generates 1.5 triangles.

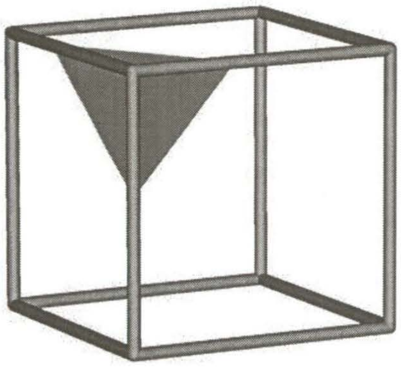
Figure 10. Case 1 by marching cubes

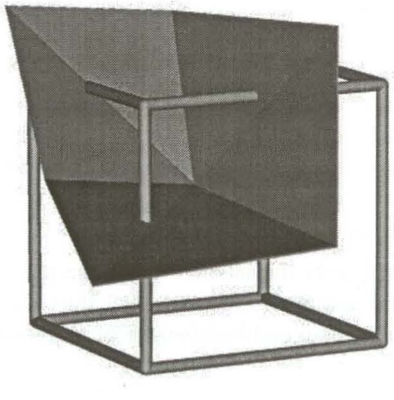
Figure 11. Case 1 by marching edges

The case number 8 of [Lorensen87] (figure 12) is one of the most common in data sets, figure 13 shows how it is triangulated by marching edges, it can be seen that it generates the same number of triangles than using marching cubes. The isopoints have been marked by black circles.

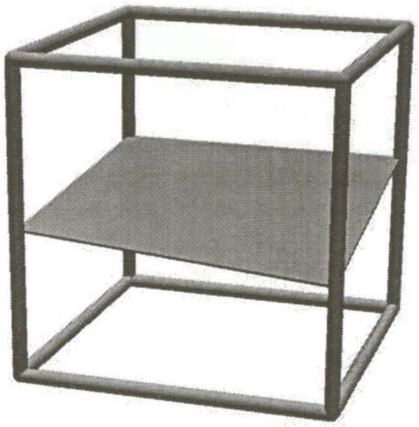
Figure 12. Case 8 by marching cubes

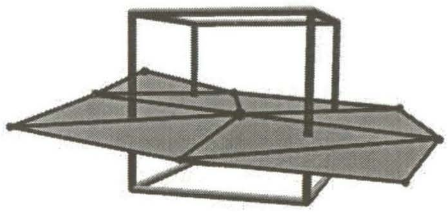
Figure 13. Case 8 by marching edges

The number of triangles built by our method can be between a 25 % less and a 50 % greater than when the marching cubes method is used.

However, the advantages of our proposal are:

- The border between cells of different size will not have any crack, because triangle's vertices will not be on the faces, and so, they will not depend on the different number of samples for that border in the large cell with respect to the small one. Triangles will cross those borders (see figure 14).
- Thanks to the previous advantage, the prune critera used in [Velasco01] can be less restrictive; only the monotony conditions are used, the condition about the center of the faces are not needed. So more prunes are done. Moreover, any property value needs to be changed at any vertex.
- The more prunes are done, the less storage space is needed.

The edges on the limit of the volume do not generate triangles even if they are actives, because there are no cells on the other side. If it is needed, volume must be resized or repositioned.

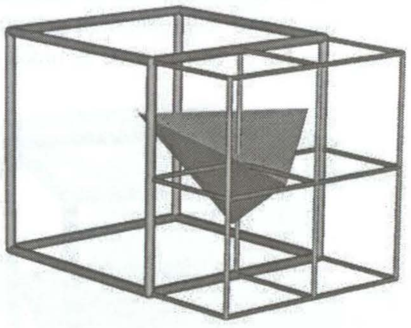

Figure 14. The advantage of no cracks

4. IMPLEMENTATION

Every active edge generates one or two triangles. The vertices of these triangles are into the cells sharing the edge. For every active cell the location of the triangle vertex, that is unique, is computed using only information local to the cell. This ensure that all triangles crossing the cell are joint at the same vertex avoiding crash. We will show now how the location of the vertex, that we call *isopoint*, is computed.

4.1. Computation of Isopoints

We present the method we use to calculate the isopoint that uses information about the cell and the threshold. Other methods like using a fix relative position can be faster but also they can generate worse isosurfaces.

Let $c = (c.x, c.y, c.z, c.s)$ be an active cell where $(c.x, c.y, c.z)$ is the cell's vertex nearest to the origin, and $c.s$ is the size of the cell. And let v_k be the threshold value. The isopoint of the cell c will be a point $P \in c$ that represents the cell in order to build triangles.

In order to calculate the isopoint P, (see the figure 15) the central point p_c and its property value is computed by the function F. For every cell's vertex p_i we can know if the isosurface crosses the virtual line l_i between p_i and p_c. If l_i is crossed, an isopoint P_i can be computed by linear interpolation. The isopoint $P = (P.x, P.y, P.z)$ will be the average point of all the P_i computed in the cell.

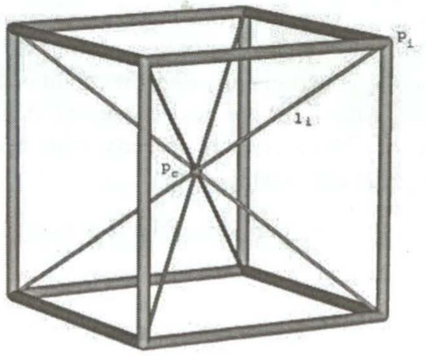

Figure 15. Computation of isopoints

The gradient vector $G = (G.x, G.y, G.z)$ at P is computed as:

$$G.x(P) = (G.x^0 (1-\Delta y) + G.x^1 \cdot \Delta y) \cdot (1-\Delta z) + (G.x^2 (1-\Delta y) + G.x^3 \cdot \Delta y) \cdot \Delta z$$

being

$$\Delta x = \frac{P.x - c.x}{c.s}$$
$$\Delta y = \frac{P.y - c.y}{c.s}$$
$$\Delta z = \frac{P.z - c.z}{c.s}$$

and

$$G.x^0 = F(c.x + c.s, c.y, c.z) - F(c.x, c.y, c.z)$$
$$G.x^1 = F(c.x + c.s, c.y + c.s, c.z) - F(c.x, c.y + c.s, c.z)$$
$$G.x^2 = F(c.x + c.s, c.y, c.z + c.s) - F(c.x, c.y, c.z + c.s)$$
$$G.x^3 = F(c.x + c.s, c.y + c.s, c.z + c.s) - F(c.x, c.y + c.s, c.z + c.s)$$

$G.y$ and $G.z$ are computed in a similar way.

The gradient vector is used to allow a shadowed rendering, like the one displayed by the phong method [Bui-Tuong75].

Every isopoint is computed just once, they are stored in a set of isopoints to use them when it is necessary.

4.2. Edge Labeling

Every edge can be shared by up to four cells. If the volume is processed cell by cell, and for each cell, all its edges are processed, every edge can be processed up to four times. So every cell has a label to indicate what edges must be processed. Every edge in the volume is processed just once.

A cell octree can have cells of different size, the edges that are shared by cells of different size (figures 8 and 9) are labeled and processed in the smallest cell. This is so in order to make easy the searching of neighbourg cells to find the needed isopoints to build triangles. When a labeled active edge is being processed, it is necessary to find the isopoints (to build triangles) inside the neighbour cells. Note that our method can process only edges that are shared by up to 4 cells. When an edge is shared by cells of different size, we process the edge on the smallest cell.

In every cell, only the labeled edges are processed, then the minimun and maximun property values stored at each cell are only computed in function of the labeled edge's vertices, so the intervals $[min, max]$ stored in the tree's nodes can be smaller and some tree's branches will not be processed with a higher probability and the visualization time will be shorter.

Note that the prune criteria and the edges labeling is independent on the threshold value.

4.3. Visualization

The pseudocode of the procedure to render the volume is shown in the figure 16, where V1 and V2 are the vertices of the edge E and F(vertex) is the estimated volume function.

```
render_volume (node N,threshold T)
{
  if (N has sons)
    for each son S of N
      if (T is between min and max of S)
        render_volume (S,T)
  else /* it is a leaf node */
    for each labeled edge E=(V1,V2) of N
      if (T is between F(V1) and F(V2))
      {
        classify E
        compute the isopoints
        show the triangles
      }
}
```

Figure 16. Pseudocode of render_volume

5. RESULTS

The proposed method has been implemented and compared with bono using marching cubes, and cell octree using marching cubes. Tests have been performed with the volumes showed in figure 19. The volumes have been modelled from TAC and the data have been normalised between 0 and 255 (the threshold for the tests is 60).

Table 1 shows the storage requirements for the octree. Table 2 shows the time used to build the tree. Table 3 shows the number of triangles of the isosurface. The time used to build the isosurface is shown in table 4. Figures 17 and 18 shows a plot of these data against the size of the volume.

In the diagrams about the isosurface the lines of *bono* and *cell octree (m. cubes)* are in the same position as the results of both methods are very similar (see tables 3 and 4).

We can see that there is a reduction in the storage requeriment and in the time to build the isosurface whereas the quality of the images is good, figure 20 shows on the left side a zoomed picture by marching cubes and on the right side the same subvolume rendered by marching edges. The time to build the tree is much longer, but this process is done just once. The number of triangles of the isosurface is greater than using marching cubes.

The reduction in the storage requeriment shows how the prunning criteria is less restrictive.

The number of triangles can be reduced by decimation. Schroeder et al. propose deleting points of the triangle mesh and making a retriangulation [Schroeder92]. Montani et al. propose to build the isosurface so that triangles can easily be in the same plane, and then retriangulate them [Montani94]. In our proposal, if the isopoints that are chosen to represents the cells are in the center of the cells, many triangles will be in the same plane too.

6. CONCLUSIONS AND FUTURE WORK

A new method for volume visualization by surface extraction has been proposed. We have presented a method to build the triangles edge by edge instead of cell by cell. This method reduces the number of distinct cases from 15 (marching cubes [Lorensen87]) to 3. Also, it avoids that cracks are arisen when the surface crosses the border between cells of different sizes, neither pre-process nor post-process are needful. Thanks to it, the prunning criteria used in [Velasco01] can be less restrictive. So, more prunes can be done and at a higher level of the tree. As a consequence of this, the storage space requirement of the tree has been reduced.

We are currently working on an improvement to store the properties in the tree avoiding the use of the grid, to obtain further reduction in space used. Also, we are studying particular cases of cells in order to reduce the number of triangles built.

7. ACKNOWLEDGMENTS

This paper has been supported by the *Ministerio de Ciencia y Tecnología* and by *FEDER* through grant TIC2001-2099-C03-02.

8. REFERENCES

[Bajaj96] Bajaj, C., Pascucci, V., and Schikore, D. (1996). Fast isocontouring for improved interactivity. In *ACM Symposium on Volume Visualization*, pages 39–46,99, San Francisco, USA.

[Brodlie01] Brodlie, K. and Wood, J. (2001). Recent advances in volume visualization. *Computers Graphics forum*, 20(2):125–148.

[Bui-Tuong75] Bui-Tuong, P. (1975). Ilumination for computer generated pictures. *CADM*, 18(6):311–317.

[Cignoni97] Cignoni, P., Marino, P., Montani, C., Puppo, E., and Scopigno, R. (1997). Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170.

[Edelsbrunner80] Edelsbrunner, H. (1980). Dynamic data structures for orthogonal intersection queries. Technical Report F59, Inst. Informations-verarb., Tech. Univ. Graz, Graz, Austria.

[Engel99] Engel, K., Westermann, R., and Ertl, T. (1999). Isosurface extraction techniques for web-based volume visualization. In *IEEE Visualization*, pages 139–146.

[Haber90] Haber, R. and McNabb, D. (1990). *Visualization idioms: a conceptual model for scientific visualization systems*, chapter of Visualization in Scientific Computing, pages 74–93. B. Shriver, G.M. Nielson and L.J. Rosenblum (eds).

Volume	Finger	Eye	Head
Size	17 × 32 × 32	64 × 64 × 64	128 × 128 × 128
Bono	24.144	299.600	2.396.752
Cell Octree (m. cubes)	23.248	190.032	1.587.088
Cell Octree (m. edges)	18.775	144.535	1.330.857

Table 1. Storage required by tree (bytes)

Volume	Finger	Eye	Head
Size	17 × 32 × 32	64 × 64 × 64	128 × 128 × 128
Bono	1	8	58
Cell Octree (m. cubes)	4	29	231
Cell Octree (m. edges)	8	64	590

Table 2. Time to build the tree (ms)

Volume	Finger	Eye	Head
Size	17 × 32 × 32	64 × 64 × 64	128 × 128 × 128
Bono	3.586	34.232	376.998
Cell Octree (m. cubes)	3.586	34.226	376.804
Cell Octree (m. edges)	4.828	40.328	463.682

Table 3. Number of triangles of the isosurface

Volume	Finger	Eye	Head
Size	17 × 32 × 32	64 × 64 × 64	128 × 128 × 128
Bono	5	45	513
Cell Octree (m. cubes)	4	45	512
Cell Octree (m. edges)	2	18	236

Table 4. Time to build the isosurface (ms)

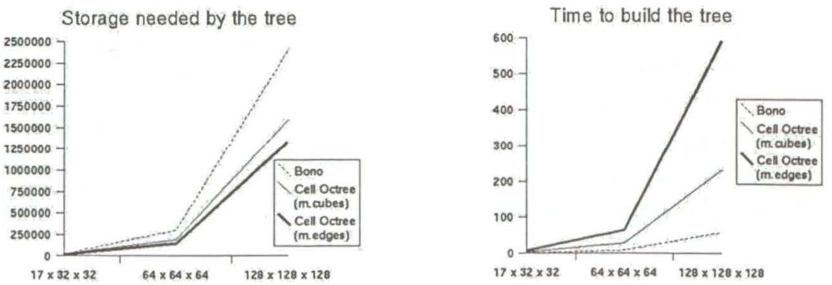

Figure 17. Diagrams about the tree.

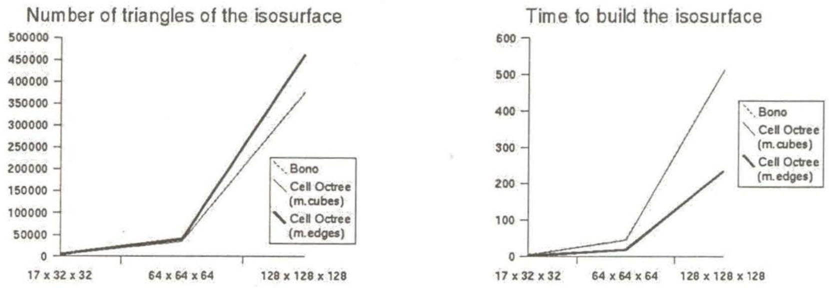

Figure 18. Diagrams about the isosurface.

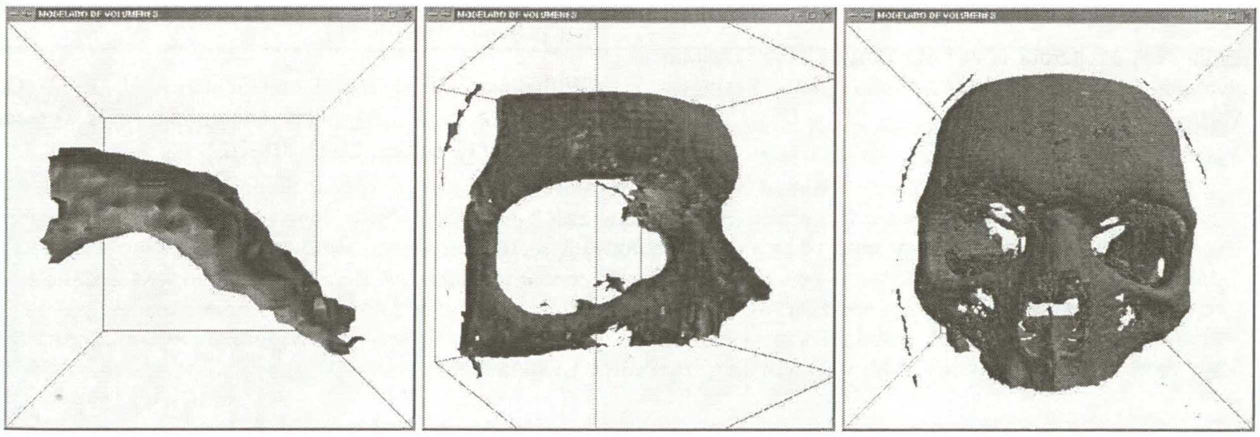

Figure 19. Images from volumes used

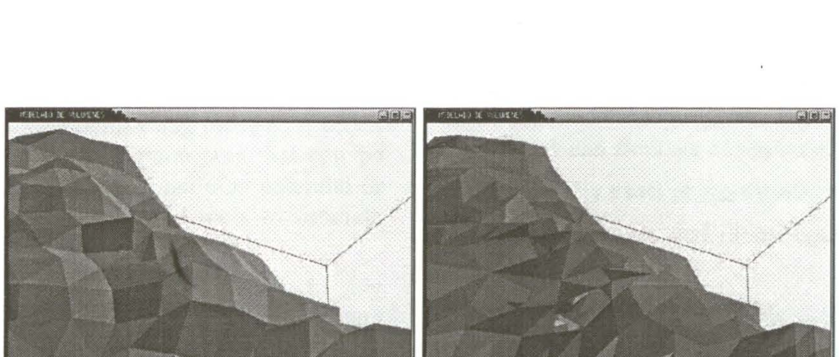

Figure 20. Marching cubes vs. marching edges

[Livnat98] Livnat, Y. and Hansen, C. (1998). View dependent isosurface extraction. In *IEEE Visualization*, pages 175–181.

[Lorensen87] Lorensen, W. and Cline, H. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169.

[Meagher80] Meagher, D. (1980). Octree encoding: a new tecnique for the representation, manipulation and display of arbitrary three dimensional objects by computer. Technical Report IPL-TR-80-111, Polytechnic Inst., Revisselaer.

[Montani94] Montani, C., Scateni, R., and Scopigno, R. (1994). Discretized marching cubes. In *Proceedings of Visualization*, pages 281–287, Los Alamitos, CA, USA.

[Ohlberger97] Ohlberger, M. and Rumpf, M. (1997). Hierarchical and adaptive visualization on nested grids. *Computing*, 59(4):365–385.

[Preparata85] Preparata, F. and Shamos, M. (1985). *Computational Geometry: An Introduction*. Springer-Verlag.

[Schroeder92] Schroeder, W., Zarge, J., and Lorensen, W. (1992). Decimation of triangle meshes. *ACM Computer Graphics*, 25(2):65–70.

[Shekhar96] Shekhar, R., Fayyad, E., Yagel, R., and Cornhill, F. (1996). Octree-based decimation of marching cubes surfaces. In *Visualization'96*, pages 335–342,499, San Francisco, USA. IEEE.

[Shu95] Shu, R., Zhou, C., and Kankanhalli, M. (1995). Adaptive marching cubes. *The Visual Computer*, 11:202–217.

[Velasco01] Velasco, F. and Torres, J. (2001). Cell octree: A new data structure for volume modeling and visualization. In *VI Fall Workshop on Vision, Modeling and Visualization*, pages 151–158, Stuttgart, Germany.

[Westermann99] Westermann, R., Kobbelt, L., and Ertl, T. (1999). Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15:100–111.

[Wilhelms92] Wilhelms, J. and Gelder, A. V. (1992). Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227.