Computer Physics Communications 184 (2013) 777–782

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

# An improved Marching Cube algorithm for 3D data segmentation☆

G.L. Masala [a,*], B. Golosio [b], P. Oliva [b]

[a] University of Sassari, via Piandanna n. 4, 07100, Sassari, Italy

[b] University of Sassari and INFN Sez. di Cagliari, Italy

ARTICLE INFO

*Article history:*
Received 27 January 2012
Received in revised form
22 August 2012
Accepted 25 September 2012
Available online 29 September 2012

*Keywords:*
3D imaging
Surface triangulation

ABSTRACT

The marching cube algorithm is one of the most popular algorithms for isosurface triangulation. It is based on a division of the data volume into elementary cubes, followed by a standard triangulation inside each cube. In the original formulation, the marching cube algorithm is based on 15 basic triangulations and a total of 256 elementary triangulations are obtained from the basic ones by rotation, reflection, conjugation, and combinations of these operations.

The original formulation of the algorithm suffers from well-known problems of connectivity among triangles of adjacent cubes, which has been solved in various ways. We developed a variant of the marching cube algorithm that makes use of 21 basic triangulations. Triangles of adjacent cubes are always well connected in this approach. The output of the code is a triangulated model of the isosurface in raw format or in VRML (Virtual Reality Modelling Language) format.

**Program summary**

*Program title:* TRIANGOLATE

*Catalogue identifier:* AENS_v1_0

*Program summary URL:* http://cpc.cs.qub.ac.uk/summaries/AENS_v1_0.html

*Program obtainable from:* CPC Program Library, Queen's University, Belfast, N. Ireland

*Licensing provisions:* Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html

*No. of lines in distributed program, including test data, etc.:* 147558

*No. of bytes in distributed program, including test data, etc.:* 26084066

*Distribution format:* tar.gz

*Programming language:* C.

*Computer:* Pentium 4, CPU 3.2 GHz and 3.24 GB of RAM (2.77 GHz).

*Operating system:* Tested on several Linux distribution, but generally works in all Linux-like platforms.

*RAM:* Approximately 2 MB

*Classification:* 6.5.

*Nature of problem:* Given a scalar field $\mu(x, y, z)$ sampled on a 3D regular grid, build a discrete model of the isosurface associated to the isovalue $\mu_{Iso}$, which is defined as the set of points that satisfy the equation $\mu(x, y, z) = \mu_{Iso}$.

*Solution method:* The proposed solution is an improvement of the Marching Cube algorithm, which approximates the isosurface using a set of triangular facets. The data volume is divided into logical volumes where the topology of the triangulation is selected through a look-up table, while the metric is computed by linear interpolation.

*Running time:* It is dependent on the input data, but the test provided takes 8 seconds.



## 1. Introduction

Surface triangulation algorithms are important tools in several fields of science and technology [1], ranging from surface modeling to three-dimensional data visualization such as boundary segmentation and surface modeling in a scalar field representation.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author. Tel.: +39 079229486.
*E-mail addresses:* gilmasala@uniss.it (G.L. Masala), golosio@uniss.it (B. Golosio), oliva@uniss.it (P. Oliva).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.09.030

Commonly, triangulation algorithms are applied to volumetric data. Scalar volumetric data can be defined as a function $\mu(P_i)$ that associates a scalar value $\mu$, or some other property, to a collection of three-dimensional points $P_i = (x_i, y_i, z_i)$ typically arranged on a regular 3D grid [2]. Each node of such a grid is called a voxel. A voxel (volumetric pixel) is a volume element, and a scalar value (voxel content) is associated to it.

Marching cubes (MC) is a computer graphics algorithm, described in 1987 by Lorensen and Cline [3], for extracting a polygonal mesh of an isosurface from a three-dimensional scalar field. It combines simplicity with high speed because it works almost entirely on look-up tables.

There are many applications for this type of technique; two very common ones are as follows:

- Reconstruction of a surface from medical and biological volumetric datasets. For example, CT scans result in a 3D volume of samples at the nodes of a regular 3D grid [4]. An important application of isosurface triangulation algorithms is Lung CT segmentation. In Ref [5], the improved version of the MC algorithm described in this work was used for modeling the pleural surface of the chest wall.
- Creating a 3D contour of a mathematical scalar field. In this case the function is known everywhere but is sampled at the nodes of a regular 3D grid [6].

The original MC algorithm has connectivity problems between triangles of adjacent cubes. These problems have been solved using different approaches in the literature.

All methods based on MC use a simple or complex look-up table to triangulate the isosurface; use of a modified look-up table is an attractive strategy because it can be both fast and accurate [1,7]. Some authors [7–12] developed an extended look-up table. Some authors have used a richer collection of maps including mirroring maps and complementation (with respect to being above or below the isosurface threshold); in [9,10] they use only rotation maps. Heiden et al. [8] developed an extended look-up table strategy that is based on their finding that for some cases of the standard MC, reflection need not be used because the reflective-symmetric cases for these can be alternatively expressed by exploiting rotation. Additionally, Montani and Scateni [11] use a modified table strategy to resolve face ambiguity; they use one additional facetization pattern for the reflectively-symmetric instances of the six scenarios that exhibit face ambiguity.

Other authors [13] propose a triangulation strategy without using a look-up table and performing complementary and rotational symmetry operations; the look-up table scheme is substituted by an automatic triangulation based on critical points located on the characteristic positions of the isosurface. Another approach [14] generates the resulting triangles in every possible cube configuration without resorting to any predefined cases: the method orders the isosurface points directly in polygons rather than triangles; producing less triangles. Some other authors [15] instead work using tetrahedral decomposition: a decomposition of each 8-cell associated with a voxel into five tetrahedra. In such a method, the entire surface is wrapped by a collection of triangles that form a graph structure in which each triangle is contained within a single tetrahedron.

One of the key shortcomings of this approach is the quality of the resulting meshes, which tend to have many poorly shaped and degenerate triangles. This issue is often addressed through post-processing operations such as smoothing. In Ref. [16] the authors present a possible solution to the connectivity problem and demonstrate it in experiments with several data sets. The algorithm, called Macet, combines the MC algorithm with edge transformations to generate an output mesh with the same connectivity and small error. They are based on the observation that MC cells generate well-shaped triangles in many, but not all, of the possible intersections with a planar isosurface. By modifying the grid before the mesh generation, the triangle quality can be significantly improved. However, although their method improves the mesh, it does not remove all degeneracies and results in an increased and unbounded error between the resulting mesh and the original isosurface.

In this paper, we present a new version of the algorithm that solves the problems of connectivity in a simple and efficient way based on an extended modified look-up table. The implementation of this algorithm is complemented by a code to convert the raw output to the VRML(Virtual Reality Modelling Language) format.

## 2. The standard MC algorithm

The MC algorithm is an isosurface triangulation algorithm. Given a scalar value $\mu_{Iso}$ (isovalue), the corresponding isosurface is defined as the set of points $(x, y, z)$ that satisfy the equation $\mu(x, y, z) = \mu_{Iso}$. Basically, the isosurface divides the data volume into two regions, one with $\mu > \mu_{Iso}$ inside the isosurface itself and the other with $\mu < \mu_{Iso}$ outside. An isosurface triangulation algorithm builds a discrete model of the isosurface, approximated by a collection of triangular facets.

The original version of the MC algorithm [3] begins by creating a triangular mesh that will approximate the isosurface and then calculates the surface normals at each triangle vertex.

Marching cubes uses a divide-and-conquer approach. The volume data grid is divided in logical cubes, each created from eight neighboring voxels, as shown in Fig. 1. The triangulation operates on two slices at time. The content of the voxels at the eight vertexes of the logical cube is compared against the isovalue $\mu_{Iso}$ and a binary number (zero or one, respectively, if the voxel content is below or above the isovalue $\mu_{Iso}$) is associated with each vertex.

The voxels contained in the volume inside the isosurface have $\mu > \mu_{Iso}$, and therefore they are associated with the binary number 1, while the voxels outside are associated with the binary number 0. The basic idea of the MC algorithm is that the type of triangulation (topology) depends only on this binary number, while the metric (i.e., the exact position of the triangle vertexes) depends on the actual voxel contents.

The surface normals are computed from the scalar field gradient, which is first evaluated in each voxel by the finite difference method.

Because this method uses the content of the six nearest neighbors to each voxel, the actual implementation of the algorithm requires that four slices are loaded in memory at a time for gradient calculation.

The type of triangulation that must be performed inside each logical cube depends only on an eight bit binary number. Therefore, the total number of elementary triangulations is $2^8 = 256$. The authors [3] use three types of symmetries to reduce the magnitude of the problem from 256 cases to 15 basic patterns:

- Conjugation: the binary numbers associated with the logical cube vertexes are swapped from zero to one.
- Reflections
- Rotations.

The simplest pattern, 0, occurs if all vertex values are above (or below) the selected value and produces no triangles. The next pattern, 1, occurs if the surface separates one vertex from the other seven.

It can be observed from Fig. 1 that the triangle vertexes always lie on the logical cube edges. The exact position of the triangle vertexes on the edges is computed by a linear interpolation between voxel contents.

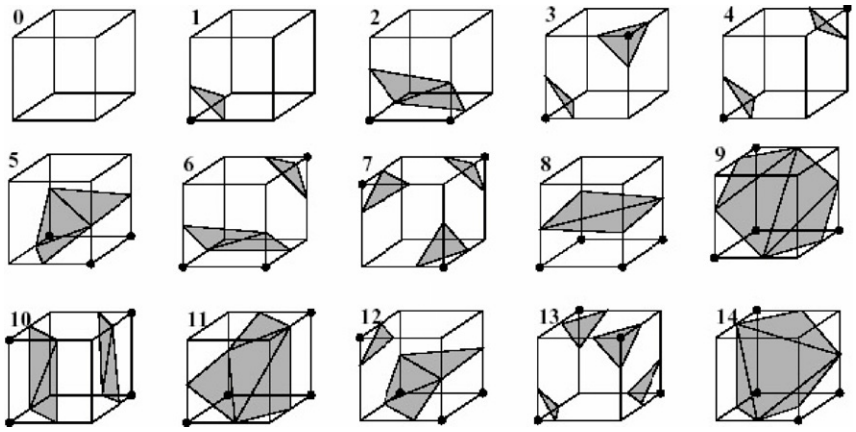


**Fig. 1.** Basic combinations for the marching cube algorithm. Black circles represent voxels inside the isosurface. A standard triangulation of the surface is performed within each cube. All possible combinations are obtained from the fundamental ones by rotations and reflections.

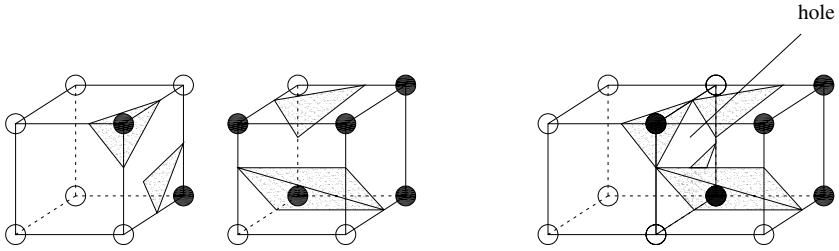


**Fig. 2.** Hole problem.

A linear interpolation is also used to evaluate the gradient on the triangle vertexes from its values previously evaluated on the voxels.

In summary, the algorithm proceeds through the following steps:

1. Load 4 slices in memory.
2. Loop on the two central slices, creating a logical cube from the eight neighboring voxels.
3. Evaluate the scalar field gradient on the eight voxels by the finite differences method.
4. Evaluate the binary index associated with the cube by comparing the voxel contents against the isovalue.
5. Get the triangulation type from the look-up table.
6. Compute the triangle vertex positions by linear interpolation.
7. Compute the scalar field gradient on the triangle vertexes by linear interpolation.
8. Store the triangle vertex coordinates and the gradient components in a file.

## 3. The novel MC algorithm

Fig. 2 shows an example of the triangulation of two adjacent logical cubes using the original MC algorithm in which the "hole problem" occurs.

In general, this problem can occur at the common face of adjacent logical cubes when two opposing voxels are above the threshold and the other two are below.

To solve the hole problem we propose a variation of the MC algorithm that does not use conjugation symmetry but rather additional new basic triangulations. We removed also the basic pattern "14" (see Fig. 2) because in the original algorithm it is equivalent to pattern "11" which differs from it by a reflection and a rotation. We introduce 7 new patterns as shown in Fig. 3.

Triangles of adjacent cubes are always well connected in this approach. Compared to other solutions, it is relatively simple and does not require substantial modifications of the original formulation of the algorithm; only the look-up table for the basic triangulations is modified and extended.

Our code produces its output in raw-binary format, as described in the next paragraph. However, we also provide a script to convert this output into the standard VRML (Virtual Reality Modelling Language) format. VRML is a text file format where, e.g., vertices and edges for a 3D polygon can be specified along with the surface color, UV mapped textures, shininess, transparency, and so on [17].

To compare the performance of the original MC algorithm with respect to our improved algorithm, we produced a simple object starting from a critical distribution of volumetric data (shown in Fig. 4).

It is possible to observe the produced holes in two views (shown in Fig. 5) using the VRML viewer.

## 4. Compiling and using the program

The program can be compiled by any standard C compiler; however, it was only tested using the gcc compiler. The program works on the Linux bash shell. A shell script (make.sh) is provided for compiling the source code using the gcc compiler.

This script compiles the source code and produces two executable binary files, "triangulate" and "trvtx2vrml". The first executable produces the triangulated isosurface, while the second can be used to convert the output format to the standard VRML.

The input volumetric data must be provided in raw binary format. The input file should be a 3D volumetric data file containing $N_x^* N_y^* N_z$ elements in binary (raw) format: $N_x^* N_y^* N_z$ floats, with $N_x$ running faster.

The "triangulate" executable reads the input volumetric data file and produces a triangulated model of the isosurface as output. This model is stored in two files, both in its own binary format:

Vertex file: file with vertex coordinates and gradient vector components in binary format:

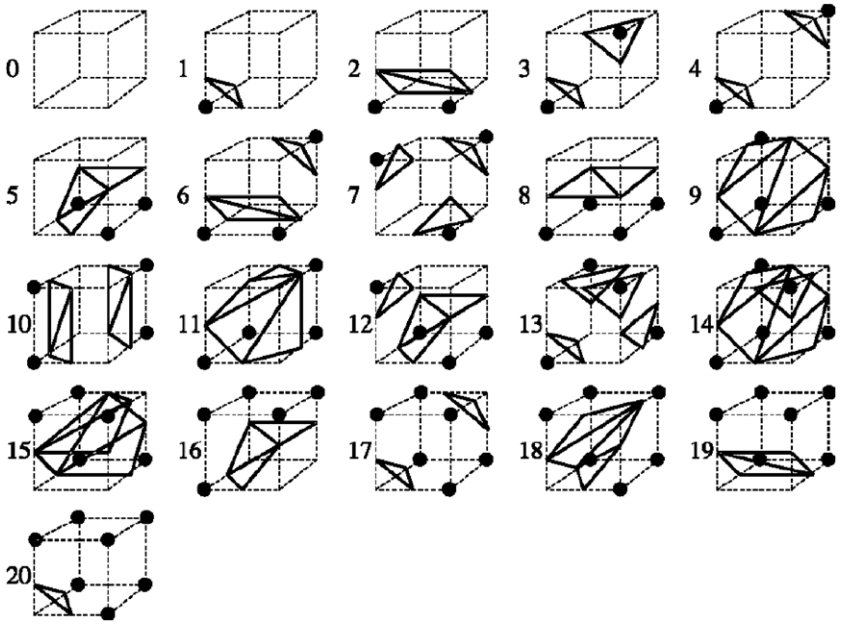


**Fig. 3.** Extension of the basic combinations for the marching cube algorithm using 21 patterns.

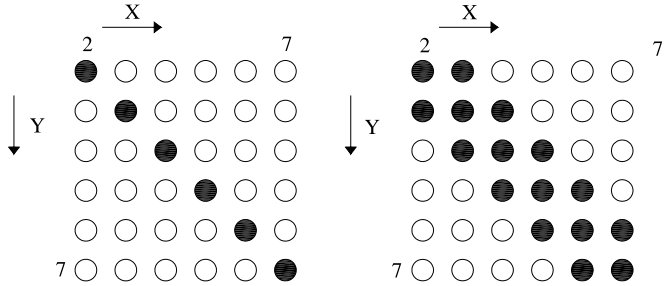


**Fig. 4.** Example of volumetric data for which the original version of the marching cube algorithm produces a triangulated surface containing holes, while the improved version described in this work does not. The volumetric data array is composed of 10 × 10 × 10 voxels, i.e., 10 slices each with 10 × 10 voxels. The content of all voxels is zero, except those of the slices *n*. 4, 5 and 6, which have nonzero elements on the main diagonal (slice 5, shown on the left) and on the three main diagonals (slices 5 and 6, which are equal to each other and are shown on the right).

$N_v$ (int type): number of vertexes in the mesh
$x_1, y_1, z_1$ (float type): coordinates of vertexes $n.1$
$v_{x1}, v_{y1}, v_{z1}$ (float type): gradient components of vertexes $n.1$
…
$x_{Nv}, y_{Nv}, z_{Nv}$ (float type): coordinates of vertexes $n.N_v$
$v_{xNv}, v_{yNv}, v_{zNv}$ (float type): gradient components of vertexes $n.N_v$

Triangle file: file with triangle vertex indexes binary format:

$N_t$ (int type): number of triangles in the mesh
$i_1, j_1, k_1$ (float type): indexes of the mesh vertexes corresponding to the 3 vertexes of triangle $n.1$
…
$i_{Nt}, j_{Nt}, k_{Nt}$ (float type): indexes of the mesh vertexes corresponding to the 3 vertexes of triangle $n.Nt$

The triangulate executable takes 11 arguments:

(1) input file name: a raw file with volumetric data array ($N_x^* N_y^* N_z$ floats)
(2) output 1 triangles file name: name of the file containing triangle vertex indexes

**Table 1**
Sign parameter.

| Rule inclusions | Settings |
|---|---|
| $<$ | 0 |
| $\leq$ | 1 |
| $\geq$ | 2 |
| $>$ | 3 |

(3) output 2 vertex file name: name of the file containing vertex coordinates and gradient vector components
(4) $N_x, N_y, N_z$: volumetric data array dimensions
(5) $\text{Side}_x, \text{Side}_y, \text{Side}_z$: voxel sides
(6) Threshold: iso-value, i.e., density threshold used to define the isosurface (i.e., $\mu$ density of tissue)
(7) Sign: INCLUSION RULE, i.e., type of comparison used to decide whether a voxel center is inside the isosurface. Note that the isosurface is not invariant if the comparison is reversed. The possible values of the command line argument and the corresponding inclusion rules are given in Table 1.

We provide two other examples for testing the code. Both examples can be visualized using any 3D data visualization program that can read the VRML format. For instance, we visualized them through the plugging of the browser Cortona 3D Viewer 7.0 [16], which can visualize the VRML format.

The executable trvtx2vrml can be used to convert the output files of the triangulation algorithm to the standard VRML format.

The executable takes 3 arguments:

(1) input 1, triangle file name: name of the file containing triangle vertex indexes, produced by the program "triangulate";

(3) input 2, vertex file name: name of the file containing vertex coordinates and gradient vector components, produced by the program "triangulate";

(4) output file name: output file containing the triangulated model of the isosurface in VRML format.

The first example (named 'visual') that we used as a benchmark to measure the performance of the system is shown in Fig. 6.

For the test described, we use the following command:

./triangulate visual.dat triangles.dat vertex.dat 200 200 200 1 1 1 0.5 0

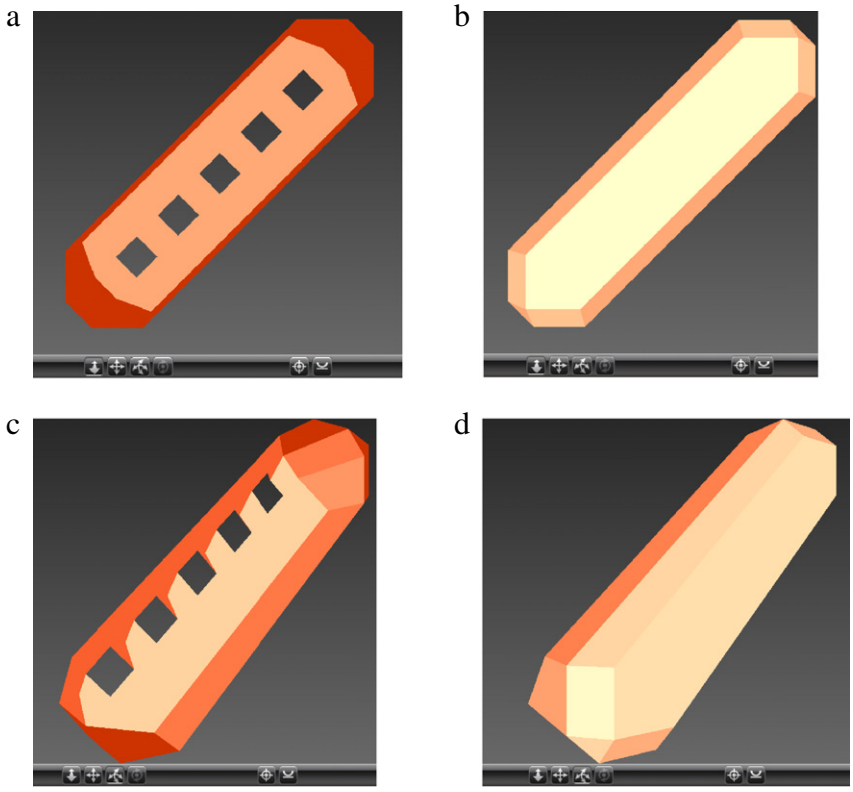


**Fig. 5.** Triangulated isosurfaces produced by the original marching cube algorithm ((a), (c)) and by the improved algorithm described in this work ((b), (d)) using the volumetric data described in Fig. 4.

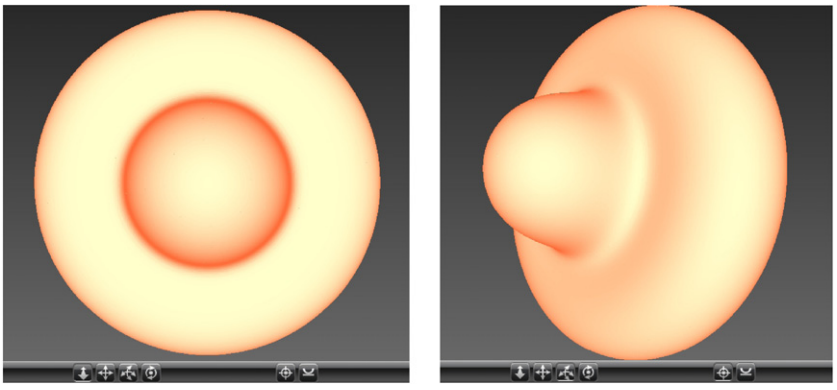

**Fig. 6.** Example 1: 'visual' in Cortona 3D Viewer.

The output is then converted to the VRML format through the following command:

./trvtx2vrml vertex.dat triangles.dat visual.wrl

The second example (named 'net' and shown in Fig. 7) can be produced by launching the following command, changing the threshold with respect to the previous test:

./triangulate net.dat triangles.dat vertex.dat 200 200 200 1 1 1 1.05 0

The output can be converted to the VRML format through the following command:

./trvtx2vrml vertex.dat triangles.dat net.wrl

To run the example (shown in Fig. 5) demonstrating the differences between the original MC algorithm and our proposed method, the following command must be launched:

./triangulate MC_example.dat triangles.dat vertex.dat 10 10 10 1 1 1 0.5 0

The output can be converted to the VRML format through the following command:

./trvtx2vrml vertex.dat triangles.dat MC_example.wrl

The VRML file, produced using the original marching cube algorithm, is also provided with the package and is called original_MC_example.wrl.

## 5. Program features

The following section highlights some useful features of the program.

The software is very fast and does not require excessive computational power. The RAM usage is optimized to be independent from the size of the data; the algorithm loads 4 slices at a time into memory in a stack, performs the triangulation on the central slices and appends the result on the vertex and triangle files.

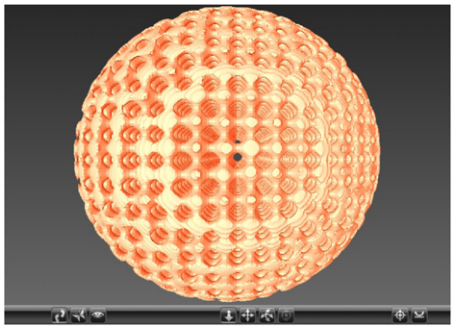

**Fig. 7.** Example 2: 'net' in Cortona 3D Viewer.

It is possible to include it in a simple script, i.e., using a loop, to obtain a multithreshold reconstruction of the image data.

The software provides the output in raw binary format, which can also be converted to the VRML output format that is compatible with the most common 3D visualization software. To optimize the visual representation we also included a Gouraud shading correction, which yields a more realistic visualization of the surface lighting [18].

## Acknowledgments

Some of the results reported in this paper have been produced in the framework of the Cybersar project funded from the Ministero dell'Università e della Ricerca (MIUR) by the Piano Operativo Nazionale "Ricerca Scientifica, Sviluppo Tecnologico, Alta Formazione" PON 2000–2006; see also http://www.cybersar.com.

## References

[1] Timothy S. Newman, Hong Yi, A survey of the marching cubes algorithm, Computers & Graphics 30 (5) (2006) 854–879.
[2] M. Jones, A. Leu, R. Satherley, S. Treavett Glossary, M. Chen, A. Kaufman, R. Yagel (Eds.), Volume Graphics, Springer, London, 2000, pp. 395–406.
[3] William E. Lorensen, E. Harvey, Cline: marching cubes: a high resolution 3D surface construction algorithm. In: Computer Graphics, July 1987, vol. 21, Nr. 4.
[4] B. Golosio, A. Brunetti, R. Cesareo, S.R. Amendolia, D.V. Rao, S.M. Seltzer, Images of soft materials: a 3D visualization of interior of the sample in terms of attenuation coefficient, Nuclear Instruments and Methods A 465 (2001) 577–583.
[5] B. Golosio, G.L. Masala, A. Piccioli, P. Oliva, M. Carpinelli, R. Cataldo, F. De Carlo, F. Falaschi, M.E. Fantacci, G. Gargano, P. Kasae, M. Torsello, A novel multithreshold method for nodule detection in lung CT, Medical Physics 36 (8) (2009) 3607–3618.
[6] A. Watt, M. Watt, Advanced Animation and Rendering Techniques, Addison-Wesley, 1992.
[7] A. van Gelder, J. Wilhelms, Topological considerations in isosurface generation, ACM Transactions on Graphics 13 (4) (1994) 337–375.
[8] W. Heiden, T. Goetze, J. Brickmann, Fast generation of molecular surfaces from 3D data fields with an enhanced marching cube algorithm, Journal of Computational Chemistry 14 (2) (1993) 246–250.
[9] G. Nielson, A. Huang, S. Sylvester, Approximating normals for marching cubes applied to locally supported isosurfaces, In: Proceedings of visualization'02, Boston, 2002, pp. 459–466.
[10] G. Nielson, On marching cubes, IEEE Transactions on Visualization and Computer Graphics 9 (3) (2003) 283–297.
[11] C. Montani, R. Scateni, A Scopigno modified look-up table for implicit disambiguation of marching cubes, Visual Computer 10 (6) (1994) 353–355.
[12] C. Zhou, R. Shu, M. Kankanhalli, Handling small features in isosurface generation using marching cubes, Computers Graphics 18 (6) (1994) 845–848.
[13] X. Renbo, L. Weijun, W. Yuechao, A robust and topological correct marching cube algorithm without look-up table, Proceedings of the Fifth International Conference on Computer and Information Technology IEEE (2005) 565–569.
[14] K.S. Delibasis, G.K. Matsopoulos, N.A. Mouravliansky, K.S. Nikita, A novel and efficient implementation of the marching cubes algorithm, Computerized Medical Imaging and Graphics 25 (2001) 343–352.
[15] A. Gueziec, R. Hummel, Exploiting triangulated surface extraction using tetrahedral decomposition, IET Computer Vision and Computer 1 (1995) 328–342.
[16] C.A. Dietrich, C.E. Scheidegger, J. Schreiner, J.L.D. Comba, L.P. Nedel, C.T. Silva, Edge Transformations for Improving Mesh Quality of Marching Cubes, Visualization and Computer Graphics, IEEE Transactions on, vol. 15, no. 1, Jan–Feb 2009, pp. 150–159.
[17] Version 1.0 Specification". Web3d.org Retrieved 2010-02-23 http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html.
[18] H. Gouraud, Continuous shading of curved surfaces, IEEE Transactions on Computers 20 (6) (1971) 623–628.