Cells Octree: a new data structure for Volume Modeling and Visualization

Francisco Velasco, Juan Carlos Torres

University of Granada, E.T.S. Ingeniería Informática
Computer Graphics Research Group
Avda. Andalucía, 38, Granada 18071, Spain
Email: {fvelasco,jctorres}@ugr.es

Abstract

When rendering a volume represented as a 3D grid, it takes a long time to check the cells that do not intersect the surface. Wilhelms et al. proposed using an octree as a hierarchical index, keeping the maximum and minimum cell value for each voxel, in order to avoid processing empty cells. Their method used a complete octree which required a large amount of space.
We propose a new indexing method that uses an incomplete octree and which requires less memory to be stored.

1 Introduction

Volumetric models are usually rendered as the boundary surface for a given property threshold, producing one isosurface of the stored property field. The simplest way to render this surface is to process each volume cell independently, checking for cells which contain the boundary. A well-known algorithm using this strategy is the marching cubes algorithm [1]. In order to process a cell, the first step is to compare the threshold value with the property values of its eight vertexes in order to decide whether the cell intersects the boundary surface. If this is the case, a triangle mesh that approximates the isosurface inside the cell (active cell) is built from the orientation of every cell vertex with respect to the boundary surface.

Rendering a surface using marching cubes requires a long time, most of which is taken up processing cells that do not have isosurfaces inside. In order to reduce this time, several methods to search active cells have been proposed, which can be classified according to search criterion in:

- *Range-based*, each cell is identified with the interval it spans in the range of property field, and from the threshold, intervals that contain it are searched. Cignoni et al. propose to use an interval tree structure [2, 3] in order to retrieve these intervals from which active cells are found [4]. 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 drawn are processed [5]. This solution does not visualize 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 renderized. Bajaj et al. propose a method in order to generate seed sets that, for all threshold possible, all components are renderized [6].
- *Space-based*, domain spanned by the dataset is partitioned in order to search active cells. This partition can be hierarchical [7].

In this respect, Wilhelms et al. propose to build an octree [8] that indexes the grid, and which stores in every octree node the maximum and minimum property value of the subvolume covered by the node [7]. This information is used when traversing the volume and allows those nodes for which the property interval does not contain the threshold to be skipped. This method, 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, volume can be rendered at differents levels of detail by pruning the tree at some level when the process of building the isosurface is running. Furthermore, an adaptive isosurface can be built if differents branches of the tree are pruned at differents levels. Works like [9, 10] use these issues, however, the prune condition is dependent on threshold and, in the case of [10], on the point of view too.

Engel et al. use [10] with several improvements in order to implement a web-based volume visualization system [11].

Livnat et al. use a bono in order to do a hierarchical front-to-back traversal of the dataset with dynamic pruning of sections that are hidden from the point of view by previously extracted sections of the isosurface [12]. Only the isosurface that is visible from the point of view is built, when the view point changes, the isosurface must be re-built.

In this paper, we present an improvement of bono approach which uses an incomplete octree with a smaller memory. The method indexes a set of cells as a block when their property value is nearly uniform. This block is rendered as the biggest cell when the threshold value is included in the cell range, thereby avoiding the need to visit every grid cell within the block. The main difference of our proposal whith others related works is that our prune condition is independent on threshold and point of view.

Consequently, this produces a compression of the cells. The structure allows adjacent blocks of different sizes to be rendered (see figure 1). When the surface crosses the border between blocks of different sizes, some discontinuities can arise, since as on one side of the separation face between the two blocks the surface is built with fewer points than on the other side. The proposed method preprocesses the grid data in order to avoid these cracks.

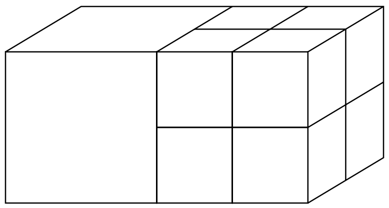
Figure 1: Compressed cell joined to uncompressed cells

The next section shows the bono structure. Section three explains our proposal. Section four discusses how to avoid cracks and finally section five shows data and images from real volumes.

2 Bono

The goal of bono [7] is to accelerate the rendering of the surface by preventing cells which do not have an inside isosurface from being processed.

Every cell can be represented by an entity (noted leaf node) that stores the minimum and maximum property value of the set of eight values of the cell. Every node represents a subvolume. The eight nodes that represent the eight subvolumes which constitute a cube can be represented by another node (noted internal node) that stores the minimum and maximum property value of the subvolume represented. We can say that the internal node created is the *father* of the eight nodes that it represents, and these nodes are its *sons*.

This continues successively until just one node (the noted root node) totally represents the volume. The result is an octal tree where in every node the minimum and maximum property value of the subvolume that it covers is stored.

When we need to render a surface for a given threshold value, we must process the root node. When a node is processed, the threshold value is compared with the minimum and maximum values stored in the node. If the threshold value is not between them, there is no surface inside the subvolume represented by the node; if the threshold value is between the minimum and maximum, the son nodes are processed (or, if the node is a leaf node, the cell that it represents is processed using the marching cubes algorithm).

In this way, the branches of the tree that do not index cells containing the isosurface are not processed. That is to say, the isosurface building time is reduced because empty cells are not processed.

In order to reduce the tree size, Wilhelms does not store the last level of the tree, so each leaf node represents eight cells.

When the number of cells of any dimension of the volume is not a power of two, or is not equal to the number of cells of another dimension of it, nodes will not have all eight sons. In such cases, nodes will have a larger ramification in the levels which are close to the leaf nodes, and a smaller ramification in the levels close to the root node. Consequently, the storage requirement is smaller.

3 Cells octree

Given an enumeration of cells with an index tree like a bono, we propose that cells can be compressed as showed in figure 2. With every compression, some values are lost in the sense that they are not used to build the isosurface inside the compressed cell. Moreover, tree's nodes that represent cells that have been compressed are eliminated from the index. We now have the following types of nodes:

- *Root node:* The tree's root.
- *Internal node:* A node that is the father of other nodes.
- *Eight cells leaf node:* A leaf node that represents a subvolume of eight cells. Cells can be larger than the minimal cubic cell volume (when they are compressed), but the eight must of course be of the same size.
- *One cell leaf node:* A leaf node that represents a subvolume of one cell. Like the eight cells leaf node, these cells can be bigger than the minimal cubic cell volume. There will be nodes of this kind when it is not possible to build an eight cells leaf node. That is to say, the *brothers* of a one cell leaf node are not all of the type one cell leaf node (if all the eight brothers are one cell leaf node, their father would be an eight cells leaf node instead of an internal node and all the eight one cell leaf nodes would not exist).

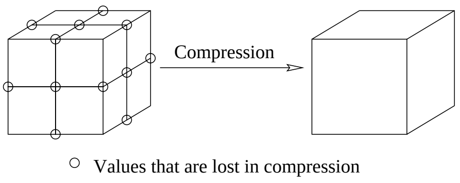

Figure 2: Compression of cells.

We note the *cells octree* to both the enumeration of cells and the compressed octal tree together, see figure 3. The main advantage is that the tree is not complete. The storage requirement is therefore reduced in relation to the bono. In the example of figure 3, if the volume is represented by one bono, then all three levels of the tree will be complete.

Every time a compression is performed, there are cells that are no longer indexed, and so they are not used to build the isosurface. Compressed cells are

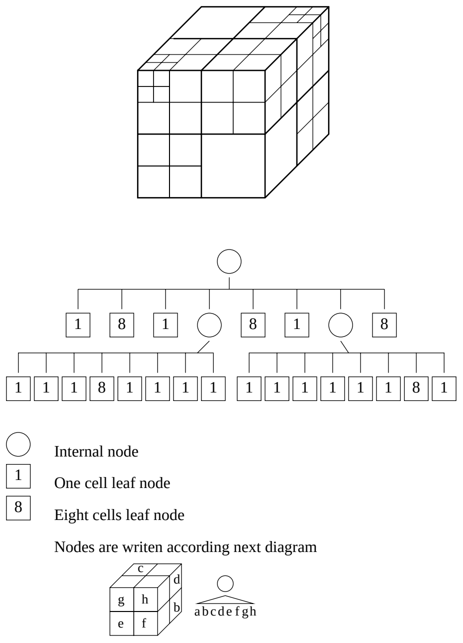

Figure 3: Cells octree example.

used instead, but the isosurface of this subvolume is built using 8 points instead of 27. The compression can only be performed when the information lost is not significant. Specifically, if there is a threshold value such that its isosurface enters and leaves a group of cells through the same side of the group, this group will not be compressed. In figure 4 we show a 2D example of a compression which must not be allowed since the isosurface would present a hole and an area of large curvature would be lost.

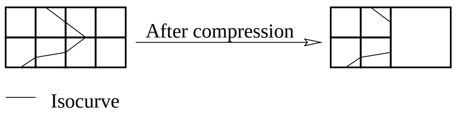

Figure 4: Not allowed compression.

In figure 5 we show a 2D example of an allowed compression where the isosurface has suffered a little modification, it is broken but it can be closed by moving the isosurface inside the small cells until the isosurface is inside the compressed cell.

Let us now define some concepts in order to defin the compression criterion we propose.

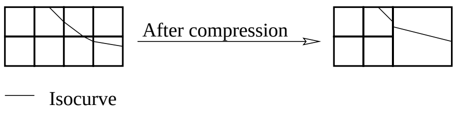

Figure 5: Allowed compression.

When the 8 property values of a cell are such that for all threshold values no one isosurface will enter or leave the cell through the same face or the same edge, we say that this cell is *monotonous*.

When the 27 property values of a group of eight cells are such that for all threshold values no one isosurface will enter or leave the group through the same face or the same edge, we say that this group is monotonous, and it can be compressed. In figure 6, the left cell is monotonous and the right cell is not.

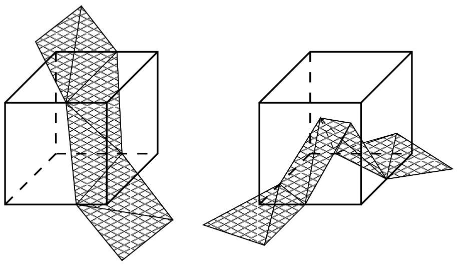

Figure 6: Monotonous and non-monotonous cells.

A cell will be monotonous if all its faces are monotonous, *a face will be monotonous* when for all threshold values, the isosurface crosses it less than twice. This can be tested by visiting the face's vertexes cyclically and testing their property values: if there is only one maximum and one minimum the face is monotonous. If there are two maximums and two minimums, there will be one local maximum and one local minimum, and for all threshold values between the local minimum and the local maximum, the isosurface will cross the face twice. See figure 7.

A group of eight cells will be monotonous when (see figure 8):

1. Every cell of the group is monotonous.
2. The cell after the compression is monotonous.
3. For every B point (half edge point), its property value is between the property values of the corner edge points.

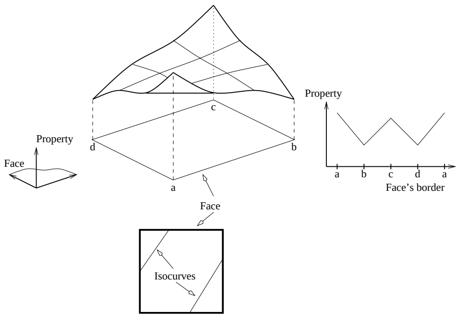

Figure 7: Local minimum and maximum values in a non-monotonous face.

4. For every C point (center face point), its property value is between the property values of the corner face points.
5. For the D point (center cube point), its property value is between the property values of the corner cube points.

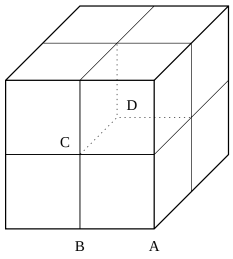

Figure 8: Group of cells to be compressed.

We define the **compression criterion** in the following way: *A group of cells will be compressed when the group is monotonous*.

With every compression, an aproximation error is produced but this is not important. This is because the compression criterion is very restrictive and only a compression is done when property chages uniformly. In section 5 we show data about it.

4 Cracks

With compression, large cells can be joined to small ones. This can cause holes (or cracks) on the isosurface, see figures 5 and 9. This is because the isosurface inside the large cell (a more compressed cell) has been built from less information than the isosurface inside the small cells (less compressed cells). In the join plane between the large cell and the small cells, the isocurve on one side of the plane has been built from 4 values, and on the other side, the isocurve has been built from 9 values.

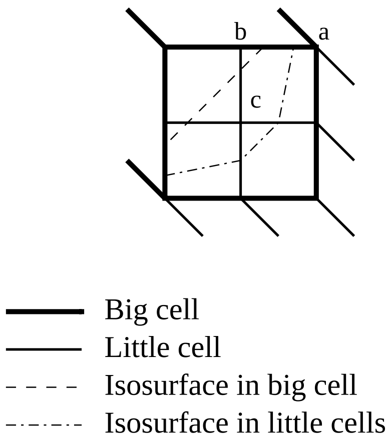

Figure 9: Crack in the join plane between cells of different size

Shu et al. propose covering the crack with a polygon [13]; Shekhar et al. propose covering it by moving the vertex of the triangle in the small cells so that they coincide with the edges of the triangle in the big cells [5]; Westermann et al. propose replace a triangle in the large cell for a fan of triangles in order to adapt the isosurface inside the large cell to isosurface inside the small cells [10]. In these cases, the process is performed after the triangle mesh has been built or when it is been built.

We propose that the property value of some of the volume points is modified, so that when the isosurface is built inside the small cells, this is joined to the isosurface inside the large cell. The process is carried out before the triangle mesh is built.

These modifications produce an aproximation error in the isosurfaces inside the small cells, but it is the same error that happen in the big cell, we have showed that this error is not significant because compression criterion is very restrictive.

Process is done by traversing the tree in a breadth-first manner in order to visit the large cells before the small cells. For every cell visited, we test to see if this cell is joined to smaller cells. If this is the case, the faces that are joined to the small cell are processed.

This process consists in modifying the values of the type b and c vertexes (see figure 9) in order to obtain the same isocurve in the small cells as in the large cell. Figure 10 shows how the process works. The triangles of the small cells have been built to be joined to the triangle of the large cell.

Type b values can be calculated independently of the threshold, but type c values cannot. Moreover, there are cases where the type c point needs two different values such as in figure 11. The value of the c point must be such as one to make the $c1$ points in both the big and small cells to coincide; and it must be another different one to make the $c2$ points to coincide in both the big and the small cells.

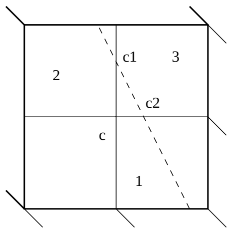

Figure 11: Central point that needs two values.

In order to avoid this, the compression criterion is modified so that if the property values of a group are such that it is possible that two different values are needed for a central point, then compression is not performed.

5 Implementation and results

Both bono and our proposal have been implemented and data about the storage requirements of the tree (table 1), about the time of building the tree (table 2), about the number of triangles of the isosurface (table 3) and about the time of building the isosurface (table 4) have been obtained (see diagrams of tables in figures 12 and 13). The volumes used are showed in figure 14. The volumes have been modeled from TAC, the data have been normalized between 0 and 255, and the threshold is 60.

We can see that there has been a reduction in the storage requirement whereas the quality of the images is good; for example in the model of head, 144082 compressions have been done and the average deviation of the new isosurface is 0.14 units with a maximun of 1 unit. The time of building the

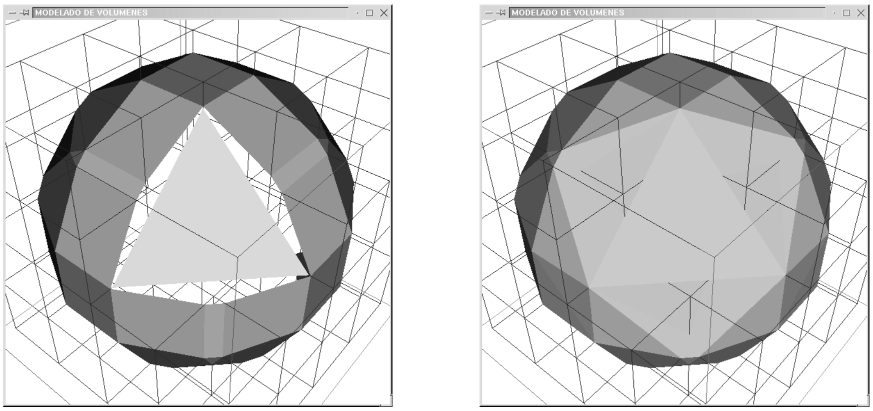

Figure 10: Volume with cracks and without cracks.

Volume	Finger	Eye	Head
Size	$17 \times 32 \times 32$	$64 \times 64 \times 64$	$128 \times 128 \times 128$
Bono	24.144	299.600	2.396.752
OctCell	23.248	190.032	1.587.088

Table 1: Storage required by tree (bytes)

Volume	Finger	Eye	Head
Size	$17 \times 32 \times 32$	$64 \times 64 \times 64$	$128 \times 128 \times 128$
Bono	1	8	58
OctCell	4	29	231

Table 2: Time of building the tree (ms)

Volume	Finger	Eye	Head
Size	$17 \times 32 \times 32$	$64 \times 64 \times 64$	$128 \times 128 \times 128$
Bono	3.586	34.232	376.998
OctCell	3.586	34.226	376.804

Table 3: Number of triangles of isosurface

Volume	Finger	Eye	Head
Size	$17 \times 32 \times 32$	$64 \times 64 \times 64$	$128 \times 128 \times 128$
Bono	5	45	513
OctCell	4	45	512

Table 4: Time of building the isosurface (ms)

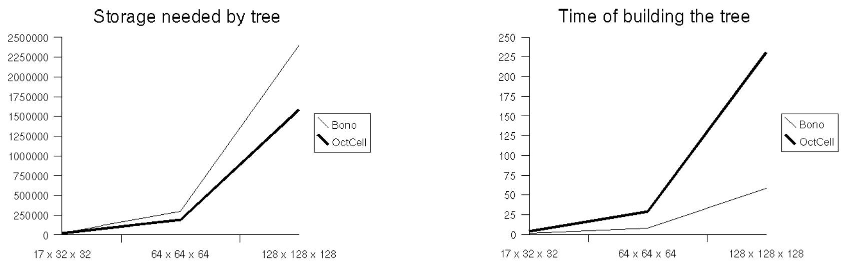

Figure 12: Diagrams about the tree.

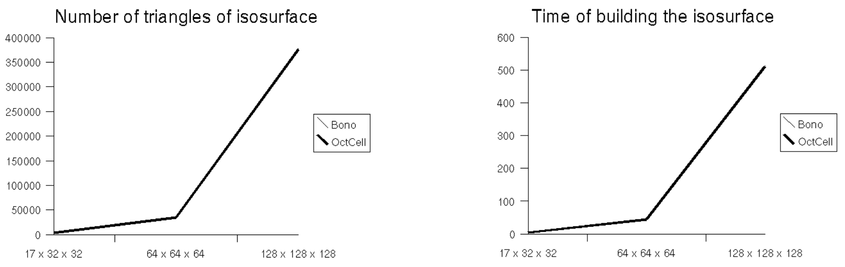

Figure 13: Diagrams about the isosurface.

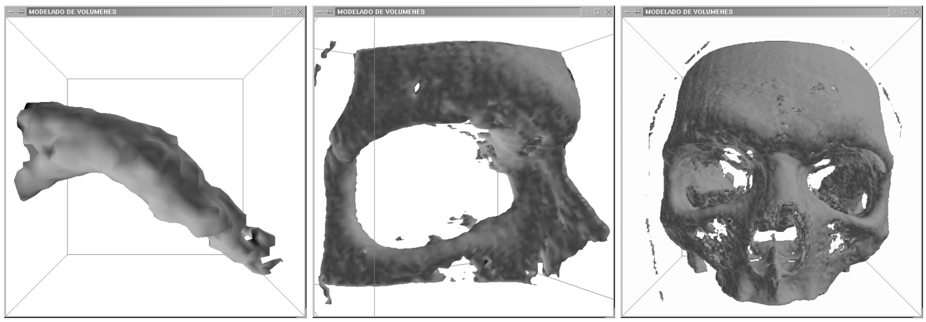

Figure 14: Images from volumes used.

tree is much longer, but it is a process that is just done once. The number of triangles of the isosurface and the time for building it is very similar; this is because the isosurface built is nearly the same in both bono and octcell.

The number of triangles can be reduced in order to reduce the rendering time. In order to do so, Schroeder et al. propose a way to reduce the triangles by deleting points of the mesh of the triangle and making a retriangulation [14]. Montani et al. propose that the isosurface be built in such a way that triangles can easily be in the same plane, then this polygon can be retriangulated [15].

6 Conclusions and future work

An improvement of bono, noted as *cells octree*, has been showed. This improvement is achieved by cell compression, so less storage is required. We have presented tables and diagrams. The quality of the images obtained using our method is good for two reasons: first, the compression criteria is such that there has been little modification in the isosurface, and secondly, we have proposed and implemented a way to fill cracks.

We are currently working on an improvement so that property values may be stored in the tree, so it is not necessary to store the grid and there will be further reductions in the storage requirements.

7 Acknowledgment

We would like to thank the reviewers for their helpful comments.

This paper has been supported by the comisión interministerial para la ciencia y la tecnología through grant TIC98-0973-C03-01.

References

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

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

[3] F. Preparata; M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

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

[5] R. Shekhar; E. Fayyad; R. Yagel; F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Visualization'96*, pages 335–342,499. IEEE, 1996.

[6] C.L. Bajaj; V. Pascucci; D.R. Schikore. Fast isocontouring for improved interactivity. In *ACM Symposium on Volume Visualization*, pages 39–46,99, 1996.

[7] J. Wilhelms; A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions of Graphics*, 11(3):201–227, July 1992.

[8] D. Meagher. 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, 1980.

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

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

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

[12] Y. Livnat; C. Hansen. View dependent isosurface extraction. In *IEEE Visualization*, pages 175–181, 1998.

[13] R. Shu; C. Zhou; M.S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11:202–217, 1995.

[14] W.J. Schroeder; J.A. Zarge; W.E. Lorensen. Decimation of triangle meshes. *ACM Computer Graphics*, 25(2):65–70, July 1992.

[15] C. Montani; R. Scateni; R. Scopigno. Discretized marching cubes. In *Proceedings of Visualization*, pages 281–287, 1994.