Summer 1996

The implementation of a disambiguation marching cubes algorithm

Xiaodan Liu

The Implementation of a Disambiguation Marching Cubes Algorithm

---

A Thesis

Presented To

Eastern Washington University

Cheney, Washington

In Partial Fulfillment of the Requirements

for the Degree

Master of Science

Computer Graphics / Mathematics

by

Xiaodan Liu

August 1996

Abstract

This thesis first systematically analyzes a classic surface generation algorithm, the marching cubes algorithm, in computer volume visualization, with emphasis on the mathematical background and the ambiguity problem of the algorithm. A simple and elegant disambiguation algorithm is then described and implemented. Finally, generated data from mathematical functions and real world data from scientific experiment are used to test the original marching cubes algorithm and the disambiguation algorithm.

Acknowledgments

With the completion of this thesis, the two-year overseas study comes to an end. I wish to give my heartfelt thanks to everybody who has cared, encouraged, and supported me during the journey.

My very special thank goes to my advisor, Dr. Clark, who has given me lots of invaluable advice and support in both computer graphics area and the real world.

I would also like to thank the mathematics department and the computer science department who have collaborated so well to give me a chance to work on this interdisciplinary program.

Finally, I want to dedicate this thesis to my dear parents. They have so many dreams that they could not accomplished in their generation. But they have helped me enormously to make my own dream come true.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scientific experiments and simulations generate huge data sets, which are difficult to interpret and analyze without computerized aids. One of the important tools is scientific visualization, in which the data is displayed visually [RAPP91].

Volume data is a discrete three dimensional array which represents a finite space or volume. A *voxel* is a cubic unit of the volume. The source of volume data is sampled data of real objects or phenomena, computed data produced by a computer simulation, or modeled data generated from geometric model. Examples of applications generating sampled data occur in wazzu medical imaging such as computed tomography (CT), magnetic resonance imaging (MRI), and ultrasonography etc. Examples of applications that generate computed data sets, typically running a simulation in a supercomputer, occur in meteorology (storm prediction), computational fluid dynamics (water flow) etc. [KAUF93].

Visualization of volume data is tightly related to the topic of *volume visualization* which has emerged recently as an important technique in the fields of visualization, computer graphics, and computer imaging. Volume visualization is concerned with the tasks of representing, manipulating, and rendering volume data. It encompasses an array of techniques which provide the mechanisms that make it possible to reveal and explore the inner or unseen structures of volume data and allow visual insight into opaque or complex data sets.

Two major techniques in volume visualization are *surface rendering* and *volume rendering*. In surface rendering, the volume data is first converted into geometric primitives (e.g. polygon mesh), then the geometric primitives are rendered to the screen by conventional computer graphics rendering techniques [KAUF91]. This method implicitly assumes that there is some underlying surface behind the data [RAPP91]. Volume rendering involves displaying the volume directly by attaching properties such as opacity and color to the individual voxels of the data set instead of converting the data to an intermediate surface representation.

Although volume rendering has several advantages such as the insensitivity to scene and object complexity, the capability to represent inner information, etc. , the computation is very expensive and the data storage space is large . On the other hand, in surface rendering, surface representation results in a significant reduction of data storage, and can exploit existing computer graphics techniques for fast manipulation and display.

Therefore, surface rendering provides a quick , handy way to visualize volume data in many scientific applications, especially real-time interactive applications.

Surface generation, a process of generating surface representation (usually a polygon mesh) out of volume data , is a principal issue in the surface rendering technique. This paper analyzes a popular surface generation algorithm, especially the mathematical theory behind it, and investigates the discontinuities in the output of the algorithm. An improved version of the algorithm which gives emphasis to solving this problem simply and elegantly is described and implemented.

Chapter 2

Background & Literature Review

In volume visualization, one primary source of volume data is generated by a sampling device or a computer model that creates discrete sampling points of a real or simulated object; usually as a sequence of cross-sectional scans. This empirical data is processed by 2D image processing techniques and is then 3D reconstructed into a 3D volume data set by stacking the cross sections. Another source of volume data is by *voxelization* which converts a geometric model into a set of voxels that "best" represents the synthetic model within the discrete voxel space [KAUF91]. Different data geometries are used by these data.

2.1 Data Geometry

The most general means for storing two dimensional (or three dimensional) data is as an unordered collection of independent points. However, this format is not only expensive to store and manipulate. it is also rare in simulation and computational data analysis. Usually, the data geometries fall into three major categories:
uniform scalar data, *rectilinear data* and *irregular data*.

In two dimensions, the three data geometries are defined as follows:
(1) Uniform scalar data:

The data points are arranged in a regular grid with even spacings (see Figure-1).

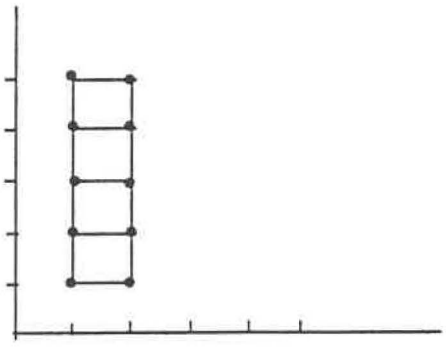


Figure-1

(2) Rectilinear data

Like the uniform scalar data, the data points are also arranged in a rectangular grid; however, the spacings may be uneven (see Figure-2).

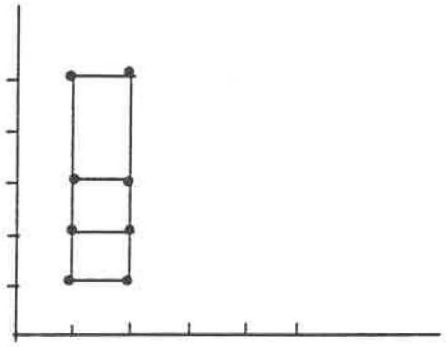


Figure-2

(3) Irregular data

The data points can locate in any place within the dimension area. Thus all the data form a connected grid of irregular-shaped cells (see Figure-3).

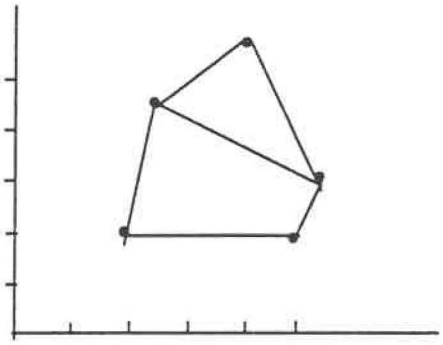


Figure-3

The above data geometries defined in two dimensions can be extended into three dimensional space. Generally, data in three dimensional space is also called *volume data*. The three types of volume data are: *uniform scalar volume data*, *rectilinear volume data*, *irregular volume data*.

Uniform scalar volume data are often used to represent medical images such as magnetic resonance images, computed tomography images etc. . Rectilinear volume data usually arises in simple computational fluid dynamics applications, in which cell density increases in regions of interest. There are also some applications which use the irregular volume data. For example, a flow field around an airplane must provide boundaries which conform to the shape of the airplane [SHIR89].

All the volume data discussed in this paper are uniform scalar volume data. For convenience, the uniform scalar volume data is refered to as volume data in this paper. The volume data set is represented as a 3D discrete regular grid. A *voxel* is a cubic unit of volume centered at the integral grid point. As a unit of volume, a voxel is the 3D counterpart of a 2D pixel, which represents a unit of area. Each voxel has a numeric value (usually called *density value*) associated with it, which represents some kind of

measurable property of the object residing in the volume data. For example, this property can be the radio-opaque in 3D medical volume data.

Surface rendering technique is one of the major techniques to visualize volume data. A polygonal surface is generated from the data and rendered to the screen by conventional computer graphics rendering techniques. The generated surface which is an approximation of the real surface of the object residing at the volume data helps to reveal and explore the inner or unseen structures of volumete data. Many techniques have been developed to generate a surface from the volume data.

2.2 Surface Generation Techniques

2.2.1 Cuberille Model

Chen et al. [CHEN85] proposed a surface shading method in the cuberille environment. They defined a cuberille model as "dissection of space into equal cubes (called voxelsthe) by three orthogonal sets of parallel planes". In this environment, an object is represented by a subset of the set of all voxels. The detected surface of the object is called *boundary surface* which is made up of faces of the cubes which separate voxels in the object from voxels in the background.

The boundary surface is first detected by a three-dimensional boundary detection algorithm which is developed by Liu [LIU77]. The algorithm first picks an intitial boundary element as the current bounday element, then it tries to find the next boundary element by choosing one of the neighbors of the current boundary element. If the next boundary element is found, then it is put into a pool; the current boundary element is chosen from the top of a pool and the next boundary element needs to be found. If the next boundary element is not found, then a back tracing routine is called. The process repeats until the whole boundary is found. Then the surface is displayed using common computer graphics techniques.

The advantage of the cuberille model is that it is fast since the boundary surface is built up of faces of cubes with no detail from inside the cube. The disadvantage is that the produced image is naturally blocky because all surface patches are orthogonal to one of the coordinate axes. However, the image can be made to appear smooth by the deft use of surface shading. A number of shading methods which include distance-only shading, constant shading, image-based contextual shading, normal-based contextual shading, Phong shading and gradient shading have been experimented with in this cuberille model. The best quality image rendering is produced by the normal-based contextual shading method.

2.2.2 Planar Contour Method

The planar contour method is one of the early techniques in 3D surface generation. The idea of this method is based on the structure of a 3D data set which consists of a stack of 2D cross-sectional planes. The contour in each 2D parallel cross-sectional plane is first identified. A 3D surface is then constructed by connecting the consecutive planes using planar surface elements which are named "tiles".

2.2.2.1 Surface with Triangular Tiles

Fuchs et al. [FUCH77] constructed a surface over a set of cross-sectional contours, using triangular tiles between each pair of adjacent contours. Each triangle is defined between two consecutive vertices on the same contour and a single vertex on the adjacent contour. A directed graph is used to represent all possible edge interconnections between any two contours. The edges to be created between vertices of the contours are represented as nodes in the graph and the triangular tiles are represented as arcs between nodes. The problem of which vertices from the two contours to connect is then resolved by finding a path through the graph which yields the lowest 'cost'. If the cost of moving from one node of the graph to another is chosen to represent the surface area of a tile, the best path through the graph represents a mesh with the smallest surface area.

2.2.2.2 Surface with Spline Tiles

Sunguroff and Greenberg [SUNG78] used a combination of B-splines and Cardinal splines to find the surfaces. In their 3D reconstruction system, there are three major modules. The first module is the contour extraction module which generates contours either with a gradient edge detection algorithm or under interactive control. In the second module which is called the surface forming module, B-splines are used for representation of the sectional contours and Cardinal splines are used to interpolate between sections to form the 3D surfaces. Finally, smooth-shaded images are created by the rendering module.

In this system, the polygon description of the objects has a major advantage for smooth-shaded images when compared to standard polygon techniques. Most continuous shading methods approximate the vertex normal vectors by averaging the continuous polygon normals. However, using the spline technique, a continuous surface is interpolated, and thus an exact surface normal is obtained at each vertex. Intermediate normals are then interpolated from the exact values at the vertices.

There are several drawbacks in the planar contour method [HERM79] [LORE87]. The inter-plane connectivity that exists in the original data is lost. Bias is introduced into the display by the orientation of the original cross sections. If there is more than one contour on a cross-sectional plane, ambiguity arises in determining the connection between contours.

2.3 Isosurface Generation

2.3.1 Definition of Isosurface & Isosurface Generation:

In a volume data, each voxel associates with a density value which represents some kind of measurable property of the object residing in the data. Given a threshold density value C, all the points with density values equal to C in the dimensions of the volume data form an *isosurface* for value C (which is also called the isovalue).

An algorithm which extracts a polygon mesh representing the isosurface from a volume data is known as an *isosurface generation* algorithm.

Frequently, the volume data used in isosurface generation is redefined as an aggregation of *cubes* whose vertices lie on the grid points of the data. The value of a vertex is the same as the value of the voxel that centers at the vertex. Note that a "cube" has sampled points at its vertices while a "voxel" has a sampled point at its center. A *face* is one of the six sides of a cube. An *edge* is one of four rims of a face.

Given a threshold value, a vertex is positive if its value is greater than the threshold, and it is negative if its value is less than the threshold. In this paper, a positive vertex is represented by a small black circle, and a negative vertex is not highlighted.

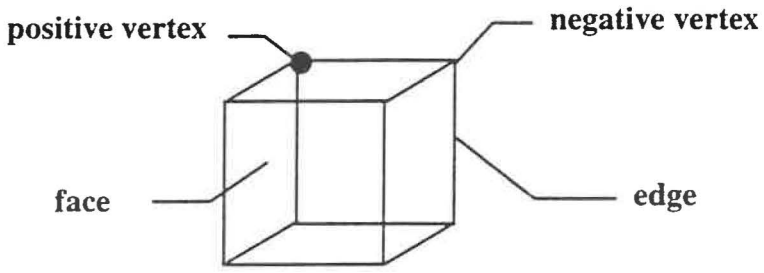


Figure-4 Cube structure

In a volume data set, a cube can be classified into one of the three categories: inside the surface, outside the surface or intersecting the surface. If all the eight vertices of a cube have values greater than the threshold value, the cube lies inside the surface. A cube is outside the surface if all its eight vertices have values less than the threshold value. If a cube has some vertices with values greater than the threshold value and some less than the threshold value, then it intersects the surface.

When an edge of a cube joins opposite signed vertices, the isosurface must pass through the edge. The point at which the isosurface crosses through edge is the *intersection point* which is calculated by interpolation. In this paper, an intersection point is represented by an empty circle.

Intersection points on all the possible edges of a cube are then connected sequentially to form polygons which are the approximation of the isosurface within the cube. Usually, these polygons are nonpalanar, and they are further tessellated into triangles which are commonly used as primitives in 3D computer graphics rendering algorithms. The result of the above process is called the *triangulation* of a cube.

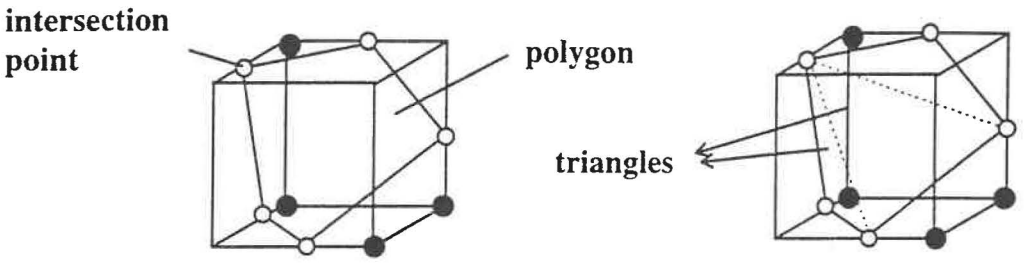


Figure-5 Triangulation of a cube

Certain assumptions are made by isosurface generation algorithms in order to simplify the implementation:

* If two vertices of an edge have the same signs with respect to the threshold, the isosurface is assumed not to pass through the edge (although the surface may actually pass through the edge any even number of times). In this situation, it is assumed that there is no intersection point on this edge.
* If two vertices of an edge have opposite signs (one positive and one negative), the isosurface is assumed to pass through the edge just once (although the surface may actually pass the edge any odd number of times). Multiple intersections can not be detected and only one intersection point is assumed to exist on this edge.

2.3.2 Assurance of Isosurface Continuity

A polygon mesh representing the isosurface of an object is built up of small triangles generated within the cubes which intersect with the isosurface. The relation of the continuity of the isosurface and the behavior of the polygon edges is revealed by the following theorem [UDUP82].

Theorem: A polygon mesh defines a continuos surface if and only if each nonboundary edge occurs exactly twice, and each boundary edge occurs only once.

A nonboundary edge of a polygon mesh is *disconnected* if it occurs only once. The isosurface represented by the polygon mesh is discontinuous if disconnected edges exist in the polygon mesh. The discontinuous isosurface is exhibited as an isosurface with holes.

An edge of a polygon mesh is *multiple-branched* if it occurs more than twice. The isosurface represented by the polygon mesh contains extra pieces which don't belong to the actual isosurface if multiple-branched edges exist in the polygon mesh.

2.3.3 Isosurface Generation Methods

2.3.3.1. Face Table Method

Wyville et al. [WYVI87] developed an early isosurface generation algorithm which constructs isosurfaces to represent the so-called *soft* objects, objects whose shape changes in response to their surrounding. This algorithm generates polygons within cubes based on a face table which tells how an isosurface intersects a cubic face with respect to the polarity of the four vertices (see Figure-6).

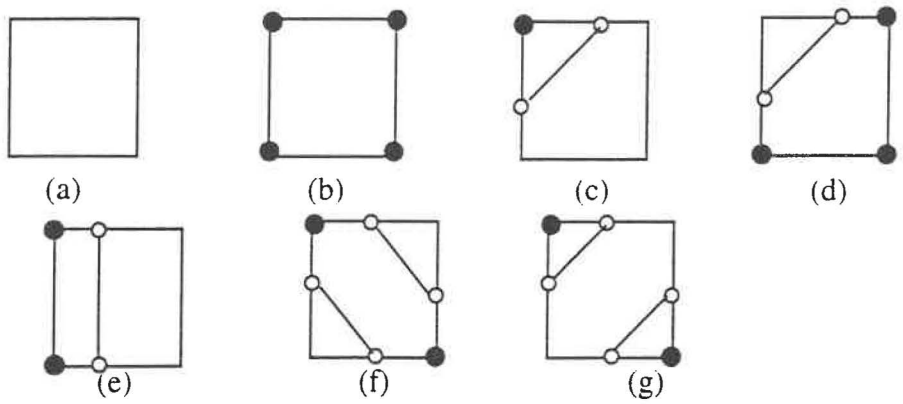


Figure-6 A face table

When the number of positive vertices on a face is four or zero, there is no surface intersection as illustrated in case (a) and (b).

When the number of positive vertices is one or three, a single line segment is created as in case (c) and (d).

There are two situations if the number of positive vertices is two. One situation is that two positive vertices connect a cubic edge and only a line segment is created as in case (e). Another possible situation is that two positive vertices are on one diagonal line and two negative vertices on another diagonal line. This situation is shown in case (f) and (g). Ambiguity arises in this situation since it is not known whether case (f) or (g) will be chosen.

The isosurface within a cube is constructed in two steps. First, the algorithm creates line segments on all the six faces of the cube. Secondly, the algorithm starts at one of the line segments, and sequentially traces its successor which has a common endpoint with the previous line segment until the algorithm comes back to the first line segment where it started.

2.3.3.2 Cube Table Method

The algorithm based on a face table which only provides the polygon connetivity on the face of a cube has to construct polygons within a cube from scratch at run-time. Some isosurface generation algorithms attempt to generate polygons from a cube table that stores the precomputed the polygon connectivity for cubes. These algorithms can be grouped into a general method which will be refered to as the cube table method which is more efficient than the face table method .

Bloomenthal [BLOO88] attempts to construct a cube table by the ordering vertices method. This method begins with any surface intersection point on an edge of a cube and proceeds towards the positive corner and then clockwise about the face to the right until another interpolation point is reached. And then the two intersection points are

connected to form an edge of a polygon. This method is repeated until the polygon is closed.

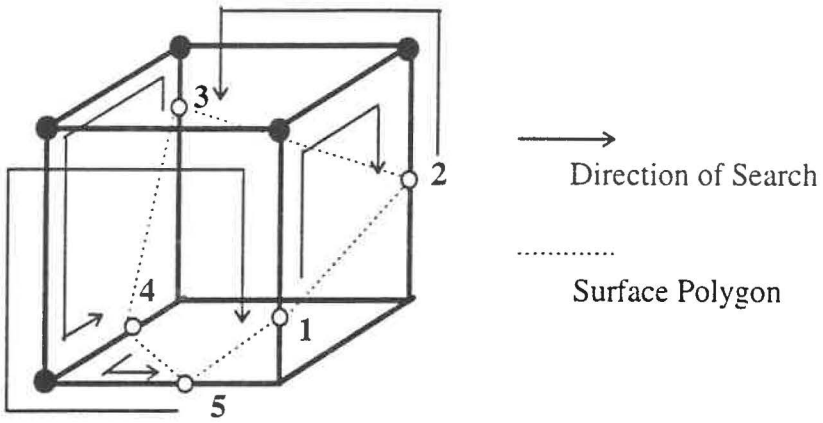


Figure-7 Ordering vertices method

Lorensen and Cline [LORE87] proposed a marching cubes algorithm which constructs the table from only 15 elementary cubes. This method is discussed in detail in the following chapter.

2.3.3.3 Dividing Cubes Method

A polygon mesh consisting of large amounts of triangles is used to represent the isosurface by many isosurface generation algorithms. However, in some cases, the resolution of the volume data is so high that the size of the triangles generated approaches the pixel size. In order to improve the efficiency of the isosurface generation by avoiding too much detail, Cline et al. [CLIN88] developed the *dividing cubes algorithm* which generates point primitives instead of triangle primitives to represent a surface.

This algorithm subdivides cubes into sub-cubes whose size is close to the pixel size on the raster display. For convenience, the term "vertex" is used to represent the 8 corners of a cube and the term "corner" is used to represent the 8 corners of a sub-cube which is obtained by the subdivision of a cube.

The algorithm calculates the gradient vector for each of the eight cube vertices . The density values at the corners of a sub-cube are calculated from the 8 vertices of a cube by linear interpolation, and the density values are used to determine whether the sub-cube is inside, outside or intersecting the surface. If a sub-cube intersects the surface, the normal vector at the center of the sub-cube is linearly interpolated from the gradient vectors of the cube vertices. The center of the sub-cube is regarded as the surface point and the normal vector at the center of the sub-cube is the surface normal at that point.

2.3.3.4 Splitting-Box Method

Muller and Stark [MULL93] proposed a splitting-box algorithm that adaptively generates the surface, i.e., adapts the size of triangles to the shape of the surface.

The splitting-box algorithm treats a 3D regular grid as a *box* whose edges are induced by linear sequences of vertices of the grid. The number of vertices of an edge is called its *length*. These authors called the edge of a box *MC* if there is at most one pair of consecutive vertices with different polarity on the edge. A box is MC if its 12 edges are all MC.

The algorithm starts with the box given by the input grid which represents the original volume data. This box is bisected perpendicular to its longest edge into two sub-boxes. Resulting boxes are recursively bisected in the same manner. The process of the bisection ends if a 2 x 2 x 2 box is reached.

Boxes arising during the process of bisection are checked to determine whether they are MC. If a box is MC, then polygons are generated for the box according to the rules of the marching-cubes configurations. The generated polygons for the box is only an approximation of the real isosurface and the quality of the approximation needs to be checked by some rules. If the approximation is acceptable, it is used as part of the output. Otherwise, the bisecting process is used to find a new approximation. At least the 2 x 2 x 2 boxes will satisfy the criterion of a satisfactory approximation.

The output of the splitting-box algorithm is a set of triangles of different size. Compared to the number of triangles generated by the cube table method, the number of triangles generated by this algorithm is significantly reduced. But this method suffers from the crack problem. When two neighboring boxes have different levels of subdivision, the surface contours generated on the splitting plane for one box might be different from the surface contours generated on the same splitting plane for another box. Therefore, cracks occur on the splitting plane.

Chapter 3

Marching Cubes Algorithm

3.1 Cubes Algorithm

3.1.1 The 15 Elementary Cases

The marching cubes algorithm was proposed by Loresen and Cline[LORE87]. It tackles the problem of isosurface construction in a local manner: the isosurface is built up of triangles generated from cubes according to 15 elementary cases. Due to its simplicity and linearity, this technique has been used in many application fields.

A cube has eight vertices, four each from adjacent planes (see Figure-8). This algorithm marches through two adjacent planes at a time, examing each cube between the two planes and generating triangles within the cube.

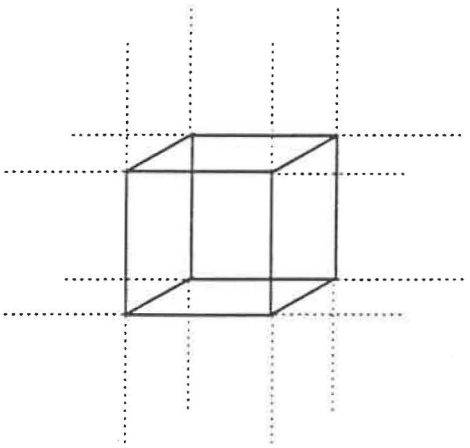
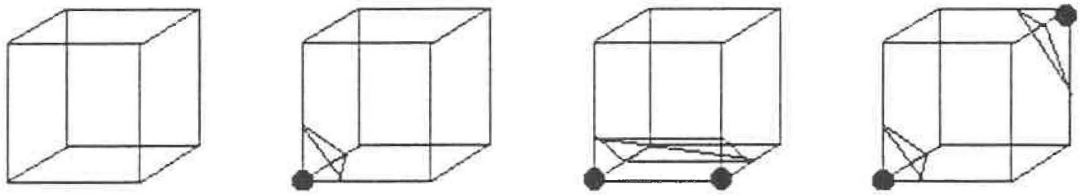


Figure-8

A vertex of a cube can be either positive or negative considering that positive means inside a surface and negative means outside a surface. A configuration of a cube represents a situation of the polarity of the eight vertices of the cube. There are $2^8 = 256$ configurations for a cube with eight vertices. But the 256 cases of cubic configuration can be reduced to only 15 elementary cases by applying two types of symmetry: inversion symmetry and rotation symmetry.

Triangles are generated within each of the 15 elementary cases. (See Figure-9)



0 1 2 3

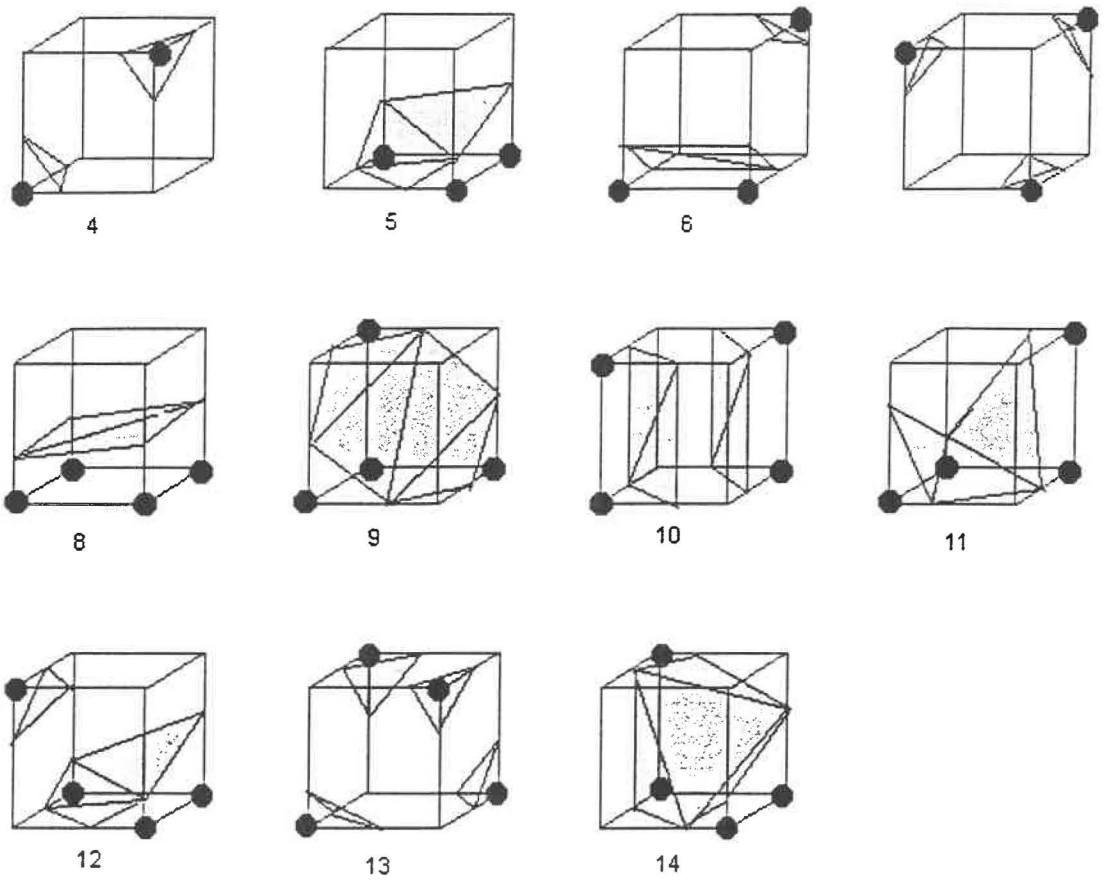


Figure-9 Triangulations of the 15 elementary cases of the marching cubes algorithm

3.1.2 Inversion Symmetry

Inversion symmetry implies that two configurations are equivalent if the positive and negative vertices are switched. That is, configurations with five, six, seven or eight positive vertices are equivalent to configurations with three, two, one or zero positive vertices. Due to inversion symmetry, only configurations with zero to four positive vertices need to be considered. This reduces 256 cases to 128 cases.

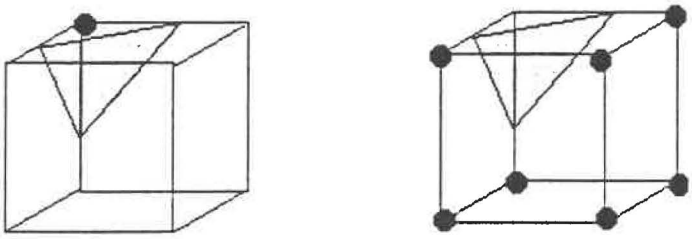


Figure-10 Inversion symmetry: the same surface triangle is generated whether one vertex or seven vertices are positive

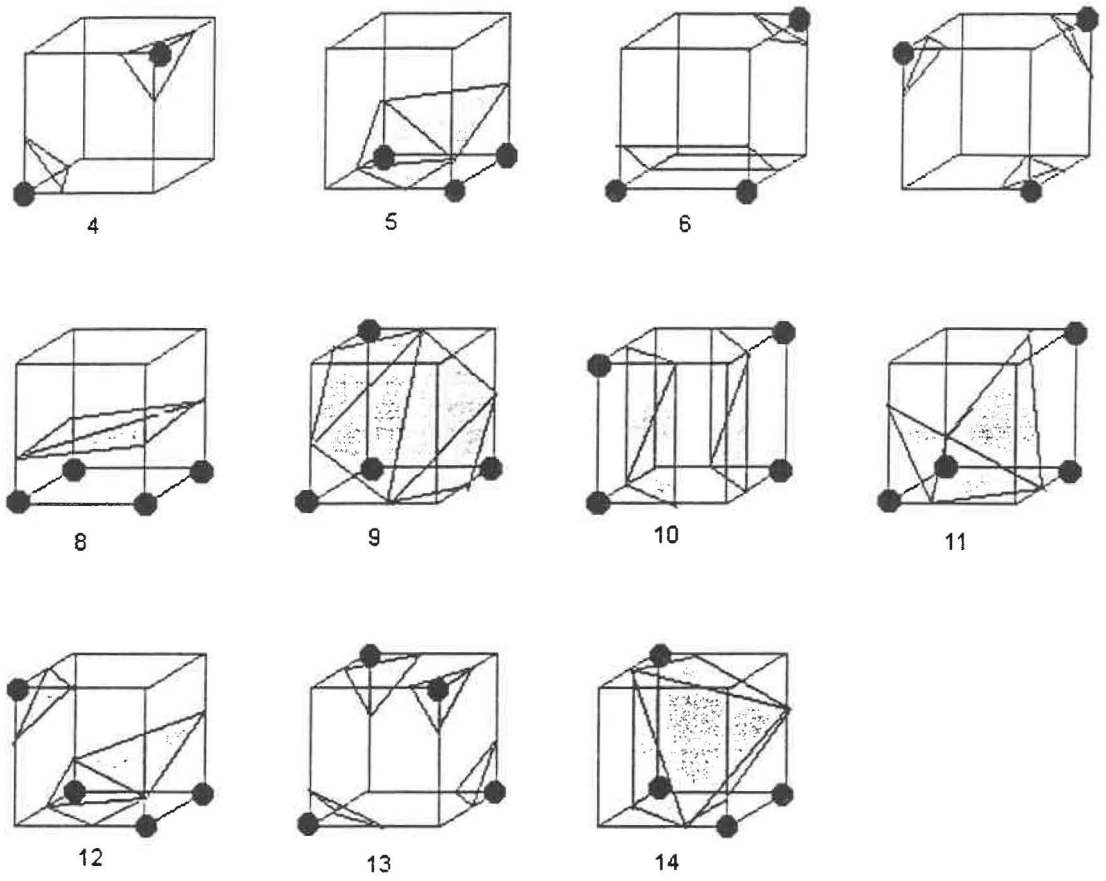


Figure-9 Triangulations of the 15 elementary cases of the marching cubes algorithm

3.1.2 Inversion Symmetry

Inversion symmetry implies that two configurations are equivalent if the positive and negative vertices are switched. That is, configurations with five, six, seven or eight positive vertices are equivalent to configurations with three, two, one or zero positive vertices. Due to inversion symmetry, only configurations with zero to four positive vertices need to be considered. This reduces 256 cases to 128 cases.

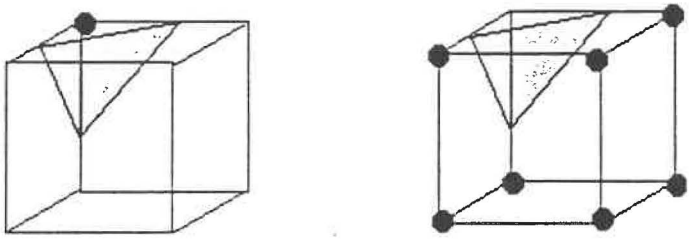


Figure-10 Inversion symmetry: the same surface triangle is generated whether one vertex or seven vertices are positive

3.1.3 Rotation Symmetry

Futhermore, two configurations are also equivalent if one is obtained from another by some kind of cubic rotation. There are 24 distinct cubic rotations which may be counted by finding all axes of symmetry together with the number of distinct rotations about each axis.

First, label a cube as Figure-11 where V_k and V_k' denote a pair of diagonal vertices of the cube ($1 \le k \le 4$). One of the rotations is the identity rotation. According to the type of symmetry axis, the other 23 cubic rotations can be classified as follows:

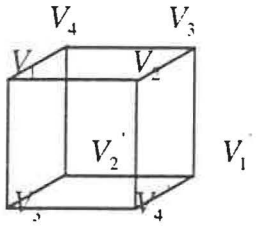


Figure-11

Type 1: Let the diagonal joining V_k and V_k' be the axis (see Figure-12):

There are four axes of this type Marching where k = 1,2,3,4. A cube can be rotated by 120° and 240° by fixing each axis. Altogether, there are 4*2=8 such rotations.

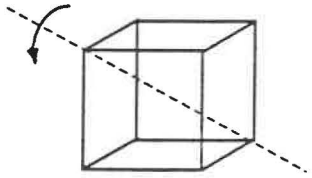


Figure-12

Type 2: Let the line joining centers of opposite faces be the axis (see Figure-13):

There are 3 pairs of opposite faces for a cube: front and back, bottom and up, left and right. We can fix the centers of each pair , then rotate the cube by ±90° and 180°, which provides 3*3=6 such rotations.

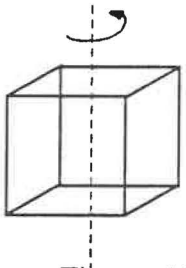


Figure -13

Type 3: Let the line connecting the midpoints of opposite edges be the axis (see Figure-14):

The 6 pairs of opposite edges of a cube are: V_1V_2 and $V_1'V_2'$, V_1V_3' and $V_1'V_3$, V_1V_4 and $V_1'V_4'$, V_2V_3 and $V_2'V_3'$, V_2V_4' and $V_2'V_4$, V_3V_4 and $V_3'V_4'$. Fixing each axis, a cube can be rotated by 180°. There are 6*1=6 rotations of this type.

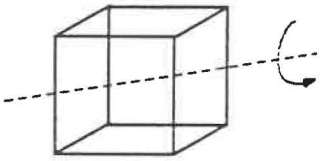


Figure-14

It has been proved that all the cubic rotations form a *rotation symmetry group* that is isomorphic to the symmetric group S_4 [ARMM88]. The idea in proving this theorem is to label the four diagonal lines of a cube V_kV_k' (k=1,2,3,4) (see Figure-11)as four numbers 1, 2, 3, 4 , and then to show that every cubic rotation corresponds to just one of the permutations of the diagonals, and vice versa.

By taking the rotation symmetry of a cube into account, the 128 cases with zero to four positive vertices is reduced to only 15 elementary cases.

3.1.4 Implementation of the Marching Cubes Algorithm

3.1.4.1 Equivalence Classes of the 15 Elementary Cases

It is mentioned previously that the 128 cases with zero to four positive vertices can be reduced to the 15 elementary cases by rotation symmetry. Actually, the 128 cases are partioned into 15 equivalence classes and each equivalence class corresponds to one of the 15 elementary cases. For example, the equivalence class of case 1 contains 8 cases of configurations which are equivalent under rotation symmetry (see Figure-15).

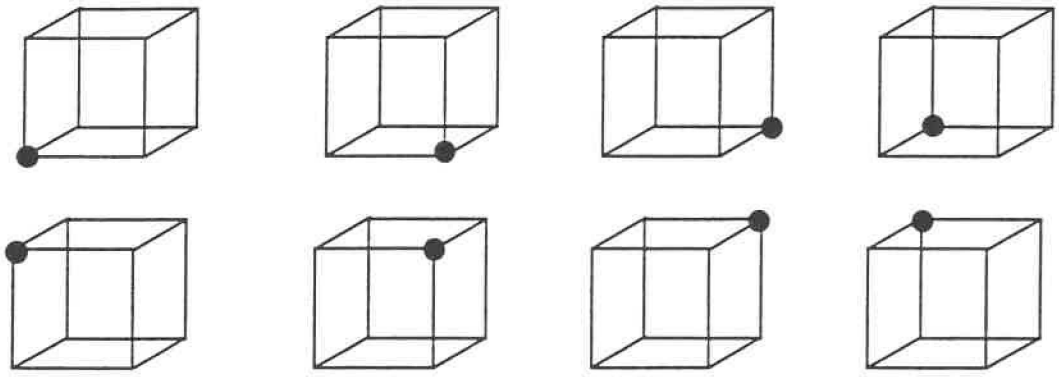


Figure-15 The equivalence class of elementary case 1 under rotation symmetry

In the implementation of the marching cubes algorithm, these equivalence classes should be generated automatically from the 15 elementary cases (see Appendix A for detail).

3.1.4.2 A Look-up Table

Configurations in the same eqivalent class should yield the same general shape of isosurface. For example, every configuration in the equivalence class of case 1 in Figure-15 yields only one triangle as in Figure -16.

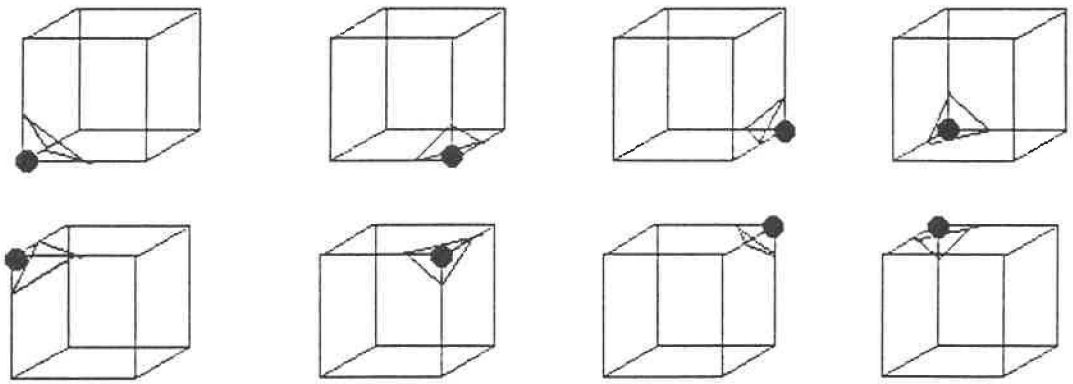


Figure-16 Triangulations in the equivalence class of elementary case 1

Once the triangulation of an elementary case is given , the triangulations of the other configurations in the same equivalence class are generated similarly. Therefore, the triangulations of all the 128 cases with zero to four positive vertices can be generated from the triangulations of the 15 elementary cases.

By inversion symmetry, the triangulations of 256 cases of a cube will be generated from the 128 cases.

A look-up table is created to store all the triangulations of 256 cases.

An index is created for each case, based on the polarity of the vertex. Using the vertex number in Figure-17 (a), the eight bit index contains one bit for each vertex. For example, wazzu the index of a cube configuration as in Figure-17 (b) is 00000001 where 1 represents positive and 0 represents negative .

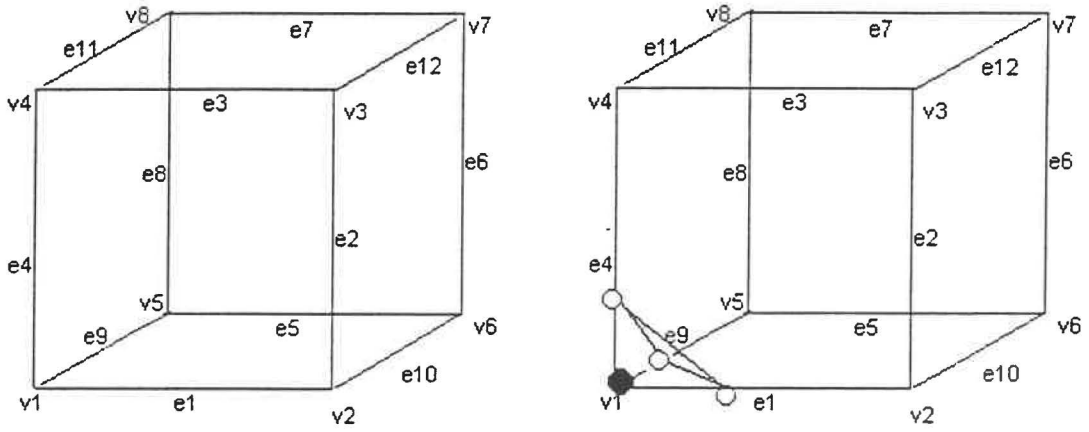


Figure-17 Cube Numbering

The index serves as a pointer to the look-up table which gives all edge intersections for a given cube configuration. For example, using 00000001 as a pointer , the triangulation of the cube configuration in Figure-17 (b) given by the look-up table will be {1, 4, 9} which indicates the isosurface intersects the cube at edges 1, 4 and 9.

3.1.4.3 Linear Interpolation of a Triangle Vertex

A vertex of a generated triangle is the intersection point of the isosurface with a cubic edge. Suppose two adjacent vertices of an edge are p, q which have density values of f_p and f_q. The location of p is (x_1, y_1, z_1) and the location of q is (x_2, y_2, z_2). The threshold value defined for the isosurface is C. If q is positve and p is negative, then the isosurface intersects the edge pq. The location of the intersection point is (x, y, z) which can be caculated by linear interpolation as follows:

$$m = \frac{C - f_p}{f_q - f_p}$$
$$x = x_1 + m(x_2 - x_1)$$
$$y = y_1 + m(y_2 - y_1)$$
$$z = z_1 + m(z_2 - z_1)$$

Although this 'linear interpolation' is not the true intersection, it is much cheaper to calculate and, provided the cubes are small enough that the polygon approximation to the surface is reasonable, it is good enough.

In order to produce a smooth-shaded image, the marching cubes algorithm also caculates the gradient vector of a triangle vertex by linearly interpolating between the gradient vectors of the two vertices of the cubic edge. The gradient at a cube vertex (x_0, y_0, z_0) is estimated using central differences along the three coordinated axes as:

$$g_x = \frac{f(x_0 + 1, y_0, z_0) - f(x_0 - 1, y_0, z_0)}{2}$$
$$g_y = \frac{f(x_0, y_0 + 1, z_0) - f(x_0, y_0 - 1, z_0)}{2}$$
$$g_z = \frac{f(x_0, y_0, z_0 + 1) - f(x_0, y_0, z_0 - 1)}{2}$$

where f represents the density function.

3.2 Ambiguity Problem in the Marching Cubes Algorithm

Unfortunately, the marching cubes algorithm is susceptible to the ambiguity problem which can result in discontinuous polygonal surfaces or surfaces with "holes".

A face of a cube is ambiguous if it contains two positive and two negative vertices on the diagonal lines. In this paper, *positive diagonal line* is used to represent the diagonal line connecting the two positive vertices and *negative diagonal line* is used to represent the diagonal line connecting the two negative vertices. One intersection point lies on each of four edges of a face. There are two possible ways to connect the intersection points which induce two topologically different surfaces in Figure-18 . The left shows that two separated surfaces are generated by connecting the intersection points in the same direction as the positive diagonal line; the right indicates that one continuous surface is created by joining the intersection points in the same direction as the negative diagonal line.

Figure-18 An ambiguous face

An ambiguous cube is a cube that has at least one ambiguous face. When two neighbor cells make inconsistent connectivity decisions among the common ambiguous face, a discontinuous surface with holes could be generated.

The ambiguity problem of the marching cubes algorithm was first pointed out by Durst [DURS87] illustrated as in Figure-19. Two adjacent cubes share an ambiguous face. The left continuous cube has only two positve vertices which is case 3 of the marching cubes algorithm. The right cube has six positive vertices which happens to be the inverted case of the left cube. In the left cube, the intersection points are connected in the same direction as the negative diagonal line; in the right cube, the intersection points are joined in opposite direction as the left. A quadrilateral hole arises in the constructed isosurface because of the inconsistent connections on the common face.

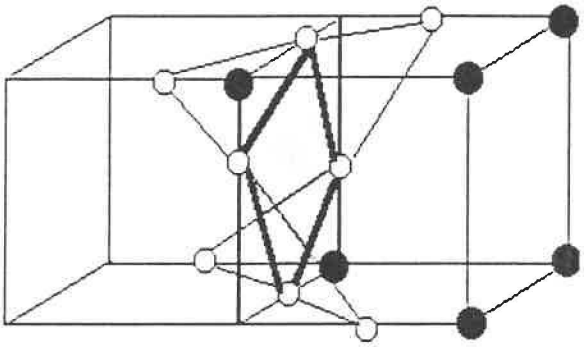


Figure-19 A hole in the isosurface

Among the 15 elementary cases of the marching cubes algorithm, case 3 and 6 contain one ambiguous face, case 10 and 12 contain two ambiguous faces, case 7 and 13 contain 6 ambiguous faces. These cases are called the ambiguous cases of the marching cubes algorithm. Isosurfaces generated from these ambiguous cases may contain holes.

3.3 Disambiguation Techniques

Several techniques have been developed to solve the ambiguity problem of the marching cubes algorithm. *Tetrahedral decomposition* and *topology inference* are the two primary techniques.

3.3.1 Tetrahedral Decomposition

The ambiguity problem can be solved if an ambiguous cube can be decomposed into unambiguous tetrahedrons and the isosurface is generated from those tetrahedrons [NING93].

A cube can be decomposed into five or six tetrahedrons without requiring additional vertices. Figure-20 gives a decomposition of a cell into five tetrahedrons.

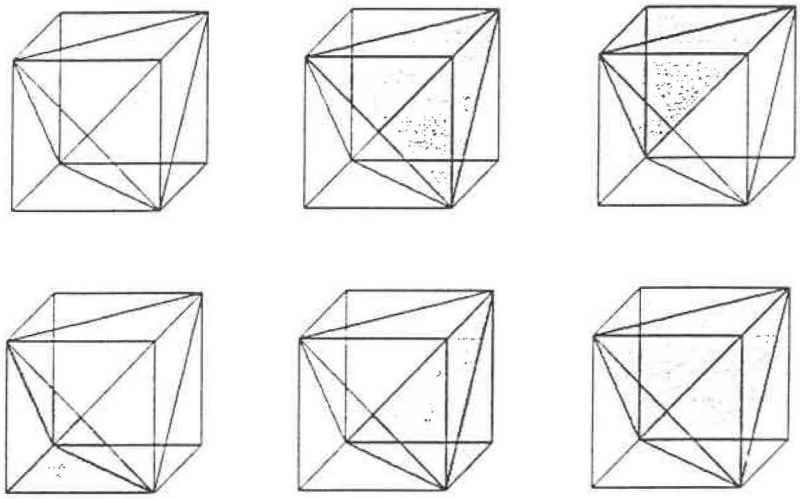


Figure-20 Decomposition of a cell into five tetrahedrons

If a vertex can be either positive or negative, there are altogether $2^4 = 16$ configurations of a tetrahedron. By rotation symmetry, the 16 cases can be reduced to only 5 cases given in Firgure-15. No triangle is generated within the tetrahedron in case (a) or (e), one triangle is generated for the tetrahedron in case (b) or (d), and two triangles are generated for case (c). There is no ambiguity in any case. That means, tetrahedrons are unambiguous.

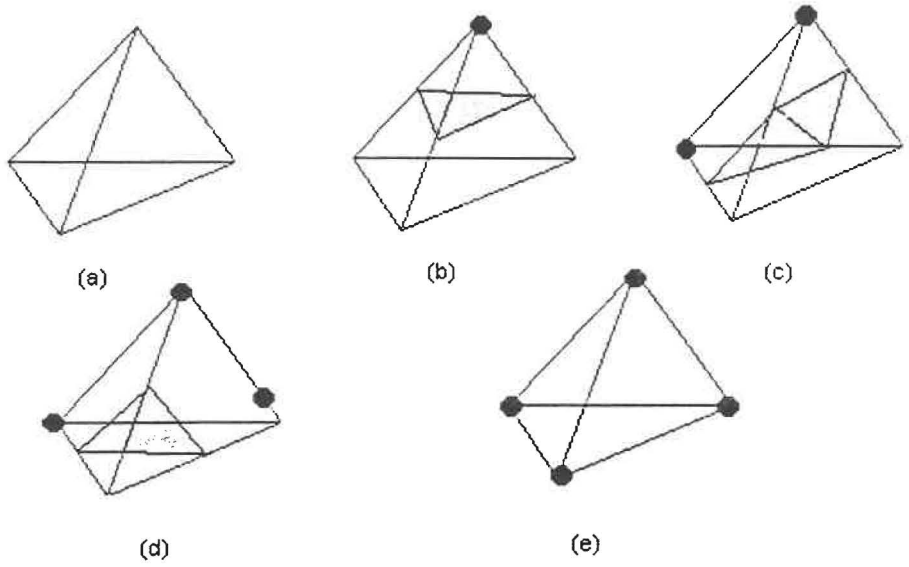


Figure-21 Triangulations of tetrahedrons

These decompositions into tetrahedrons introduce diagonals on the cubic faces. When a face is ambiguous, the diagonals on the face helps to resolve the ambiguity as in Figure-22. The orientation of the diagonal decides how to connect the intersection points on the face.

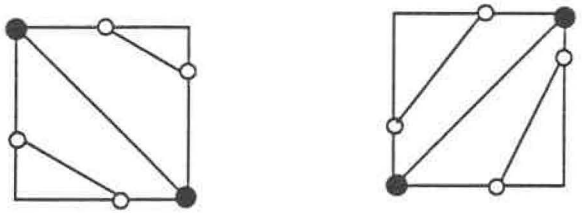


Figure-22 Diagonals on the ambiguous faces

As long as neighboring cubes are decomposed so that they share common tetrahedral faces at their boundaries, a consistent polygonization will result and a isosurface will be generated. For example, Payne et al. [PAYN] decompose adjacent cubes into mirror-image patterns to assure the continuity of the isosurface. (See Figure-23)

Figure-23 Mirror image cubes

The implementation of tetrahedral decomposition within the marching cubes algorithm first decomposes each of the 15 elementary cases into five tetrahedrons, and then generates triangles within those tetrahedrons to form the isosurface for the case.

It needs to be pointed out that it is not enough to only decompose the ambiguous cases of marching cubes into tetrahedrons in the implementation. The reason is that inconsistent surface contours might be generated on the common face of two adjacent cubes if one of the cubes is decomposed into tetrahedrons, and the other is not.

An example in Figure-24 shows the isosurface generated within tetrahedrons from the decomposition of a cube. The left represents the triangulation of case 1 of the marching cubes algorithm which contains only one triangle. The right represents the triangulation of case 1 when the cell is decomposed into five tetrahedrons. The isosurface within the right cell is composed of four triangles.

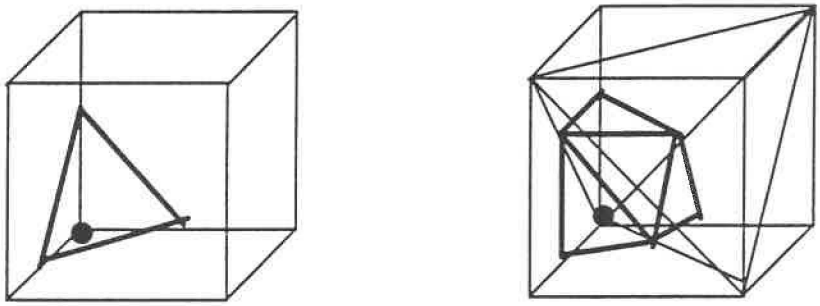


Figure-24 Isosurface generated within tetrahedrons

One obvious disadvantage of tetrahedral decomposition is that more triangles are generated than the original marching cubes algorithm According to experimental results, tetrahedral decomposition yields over twice as many triangles as does the original marching cubes technique.

3.3.2 Topology Inference

Rather than decomposing a cube into unambiguous tetrahedrons, some methods attempt to infer the correct topology of an ambiguous face from some data values calculated at run time, then construct the appropriate isosurfaces for the face [NING93]. The three principal inference schemes are *facial center value, gradient heuristics* and *bilinear contours* [WYVI86] [NIEL91] [WILL90]. All of these methods make consistent inferences across adjoining cells, therefore, no holes result.

(1) Facial Center Value:

In the facial center value method, the value of the center of an ambiguous face is approximated by the average value of the four vertices of the face, then the topology of the isosurface for the face is determined by the polarity of the center value [WYVI86].

Suppose the ambiguous face is on the x-y plane and its four corners are represented as (0,0), (1,0), (0,1), (1,1). Then the value of the center is:

$f_c = \frac{1}{4}(f_{00} + f_{10} + f_{01} + f_{11})$ where f_{00}, f_{10}, f_{01}, f_{11} are the density values of the four corrners of the ambiguous face.

The center has positive value if its value is greater than the threshold value; otherwise it is negative if its value is less than the threshold value. The sign of the center determines how the intersection points are to be connected. (see Figure-25)

Figure-25 If the center is positive, the intersection points are joined in the same direction as the positive diagonal line; if the center is negative, the intersection points are joined in the same direction as the negative diagonal line .

(2) Gradient Heuristics:

Gradient heuristics use the gradients at the cell vertices to decide the topology of the isosurface in an ambiguous case [WILL90]. The gradient direction is normal to the isosurface, and its magnitude indicates how rapidly the underlying function of the isosurface is changing. An advantage in using gradients is that they are also needed for the shading model, so no additional computational cost is incurred to calculate them. One representative method of gradient heuristics is *center-pointing gradient.*

The center-pointing gradient is defined to be the component of the gradient in the direction from the cell vertex toward the center of the face. Only the component of the gradient in the plane of a face is calculated. Suppose a face is in the x-y plane and the corners of the face are indexed as (0,0), (1,0), (0,1), (1,1). f_{ij} represents the density value and $\nabla f_{xij}, \nabla f_{yij}$ represent the gradients in the x and y directions at a point (i, j) where $i, j = 0,1$. Then the estimate of the function of the isosurface at the center of the face is:

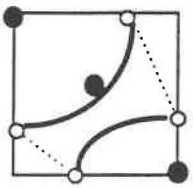
$$f_c = \frac{1}{4}(f_{00} + f_{10} + f_{01} + f_{11})$$

$$+\frac{1}{16}\left(\nabla f_{x00} + \nabla f_{y00} - \nabla f_{x10} + \nabla f_{y10} + \nabla f_{x01} - \nabla f_{y01} - \nabla f_{x11} - \nabla f_{y11}\right)$$

Comparing the formula of f_c here with the formula of f_c in the facial center value method, the center-pointing gradient method can be thought of as providing a "correction term" to the f_c calculated by the facial center value method. Therefore, the center-pointing gradient method calculates the value of the center in a more accurate way than the facial center value method does.

(3) Bilinear Contours:

Both the facial center value method and the center-pointing gradient method attempt to use the value of the center of the face to make consistent decisions on the common face which guarantees the generated isosurface is continous. However, this continous surface is not necessarily correct all the time. For example, in Figure-26, although the value at the face center indicates that the intersection points should be connected in a way such that a connected isosurface is formed on the face, actually the real isosurface consists of two separated parts.



---- isosurface decided by facial center value

—— real isosurface

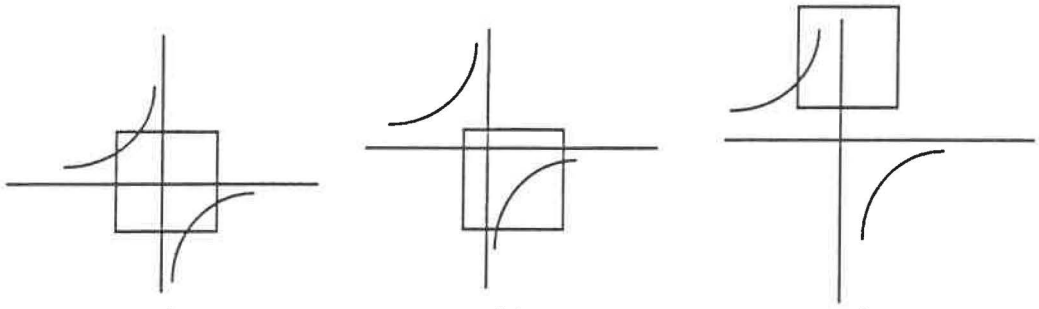
Figure-26

An alternative technique, the bilinear contours method, makes an effort to infer the topology of the surface from the saddle point of a hyperbola on an ambiguous face [NIEL91].

This approach is based on the assumption that the isosurface on a cubic face can be bilinear interpolated. Suppose a cubic face can be represented as a unit square $\{(s,t):0 \le s \le 1,0 \le t \le 1\}$ and the values of the corner vertices are $f_{00}, f_{10}, f_{01}, f_{11}$. Then the bilinear interpolation of a point (s,t) on the face is

$$f(s,t) = (1-s, s)\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix}\begin{pmatrix} 1-t \\ t \end{pmatrix}$$

It can be verified that for a threshold value C, the set $\{(s,t): f(s,t) = C\}$ which represents the isosurface contour on the face, is a hyperbola. Some possibilities as to how these contour hyperbolas and their asymptotes relate to the cell faces are shown in Figure-27



a) b) c)

Figure-27 Countours of bilinear interpolation

In case a), the ambiguity arises when both components of the hyperbola intersect the face. The disambigutity decision is based on the sign of the saddle point (the intersection point of the asymptotes) of the hyperbola. The value of the saddle point fp can be calculated as:

$$fp = \frac{f_{00}f_{11} + f_{10}f_{01}}{f_{00} + f_{11} - f_{01} - f_{10}}$$

If the saddle point has a positive value, the isosurface generated for the face is illustrated at the left of Figure-28. If the saddle point is negative, then the isosurface is connected as illustrated at the right of Fibure-28.

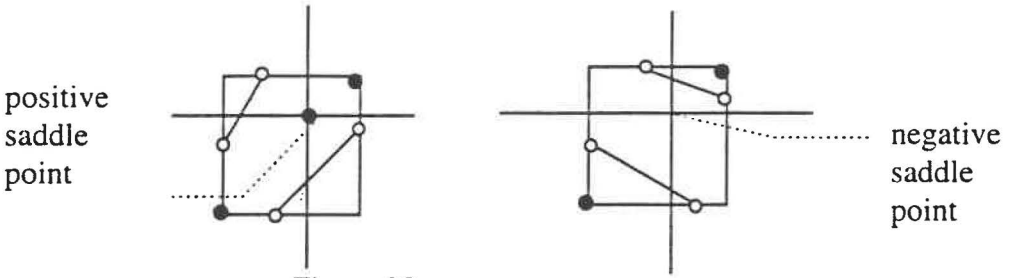


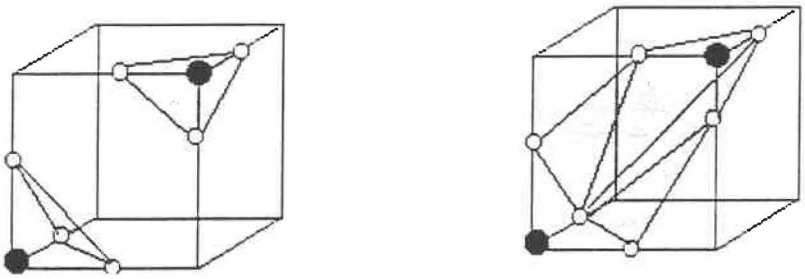
Figure-28

(4) Implemtation of Topology Inference:

Although the means of calculating the data used for decision making on an ambiguous face are different in various topology inference methods, the implementations of translating these decisions into actual polygon connectivity are very similar. The idea of these implementations is to create two kinds of tables: a major table which stores the 15 elementary cases of the marching cubes algorithm and some subcase tables which are used to handle the ambiguous cases [WILL90] [NIEL91].

If a case is unambiguous, the major table simply contains the polygon connectivity of this case. However, if a case is ambiguous, the major table contains a list of entry to a subcase table for this ambiguous case. The subcase table stores different kinds of polygon connectivity corresponding to all the possible decisions made from the status of the ambiguous face.

For example, case 3 in an ambiguous case with only one ambiguous face. The major table is connected to a subcase table of case 3 which contains two different kinds of polygon connectivity corresponding to two possible decisions made for case 3 (see Figure-29).



decision 0: connecting negative vertices

decision 1: connecting positive vertices

Figure-29 Two subcases of elementary case 3

The complexity of the subcase table increases dramatically when the number of ambiguous faces in an ambiguous case increases. For example, in case 13, all six faces of the cube are ambiguous. Since two decisions can be made for each ambiguous face, the subcase table needs to have $2^6 = 64$ different polygon connectivity corresponding to all possible decisions for case 13. The creation of a subcase table for case 13 is very difficult due to its great complexity.

Chapter 4

An Eifficent Disambiguation Marching Cubes Algorithm

4.1 Objectives of Isosurface Generation Algorithms:

Many isosurface generation techniques in the literature were designed for specific applications. As a result, they may have implicit assumptions about the nature of the data that would not hold in another application. As applications proliferate, it becomes important to have a general-purpose method that is free of application dependencies. Toward this end, a number of desirable features of a polygonal isosurface generation algorithm are identified as follows:

1. The algorithm should yield a continuous isosurface.
   Each edge of a polygon mesh should be shared by exactly two polygons or lie in an external face of the entire volume. Therefore, the isosurface approximated by the polygon mesh should not contain any holes.
2. The algorithm should decide consistently which positive vertices belong to the same object. Positive vertices belonging to the same object are to be covered by the same surface.
3. The algorithm should generate a smooth isosurface.
   The gradients of the surface should be calculated for a gradient shading to produce a smooth, pleasing and realistic isosurface.
4. Algorithmic construction of the polygon connectivity table is desired.
   The polygon connectivity for each case entry needs to be generated automatically based on the algorithm and to be stored in a look-up table for fast access. Manual construction of the table is tedious and prone to error.
5. The isosurface should not create artifacts not implied by the data.
   Extraneous polygons which are the results of the existence of multiple branched edges in the polygon mesh should not be generated in the isosurface.
6. The algorithm should be simple wazzu and efficient.
   The algorithm should be as simple as possible for easy implementation. Also the algorithm should be efficient enough for real-time interactive use.

Some of these criteria may not seem important when the resolution is fine enough that the eye does not notice an occasional "glitch". However, visualization systems will inevitably provide a zoom ability for a close-up examination of "interesting" features of a scene. Isosurfacess generated incorrectly can lead to misleading or at least confusing images under close-up examination, regardless of the original resolution [WILL90].

The simplicity and fast speed of the marching cubes algorithm makes it a good candidate for a general-purpose isosurface generation algorithm. Actually, the marching cubes algorithm successfully fulfills most of the objectives except the first one. The failure of the first objective is due to the ambiguity problem of the algorithm which might allow holes in a constructed isosurface.

As described in chapter 2, the two major disambiguation techniques, tetrahedral decomposition and topology inference, have been developed to resolve the ambiguity problem. While the first objective is satisfied when these techniques are applied, some other objectives which indicate the advantages of the marching cubes algorithm are lost.

According to experimental results, tetrahedral decomposition yields over twice as many triangles as does the marching cubes technique. The generation and display of these large amounts of triangles is extremely computational expensive which makes it difficult for effective interactive use.

In topology inference, the creation of different subcase tables for all the ambiguous cases is very complicated and time consuming since these tables are usually generated manually. This can not meet the fourth objective which requires the automatic generation of look-up tables to prevent errors from occuring in the polygonization. Besides, topology inference requires additional computation in order to make a decision on an ambiguous face which lowers the efficiency of the algorithm.

4.2 An Efficient Disambiguation Method

4.2.1 Durst method

When Durst pointed out the ambiguity problem, he suggested a modified method which always adds a quadrilateral to the polygon mesh whenever there is an ambiguous face. Unfortunately, this method could sometimes lead to multiple-branched edges which creates extraneous polygons in the isosurface. An example is given in Figure-30:

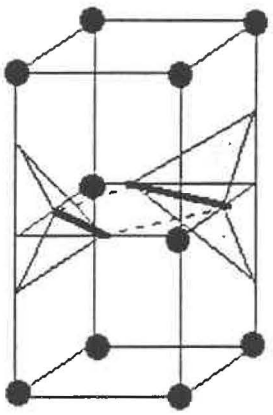


Figure-30 A quadrilateral is added on common face

In Figure-30, each of the two edges on the common ambiguous face occurs twice when the original marching cubes algorithm is used. However, the additional

quadrilateral added by this modified method increases the number of occurrences of each edge by one. Therefore, the two edges turn into multiple-branched edges which are not allowed in the isosurface generation.

4.2.2 An Efficient Disambiguation Method

In spite of the fact that the modified method by Durst fails to solve the ambiguity problem completely, the method provides some clues leading to the development of a very efficient disambiguation method [ROLL95].

The example in Figure-30 has shown that holes do not always occur on an ambiguous face. However, if there is a way to find out whether or not holes occurr on an ambiguous face and to fill the holes only when they occur, then the isosurface produced is a continuous isosurface without holes. Thus, the ambiguity problem is solved.

The formation of holes is due to disconnected edges in the polygon mesh representing the isosurface. An edge of a polygon mesh lies either on the face of a cube or inside a cube. An edge inside a cube occurs exactly twice in the polygon mesh because this edge is created by the tessellation of a nonplanar polygon into triangles (see Figure-31). Therefore, holes may only arise on the faces of a cube where the edges are disconnected .

An edge inside a cube always occurs exactly twice in two neighboring triangles

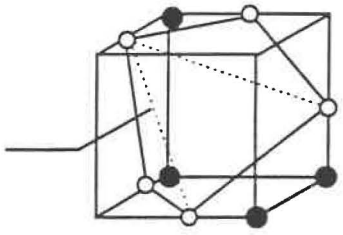


Figure-31 Polygon edges

Since two cubes share a common face, the number of occurrences of the triangle edges on the common face needs to be examined from a combination of these two adjacent cubes.

Two adjacent cubes sharing an ambiguous common face are illustrated in Figure-32. Three parallel faces of the two cubes are named as upper face, common face and lower face. Keeping the vertex configuration of the common face fixed, the possible combinations

of the other vertices (four on the upper face and four on the lower face) are $2^8 = 256$.

The number of 256 possible combinations can be reduced to a smaller number of distinct combinations by making use of two types of symmetry.

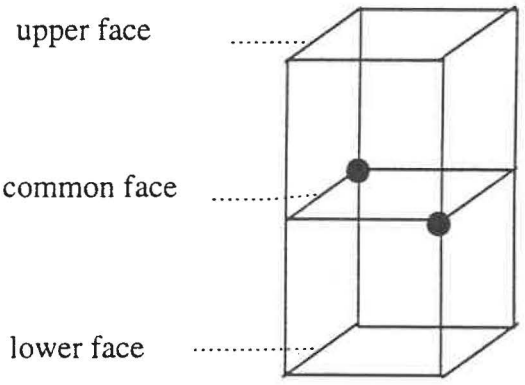
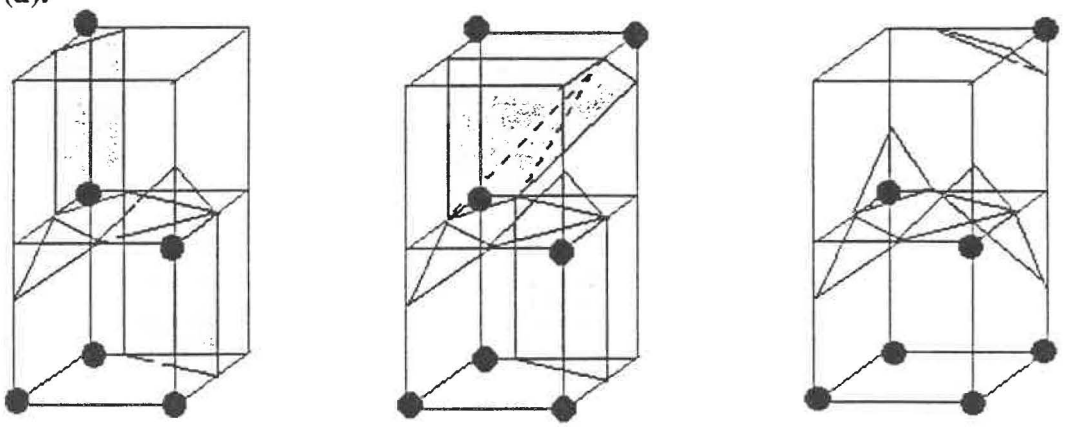


Figure-32 Two adjacent cubes sharing an ambiguous face

Swapping symmetry is defined by interchanging the upper and the lower face. That is, the combination of i positive vertices in the upper face and of j in the lower face, is equivalent to the combination of j positive vertices in the upper face and of i in the lower face. It reduces 256 combinations into 128. Another symmetry, rotation symmetry, is defined by fixing the centers of the upper face and the lower face and then rotating the two cubes by 180°. By this symmetry, the number of the distinct combinations is further reduced to only 64.

For each of the 64 distinct combinations, polygons are generated within the upper and lower cube according to the 15 elementary cases of the marching cubes algorithm. The number of times each triangle edge occurs on the common face of each combination is then examined. Some combinations are illustrated as in Figure-33 . A combination is regarded as containing holes if some triangle edges occur only once on the common face. The result of the examination of the 64 combinations is given in Table-1. The number of combinations means the number of different vertex arrangements.

(a).



(b).

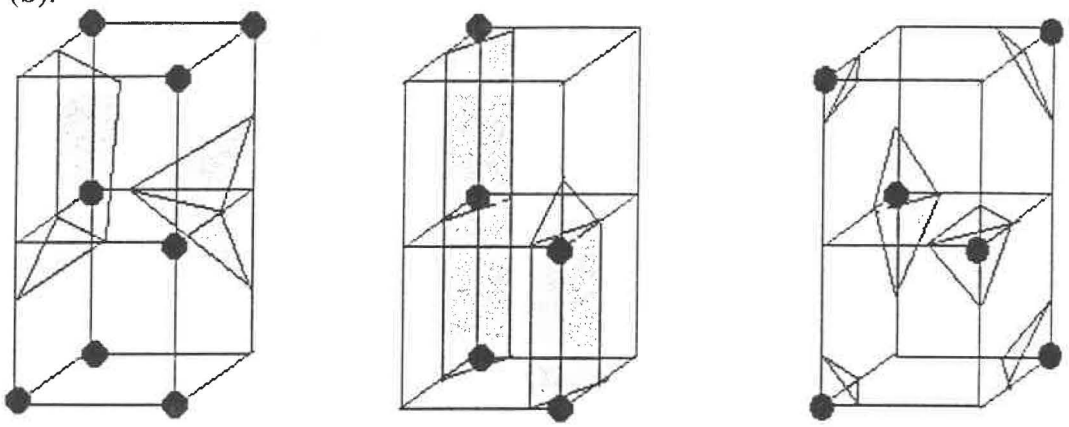


Figure-33 The Polygon Connectivity of Some Combinations

Number of combinations

Number of positive vertices in the upper cube

Number of positive vertices in the lower cube

	2	3	4	5	6
2	1	2	3	2*	1*
3	-	6	12	6*	2*
4	-	-	9	8*	3*
5	-	-	-	6	2
6	-	-	-	-	1

Table-1 22 out of the 64 combinations exhibit holes which are marked by *.
(Note: the symbol - represents symmetries data.)

In Table-1, the combinations in which holes are presented (numbers marked by *) are those having at least 5 positive vertices in the upper cube and no more than four positive vertices in the lower cube. By the swapping symmetry, the combinations with at least 5 positive vertices in the lower cube and no more than four positive vertices in the upper cube also contain holes on the common face. An inverted case is a case that has five or six positive vertices and a non-inverted case is a case that has two, three or four positive vertices. A very interesting conclusion can be drawn from the above observation: *a hole arises if and only if one of the adjacent cubes in the combination represents an inverted caseand the other represents a non-inverted case.*

A combination with an inverted case and a not-inverted case is called *inverted combination*. Figure-33 (a) shows some examples of inverted combination. On the common face, four edges forming a quadrilateral remain unpaired which induces holes. Figure-33 (b) represents some other examples where an inverted case meets an inverted case, or a not-inverted case meets a not-inverted case. All edges are properly connected. Separated isosurfaces are constructed on the common face and no holes arise.

Inspection of all combinations in Table-1 reveals another simple property, which implies how the holes are to be mended: *all holes consist of four disconnected edges forming a quadrilateral on the common face.* Therefore, a hole is mended by filling the four intersection points of the ambiguous face with two triangles forming a quadrilateral.

A disambiguation marching cubes algorithm is implemented based on the above results. The local isosurface construction within a cube by this method can be summarized in three steps as follows:

(1) Initial isosurface generation.
Triangles are first generated within the examined cube according to the 15 elementary cases of the marching cubes algorithm. The construction process continues if the cube represents an ambiguous case; otherwise, it stops here.

(2) Hole detection.
For each ambiguous face of the examined cube, the neighbor cube sharing the ambiguous face also needs to be inspected in order to determine whether or not the combination is inverted. The process continues if it is inverted which indicates the existence of holes; otherwise, it stops.

(3) Hole fixing
Holes on the ambiguous face of an inverted combination can be mended by adding a quadrilateral which consists of two triangles to the initial isosurface generated in the first step. All the triangles form the isosurface of the examined cube.

The implementation of the last two steps, hole detection and hole fixing is very direct, simple and rapid based on Table-1. These two steps can be viewed as an extension of the local construction of the marching cubes algorithm and are only needed when a cube is ambiguous. Consequently, This disambiguation method efficiently generates a continous isosurface without holes while keeping the simplicity and fast speed of the original marching cubes algorithm.

Chapter 5

Test Results

5.1 Implementations of Three Isosurface Algorithms

Three isosurface generation algorithms which are the original marching cubes algorithm, Durst algorithm and the disambiguation algorithm described in chapter 4 are implemented. Generated data from mathematical functions are used to test the algorithms, and the test results are compared and explained. Also the isosurface of a medical volume data is generated using the disambiguation algorithm

5.2 Two Mathematical Functions Test:

5.2.1 Definition of F_1 and F_2

The following two quadratic functions are very useful in testing the ability of isosurface generation algorithms [WILL90].

$$F_1(x,y,z) = 4y + 4(x-z)^2 - 5$$

$$F_2(x,y,z) = 4(y-1)^2 + 2(x-z)^2 - 2(x+z-3)^2 + 1$$

The actual isosurfaces for real numbers ranging from 0 to 3 in the three dimensional space are shown in Figure-34 (a) and (b). The threshold value that defines the two isosurfaces is zero. The isosurface of F_1 is a single continuous surface, and the isosurface of F_2 is two separate lobes of a hyperboloid.

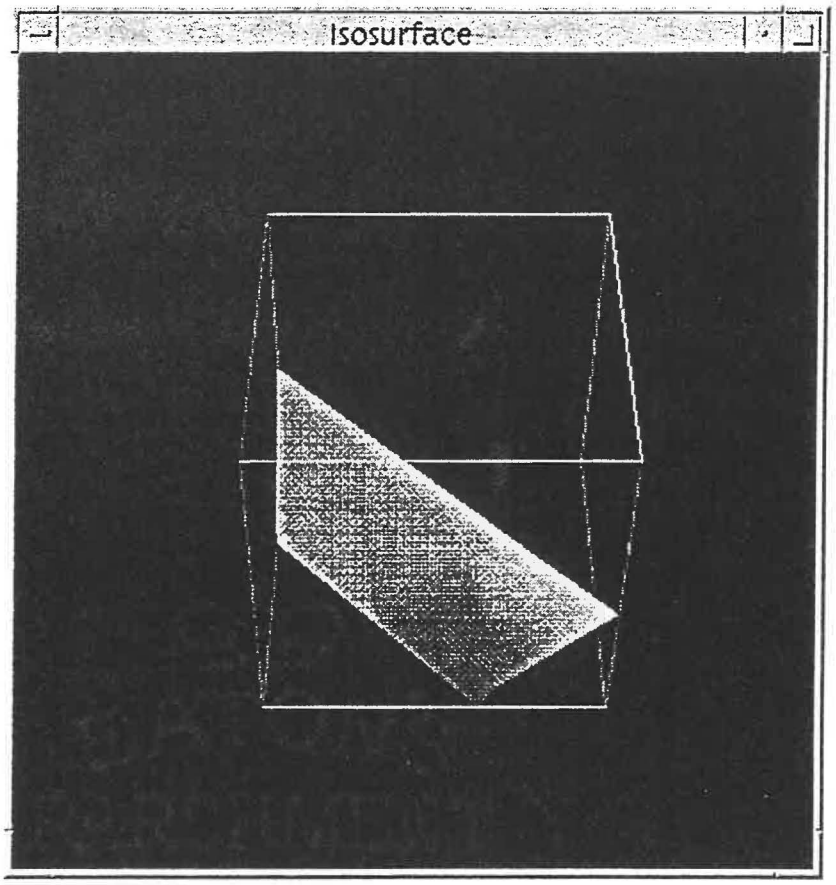


Figure-34 (a) The actual isosurface of F_1

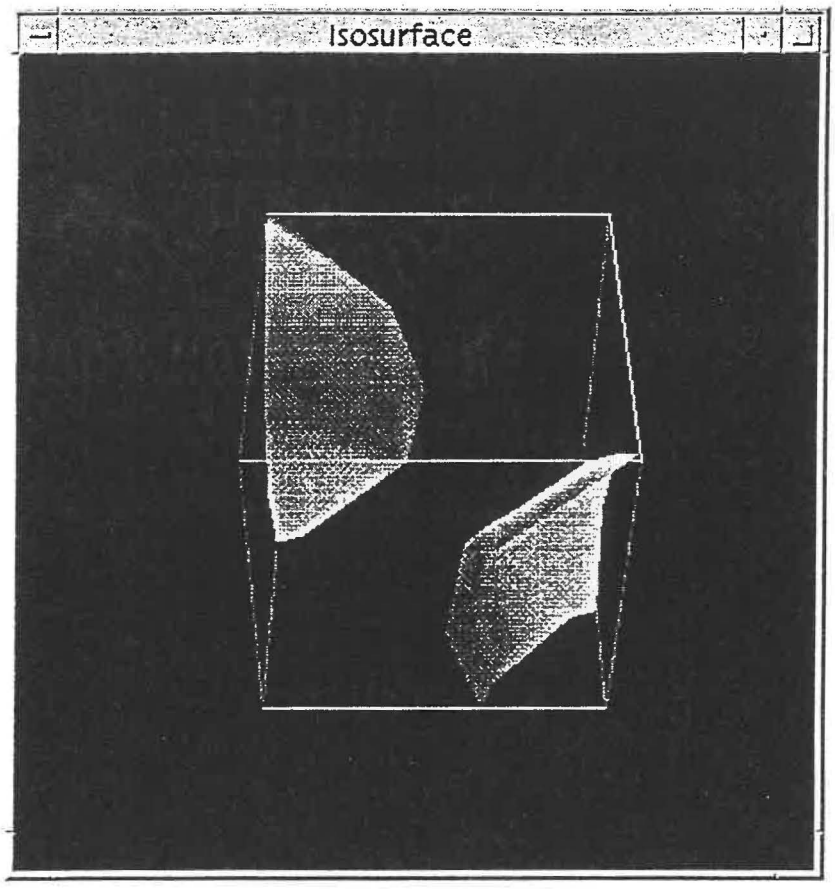


Figure-34 (b) The actual isosurface of F_2

Each of the two functions is sampled at the integers 0 through 3 for x, y and z respectively to produce a $4 \times 4 \times 4$ volumetic data which will be used to test several isosurface generation algorithms.

5.2.2 Center Cube and Center-Lower Cube of F_1 and F_2

Figure-35 shows the vertex values of the center cube and the center-lower cube for F_1 and F_2. The center cube is actually the cube which locates at the center of the volume which consists of three cubes in each of the three dimensions. The center cube of F_1 has exactly the same vertex values as the center cube of F_2. As a result, the ambiguous face between the center cube and the center-lower cube of F_1 is the same as the ambiguous face of F_2. The polygon connectivity within those cubes by the original marching cubes algorithm is also illustrated in the Figure-35.

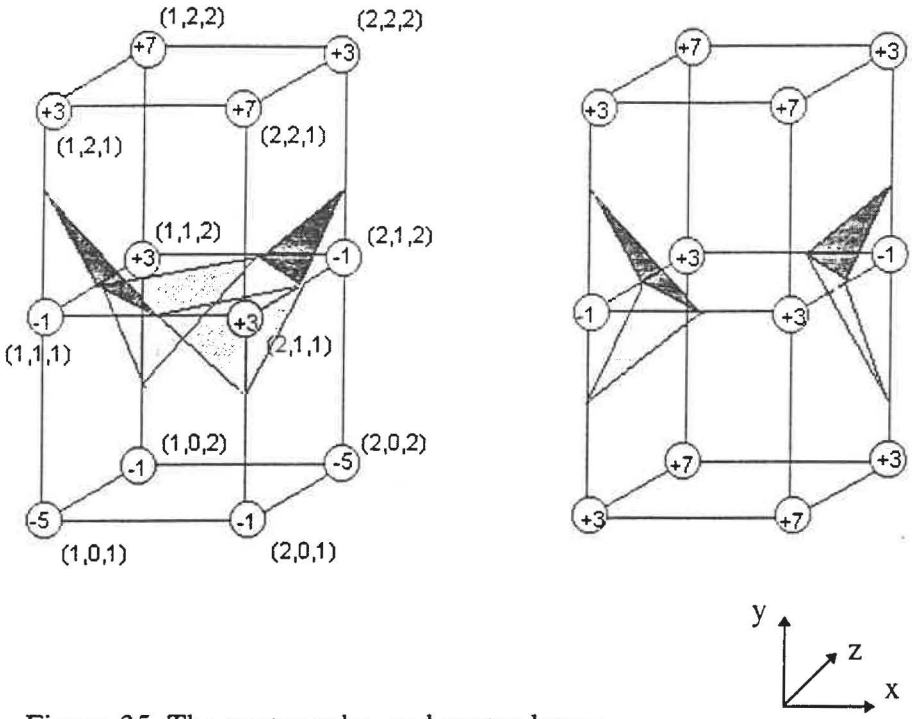


Figure-35 The center cube and center-lower cube of F_1 (left) and F_2 (right)

(Note: the (x,y,z) coordinates of F_2 is the same as the coordinates of F_1)

5.2.3 Test Results of Isosurface Generation Algorithms

5.2.3.1 Test Results of the Original Marching Cubes Algorithm

Isosurfaces which are constructed by the original marching cubes algorithm from the two volume data sets of F_1 and F_2 are demonstrated in Figure-36 (a) and (b):

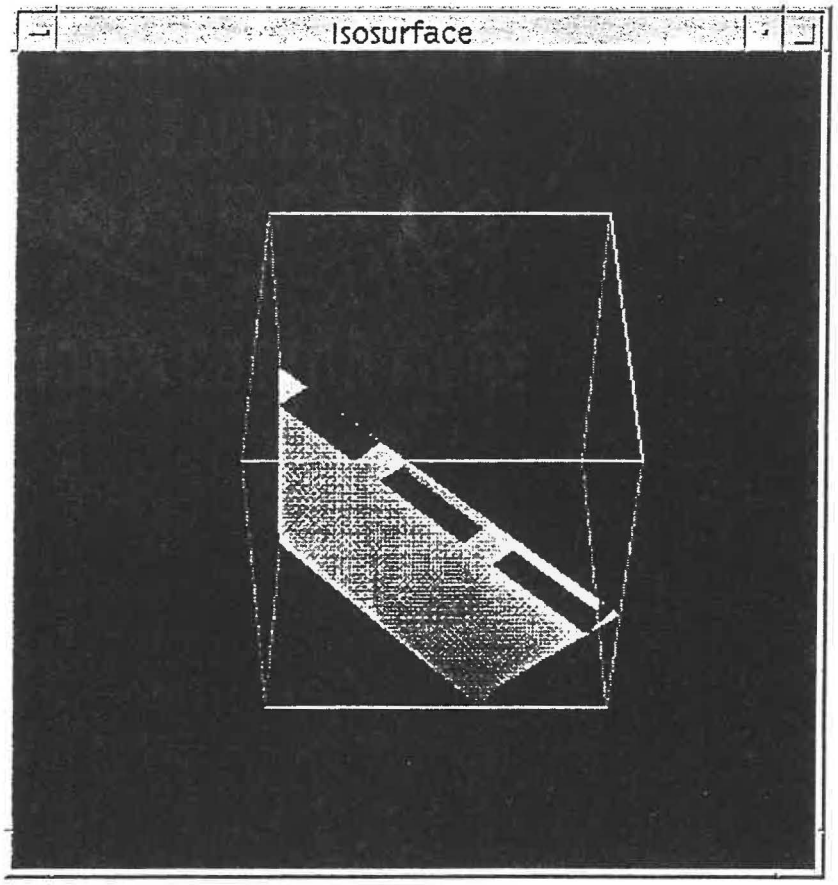


Figure-36 (a) The generated isosurface of F_1 by the original marching cubes method

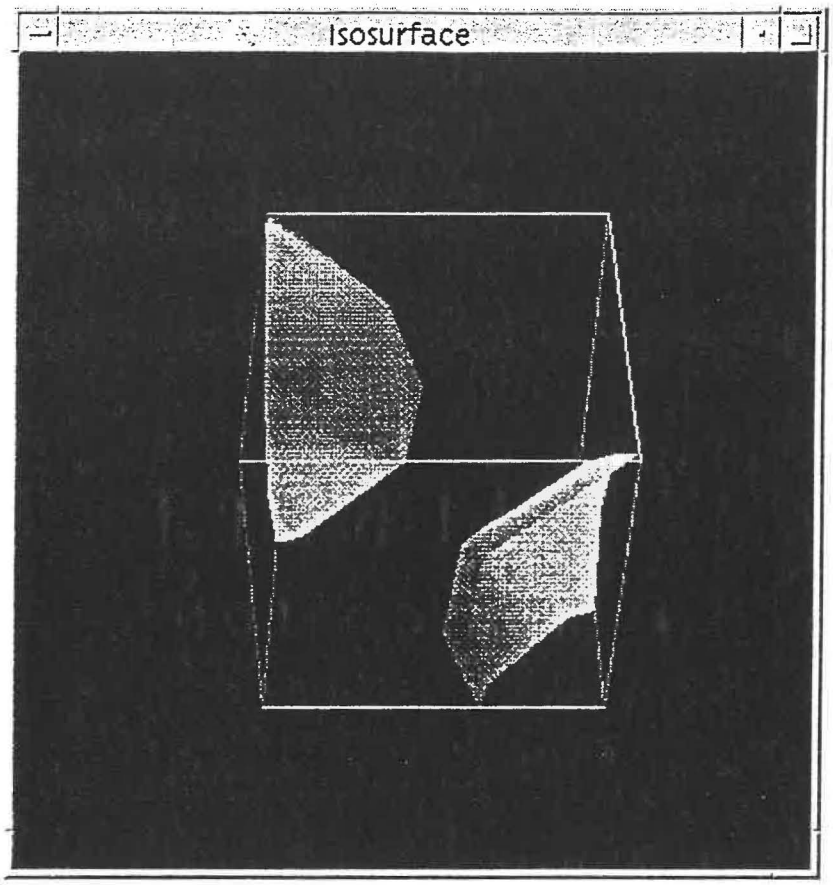


Figure-36 (b) The generated isosurface of F_2 by the original marching cubes method

Compared with the actual isosurface of F_1 in Figure-34 (a), the constructed isosurface of F_1 exhibits holes, which proves the ambiguity problem of the marching cubes algorithm pointed out by Durst. The reason for the exhibition of holes in the constructed isosurface can be explained by the center cube and the center-lower cube of F_1 in Figure-35, left. Each of the four edges on the common ambiguous face is a disconnected edge, i.e. an edge which occurs just one time in the polygon mesh. By the theorem discussed in chapter 2, holes are induced on the ambiguous face.

Nevertheless, the constructed isosurface of F_2 doesn't have any holes, and it is very similar to the actual isosurface of F_2. This can also be interpreted by the center cube and the center-lower cube of F_2 in Figure-35, right. Since all the edges on the common face are connected correctly (each edge occurs exactly twice in the polygon mesh), there are no holes on the common face even if the face is ambiguous. Thus, the constructed isosurface doesn't contain any holes.

5.2.3.2 Test Results of Durst Algorithm

The volume data of F_1 and F_2 are also used to test the modified marching cubes algorithm suggested by Durst. The generated isosurfaces are illustrated in Figure-37 (a) and (b).

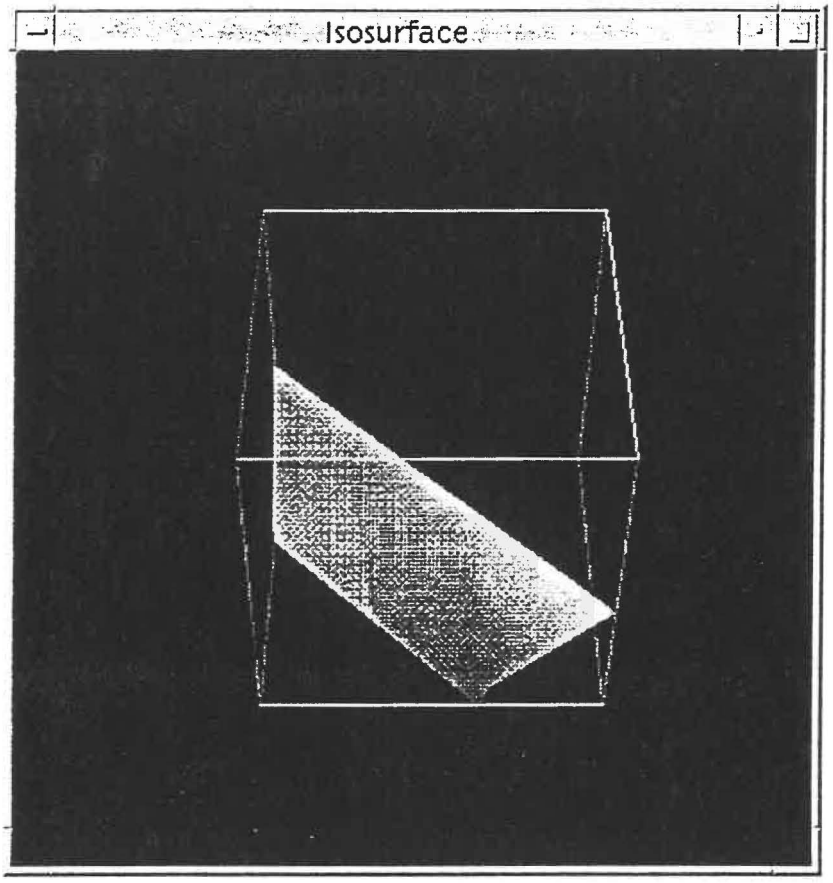


Figure-37 (a) The generated isosurface of F_1 by Durst method

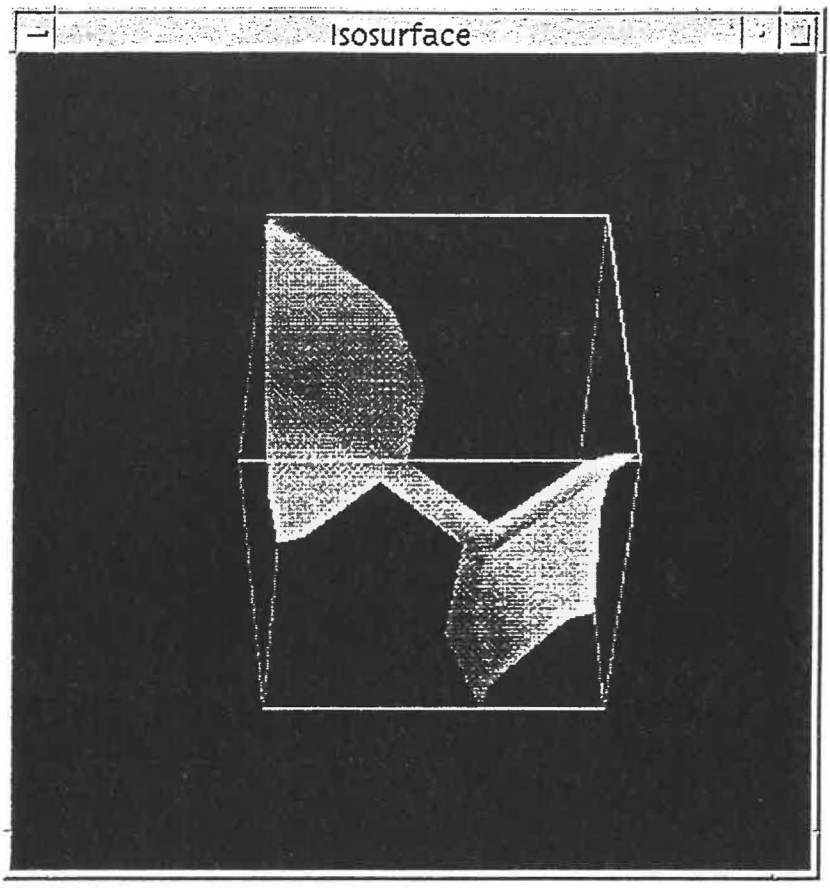


Figure-37 (b) The generated isosurface of F_2 by Durst method

This method successfully repairs the holes for F_1. But it adds two extraneous polygons between two lobes of the hyperboloid for F_2.

It has been shown previously that all the edges occur only once on the common face of the center cube and the center-lower cube of F_1 by the original marching cubes algorithm. When the Durst method adds a quadrilateral on the common face, all the disconnected edges (occuring just one time) on the common face are connected properly (occuring exactly twice). Hence, the isosurface of F_1 turns out to be a continuous surface without holes. However, although all the edges on the ambiguous common face of the center cube and the center-lower cube of F_2 have already occurred twice by the original marching cubes algorithm, an additional quadrilateral is still supplemented on the common face by the Durst method. As a result, the edges on the common face occur three times which creates extraneous polygons in the isosurface.

5.2.3.3 Test Results of the Disambiguation Algorithm

The disambiguation marching cubes algorithm described in chapter 4 is also used to generate isosurfaces for F_1 and F_2 as illustrated in Figure-38.

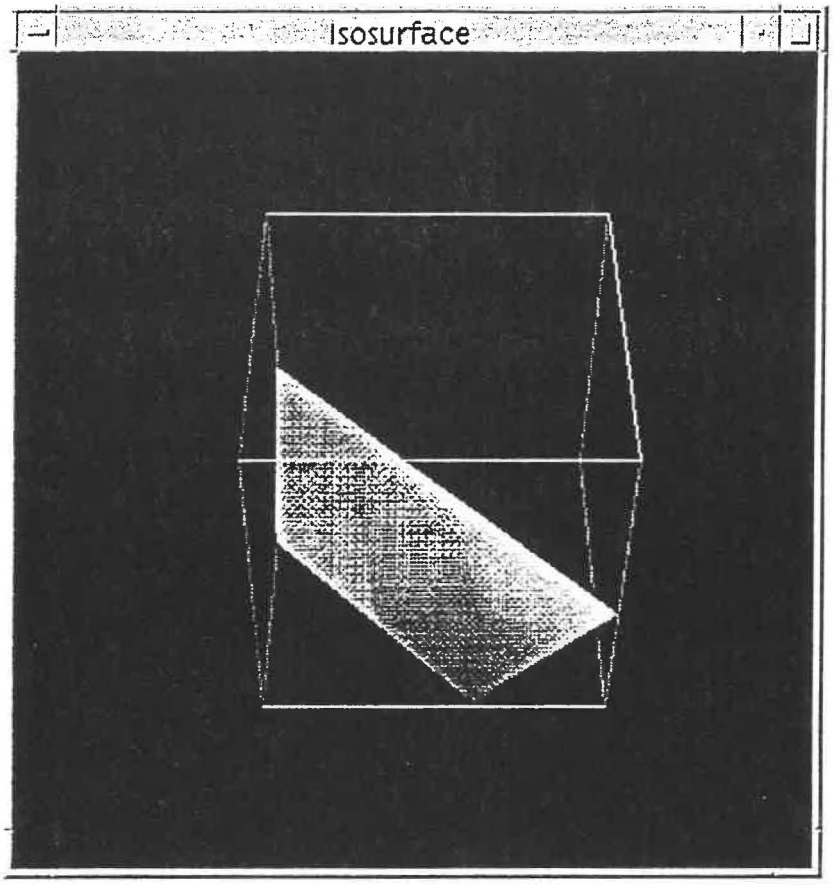


Figure-38 (a) The generated isosurface of F_1 by the disambiguation method

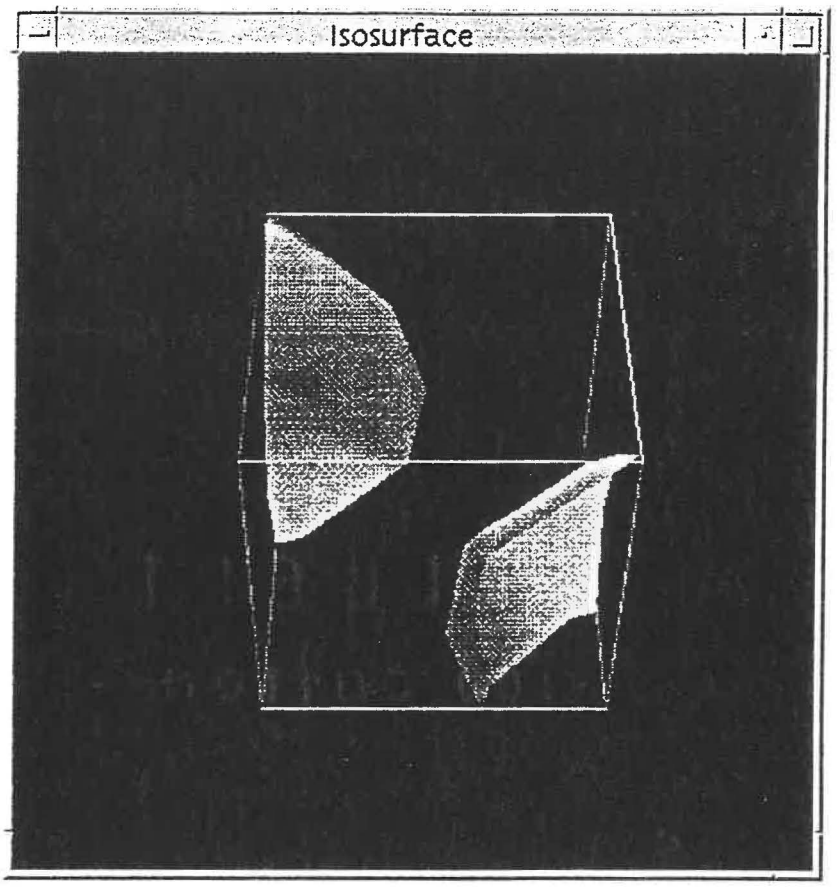


Figure-38 (b) The generated isosurface of F_2 by the disambiguation method

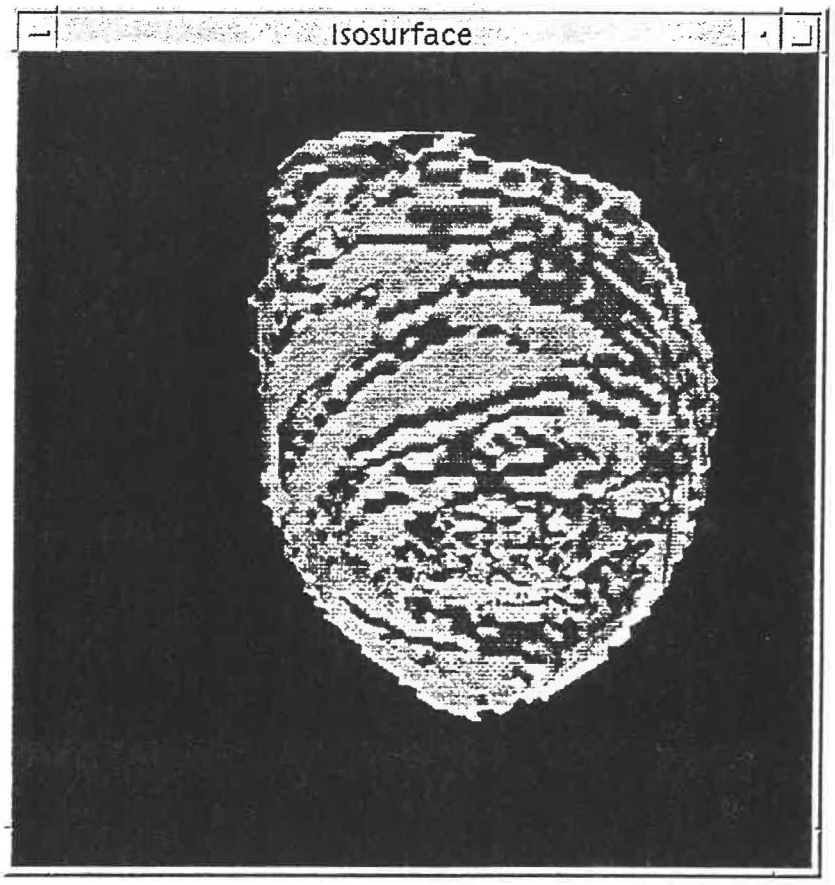
The outcome of this disambiguous method is very heartening. Both the generated isosurfaces are very close to the actual isosurfaces. There is no hole in F_1 and there is no extra piece in F_2.

In the case of the center cube and the center-lower cube of F_1, the center cube is an inverted case which has six positive vertices, and the center-lower cube is a non-inverted case which has two positive vertices. The combination of these two cubes is inverted which implies that there are holes on the common face. The disambiguation method mends the holes by generating a quadrilateral on the common face.

As for the case of the center cube and the center-lower cube of F_2, both the center cube and the center-lower cube are inverted since each cube has six positive vertices. Then the combination of these two cubes is not inverted and there are no holes on the common face. Therefore, the isosurface within the two cubes is generated only by the original marching cubes algorithm and no quadrilateral needs to be added on the common face.

5.3 Medical Experimental Result:

The experimental result of generating an isosurface from a medical volume data by the disambiguation marching cubes algorithm is shown in Figure-39 :



The medical volume data is a presegmented Cine-CT of the left ventricle of the heart. The dimensions of the data is $85 \times 80 \times 18$. The data set contains values between 0 and 255. The threshold value used to determine the isosurface in Figure-39 is 79.0 which is chosen based on the histogram of the data so that the result shows the boundary of the heart as well as the structure inside. The isosurface consists of 23648 triangles, among which 23572 triangles are generated by the original marching cubes , and 76 triangles are generated to mend the holes. The number and the frequency of the 15 elementary cases of the volume data is illustrated in Table-2.

	Number of Cases	*Percentage (%)*
Case 0	1009877	89.51
Case 1	3009	2.67
Case 2	3165	2.81
*Case 3**	85	0.075
Case 4	2	0
Case 5	2175	1.92
*Case 6**	32	0.028
*Case 7**	2	0
Case 8	2987	2.65
Case 9	293	0.26
*Case 10**	2	0
Case 11	42	0.037
*Case 12**	2	0
*Case 13**	0	0
Case 14	38	0.034
Total Cases	112821	

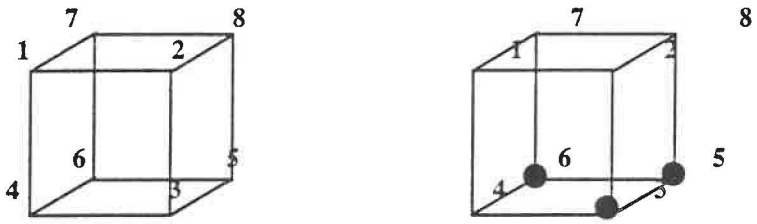
Table-2 The number and frequency of the 15 elementary cases
(Note: numbers marked with * represent ambiguous cases)

It takes about two minutes to generate the polygon mesh representing the isosurface from the heart data on a Sun Workstation using the disambiguation marching cubes algorithm. And it takes about another one minutes to render the polygon mesh to get the image as in Figure-39 on the same machine. The isosurface generated by the disambiguation algorithm is continuous without any holes. And the speed of this algorithm is reasonable and practical. Therefore, the disambiguation algorithm can be used in many scientific applications.

Appendix A

The Generation of the Equivalent Classes of the 15 Elementary Cases of the Marching Cubes Algorithm

If a cube is labeled as Figure-40(a), then the configuration of a cube can be represented as a subset of {1,2,3,4,5,6,7,8}. For example, a configuration as Figure-40(b) can be represented as {3, 5,6} which only contains positive vertices.



(a) (b)

Figure-40

Then all the 15 elementary cases of the marching cubes algorithm can be represented by a set M =={{}, {4}, {4,3}, {2,4}, {4,8}, {3,5,6}, {4,3,8}, {1,3,8}, {3,4,5,6}, {4,5,6,7}, {1,4,5,8}, {4,5,6,8}, {1,3,5,6}, {4,5,2,7}, {3,5,6,7}}. Each element of set M corresponds to one of the elementary cases.

It is known that the rotation symmetry group is isomorphic to symmetry group S_4. It is also isomorphic to a subgroup of S_8 which consists of some permutations of 1,2,3,4,5,6,7,8.
In this paper, a permutation of S_8 represented as [2,3,4,1,6,7,8,5] is the same as

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 4 & 1 & 6 & 7 & 8 & 5 \end{pmatrix}.$$

If a cube is labeled as Figure-40, a subgroup of S_8 which is isomorphic to the rotation symmetry group can obtained:
S={[2,3,4,1,6,7,8,5], [6,5,8,7,2,1,4,3], [1,4,6,7,5,8,2,3], [2,8,5,3,6,4,1,7], [5,6,4,3,1,2,8,7], [2,1,7,8,6,5,3,4], [5,3,2,8,1,7,6,4], [6,7,1,4,2,3,5,8], [4,1,2,3,8,5,6,7], [7,6,5,8,3,2,1,4], [8,2,1,7,4,6,5,3], [1,7,8,2,5,3,4,6], [5,8,7,6,1,4,3,2], [3,2,8,5,7,6,4,1], [4,6,7,1,8,2,3,5], [7,8,2,1,3,4,6,5], [7,1,4,6,3,5,8,2], [4,3,5,6,8,7,1,2], [6,4,3,5,2,8,7,1], [3,4,1,2,7,8,5,6], [8,5,3,2,4,1,7,6], [8,7,6,5,4,3,2,1], [1,2,3,4,5,6,7,8], [3,5,6,4,7,1,2,8]}

The equivalence class of each elementary case is obtained by letting the group S act on the elementary case . For an example, case 4 is represented as {4,8} and its equivalence class is {{3,7}, {8, 4}, {5,1}, {6,2}}. Case 5 is represented as {6,5,3} and its equivalence class is { {7,6,5}, {2,8,7}, {7,4,6}, {7,8,1}, {8,7,6}, {5,8,7}, {5,6,8},{2,3,8}, {4,3,6}, {3,6,5}, {6,1,7}, {5,8,3}, {8,2,5},{4,6,5}, {5,4,3}, {7,4,1}, {8,2,1}, {1,6,4}, {5,3,2}, {2,1,7}, {4,3,1}, {2,4,3}, {1,2,4}, {1,3,2}}

REFERENCES

ARMS88 M. A. Armstrong, "Groups and Symmetry", Springer-Verlag, 1988.

BLOO88 J. Bloomenthal, "Polygonization of Implicit Surfaces", Computer Aided Geometric Design, Vol. 5, No. 4, Nov. 1988, pp. 341-355.

CHEN85 Chen, G. R. Herman, R. A. Reynolds and J. K. Udupa, "Surface Shading in the Cuberille Environment", Computer Graphics and Applications, Vol. 5, No.12, Dec. 1985, pp. 33-43.

CLIN88 H. E. Cline, W. E. Lorensen, and S. Ludke, "Two algorithms for the three-dimensional reconstruction of tomograms", Medical Physics, Vol. 15, No. 3, May/June 1988, pp. 320-327.

DURS88 M. J. Durst, "Additional Reference to Marching Cubes", Computer Graphics, Vol. 22, No. 2, April 1988, pp. 72-73.

FUCH77 H. Fuchs, Z. M. Kedem and S. P. Uselton, "Optimal Surface Reconstruction from Planar Contours", Communications of the ACM, Vol. 20, Oct. 1977, pp. 693-702.

HERM79 G. T. Herman and H. K. Liu, " Three-Dimensional Display of Human Organs from Computed Tomograms", Computer Graphics and Image Processing, Jan. 1979, pp. 1-21.

KAUF91 A. Kaufman, "Introduction to Volume Visualization", Volume Visualization, IEEE Computer Scoiety Press Tutorial, 1991, pp. 1-18.

KAUF93 A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics", Computer, July 1993, pp. 51-64.

LIU77 H. K. Liu, "Two and Three-Dimensional Boundary Detection", Computer Graphics and Image Processing, No. 6, June 1977, pp. 123-124

LORE87 W. E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics, Vol. 21, No. 4, July 1987, pp. 163-169.

MULL93 H. Muller and M. Stark, "Adaptive Generation of Surfaces in Volume Data", Visual Computer, Vol. 9, No. 4, 1993, pp. 182-199

NIEL91 G. M. Nielson and B. Hamman, "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes", Visualization '91, IEEE, 1991, pp. 83-90.

NING93 P. Ning and J. Bloomenthal, "An Evaluation of Implicit Surface Tiles", IEEE Computer Graphics, Vol.13, No. 6, November 1993, pp. 33-41.

PAYN90 B. Payne and A. W. Toga, "Surface Mapping Brain Function on 3D Models", IEEE Computer Graphics & Applications, Sept. 1990, pp. 33-41.

RAPP93 A. Rappoport, R. Kosloff, and R. Mayer, "Visualizing Wave Functions in Molecular Dynamics Using Polygon Encoding", Scientific Visualization, Advanced Software Techniques, Ellis Horwood, 1993, pp.125-132

ROLL95 S. Roll, A. Haase and M. Kienlin, "Fast Generation of Leakproof Surfaces from Well-Defined Objects by a Modified Marching Cubes Algorithm", Computer Graphics Forum, Vol. 14, No. 2, 1995, pp. 127-138.

SHIR89 P. Shirley and H. Neeman, "Volume Visualization at the Center for Supercomputing Research and Development", Proceedings of the Chaperl Hill Workshop on Volume Visualization, May 1989, pp. 17-20.

SUNG88 A. Sunguroff and D. Greenverg, "Computer Generated Images for Medical Application", Computer Graphics, Vol. 12, No. 3, Aug. 1978, pp. 196-202.

UDUP82 J. K. Udupa, S. N. Srihari, and G. T. Herman, "Boundary Detection in Multidimensions", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No. 1, Jan. 1982, pp. 41-49.

WILH90 J. Wilhelms and A. V. Gelder, "Topological Considerations in Isosurface Generation", Computer Graphics, Vol. 24, No. 5, Nov. 1990, pp. 79-86.

WYVI86 G. Wyvill, C. M. Mcpheeters and Brian Wyvill, "Data Structure for Soft Object", Visual Computer, Vol. 2, No. 4, Aug. 1986, pp. 227-234.