Tetra-Cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra

BERNARDO PIQUET CARNEIRO‡
CLÁUDIO T. SILVA* †
ARIE E. KAUFMAN*

IMPA–Inst. de Matemática Pura e Aplicada‡
Estrada Dona Castorina, 110
22460 Rio de Janeiro, RJ, Brasil
piquet@fluid.impa.br

Department of Computer Science*
Department of Applied Mathematics & Statistics†
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{csilva,ari}@cs.sunysb.edu

Abstract. We present the *tetra-cubes* iso-surface extraction technique. *Tetra-cubes* is similar to the well-know *marching-cubes* technique, but its basic building block consists of tetrahedral cells instead of cubes. When the original cells are not tetrahedral cells, tetra-cubes converts them to tetrahedra. We show that our technique is simpler than marching-cubes and does not have ambiguity configurations, leading to simpler iso-surface generation. We advocate the use of tetra-cubes due to its simplicity of implementation and generality.

Keywords

Isosurfaces, Marching-cubes, Tetrahedral cells, Scalar fields.

1 Introduction

Many applications in industrial and scientific areas such as computed tomography (CT) and magnetic resonance (MR) in the medical field, and numerical simulations in the field of petroleum extraction in deep sites, generate 3D scalar fields. These data must be presented to the scientists in a suitable visual form in order to facilitate analysis, interpretatioan, and exchange of information. Visualization of 3D scalar fields is a necessary step even during debugging large scientific applications. There are several methods for the visualization of 3D scalar fields. The two most common are direct volume rendering and 3D iso-surface extraction (see [3] for an overview of techniques). In this paper we study iso-surface extraction techniques.

The isosurface extraction problem consists of generating a polygonal representation for the implicit surface $f(x, y, z) = \lambda$, where $\lambda \in \mathcal{R}$ is the value of interest in the scalar function $f : \mathcal{R}^3 \rightarrow \mathcal{R}$, which is usually given by discrete samples. There are several methods to generate isosurfaces [4, 5, 6, 7], a popular one being the marching-cubes method [4]. Isosurfaces have a clear advantage over direct volume rendering when it comes to interactivity. Once the models have been polygonized (and simplified [9]; marching-cubes usually generates a large number of triangles.), a hardware supported graphics workstation can be used to speed up the rendering. Isosurfaces also have some disadvantages, such as lack of fine detail and flexibility during rendering (especially for handling multiple transparent surfaces), and its binary decision process where surfaces are either inside or outside a given voxel tends to create artifacts in the data (there is also an *ambiguity* problem in the marching-cubes technique, which has been addressed by later papers such as [7]).

Our main reason for using 3D iso-surface techniques in our work is its generality as opposed to direct volume rendering. Volumetric data comes in a variety of formats, the most common being cartesian or regular data (we are using the taxonomy introduced in [11]). Cartesian data is typically a 3D matrix composed of voxels (a *voxel* can be defined in two different ways, either as the datum in the intersection of each three coordinate planes, or as the small cube; either definition is correct as long as used consistently), while the regular data has the same representation but can also have a scaling matrix associated with it. Irregular data is represented in several different

ways, including curvilinear data, that is, data defined on a *warped* regular grid, or in general, one can be given scattered (or unstructured) data, where no explicit connectivity is defined. In general, scattered data can be composed of tetrahedra, hexahedra, prisms, etc. Our main interest in this work is in tetrahedral grids. They have several advantages, including easy interpolation, simple representation (especially for connectivity information), and the fact that any other grid can be interpolated to a tetrahedral one (with the possible introduction of Steiner points [8]). Among their disadvantages is the fact that the size of the datasets tends to grow as cells are decomposed into tetrahedra. In the case of curvilinear grids, for instance, an accurate decomposition could make the cell complex contain even six times as many cells (see [1] for six tetrahedra decomposition) or five times as many cells as in our approach.

Our approach consists of a single iso-surface extraction technique, that uses tetrahedral cells as its building block. For any other type of grid, a *converter* is written to take that grid format into a tetrahedral complex, which can be used as input to our algorithm.

In the next sections we will introduce a very easily implementable alternative to the marching-cubes algorithm called the *tetra-cubes* algorithm. The tetra-cubes algorithm explores the correlation among hexahedra and tetrahedra to solve ambiguities found in the marching-cubes algorithm and provides a very general and fast way of creating isosurfaces.

2 Tetra-Cubes Algorithm

Following the idea in the marching-cubes algorithm, the data is represented in slices where logical cubes are created from eight adjacent pixels; each four from two adjacent planes.

The next step, consists of extracting tetrahedra from the logical existent cubes (hexahedra), since the algorithm uses the correlation among hexahedra and tetrahedra. Each cube can be divided into a collection of five tetrahedra, where the vertices of the tetrahedra coincide with those of the original cubes.

Once the tetrahedra are created, the algorithm intersects the surface with each tetrahedron in one group of five tetrahedra and then moves on to the next collection of tetrahedra, which was extracted from an adjacent hexahedron. As in the marching-cube, in order to find the intersection of a surface with a tetrahedron, we assign a "1" to a tetrahedron's vertex if the data value at the vertex exceeds or equals the value of the isosurface we are constructing, which represents the vertices that are inside or on the surface. The vertices of the tetrahedra with values below the surface are assigned a "zero" value and are considered outside the surface. Thus, the edges of the tetrahedron which have a vertex inside the surface and the other outside are intersected by the surface. This fact allows us to find the topology of the surface within the tetrahedron in a binary manner. Since there are four vertices in each tetrahedron, and two possible values for each of them, there are exactly $2^4 = 16$ ways a tetrahedron can be intersected by a surface, including the two cases where a tetrahedron is completely inside or outside the surface, thus, no edge is intersected.

However, using symmetry one is able to narrow down these cases to only three (see Figure 1).

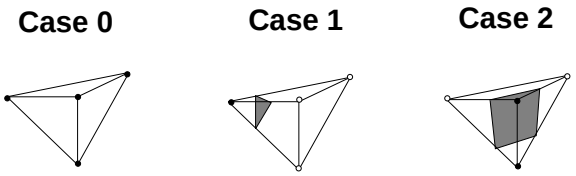


Figure 1: *Reduced possible tetrahedron's intersections.*

With these three cases, we create a table to look up surface-edge intersections, where the table contains the edges intersected for each case. Still following the marching-cubes idea, we create an index for each case, based on the state of the vertex. Thus, each vertex is represented by a binary position in this index, which describes the case we are treating.

This index is used as a pointer to an edge table that gives all edge intersections for a given tetrahedron configuration. Using the index to find which edge the surface intersects, we use linear interpolation to place the intersection point along the edge. Finally, once the interpolation is done for all the tetrahedron edges, the triangle contained in the surface is created connecting all the intersection points within this cell. The same task is repeated, cell by cell, generating the complete isosurface.

3 Grid Connection

Due to the implicit symmetry of the new basic cells used, the tetrahedra, a problem with the connection among the collections of five tetrahedra extracted from neighboring hexahedra arises. This problem arises from the fact that adjacent tetrahedra in different adjacent hexahedra must share identical faces with identical edges and vertices. This problem does not occur if the same collection of tetrahedra is extracted from adjacent cubes (Figure 2).

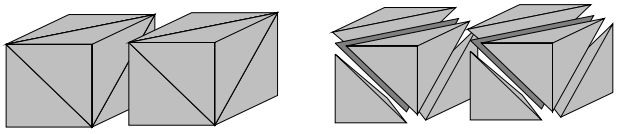


Figure 2: *Erroneous Connection among adjacent cubes.*

In order to avoid this erroneous situation, we have

created a new collection of tetrahedra to use in adjacent cubes. This new collection is obtained by a ninety degrees rotation of the previous one. In other words, we have implemented a"zig-zag", or a "chess-table configuration" alternating the collections of tetrahedra in the three axes directions to avoid this problem (Figure 3).

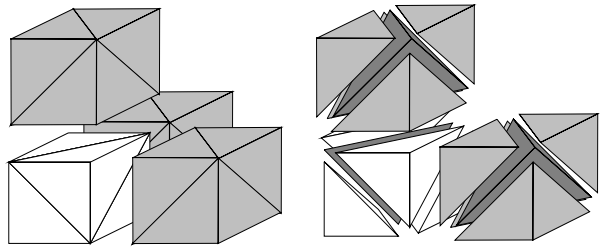


Figure 3: *Zig-zag connection.*

4 No Ambiguities

The binary approach used to distinguish the cases leads to some ambiguities in the marching-cubes implementation. This does not occur in the implementation of the tetra-cubes algorithm due to the geometrical differences between the hexahedron and the tetrahedron, as we can see by comparison in Figure 4, which depicts the same ambiguity case in the two different basic cells.

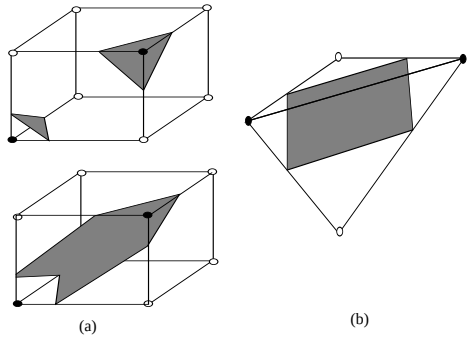


Figure 4: *(a) Example of two possible surfaces within a cube, based upon the same two "black" vertices, which defines an ambiguity. (b) The related case described on figure (a) using tetrahedra as base cell. No ambiguity occurs, only one surface is possible.*

Since no ambiguity is found using tetrahedra as basic cells, a fast implementation for the marching-cubes algorithm may be accomplished combining the marching-cubes and the tetra-cubes algorithms. Cubes would be used as basic cells, keeping track of the "chess-color" of the cubes. Once an ambiguity is found, the tetra-cubes algorithm could be used to intersect the surface with this specific problematic cube, after dividing it into the right collection of five tetrahedra. The marching-cube algorithm would be applied to all other non-problematic hexahedra. Using this idea, the tetra-cube algorithm provides a consistent and elegant rule for resolving the ambiguities present in the marching-cube algorithm. (It is important to note that no ambiguity exists if we consider the linear interpolation as calculated with respect to the tetrahedra. See [12] for a technique that handles the case where the interpolation is performed with respect to the original cubes.)

Irregular Grid Approach: Unlike the marching-cube approach that receives as data only the density values, implicitly assuming a regular grid of cubes, our approach not only receives the density values for each pixel, but also the 3D coordinates of each pixel. This approach does not assume a regular grid; in fact, it creates a grid of irregular tetrahedra, since the coordinates read from the data may be completely irregular. Thus, the tetra-cubes algorithm receives as data the coordinates of each pixel, which may be completely irregular, and it is still able to generate the irregular tetrahedron grid. These irregular coordinates are then used by the algorithm as the coordinates of the irregular tetrahedron's vertices and, based on these vertices, the tetra-cubes algorithm generates the high resolution 3D isosurface of the irregular grid.

5 Implementation Details

5.1 Look up Tables

The first implementation aspect to be analyzed is the look up tables.

Pattern Tables: This table establishes the relation between the value of the binary index found using the vertices' status and the pattern in the three possible symmetric cases. The table is implemented as a one-dimension array.

Edge Tables: These tables contain two indices for the array of vertices. These two vertices are the end vertices for the edge being intersected by the isosurface. There is one edge table for each tetrahedron configuration. Since within each tetrahedron we have seven edges and each edge has two end vertices, these tables are 7 x 2 matrices.

Case Tables: These tables contain the intersected edges for each of the three symmetric cases. They are implemented as two dimensional matrices whose size varies with the number of cases handled in each table. The implementation also uses auxiliary tables called "case_index", which are just pointers to the right index in the "cases_edges" tables. The usage of the tables is explained in the next section.

5.2 Code

The code is structured and divided in four major parts:

Data input: These are the functions related with the reading of the input files (data), creation of data struc-

tures, user interface and creation of the output file.

Extraction of tetrahedra: For each logical cube (hexahedra), the cube's vertices are used as the tetrahedron's vertices. In this way, each group of four different vertices in the cube is used to represent one tetrahedron. In Figure 5, for instance, vertices V1, V2, V4 and V5 represent one tetrahedron.

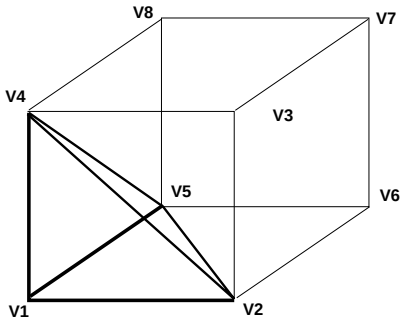


Figure 5: *Relation cube (hexahedron) - tetrahedron.*

Accessing the tables: Once the tetrahedra are extracted, the density value on each vertex is used to generate the binary index for each tetrahedron; this index is used as the index in the "pattern_table". This table then will map one of the three symmetric cases. Knowing exactly which case we are treating now, we use the "cases_index" tables to find the index of the current case in the "cases_edges" tables. The "cases_edges" tables map the edges intersected by the isosurface and finally, the informations provided by the "edges_tables" are retrieved to find the edges' vertices which will be used in the linear interpolation, in order to find the exact point of intersection along the edge.

Triangles creation: At the end, for each tetrahedron, the triangle contained in the isosurface is created and sent to the output. Then, the isosurface can be visualized using the triangle model created for the specific function/density value.

6 Results

We have tried to run tetra-cubes on a variety of inputs, including both regular and irregular grids. See Figures 6, 7 and 8 for images of iso-surfaces.

The table below shows the clock time taken to generate the isosurface models for each of the results shown in this paper. They were measured for the function density value 128 using an SGI Indigo^2 machine (Table 1).

Table 1: Clock time to generate the isosurfaces using tetra-cubes on a SGI Indigo 2 machine.

Dataset	# tetrahedra	# polygons	Time(sec)
Sphere	1,642,545	73,580	151
Bulb	32,400	7,258	12
Terrain	87,120	15,228	26
House	2,744,280	201,052	564
Blunt	187,395	5,297	9

7 Conclusion and Future Work

It is interesting to try comparing our method with other approaches published in the literature. Relative to Marching-Cubes: (1) our method is simpler to implement due to the smaller number of cases; (2) it does not have the ambiguity problem, and always generates correct C^0 functions representations. Recently, Zhou et al [12] showed that the use of tetrahedra still does not solve ambiquities, but in their paper they assume the trilinear interpolation is performed on the cube, while the isosurface sampling is performed on the tetrahedra. In our method we perform the interpolation and sampling in the tetrahedra, thus avoiding this problem.

One possible shortcoming of our approach is the large number of triangles generated. Fortunately, this can be solved with simplification techniques, such as decimation [9]. By running a topology preserving simplification algorithm on the output of tetra-cubes, we can remove the *redundant* triangles. This is also necessary when using other isosurface generation techniques, such as marching-cubes.

In summary, we have presented an extension of the marching-cubes algorithm, which works with tetrahedra-based grids. The tetra-cubes algorithm may be applied to irregular grids, as we have shown in the case of a curvilinear grid. It can also be used as a consistent and elegant way of solving ambiguities in the marching-cube algorithm.

For future work, we would like to explore further optimizations of our algorithm. Possibly we could try to incorporate the technique presented in [10] into our code. Another area of interest is the generation of *fat isosurfaces*, defined over an interval, and not just a single value (e.g., [2]). Finally, because decimation is expensive, it would be interesting to add some form of adaptive generation of triangles (such as [6] did for the case of marching-cubes) directly to our method.

Acknowledgments

Bernardo Piquet was supported by RHAE-Brazil during his MS work at Stony Brook. C. Silva is partially supported by CNPq-Brazil under a Ph.D. fellowship, by Sandia National Labs, and by the National Science Foundation (NSF), grant CDA-9626370. A. Kaufman is partially supported by NSF (CCR-9205047, DCA 9303181

and MPI-9527694) and by the Dept. of Energy under the PICS grant. We would like to thank the members of the Visualization Lab at Stony Brook for the support given to this work.

References

[1] Eugene Allgower and Kurt Georg. Simplicial and continuation methods for approximating fixed points and solutions to systems of equations. *SIAM*, 22(1):151–158, 1980.

[2] Issei Fujishiro, Yuji Maeda, and Hiroshi Sato. Interval volume: A solid fitting technique for volumetric data display and analysis. In *IEEE Visualization '95*, pages 151–158. IEEE CS Press, 1995.

[3] Arie E. Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1990.

[4] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[5] James V. Miller, David E. Breen, William E. Lorensen, Robert M. O'Bara, and Michael J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 217–226, July 1991.

[6] Heinrich Muller and Michael Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, January 1993.

[7] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91, 1991.

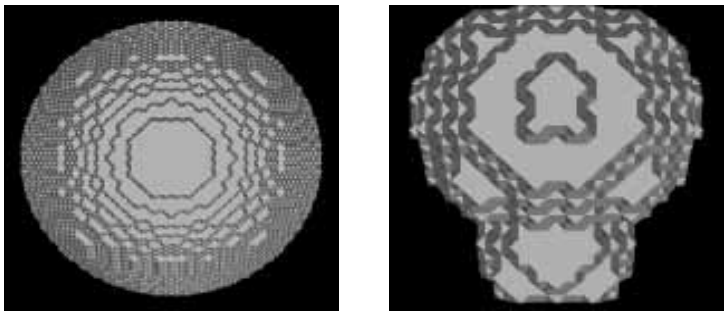
[8] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[9] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[10] Han-Wei Shen and Christopher Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization '95*, pages 143–150. IEEE CS Press, 1995.

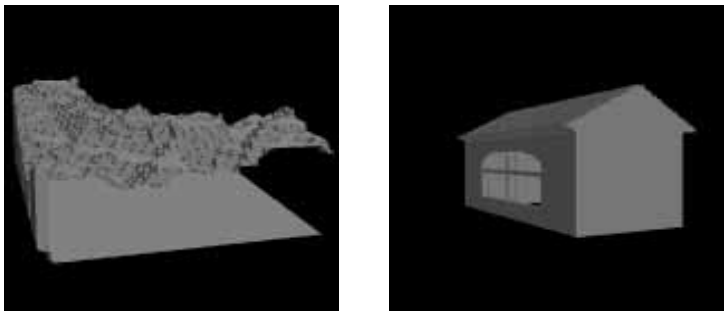
[11] Don Speray and Steve Kennon. Volume probes: Interactive data exploration on arbitrary grids. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 5–12, November 1990.

[12] Yong Zhou, Weihai Chen, and Zesheng Tang. An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. *Computer & Graphics*, 19(2):355–364, 1995.



(a) (b)

Figure 6: *(a) 3D Sphere 70 X 70 X 70 grid. (b) 3D Bulb Model 19 X 21 X 19.*



(a) (b)

Figure 7: *(a) 3D Terrain Model 34 X 17 X 34. (b) 3D House Model 127 X 67 X 67.*

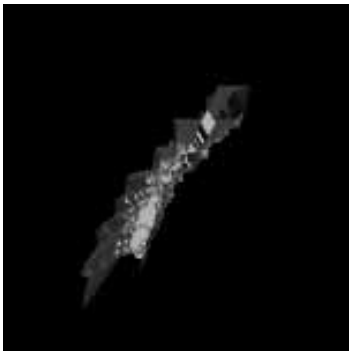


Figure 8: *3D Blunt Model 40 X 32 X 32 - Curvilinear Grid.*