Journal of University of Science and Technology Beijing
Volume 12, Number 3, June 2005, Page 203

Application of marching cubes algorithm in visualization of mineral deposits

Dewen Seng[1,2], *Zhongxue Li*[1], *Cuiping Li*[1], *and Chunmin Li*[1]

1) Civil and Environmental Engineering School, University of Science and Technology Beijing, Beijing 100083, China
2) Department of Computer Science, Zhejiang Water Conservancy and Hydropower College, Hangzhou 310018, China

(Received 2004-12-14)

Abstract: An implementation scheme of the marching cubes (MC) algorithm was presented for the visualization of mineral deposits. The basic principles, processes and pitfalls of the MC algorithm were discussed. The asymptotic decider algorithm was employed to solve the ambiguity problem associated with the MC algorithm. The implementation scheme was applied to model and reconstruct the surfaces of mineral deposits, using the geological data obtained from an iron mine in China. Experimental results demonstrate the ability of the implementation scheme to solve the ambiguity problem, and illustrate the effectiveness and efficiency of the MC algorithm in the visualization of mineral deposits.

Key words: marching cubes algorithm; visualization; surface reconstruction; mineral deposit modeling

[*This study is financially supported by the Ph.D. Programs Foundation of the Ministry of Education of China (No.20020008006).*]

1 Introduction

Two-dimensional (2D) mineral maps are routinely used and produced in mining practices. Viewing these 2D maps often gives insufficient information if the shape and morphology of a mineral deposit is to be fully comprehended. Using three-dimensional (3D) reconstruction modeling, the structure of the mineral deposit can be visualized and further manipulated in such a way that the maximum realism is achieved. Such a 3D approach can be considered as the first step towards virtual reality for mine modeling and design.

Techniques available for reconstructing a structure are generally divided into the following two categories:

(1) Voronoi diagrams [1] or graph techniques [2], in which contours of the object under consideration are produced and then the triangulation problem is solved using methods such as Delaunay triangulation.

(2) Techniques based on 3D datasets, in which appropriate triangles are produced directly from 3D datasets and processed in cubic neighborhoods of typical eight voxels. The marching cubes algorithm belongs to the techniques [3].

The second category of the techniques, particularly the marching cubes (MC) algorithm, is nowadays the most widely used technique for the extraction of iso-surfaces out of volumetric datasets [4]. The reasons for MC success include its simple logical structure implying a nearly straightforward implementation, and its computational efficiency. The MC has been incorporated in many commercial and public domain visualization systems [5]. Many papers have been published on enhancements, optimization, extensions and applications of this technique [6-7].

As a huge amount of volumetric data are created by geological or mineral activities, the ability to analyze and understand these geological data in depth necessitates visual processing. In this paper, the conventional approach to the MC was firstly described. Then, an overview of the implementation scheme of the MC algorithm for the visualization of geological and mineral datasets was discussed in depth. Finally, Experimental results based on the geological data from an iron mine in China were also presented.

2 Marching cubes algorithm

2.1 Principles

Volumetric datasets are generally organized as 3D rectilinear grids with a scalar value stored at each grid point [5]. The MC algorithm operates on a logical cube created from eight corner points (vertices); four each from two adjacent slices. The basic principle behind the algorithm is to subdivide space into a series of such small

Corresponding author: Dewen Seng, **E-mail:** 3qwen@126.com

logical cubes; then the operator is instructed to "march" through each of the cubes and determine how a surface intersects them. To find the surface intersection in a cube, the algorithm assigns "1" to a cube's vertex if the data value at that vertex exceeds or equals the threshold value of the isosurface or isovalue. These vertices are inside or on the isosurface. Cube vertices with data values below the isovalue receive "0" and are outside the isosurface. The isosurface intersects those cube edges where one vertex on the edge is outside the isosurface and the other on the edge is inside the surface. With this assumption, the topology of the isosurface within a cube is determined. **Figure 1** demonstrates the vertices, edge intersection points, and the resultant triangular facet for a sample cube that has one vertex with a scalar value above the isovalue of the surface.

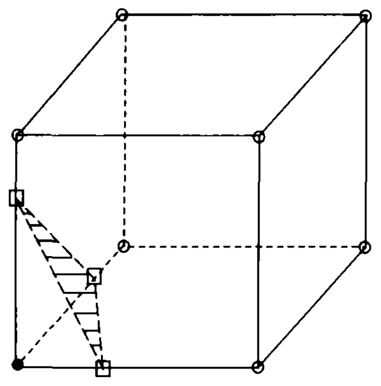


Figure 1 Sample cube with interpolated intersection locations and triangular facet: ○—vertex outside the isosurface; ●—vertex inside or on the isosurface; □—interpolated intersections.

Since there are eight vertices in each cube and two states, inside and outside, there are only $2^8 = 256$ ways that a surface can intersect the cube. By enumerating these 256 cases, the algorithm creates a table to look up surface-edge intersections, given the labeling of a cube's vertices. The table contains the edges intersected for each case.

Triangulating all the 256 cases is possible but tedious and error-prone. Two symmetries of the cube reduce the problem from 256 cases to 15 patterns [3]. First, the topology of the triangulated surface is unchanged if the relationship of the surface values to the cubes is reversed. Complementary cases, where vertices greater than the surface value are interchanged with those less than the value, are equivalent. Thus, only cases with 0 to 4 vertices greater than the surface value need be considered, reducing the number of cases to 128. Considering the second symmetry property or rotational symmetry, the algorithm can be reduced further to 15 patterns by inspection. **Figure 2** shows the triangulation for the 15 patterns. 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, resulting in one triangle defined by the three edge intersections. Other patterns produce multiple triangles. Permutation of these 15 basic patterns using complementary and rotational symmetries produces the 256 cases.

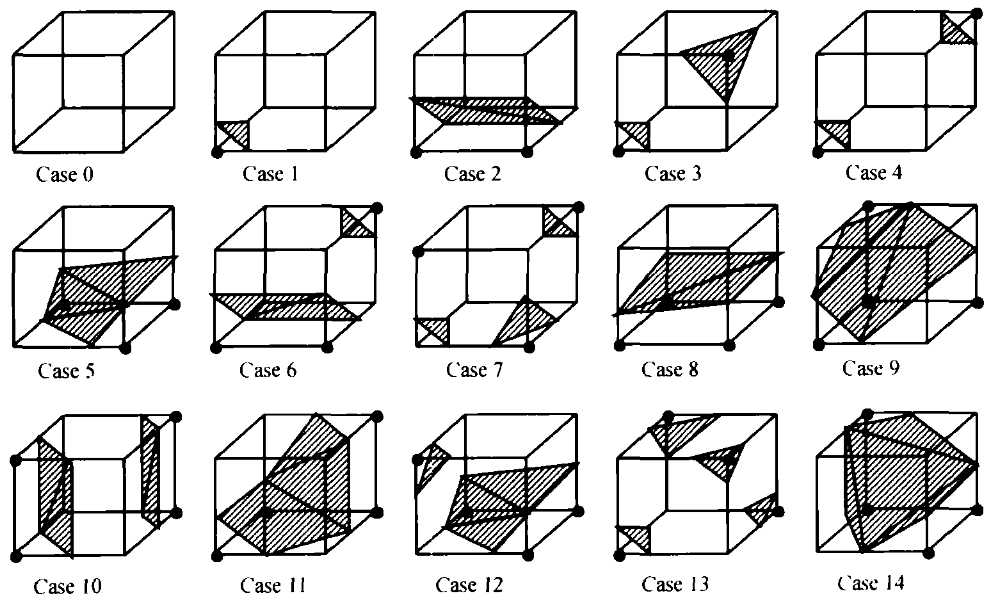


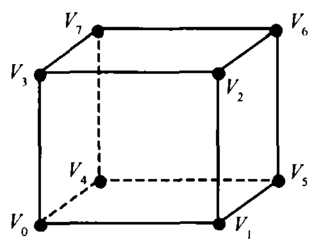
Figure 2 Triangulation of 15 predefined cube configurations.

2.2 Algorithm

There are two primary steps in the approach of the MC algorithm to the surface construction problem.

(1) To locate the surface corresponding to a user-specified value and create triangles.

The MC algorithm creates an index for each case, based on the state of the vertex. Using the vertex numbering in **figure 3**, the eight-bit index contains one bit for each vertex. This index serves as a pointer into an edge lookup table (LUT) that gives all edge intersections for a given cube configuration. Using the index to tell which edge the surface intersects, the user can interpolate the surface intersection along the edge by linear interpolation or higher degree interpolations. Since the algorithm produces at least one and at most four triangles per cube, the higher degree surfaces show little improvement over linear interpolation.



Index=	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7

Figure 3 Cube numbering.

(2) To calculate a unit normal for each triangle vertex to ensure a quality image of the surface.

The rendering algorithms use a normal to produce Gouraud-shaded images. A surface of constant density has a zero gradient component along the surface tangential direction; consequently, the direction of the gradient vector (***g***), is normal to the surface. This fact can be used to determine surface normal vector, ***n***, if the magnitude of the gradient is nonzero. Fortunately, at the surface of interest between two types of different densities, the gradient vector is nonzero. The gradient vector, ***g***, is the derivative of the density function:

$$\boldsymbol{g}(x,y,z)=\nabla f(x,y,z) \tag{1}$$

To estimate the gradient vector at the surface of interest, the algorithm first estimates the gradient vectors at the cube vertices and linearly interpolate the gradient at the point of intersection. The gradient at the cube vertex (*i*, *j*, *k*) is estimated using central differences along the three coordinate axes by:

$$\begin{cases} g_x=\dfrac{f(x_{i+1},y_j,z_k)-f(x_{i-1},y_j,z_k)}{\Delta x} \\ g_y=\dfrac{f(x_i,y_{j+1},z_k)-f(x_i,y_{j-1},z_k)}{\Delta y} \\ g_z=\dfrac{f(x_i,y_j,z_{k+1})-f(x_i,y_j,z_{k-1})}{\Delta z} \end{cases} \tag{2}$$

where $f(x_i,y_j,z_k)$ is the density at (x_i,y_j,z_k) and Δx, Δy, Δz are the lengths of the cube edges. Dividing the gradient by its length produces the unit normal at the vertex required for rendering. The algorithm linearly interpolates this normal to the point of intersection. Note that to calculate the gradient at all vertices of the cube, four slices should be kept in memory at once.

In summary, the MC algorithm creates a surface from a three-dimensional set of data as follows:

(1) Read four slices into memory.

(2) Scan two slices and create a cube from four neighbors on one slice and four neighbors on the next slice.

(3) Calculate an index for the cube by comparing the eight density values at the cube vertices with the surface constant.

(4) Using the index, look up the list of edges from a precalculated table.

(5) Using the densities at each edge vertex, find the surface edge intersection *via* linear interpolation.

(6) Calculate a unit normal at each cube vertex using central differences, and interpolate the normal to each triangle vertex.

(7) Output the triangle vertices and vertex normals.

An outline of the execution flow for the MC algorithm is shown in **figure 4**.

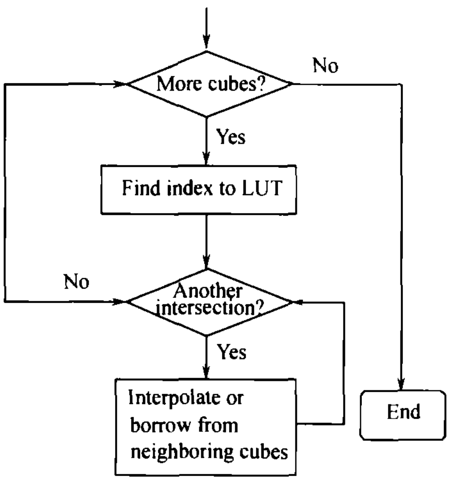


Figure 4 Execution flow of the MC algorithm.

3 Pitfalls and enhancements

As marching cubes is an object-order rather than an image-order algorithm, it may be inefficient, specifically, slow in computation and large in storage requirements. Each cube may contribute up to four facets to the final geometry; therefore, even a modest sized dataset of 100 slices at 256 by 256 resolutions may produce between 500000 and 2000000 triangles. This may have excessive processing and storage requirements, especially during the rendering stage.

Further, as the facets generated are sub-cube, many triangles are smaller than a single pixel after rendering. This is inefficient, and reveals the algorithm's object-order nature. The dividing cubes [8] algorithm attempts to avoid this problem. It is similar to marching cubes except that it projects cubes that intersect the surface into the image plane. If the cubes map to single pixels or smaller, they are rendered as single points, otherwise they are subdivided into facets as with marching cubes.

Another major problem in the MC algorithm is the ambiguity in isosurface construction. This ambiguity occurs when the assumption of one edge-surface intersection per cube edge breaks down and an improper decision is made on whether a vertex is contained by the surface, or when the triangles chosen at adjacent cubes do not fit together properly. The result is a hole in the constructed isosurface. Such a possibility is shown in **figure 5** where a cube with configuration 6 shares a face with the complement of configuration 3. In such a case, the asymptotic decider algorithm [9] is employed to solve this problem.

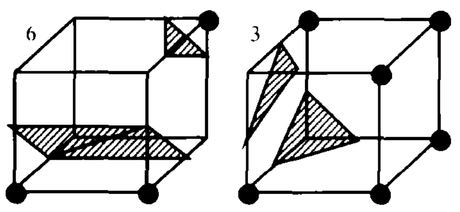


Figure 5 Ambiguity in isosurface construction.

4 Experimental results

The marching cubes algorithm was implemented in C++ on Windows 2000. The original geological data, including 144 prospecting boreholes, were sampled from an iron mine in Wuhan, China. The original data were regularly structured by Krige method [10], generating a dataset which includes the geologic range of 350 m×470 m×70 m. The cube size of the dataset is 10 m×10 m×1 m. **Figure 6** shows the surface reconstruction result of the ore body with an isovalue of ore grade of 25%, **figure 7** with an isovalue of 40%, and **figure 8** with an isovalue of 50%.

The number of triangles in a surface model is proportional to the area of the surface and it can become very large. In this study, the number was reduced using cut planes and surface connectivity. The original data was also filtered to produce a somewhat smoother surface with some loss of resolution. An in-house Z-buffer program was used to display the models. Execution time depends on the number of surfaces and resolution of the original data. Model creation time on a computer with a Pentium 2.8 GHz CPU and 512M RAM is about 2 s for the dataset of 350 m× 470m×70m.

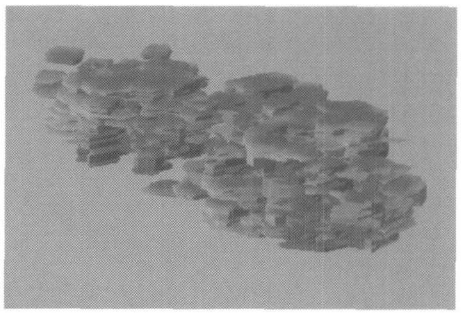


Figure 6 Surface reconstruction using the MC algorithm with an ore grade of 25%.

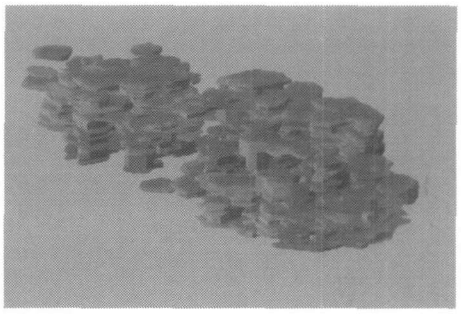


Figure 7 Surface reconstruction using the MC algorithm with an ore grade of 40%.

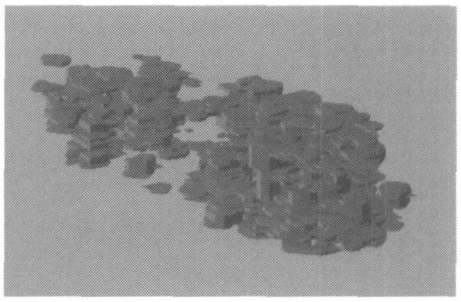


Figure 8 Surface reconstruction using the MC algorithm with an ore grade of 50%.

5 Conclusions

3D surface construction by the MC algorithm complements 2D maps by giving the mining engineer 3D views of mineral deposits. The algorithm uses a case table of edge intersections to describe how a surface cuts through each cube in a 3D dataset. Additional realism is achieved by calculating the normalized gradients from the original data. The results can be displayed on conventional graphics display systems. Although these models often contain a large number of triangles, surface cutting and connectivity can reduce this number.

A case study with data from an iron mine in China has been presented to demonstrate the quality of the constructed surfaces and the efficiency and effectiveness of the algorithm. The results are very encouraging for further studies and applications.

References

[1] J.D. Boissonat, Shape reconstruction from planar cross sections, *Comput. Vision Graphics Image Process.*, 44(1988), p.1.

[2] H. Fuchs, Z.M. Kedem, and S.P. Uselton, Optimal surface reconstruction from planar contours, *Commun. ACM*, 20(1977), No.10, p.693.

[3] W.E. Lorensen and H.E. Cline, Marching cubes: high resolution 3-D surface construction algorithm, *Comput. Graphics*, 21(1987), No.3, p.163.

[4] P. Cignoni, F. Ganovelli, and C. Montani, Reconstruction of topologically correct and adaptive trilinear isosurfaces, *Comput. Graphics*, 24(2000), p.399.

[5] T.S. Newman, J.B. Byrd, and P. Emani, High performance SIMD marching cubes isosurface extraction on commodity computers, *Comput. Graphics*, 28(2004), p.213.

[6] D.A. Rajon and W.E. Bolch, Marching cube algorithm: review and trilinear interpolation adaptation for image-based dosimetric models, *Comput. Med. Imaging Graphics*, 27(2003), p.411.

[7] K.S. Delibasis, G.K. Matsopoulos, and N.A. Mouravliansky, A novel and efficient implementation of the marching cubes algorithm, *Comput. Med. Imaging Graphics*, 25(2001), p.343.

[8] H.E. Cline, W.E. Lorensen, and S. Ludke, Two algorithms for three-dimensional reconstruction of tomograms, *Med. Phys.*, 15(1988), No.3, p.320.

[9] D.W. Seng, Z.X. Li, C.M. Li, *et al.*, 3D visual modeling system for mineral deposits, *J. Univ. Sci. Technol. Beijing* (in Chinese), 26(2004), No.5, p.453.

[10] C.P. Li, Z.X. Li, and N.L. Hu, A method for spatialization and regularization of geological data in the volume visualization of mineral deposits, *J. Univ. Sci. Technol. Beijing* (in Chinese), 24(2002), No.6, p.589.