INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Volume Data Set Visualization using Topological Zone Segmentation

by

Nazma Ferdous

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy,

The City University of New York

2001

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Approval

The manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

8/20/01
Date

Chair of Examining Committee

Sept 20, 01
Date

Executive Officer

Dr. Bhubaneshwar Mishra

Dr. Paula Whitlock

Dr. Robert Goldberg

THE CITY UNIVERSITY OF NEW YORK

Abstract

Volume Data Set Visualization using Topological Zone Segmentation

by

Nazma Ferdous

Advisor: Dr. James L. Cox

For the last two decades, different aspects of visualization have been very popular area of research due to increasing visualization demands of medical and scientific applications in the late twentieth century. A major portion of this research work has been conducted concerning visualization of volume data sets in general, and isosurface extraction in particular. We summarized some of the currently used algorithms in this field, and their efficiency issues in first few chapters. Certainly, the effort of the visualization specialists has come a long way in improving of both quality and applicability. Many more issues remain unresolved, particularly when visualizing large volume data sets, such as the bottleneck induced by excessive I/O operations.

We present an algorithm for organizing the discrete scalar volume data on external storage with important application to out-of-core visualization of extremely large data sets. The application include extraction isosurfaces in a manner that minimizes both I/O and disk seek time, topologically correct isosurface simplification and producing a visual

atlas of all topologically distinct objects in the data set, with the range of scalar isovalues that reveal each. The segmentation algorithm computes the region of space called topological zone components, so that any isosurface component is completely contained in a zone component and all contours contained in a zone component are homeomorphic. The algorithm also develops a search structure called criticality tree as by-product and both of these computation is carried out in space efficient manner. The algorithm is very generic in nature. It does not assume any specific structure of the input data or any specific interpolants and can be extended to data sets with non-unique values. Towards the end, we give a simple, efficient and provably correct algorithm for constructing isosurfaces. Finally we present the results obtained by various experiments that justifies our assumption.

Acknowledgements

First of all, I would like to thank my advisor Dr. James Cox for his endless support, advise and tremendous help throughout the last 3 years of my research, without which I could not have come up with this dissertation today. I would also like to thank Dr. D. B. Karron for letting me use his machines, setting the environment every time I asked him, and for his valuable advise. I could not have performed the experiments without it. I like to thank Dr. Paula Whitlock for doing a very tedious job of proofreading of my paper, Dr. Ted Brown for his enormous support that he provided me throughout the last 5 years, as the chairperson of Computer Science Dept. of Queens College and as the Executive officer of the program, Dr. Robert Goldberg for his encouragement and for his helps when I needed that most, Dr. Stathis Zachos for his encouragement and kind support, and Dr. B. Mishra for his valuable advise. A very special thanks to Dr. Stanley Habib for his incredible support and for taking care of me in various occasions, till he remained in his office. I would not be a part of this program today without this help. I would like to thank all the members of my Ph. D. committee for being part of it, for taking you time to examine and evaluate my dissertation and for your valuable remarks that helped to me to shape this paper in a better form. Finally I would like to thank all the doctoral faculties from whom I have learned. My earnest thanks goes to the Doctoral program in Computer Science for giving me financial support and the opportunity to become a researcher.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Objects and natural phenomenon in our spatial and temporal surrounding are sampled at discrete points to produce computational models, generating 3D volumes of data. Volume data sets are 3D (or higher) entities that have information inside, may or may not consist of surfaces, and may be too large and complex to be represented geometrically. Volume visualization is a method of extracting meaningful information from volumetric data sets, through interactive graphics and imaging. These techniques are mainly designed to manipulate, render and represent the data set, usually in the form of a multidimensional digital image. Volume visualization provides techniques to explore 3D data, inside or outside the objects. This allows understanding of the complex boundary structure, as well as peering inside the object to reveal the inner structure, and giving visual insight into opaque or complex data sets. Also, in some applications (such as computational fluid dynamics, and other scientific simulations), it gives a way to visualize the dynamics, which are not ordinarily easily comprehensible. Volume visualization, as a technique, brings a revolution to computer graphics and promises important breakthroughs in numerous applications.

Volume visualization is a very effective tool, where the acquired data set is inher-

ently volumetric. In many application areas such as medical and scientific imaging, computational fluid dynamics, X-ray crystallography, computer aided design (e.g. solid modeling), simulation and animation (e.g. flight simulation), the data sets that are usually obtained fall into this category. Different aspects of visualization methods have kept the attention of scientists since the early '80s. The technology still holds substantial challenges and promises for future extension. This report discusses different stages of visualization and effective visualization methods currently in use in the first few chapters. As the field matured, the algorithms focused on overcoming existing bottlenecks and acquiring better performance by making the process efficient, faster, easier to use, more meaningful and more interactive. Most of these methods have their special features, and each is best suited for use in specific applications. This report also includes a comparative discussion between different methods of visualization, along with the existing challenges in the field. In addition, the features and applications of a new scheme for visualization, designed to expedite correct isosurface construction, is discussed and some empirical results are presented to verify its usefulness. Due to the limitations of this report, only a few popular algorithms have been summarized despite the enormous amount of work that has been done in this field. Details of each method and related figures can be found in the associated references.

1.1 Phases of Visualization

The process of visualization is described step by step below. Figure 1.1 shows the flow chart of different phases of visualization.

Phase 1: Data Acquisition

The primary sources of volumetric data are the following:

- Sampled data of a real object or phenomenon. These are acquired by multi-channel scanners or sensors that measure the real object or phenomenon, producing a sequence of 2D cross sections or slices. For example, a real human organ is sampled with medical scanners as in Computed Tomography (CT), Magnetic Resonance Imaging (MRI) or ultrasonography, and is represented by a series of slices through the sampled organ.

- Simulated data produced by a computer, by executing a simulation of a mathematical or physical model. The data produced can be a scalar, vector, or tensor field of the 3D spatial grid, or a sequence of 2D planar slices. Examples of such computer modeling can be seen in fluid dynamics, molecular graphics, etc.

- Another source of discrete 3D data is the data generated from a geometric model, for example in Computer Aided Design.

The original data is often enhanced by changing it into a form that is more informative, filtered, and is uniformly distributed on an orthogonal grid, using well-known 2D image processing techniques. While reconstructing a contiguous 3D data set, the 2D slices are stacked together, the missing data between slices or scattered points are estimated by linear, bilinear, trilinear, or sophisticated nonlinear interpolation, depending on the accuracy level required for the application.

Phase 2: Format of 3D Discrete Data

In order to manipulate a discrete data set in an efficient way, the 3D data points are stored and ordered in some specific formats, which are described below.

Voxelization of 3D Discrete Data

Most commonly, the 3D volumetric data set resides in 3D discrete voxel space. which is a 3D integer grid of unit volume cells or elements called voxels. A voxel is a 3D counterpart of 2D pixel. Each voxel is a unit of volume and has a numerical value(s) associated with it, which represents some measurable properties of the real objects, such as color, opacity, density, velocity, etc. A voxel is commonly represented as a unit cube centered at each grid point or having each of its eight vertices located at adjacent grid points. The aggregate of the voxels occupying the whole space forms the volumetric data set.

Both data generated by a computer simulation and sampled data are often inherently voxelized. Otherwise, the data value at each grid point is approximated from the closest data points available by linear, bilinear, trilinear, or nonlinear interpolation, depending on the application. In order to create a 3D voxel image from geometric models, a 3D scan conversion algorithm is used, which generates a well-matched discrete representation of geometric models.

The synthesis and analysis of voxel models need a substantial mathematical foundation. For example, the generated 3D data must satisfy some connectivity requirements. Thus, the framework for discrete space should be a close analog of continuous space, and in this case the discrete space is said to be well-behaved. There has been a fairly extensive literature on digital topology. Discussion about the characteristics and nature of well-behaved digital spaces and connectivity requirements for digital topologies can be found in [21].

Other Formats

Sometimes, it is beneficial to store the data in an uneven grid [29], especially in cases where some regions of the data space have greater fluctuations in values, whereas in other parts of the data space, the data values do not change significantly. In such cases (e.g., computational fluid dynamics), while cell density is required to be very high in a region of interest, having the same high density throughout the data space would lead to expensive storage and data manipulation requirements. The search for an optimal solution suggests that the data geometry be a rectilinear volume, where data points are arranged in a uneven rectangular lattice.

Some other applications create differently sized and shaped cells, such as tetrahedral, or other types of polyhedral. A preprocessing phase is required to create an irregular grid of these aforementioned cells from the original discrete data. In the case of some sampling modalities, the mapping of the data onto an irregular grid is no more difficult than the transformation to a cubic lattice. Each of these mapping algorithm must maintain connectivity requirements for Digital space.

Phase 3: Filtering Volumetric Data

The visualization methodologies employed often display sensitivity to noise, and the type of noise encountered is often dependent on the data acquisition technology. Thus the next step is often the filtering of the volume data to remove distortions produced by noise. The filtering techniques used include fairly standard techniques that are normally employed in signal processing (e.g. Gaussian filters). The filtering techniques are sometimes adapted to the source of the volume data. For example, one might use different filtering techniques for MRI data than the one used for X-ray crystallographic data. The discussion of the filtering techniques and their suitability

to the data acquired by different devices, is beyond the scope of this paper.

Phase 4: Classification of Volumetric Data

This step is used to classify objects within the image, either by thresholding or material classification step. Thresholding is primarily used by the surface rendering approach, where the surfaces of each object are approximated by geometric primitives. An enormous amount of research has been done for fast and efficient surface rendering techniques. Some of the well-known algorithms are discussed in chapter 3.

The other approach, material classification, is often used by volume rendering techniques, where each voxel is classified depending on the percentage of material contained in it. Volume rendering is also a very popular area of research, and the techniques are discussed in the next chapter.

Phase 5: Mapping of the Volumetric Data

After the 3D data set is rendered, it is mapped into geometric or display primitives. Various kinds of display primitives are generated by different rendering algorithms, such as zero dimensional points or particles, one dimensional curves, lines, vectors, two dimensional polygon meshes or curved surfaces, or three dimensional voxels. These display primitives are then used to display the 3D data set as an image.

Phase 6: Volume Viewing and Shading

Once the display primitives are generated, they are projected to form a 2D screen image. The projection algorithms depend heavily on the kind of display primitives generated and the rendering algorithms used. In the case of surface rendering, the geometric primitives can be displayed using conventional viewing algorithms.

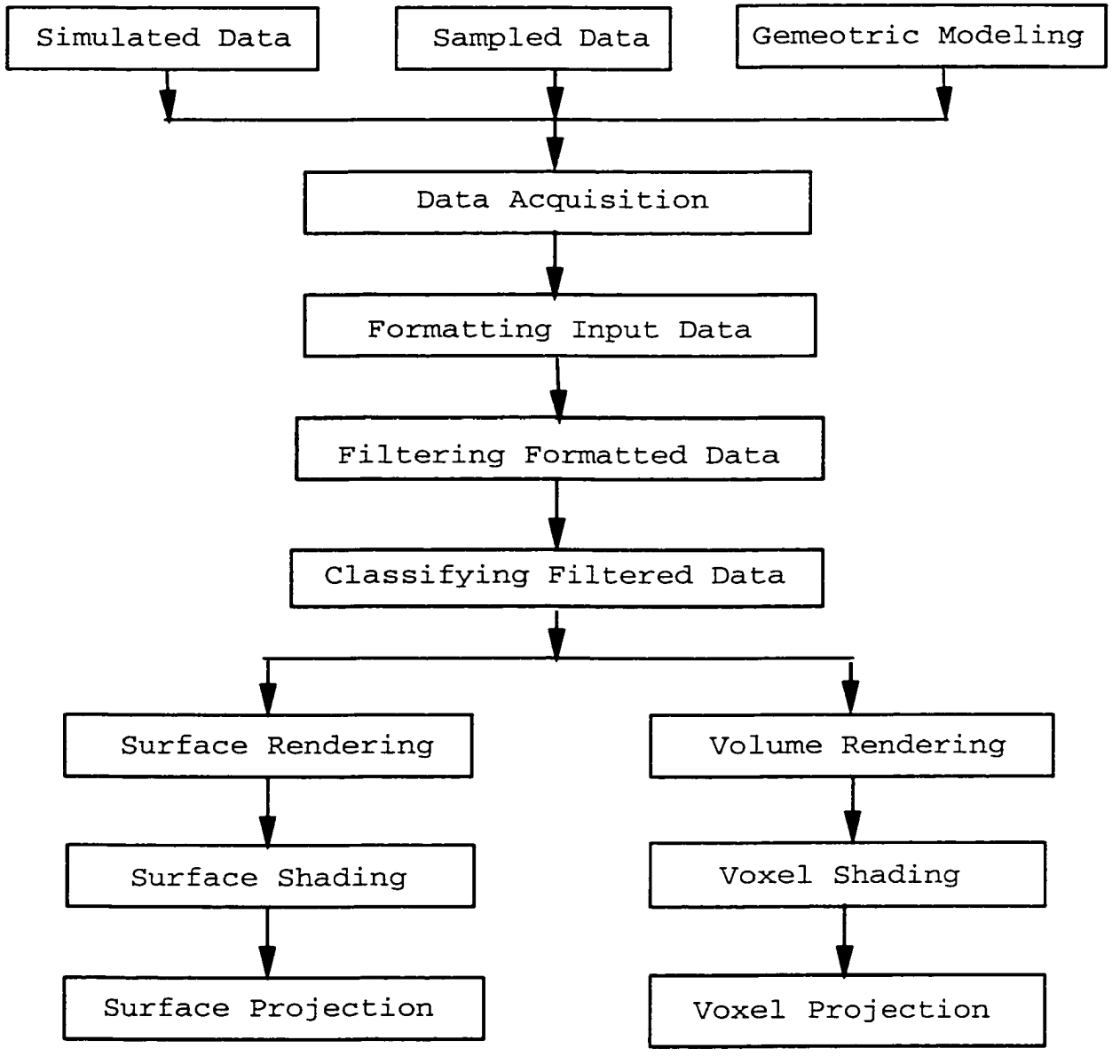

Figure 1.1: Phases of Visualization

In volume rendering, volume primitives are displayed directly and special viewing algorithms are used to capture the content of the voxels. The viewing algorithms can be categorized into two groups: *object order*, and *image order*. In object order algorithms, the object space is scanned and voxels are mapped into pixels. In image order algorithms, a ray is passed through each pixel and the color contribution for each pixel is gathered from the voxel space, by tracing the color along the path of the ray. This latter procedure, called ray tracing or ray casting, is an efficient way to implement volume rendering directly. An optical model for direct volume rendering is discussed in the next chapter.

While constructing the 2D projected image, each pixel within the image needs to be shaded. For this purpose, a surface normal has to be calculated. If geometric primitives are used to create the 2D image, conventional computation methods can be used to calculate the surface normal of each geometric primitive. Otherwise, the gradient information is recovered from volume data itself.

1.2 Hardware for Volume Rendering

Volume images are huge 3D matrices, in the order of gigabits for moderate spatial resolution, with voxel granularity as fine as that of a pixel. This places new demands on graphics systems for more memory, faster computation and communication, and a new breed of processing and display techniques. Some innovative special purpose hardware architectures, such as GODPA, SCOPE, CUBE architecture, PARCUM system, etc., have been designed for volume visualization. A detailed discussion and comparison of these architectures' organization, capabilities and performance can be found in [32]. Several contemporary general purpose systems provide some software support for volume visualization, relying on the partial hardware support from the

geometry and pixel engines. These systems include the Ardent Titan, the AT&T pixel machine, the Hewlett-Packard Turbo SRX/VRX, the Pixar Image Computer, the Pixel plane, the Silicon Graphics 4D, the Stellar GS2000, and the Sun TAAC-1.

Chapter 2

Volume Rendering

Volume rendering is a very thorough and informative method for visualizing volumetric data. In this approach, the data values at the vertices of each voxel are examined and a property value (such as light attenuation, or opacity) is assigned to the voxel, depending on the composition of the material present in it [31]. While forming the 2D image, the volume primitives are directly projected onto the screen without any intermediate representation. This technique models the volume as cloud-like cells of semitransparent material. Each cell emits light, partially transmits light to other cells, and absorbs some incoming light. Figure 2.1 shows the process diagram of the volume rendering of a typical medical CT data set, which is a mixture of fat, soft tissue, bone and air. In addition to the final image, at each stage, one slice from the stage is shown. The basic algorithm is discussed below.

At first the input volume data is converted into several material percentage volumes. The value of each voxel in each of these material percentage volume is the percentage of that particular material present in that region of space. These values can be directly received from the input device, or can be determined from the input data using probabilistic classification techniques. From the knowledge of the kind of

material present in the volume data, the percentage of each material within a voxel that has an intensity I, can be estimated by the following probabilistic method:

$$p_i(I) = \frac{P_i(I)}{\sum_{j=1}^{n} P_j(I)}$$

where,

$p_i(I)$ is the percentage of material i within the voxel

$P_i(I)$ is the probability that material i has intensity I

n is the number of materials present in the volume data.

Given the materials' property value and the material percentage volumes, a composite volume corresponding to that property can be calculated by multiplying the percentage of each material times the property value assigned to that material and summing up all the values. Using this method a composite color volume is obtained from the material percentage volumes and color and opacity values of each material.

Along with the composite volume, the boundaries between materials are detected by applying a three dimensional gradient to a density volume. Each material within the volume is assigned a density value ρ and the density volume is computed from the material percentage volume by summing the product of the percentage of material times the material's density, as follows.

$$d_j = \sum_{i=1}^{n} \rho_i p_{ij}$$

where,

d_j is the density of the voxel j

ρ_i is the density of material i

p_{ij} is the percentage of material i in voxel j

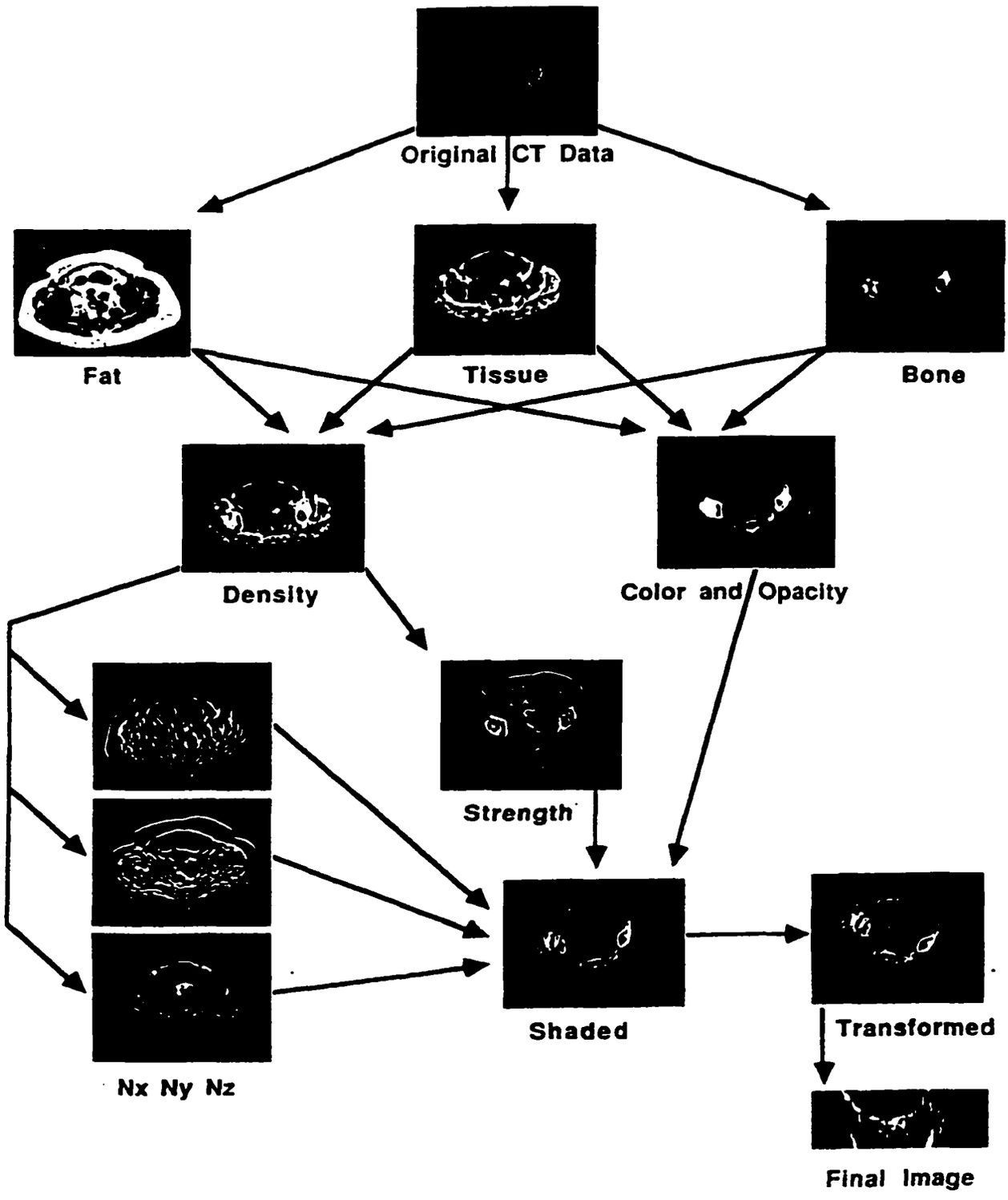

Figure 2.1: Volume Rendering Process

n is the number of materials in voxel j

The gradient is largest where there is a sharp transition between materials having two different densities. This allows one to detect the interface between the two different materials, or in other words, allows one to detect a surface. The magnitude of the gradient is stored as a surface strength volume and is used to estimate the position of the surface present in the image. The direction of the gradient is stored in a surface normal volume, and is used for the shading computation.

From the composite color volume, surface strength volume, and surface normal volume, the shaded color volume is obtained. It is the sum of the light emitted by the volume and scattered by the surface. The relative contribution of the volume emission and the surface scattering can be varied, depending on the application. The reflective component of the light from the surface can be computed from the position and color of the light source, position of the eye, composite color volume, surface strength volume, and surface normal volume. The amount of emitted light is proportional to the percentage of luminous material present in each voxel.

The image is formed from the shaded color volume. In order to lessen the computation, it is often beneficial to transform the volume, so that it lies in the viewing coordinate system. In that case, the eye is at infinity, and all rays are parallel to an axis of the volume. An image of the transformed volume is created by projecting the volumes onto the image plane, taking into account the emission and attenuation of light in each voxel along the ray, passed through each pixel. The accumulated color of the final pixel can be computed by using a simple composite scheme. In back to front composition, if the current voxel has color C, opacity α, and the incoming intensity of the color I, the outgoing intensity I' can be computed as follows, where the *over* operator is defined by the calculation:

$$I' = C\alpha + I(1 - \alpha) = CoverI.$$

Final output is created by projecting the voxels and calculating the color of each pixel on the image plane.

2.1 Optical Model for Direct Volume Rendering

The *Optical Model* for direct volume rendering utilizes front to back color composition to shade the pixels [17]. A ray is passed through every pixel on the screen. The effects of the optical properties are integrated continuously along each viewing ray to compute the final image. The method also models the light interaction with volume densities, such as, only absorption, only emission, combination of absorption and emission, single scattering of external illumination without shadows, single scattering with shadows, multiple scattering, etc. For each model, it derives the differential and integral equations for light transport and also provides calculation methods to solve them numerically. The color and opacity of each voxel is determined using these equations for absorption, emission, scattering and by considering the kind of material contained in it. To determine the color for each pixel, a ray is cast through it from the view point, and the voxels intersecting its path are traced. The final color is calculated by composition of the colors of all the voxels intersecting the ray using a formula similar to the one stated above.

2.2 The Parallel Volume Rendering (PVR) System

Traditional volume rendering methods are too slow to handle large data sets. The *PVR* system [16] implements parallel volume rendering to speed up the visualization process, allowing interactive steering of the simulation. *Parallel Volume Rendering* can exploit three main type of parallelism:

- Object based parallelism: Where each rendering node gets a portion of the data set.
- Image space parallelism: Where different nodes compute disjoint part of the image.
- Time, space, or temporal parallelism: Where different portions of the rendering process are divided, in pipeline fashion, among independent sets of nodes.

The *PVR* system discussed here, is based on a combination of data set load balancing and composition of schemes to produce the final image. One such procedure is illustrated in Figure 2.2. The processors are divided into two distinct groups of nodes. First, the rendering nodes, each receive a portion of the data set, with approximately the same number of voxels. These nodes sample and compute the accumulated color along their part of a ray. The other group is the composition of the nodes, which are responsible for turning a collection of sub-ray images into a complete image. This operation produces the correct result, because of the fact that the composition is an associative operation. So the sub-ray contribution can be combined recursively to produce the final color of the pixel. By setting the rendering nodes and compositing nodes as two stages of the pipeline, one can exploit the inherent time and space

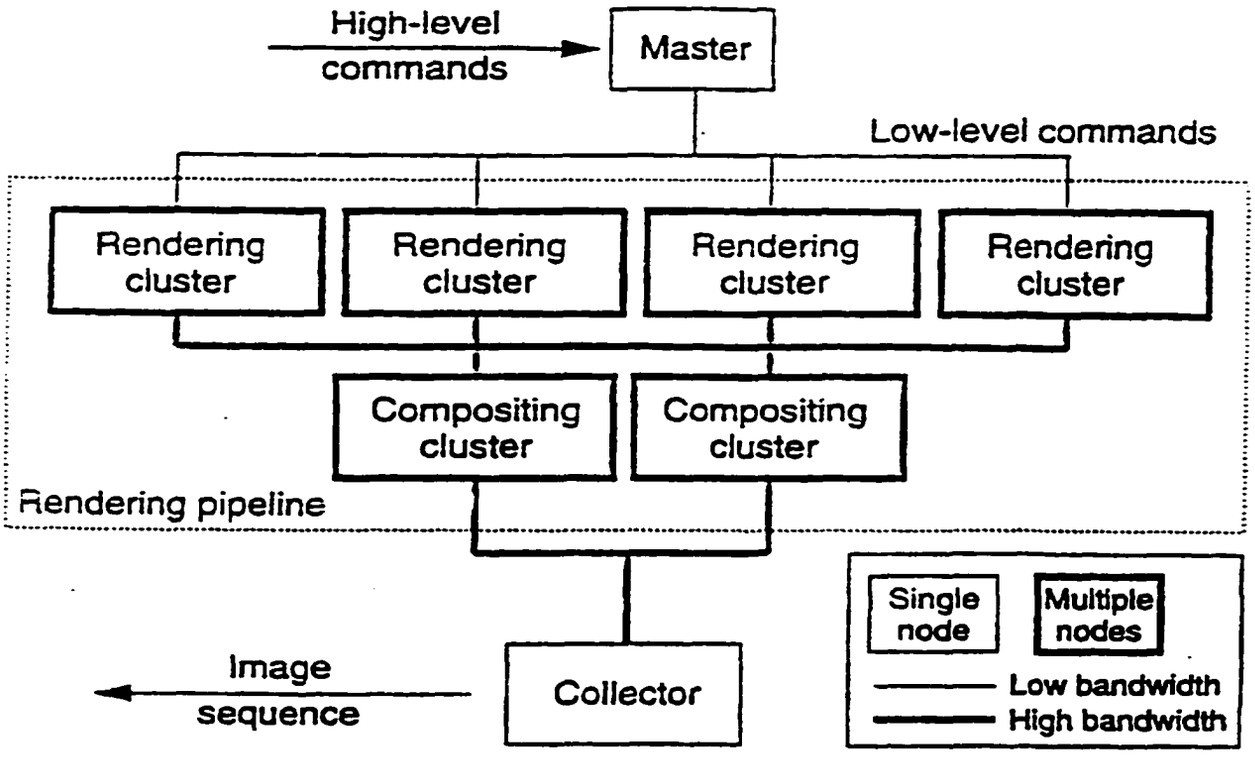

Figure 2.2: The *Parallel Volume Rendering* Process

parallelism available.

Chapter 3

Surface Rendering

Isosurface extraction is a powerful tool for investigating scalar fields within volumetric data sets. Isosurfaces are surfaces of constant scalar value. These surfaces provide information about the interfaces between regions, rather than the region itself, giving clues about underlying structure of the scalar field. Isosurface extraction techniques produce the topological boundaries of the scalar regions.

The isosurface extraction problem is often stated as follows. A scalar volume data set consists of tuples (x, g(x)), where x is a point in 3D, and g is a scalar function defined over a discrete set of these 3D points. Given an isovalue v, the isosurface extraction problem is to compute the surface consisting of the points $\{x : f(x) = v\}$, where f is a function (interpolant) which extends g to all of the 3D space. A more mathematically precise statement is to say that we are computing the topological boundary of $\{x : f(x) \geq v\}$, as the set of points of constant value v may in fact include a 3D volume in the degenerative case, and we may have multiple isosurfaces (e.g. boundary components) that surrounds the set.

Appropriate and comprehensive techniques (see *Spider Web* and *Digital Morse Theory* below) can identify particular objects and their boundary isosurfaces and not

just merely extract particular isosurfaces.

The algorithms for extracting isosurfaces are discussed below. There are numerous algorithms for extracting isosurfaces. *Marching Cubes* seems to be the most popular among them, and VtK is one of the most popular commercial visualization packages with isosurface extraction capability.

3.1 Connecting 2D Contours

An early approach [5] for constructing a surface in 3D volume data was to extract a 2D planar contour of the surface in each slice, and then join the contours of consecutive slices using triangles. Unfortunately, if more than one contour exists in a slice, ambiguity arises as to which contour to connect, which is sometimes solved by interactive intervention. This method uses the information between parallel slices. Surface rendering techniques, with a voxel oriented approach, were later developed. These techniques use the information within a voxel. Rather than finding a global solution, surfaces are approximated locally within a cell. These new techniques are often unambiguous, more accurate and reliable to represent 3D images, and can take advantage of the inherent coherence that exists between adjacent cells.

3.2 Marching Cubes

A classic algorithm, proposed by Lorenson and Cline [1], called *Marching Cubes*, constructs a polygonal mesh to represent the 3D surface, which is later projected to the screen. In order to generate an isosurface, corresponding to a user specified value, the algorithm visits all logical cubes created from eight adjacent pixels, four each from two adjacent slices, determines whether any surface component intersects with this

cubic voxel, and then moves to the adjacent voxel. To determine surface intersection within a cube, the algorithm examines all its vertices and assigns 1, if the data value is higher than or equal to the isovalue, (e.g. these vertices are considered to be inside the surface) or 0, if the data value falls below the isovalue, (e.g. these vertices are considered to be outside the surface). A cube edge is intersected by the surface if and only if one of its two vertices is inside, and the other is outside the region of space bounded the surface. The isosurfaces are assumed to be oriented manifolds. The location of the intersection point can be approximated by linear interpolation. The topology of the surface within a cube can be determined by the pattern of its vertex values (e.g. 0 or 1). Although there are 256 ways to color 8 vertices with 2 colors, taking into account rotational and complementary symmetry, there are only 15 topologically distinct patterns. A lookup table is built that includes all 15 patterns and a corresponding surface approximation for each. The surface approximation uses triangles that connect the intersection points. All the distinct patterns of surface approximation are enumerated in figure 3.1. While visiting each voxel, the algorithm determines its intersection pattern and tessellates its configuration using the look up table. Along with the triangulated surface, the algorithm calculates the unit normal at each triangle vertex, by interpolating the voxel vertex gradient vectors. When displayed, the triangles that are projected on the view plane are shaded smoothly by grading the intensity value.

3.2.1 Asymptotic Decider: Remedy for Tiling Error

It was later pointed out by Nielson [2], that the original *Marching Cubes* produced tiling errors between voxels, and he proposed an *Asymptotic Decider* to remedy this problem. The problem occurs in what Nielson terms as ambiguous faces, those faces

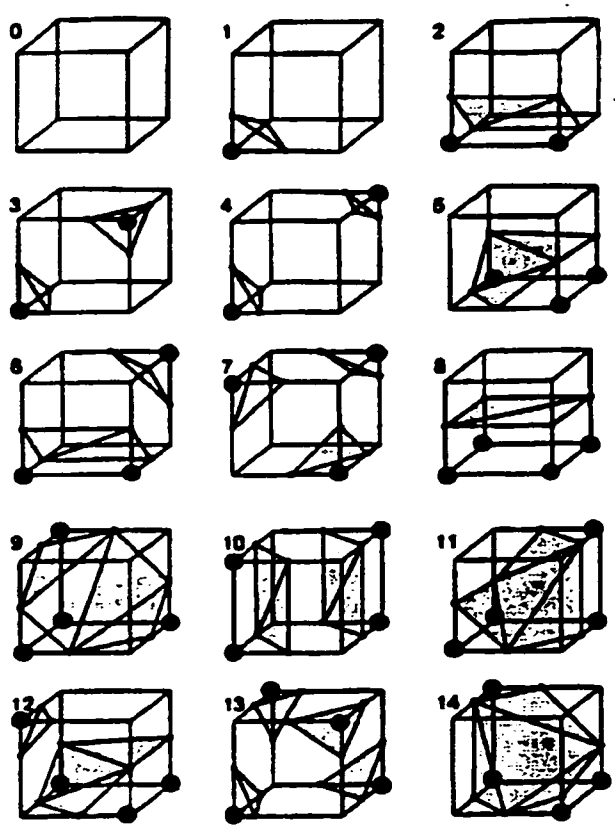

Figure 3.1: Distinct patterns of surface intersection

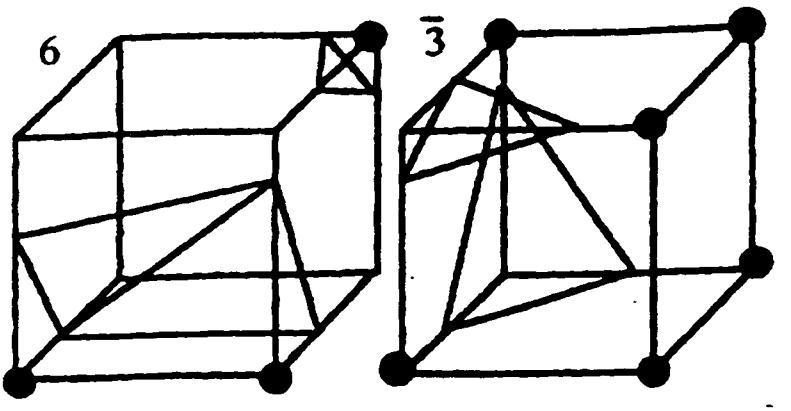

Figure 3.2: An example illustrating flaw in *Marching Cubes*

with 4 separate isosurface intersection points (e.g. we term these faces as 4 hit face). Figure 3.2 shows one such case where triangles produced by *Marching Cubes* do not produce a continuous surface. For instance, if a voxel with configuration case 6 shares a face with a voxel having a configuration $\bar{3}$, (e.g. which is complement of case 3), triangles produced by the *Marching Cubes* algorithm will create a hole in the resultant isosurface. In order to correct this problem, different triangulation can be used. Figure 3.3 depicts two possible triangulation that will result in a topologically correct isosurface. The choice of triangulation can be made using the proposed *Asymptotic Decider* which is based on bilinear interpolation of the value of an interior point in the face.

3.3 Dividing Cubes

The *Dividing Cubes* algorithm [30] uses point primitives instead of geometric primitives. As a result, scan conversion of two or three dimensional primitives can be eliminated. For complex medical images, where the number of polygonal faces is very high, the size of the polygon approaches a pixel size. Therefore, it is more efficient in time and memory to display these images using point primitives.

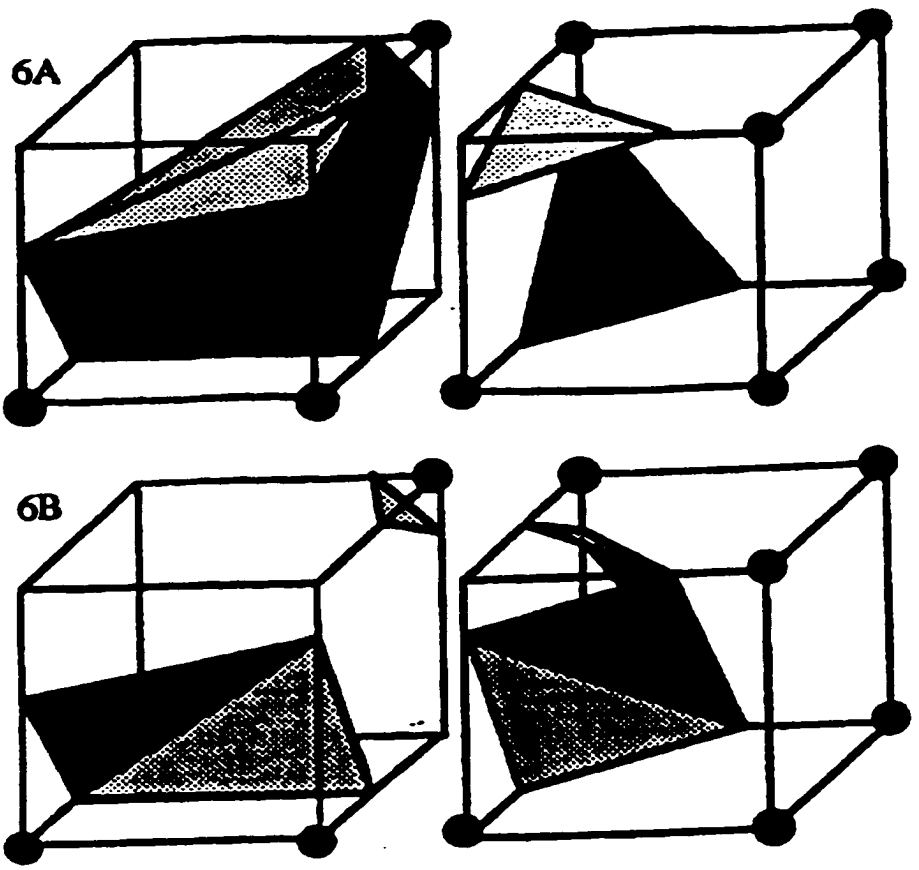

Figure 3.3: Two possible triangulation that produce correct isosurfaces

The *Dividing Cubes* algorithm subdivides the voxels into small cubes that lie on the surface of the objects. The number of divisions is chosen to make the divided cubes small enough so that each one of these is equal to the pixel size of the raster display. The density at each of the eight corners of these small cubic elements is calculated from the original voxel vertices by linear interpolation. All these generated cubes are scanned and tested for surface intersection, using the methods mentioned above in the *Marching Cubes* algorithm. The cubes inside a voxel intersected by a surface, are then classified to be inside, outside, or lying on the surface. The cube that has some, but not all the density values exceeding the surface constant, is marked as lying on the surface. The gradient vector at each cube lying on the surface, is interpolated from the gradient vector at each of the vertices of its parent voxel. Each of these surface points are then projected onto the view plane.

3.4 Spider Web

The *Spider Web* algorithm [22] [27] determines a voxel intersecting the isosurface and then determines hit points along the voxel edges using linear interpolation. In the current implementation of the algorithm [25], the pair of hit points on a voxel face with 2 hits are considered adjacent. On a voxel face containing 4 hits a decision of pair-wise hit adjacency is made, using a strategy similar to Nielson's *Asymptotic Decider*. The adjacency defines one or several connected sets of hits within the voxel. After that the centroid point of all the hits within each connected set is calculated. This point, called the articulation point (AP), is used as the voxel interior triangle vertex for all of the subsequently constructed triangles within that set of hits. Each triangle consists of the AP and a pair of adjacent hits on a voxel face. Hits on a voxel face are shared by the voxel that shares this face, and thus the algorithm can continue

surface construction in the adjoining voxels. This algorithm has an advantage for not being case table driven. It also produces topologically correct isosurfaces with no tiling errors or holes, and is completely parallelizable, as each voxel can be processed independently. But it produces more triangles than other algorithms. The primary cases where it produces more triangles are when the hit set consists of 3 or 4 hits. In the former case one can remove the articulation point to produce one triangle only. When there are 4 hits, *Marching Cubes* produces two triangles. However the inclusion of the interior AP and the subsequent 4 triangles results in a better approximation of the small scale curvature of the surface.

The algorithm is easily extendable to higher dimensions. For an $n > 3$ dimensional cell, recursively build the (hyper) surface patches in the $n-1$ dimensional boundary cells. Each of these patches will consist of $n-2$ dimensional simplices and each patch will be homeomorphic to an $n-2$ dimensional ball. Now for each connected set of hits in the n dimensional cell select an AP. For each pair of adjacent hits in the set construct 2^{n-3} simplices, each consisting of the AP and an $n-2$ dimensional simplex that contains the pair in a boundary cell. Each such simplex will consist of the pair of hits, the AP, and $n-3$ lower dimensional articulation points. The simplices thus formed with the AP for the set will form a (hyper) surface patch homeomorphic to an $n-1$ dimensional unit ball.

For example, in a 4 dimensional cell, the lower dimensional patches will be the normal triangular meshes in the 3 dimensional boundary cubes. For a connected set of hits, these patches will form a closed surface within the 4 dimensional cell, topologically equivalent to a sphere (e.g. by the proof of correctness of the original *Spider Web*). Tetrahedra will be formed from an interior AP to each triangle on the closed surface, forming a 3 dimensional surface patch equivalent to a ball. This patch will intersect the patches in neighboring hypercubic cells by completely sharing

a 2D surface patch in a 3D boundary cube, since this boundary cube will be completely shared by two neighboring 4 dimensional cells. In this way a collection of 3 dimensional manifolds is constructed.

Finally, analysis of the connectivity behavior of *Spider Web* as well as *Marching Cubes* led its inventors to develop the new techniques called *Digital Morse Theory* [23], [25] [26]. In particular, the decision to disambiguate 4 hit faces should be made using all data readings from both voxels sharing the face. Otherwise, as the isovalue is varied we can see that incorrect choices lead not only to tiling errors at a specific isovalue, but to situations that are inconsistent with a continuous interpolant for the volume data. In particular, it can lead to cases where the identical point sets satisfying $f(p) \geq c$ and $-f(p) \leq -c$ are interpolated differently. Consideration of these issues led to the formulation of a mathematical theory of the properties that results in correct isosurfaces over the entire range of isovalues.

3.5 Exploiting Tetrahedral Decomposition

Decomposing the voxel into five tetrahedral cells [7] can lead to an efficient surface extraction, primarily because of the fact that this method does not suffer from the well known inconsistency discussed before. The faces of the tetrahedral cells can only be classified (or colored) in three distinct ways, where the connectivity among the vertices can be resolved without any ambiguity. The vertices of the cells can be reordered according to their ascending values, say v1 to v4, in the initialization stage. When a cell is determined to intersect the isosurface, at most two vertices (v2 and v3) are needed to be examined. Because there are only three distinct ways that a tetrahedral cell can be intersected, a lookup table is not needed to find the approximated surface. The three possible cases, described in figure 3.4, are the following:

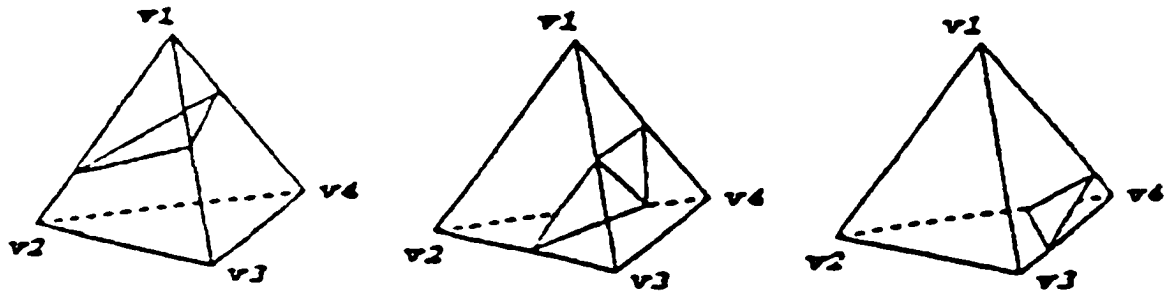

Figure 3.4: Surface intersection within tetrahedral cells

- only v1 is outside the surface, the rest are lying inside.
- v1 and v2 are outside the surface, and v3 and v4 are inside.
- only v4 is inside the surface, the rest are outside.

For each case the possible triangulation is also shown in the figure 3.4. All the 5 cells within the voxel is examined for surface intersection and the triangles are built accordingly. However, Zhou et al [33] showed that tetrahedral decomposition and linear approximation along the introduced diagonals change the original function and may lead to incorrect, though consistent topology. They proposed trilinear approximation across the diagonal of the cells as a solution to this problem, rather than applying linear approximation along all edges. This method has a disadvantage of increasing fivefold the number of cells that have to be processed individually.

3.6 Faster Active Cell Selection

In general, isosurface passes through only a small fraction of total cells (approximately $O(n^{2/3})$, where n is the number of total cells). Exhaustive search of all cells often prove to be inefficient, because a large amount of time is spent traversing the cells which do not contribute to the isosurface generation. One of the key issues of isosurface

construction algorithm is selection of active cells, which are, the cells known to have intersected the isosurface. The next group of algorithms are concerned with the faster selection of the active cells. The basic idea is to preprocess the scalar data and construct a search structure which accelerates the repeated action of isosurface generation and allows increased interactivity during the modification of the isovalue. The preprocessing algorithms presented below is classified into three major group depending on whether it decomposes the data based on geometric space or value space or topological structure.

3.7 Geometric Space Decomposition

This group of algorithms decomposes the geometric space and exploits the inherent coherence along the shared cell faces while generating the isosurface.

3.7.1 Octree

The original *Marching Cubes*, *Dividing Cubes*, and *Spider Web* algorithms did not consider the problem of optimizing the time needed active cell (e.g. those cells which actually intersect the surface) search. The issue was later addressed by Wilhelms and Gelder, who used an *Octree* to sort out the cells of interest [3]. The *Octree* is basically a hierarchical decomposition of the cell set. Several different representations of the *Octree* [6], each of which differs either by the heuristics of subdivision or by the representational data structure, have been proposed. Each node in the tree represents a volume. The root node represents the entire object space. The volume represented by any internal node is subdivided by three orthogonal planes, creating eight subvolumes, represented by eight child nodes. This process is repeated until some criterion is satisfied, or the maximum height allowed has been reached. Usually,

each volume is subdivided only when there is more than one object intersecting the volume, enabling one to trim off the portion of the space containing only one object, during the search.

However, the representation of such a tree using pointers can be very memory intensive. Solutions without pointers have been proposed, such as *Coded Tree* [5], where a list of tree nodes are arranged according to a preset tree traversal mode. Also, *Octree* decomposition are known to be sensitive to noise.

3.7.2 Extrema Graph

This method proposed by Itoh and Koyamada[8] presents a new way of searching for the intersected cells. The search starts at a seed cell known to have intersected the surface, and propagates recursively to its neighboring cells. Knowledge of how the current cell has been intersected by the isosurface, and taking advantage of the inherent coherence between cells, the search can be propagated to only those neighbors that are guaranteed to intersect the surface.

The algorithm also automatically searches for the seed cell, using an *Extrema Graph*. First, extrema points (e.g. local minimum and maximum) in a scalar field are extracted and then connected by a graph, called an *Extrema Graph*. The cells that are intersected by each arc of the *Extrema Graph* are placed in a list called the cell list. The boundary cells of the volume are placed in a boundary list. Each of these lists is sorted.

The cells residing on a closed surface can be found by visiting the cell list along the path of each arc. The cells intersected by a open surface can be found in the boundary cell list.

For a specific isovalue, the *Extrema Graph* is first scanned to locate the arcs that

span across the value. The cell list and the boundary list are searched sequentially for the seed cells. which are then put into a FIFO queue. The search is then propagated among the adjacent cells and the isosurface is constructed from the active cell list.

Storage requirements for the extrema graph may be high, since the propagation requires six links from each cell. The required size of the queue is unknown in advance.

3.8 Value Space Decomposition

The next group of methods decomposes the value space of the data, and thus results in decreased ability to exploit geometric coherence information. While the storage costs for some of the methods can be higher than a geometric decomposition of the data, they have the advantage that they can be used even when the data is residing on an unstructured or irregular grid. Also since active cells are selected by their maximum and minimum values only, the dimensionality of the problem reduces to two dimensions for 3D scalar volume data.

3.8.1 The Span Filter

The *Span Filter*, proposed by Gallagher [10], attempts to minimize the number of cells that have to be scanned for isosurface intersection by subdividing the range domain into buckets. Each cell is classified based on the bucket its minimum value resides in and on how many buckets the cell range spans. Cells are first grouped together according to their starting bucket and within each such group they are further regrouped depending on their span of the range value. Each cell is associated with a unique id and is listed under a span list of a specific span length, based on its span.

Figure 3.5 shows the organization of cells into buckets over span space. Given a

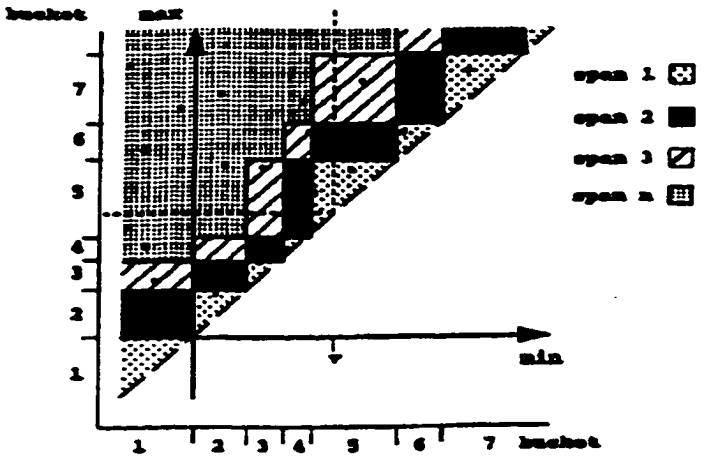

Figure 3.5: *Span Filter*: The division of the range into buckets is depicted.

isovalue v, the cells which lie above and to the left of the dashed line are selected as the active cells. This can be done only by examining those cells whose spans extents to the bucket that contains the isovalue, which is only a small fraction of the total cells. The entire organization of the domain, as well as the cells, depends on the division of the range into buckets. While the cells may be evenly distributed initially by carefully choosing the buckets, span length of these cells cannot be controlled. Moreover, the division of the range has to be done interactively and has to be crafted for each data set. The search algorithm has a complexity of O(n).

3.8.2 The Active List

A different approach was taken by Giles and Haimes [9] to find the intersected cells. The *Active List* method is based on two cell lists, ordered by the cell's maximum and minimum value respectively, and on Δ, the global maximum range of the cells. The active cell list is first initialized as all the cells with a minimum value between v and $v - \Delta$, by consulting the minimum list. The maximum list for these cells is

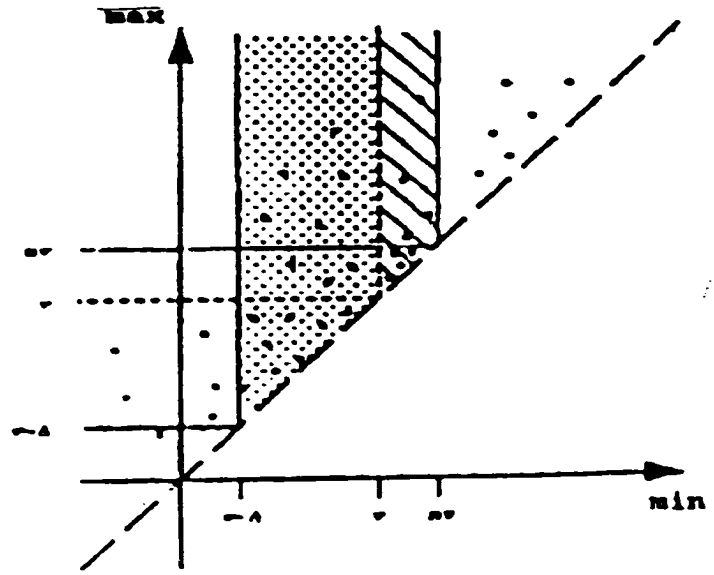

Figure 3.6: *Active List*: The organization of data points in min and max coordinates.

then examined, and cells having maximum value less then v are taken out. Figure 3.6 depicts the *Active List* algorithm over the span space. Though the algorithm does not partition the range in advance, the use of the global maximum cell span does that implicitly. The width of the area that needs to be scanned is constantly maintained by the use of Δ, thereby automating the division of the span space.

Another nice feature of this algorithm is that once the active cells for an isovalue v are detected, the active cells for a neighboring isosurface can be found with minimal effort. If the isovalue is changed by less than Δ , the active list is augmented with the cells that lie between the previous isovalue v and the new isovalue nv. The new cells are found by using one of the ordered lists, based upon whether the change was positive or negative. The active cell list is then purged again to remove the cells that do not intersect the isosurface using the other extreme value. If the change is more than Δ, a totally new list is created.

Since Δ depends on the data set, the algorithm has no control over the size of the

scanned list . If Δ is too large, the algorithm has to scan a large portion of the data set. If Δ is too small, while searching for a neighboring isosurface, any previously scanned list will be of little use. When the change of the isovalue is continuous and smooth, it gives small improvement over a non-optimized method. The complexity of the algorithm is also O(n).

3.8.3 Sweeping Simplices

The *Sweeping Simplices* method for extracting isosurfaces developed by Shen and Johnson [11] uses two ordered cell lists, a sweep list and a min list, similarly as the *Active List* algorithm. Each element in the sweep list contains a pointer to a cell, the cell's maximum value, and a flag and is sorted according to its maximum value. The min list contains the minimum value for each cell, as well as a pointer to the corresponding element in the sweep list, and is ordered by the minimum value. The *Sweeping Simplices* algorithm also augments the approach with a hierarchical organization of the value space for faster search. The root node is associated with the entire range of data values. At each level the cells are subdivided into several groups, each of which maintains its own min list and sweep list. The precomputation needs a time of $O(n \log n)$.

Given an isovalue v, a binary search in the min list marks all the cells that have a minimum value higher than the given isovalue and sets the corresponding cell's flag in the sweep list. The candidate cell that has largest maximum less than the isovalue is also determined. All candidate cells having higher maximum than this cell is reported as active cells. If any isovalue was previously given, then the min list is traversed between the old and the new isovalue and the sweep list is updated accordingly, from which the active cells are selected. The complexity of the algorithm

remains as O(n) in worst case.

Sweeping Simplices does not depend on Δ and its space decomposition may not be optimal. Each group whose range intersects the isovalue must be linearly scanned, and each such group may contain an area outside the region of interest.

3.8.4 Isosurface Extraction using Span Space

The above algorithms decompose the value space, and order the cells, by maintaining two different lists for minimum and maximum value. *Isosurface Extraction using Span Space* algorithm proposed by Livnat, Shen and Johnson [12] attempts to combine the two lists into a single one. Using the span space as the underlying domain, a search tree is employed as the means for the simultaneous ordering of the cells according to their minimum and maximum values. The authors call this search tree a Kd-tree. The median of the data values along one dimension (either min or max) is stored as a key in the root node. Data points are partitioned according to the median and recursively stored in the two subtrees. The partition at each level alternates between the min and max coordinates. The construction of the Kd-tree can be done in optimal $O(n \log n)$ time. Figure 3.7 depicts a typical decomposition of the span space by a Kd-tree.

Given an isovalue v, the Kd-tree is traversed recursively comparing v with the key stored at the current root. This automatically alternates the search for minimum and maximum value at odd and even levels, and determines the intersected cells. The worst case complexity of the query is $O(\sqrt{n} + k)$, where k is the output size.

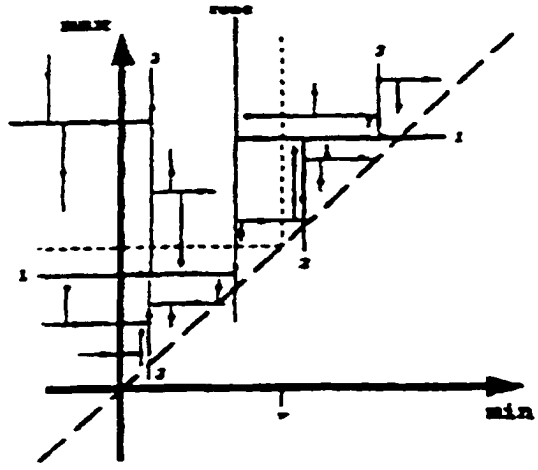

Figure 3.7: The organization of cells in a Kd-tree. The splitting of data is done by alternating between min and max coordinates

3.8.5 Interval Tree

The *Interval Tree* is an optimally efficient search structure proposed by Edelsbrunner [13] to organize a set of (possibly overlapping) intervals on the real line, so that one may retrieve those intervals that contain a given query value. Cignoni proposed an algorithm [14] that focuses on the fast selection of the active cells by organizing the cells using a similar data structure. The location of the active cells is computed as follows. Let Σ be the entire data set (e.g. structured or unstructured). Each cell $\sigma_j \in \Sigma$ is associated with an interval I_j whose extremes are a_j and b_j, the minimum and maximum value of the cell, respectively. σ_j is considered to be an active cell if the isovalue lies within its interval.

The intervals can be organized into an interval tree, which gives rise to an efficient way of selecting the cells having an interval containing the isovalue v. For each cell $j = 1, 2, 3, \ldots, m$, consider a sorted sequence of values $\mathrm{X} = (x_1, x_2, x_3, \ldots, x_h)$ corresponding to the distinct extrema (e.g. a_j and b_j) of the intervals. The interval tree is a balanced binary tree $\mathcal{T}$, whose nodes correspond to the values of X, and an

additional structure, which lists the cells within the interval, referenced by the node.

The interval tree is defined recursively as follows:

The root of the tree has a discriminant value $\delta_r = x_r = x_{\lceil \frac{h}{2} \rceil}$, which partitions the whole interval I into three subsets.

- $I_l = \{I_j \in I | b_j < \delta_r\}$ This interval corresponds to the left subtree, the cells having a maximum value less than δ_r.
- $I_r = \{I_j \in I | a_j > \delta_r\}$ This interval corresponds to the right subtree, the cells having a minimum value higher than δ_r.
- $I_{\delta_r} = \{I_j \in I | a_j \leq \delta_r \leq b_j\}$ This interval corresponds to the node itself.

The cells within each interval are arranged as two sorted lists, called AL and DR.

- AL contains all cells in the interval sorted in ascending order of their minimum value.
- DR contains all cells in the interval sorted in descending order of their maximum value.

The left and right subtrees are defined recursively, by considering the interval sets I_l and I_r and $(x_1, \ldots, x_{\lceil \frac{h}{2} \rceil - 1})$ and $(x_{\lceil \frac{h}{2} \rceil + 1}, \ldots, x_h)$ as the extreme sets, respectively. The resulting structure is a balanced binary tree of h nodes, and of height $\lceil \log h \rceil$, where h is the total number of distinct extrema of cell intervals. The interval tree can be constructed in $O(\frac{n}{2} \log \frac{n}{2})$ time, for a total of n list elements. Figure 3.8 shows a typical interval tree organization of 13 cells.

Given an isovalue v, the tree is traversed, starting at the root, to search for active cells as follows:

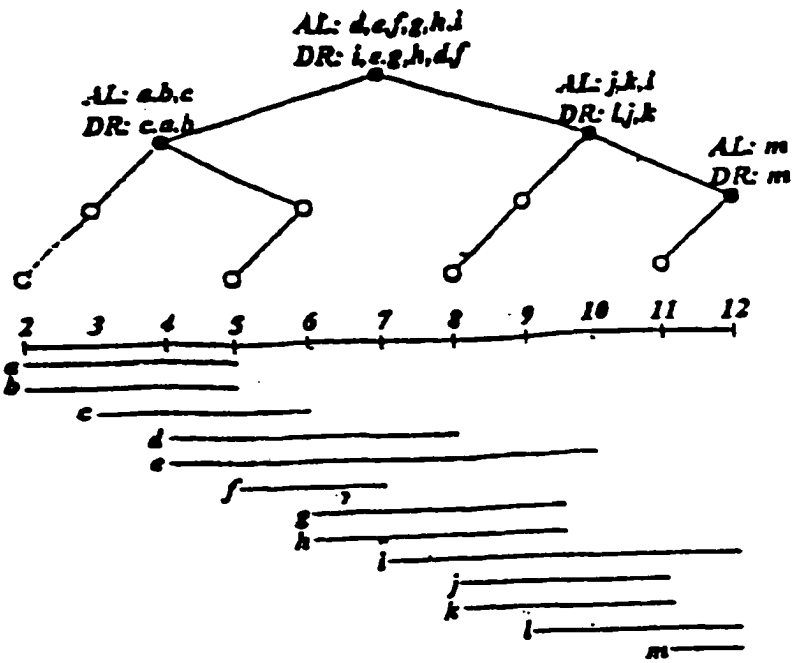

Figure 3.8: An example of an *Interval Tree.* The white dots represent nodes with empty AL and DR lists

- if $v < \delta_r$ (e.g. the discriminant value), then the AL list is scanned until a cell is found having a minimum value $a_j > v$. The left subtree is visited recursively. All the scanned cells are considered to be active.

- if $v > \delta_r$ (e.g. the discriminant value), then the DR list is scanned until a cell is found having a maximum value $b_j < v$. The right subtree is visited recursively. All the scanned cells are considered to be active.

- If v = s, then the whole AL list is reported.

While selecting the active cells, the maximum number of nodes that are visited is $\lceil \log h \rceil$. So, if k is the output size (e.g. usually $O(n^{2/3})$, where n is the total number of cells), the computational complexity for finding the active cells is $O(k + \log h)$.

The algorithm also suggests how to reduce the number of intervals to be stored,

using a 3D chessboard approach, where the intervals of the black cells are stored in the tree. The active black cells are first selected using an interval tree. All active white cells can then be located by searching the neighborhood of the active black cells, in constant time per cell. Since the number of black cells is only $1/4^{th}$ of the whole data set, this reduces the number of intervals to be stored significantly.

3.8.6 I/O Optimal Interval Tree

As a follow up of the previous idea, an *I/O Optimal Interval Tree* has been proposed and implemented by Chiang and Silva [15]. The search structure is specifically designed to reduce the I/O communication (and particularly disk seek time) between main memory and the slower external memory, specially when the entire data set is too large to fit in main memory. The algorithm preprocesses the data set to build an efficient search structure on the disk. When constructing an isosurface an output sensitive query is performed on the search structure to retrieve only active cells. During the query operation, only two blocks of main memory are used to bring in the active cells and negligible disk overhead is required.

Each node of the *I/O Optimal Interval Tree* is one block on disk, and the maximum number of children it can have is equal to b which is dependent on B, where B is the amount of cell information that can fit into one block. b is assumed to be $O(\sqrt{B})$ in the proposed algorithm, in order to fit three accessory lists associated with each node (cell) within one block. Let S be the set of all intervals and E be the set of all extremas. The *I/O Optimal Interval Tree* is defined recursively as follows:

The root u is associated with the entire range of E and the set of all intervals in S. If S contains less than B intervals, it is a leaf node containing all intervals of S. Otherwise, E is equally divided among b slabs namely, $E_0, E_1, \ldots, E_{b-1}$. The slab

boundaries are stored as keys in node u. Every interval I in S is examined, and if both its endpoints lie within the slab E_i, then it is put into the associated interval set S_i. If the two endpoints of interval I lie in different slabs, I belongs to node u. All information regarding I is stored in an additional structure associated with node u.

The subtrees u_i, for $i = 1, 2, \ldots, b$, rooted at node u are defined recursively by using range E_i and interval set S_i.

The additional structure associated with each node consists of three sorted lists named *left*, *right* and *multi*, which lists the intervals according to their minimum, their maximum, and their span, respectively. A careful implementation of this data structure arranges the lists and nodes on disk in a manner that allows each list to be retrieved with one I/O operation. The branching factor, b, is chosen in a manner, so that all information for a node can fit in one block.

An example of an *I/O Optimal Interval Tree* can be found in figure 3.9. Consider the intervals shown in the figure. The set all extremas E is divided into five slabs, E_0 through E_4, at the root node u. The interval sets for each of these slabs are S_0 through S_4. These sets contain all cell intervals that are completely contained within the corresponding slab. Thus, in this case the interval sets are $S_0 = \{I_0\}, S_1 = S_2 = S_3 = S_4 = \emptyset$. For intervals that are not part of any of these interval sets, are associated with the root node u itself. We examine each of these intervals and decide whether it should be placed on the *left*, *right*, and *multi* lists for node u. Node u will have a *left* list and a *right* list for each slab and these lists will contain all the cells whose interval has its lower bound and its upper bound contained in the corresponding slab, respectively. The *multi* lists for node u contain all the intervals who span more than one slab and completely cover at least one slab. Note that, some intervals may simultaneously be on both a *left* (or *right*) list and *multi* list

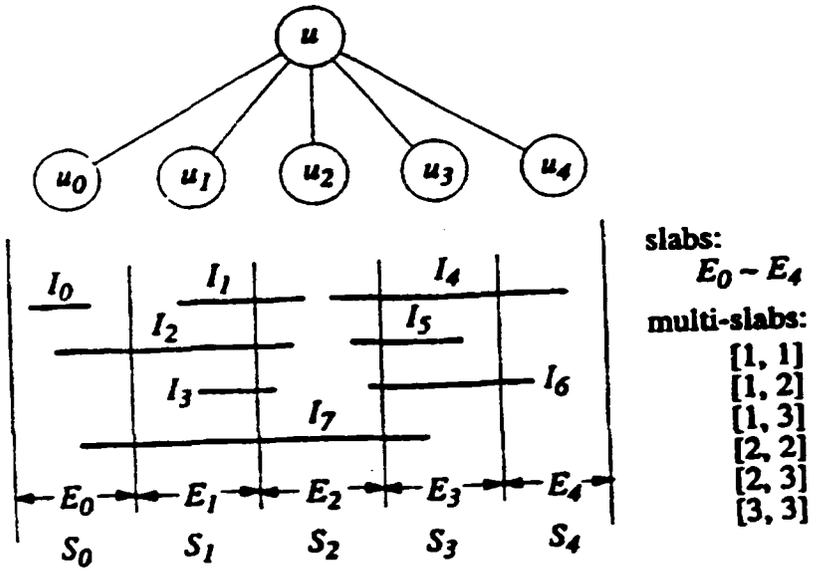

Figure 3.9: An example of *I/O Optimal Interval Tree* for branching factor b=5.

for node u. For example, the $left$, $right$, and $multi$ lists of node u are the following: $left(0) = \{I_2, I_7\}, left(1) = \{I_1, I_3\}, left(2) = \{I4, I_5, I_6\}$ and $left(3) = left(4) = \emptyset$. $right(0) = right(1) = \emptyset, right(2) = \{I_1, I_2, I_3\}, right(3) = \{I_5, I_7\}$ and $right(4) = \{I4, I_6\}$. $multi([1,1]) = \{I_2\}, multi([1,2]) = \{I_7\}, multi([1,3]) = multi([2,2]) = multi([2,3]) = \emptyset$ and $multi([3,3]) = \{I4, I_6\}$. The subtrees $u_0, u_1, \ldots, u_4$ are recursively defined using $E_0, E_1, \ldots, E_4$ as the range and $S_0, S_1, \ldots, S_4$ as the interval list, respectively.

Given an isovalue v, starting from the root, the search algorithm iteratively visits a series of nodes until a leaf node is encountered. For each node visited, the associated lists of intervals are retrieved from the disk, which amounts to a single disk block per node. From a given node, the child that is next visited is determined using the slab boundaries.

The height of the tree is $O(\log_b N) = O(\log_B N)$. The number of I/O operations needed to retrieve k active cell is $O(\log_B N + k/B)$, where $\lceil k/B \rceil$ disk reads are

necessary to report all active cells. Approximately $O(N/B)$ blocks are needed to store the interval tree.

3.9 Topology Based Decomposition

As we can see, the geometric space and value space decomposition exploit coherence in one sense by sacrificing coherence in another. In order to combine the advantages of both coherence with the goal of extracting the boundary surfaces of individual shapes, a natural and advanced approach of decomposition is utilized by the following group of algorithms. At first, the data points within individuals shapes are grouped together, and then a search structure is created for each group. This approach efficiently minimizes the search only within related data points and not further. For grouping the data points, this set of algorithms records the creation, disappearance, split, merge and change in genus of individual components using Morse theory. Morse theory studies all changes in topology of the level sets of a function f, as its parameter x changes. The level set is defined as follows. Let, $p_1, p_2, \ldots, p_n$ is a set of n points in a d-dimensional space R^d, with corresponding scalar values $h_1, h_2, \ldots, h_n$. A piece-wise linear function f interpolates the values from the data points in the entire region such that, $f(p_i) = h_i$ for all $i = 1, 2, \ldots, n$. The level set of f for some parameter x is the set $p \in R^d \mid f(p) = x$. Points at which the topology change occurs in the level sets are called critical points. The following algorithms detects the location of critical points and partitions the data accordingly.

3.9.1 Contour Tree

The *Contour Tree* [35] [34] is a graph that tracks the components of level sets as the they split and appear or join and disappear, by incrementally increasing the

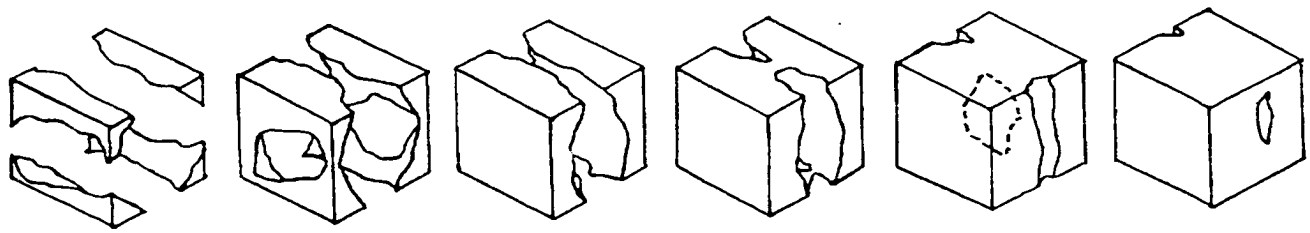

Figure 3.10: Level set of f as the parameter x decreases

parameter. The *Contour Tree* is defined as follows:

1. Each leaf vertex represents the creation (respectively deletion) of a component at a local maximum (respectively minimum) critical point.

2. Each interior vertex represents the joining and/or splitting of two or more different components at saddle critical point.

3. Each edge represents a component in the level sets for all values of the parameter between the values of the vertices it joins.

Figure 3.10 depicts the level set of a function f. As the parameter x is decreased, it evolves from four sticks, to two rings, to two disks, to a single ring, to a hollow ball, to a solid. The corresponding *Contour Tree* is shown in the figure 3.11. Starting from the bottom, the four leaves (e.g. 7 - 10) correspond to four sticks (figure 3.10 (leftmost)), which merge in pair at node 5 and 6 forming two rings (figure 3.10 (2nd from the left)) followed by two disks (figure 3.10 (3rd from the left)). The two disks then merge at node 4 forming a single component (figure 3.10 (3rd from the right)). At node 3 it encloses a hollow (figure 3.10 (second from the right)), where two separate boundaries are created. The inner boundary slowly contracts and disappear at node 2 forming a solid (figure 3.10 (rightmost)). The change in genus within a single component (e.g. from ring to disk) is not recorded because the isosurface can still be traced from a single seed cell.

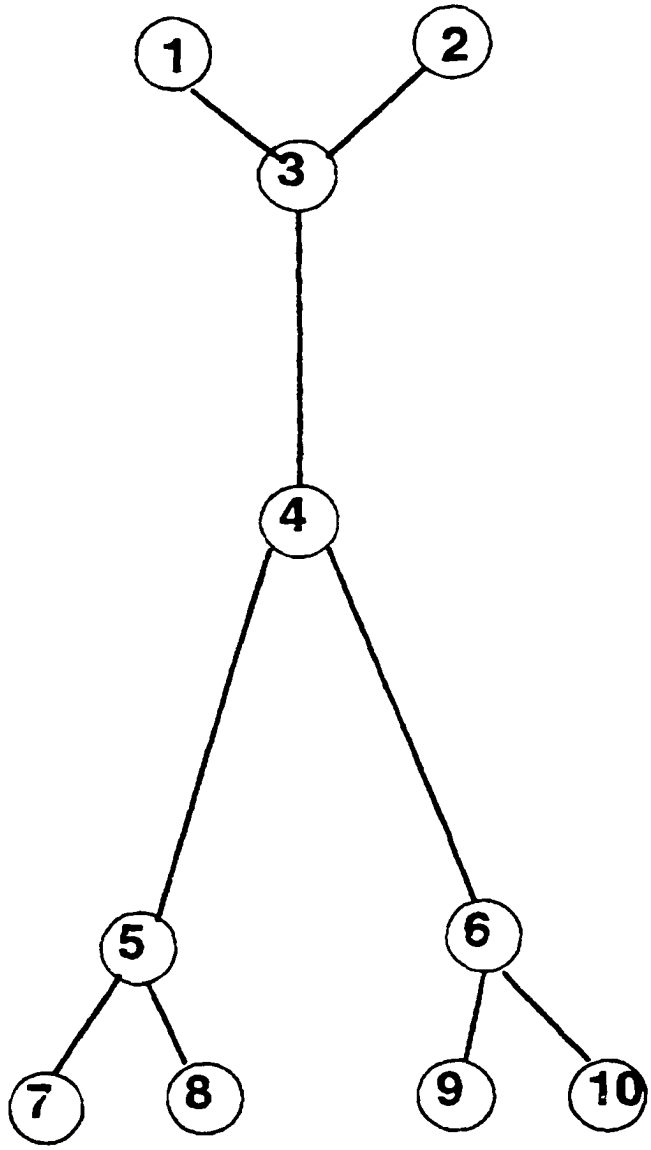

Figure 3.11: *Contour Tree* constructed for the level set shown 3.10

The vertices and edges of the *Contour Tree* are called *super-nodes* and *super-arcs*, respectively. All the *super-arcs* are augmented with the left over points swept through by the respective components, which are then connected in sorted order. The next phase is to generate seed set for each component for a all range of isovalue to be used for isosurface construction. Note that each individual component whose cells span across the isovalue gives rise to an individual isosurface. Due to this property, search for seed cells is restricted only within the cells of that particular component. Several methods for efficient seed set generation are discussed in [36]. Given a isovalue v, the seed set of the components intersected by isosurface is searched for appropriate seed cells and the search propagates recursively through all neighbors guaranteed to intersect the isosurface.

3.9.2 Digital Morse Theory

This method identifies the critical points at which topology of the level set changes as before, and constructs a search structure called *Criticality Tree* which traces the creation, deletion, merge, split of different components as well as changes in genus, as the isovalue is decreased. The algorithm also segments and organizes the data set into individual zones, each of which contains the data points in between neighboring critical points. The motivation behind the thought and the algorithm is discussed in greater detail in the following chapters. The *Criticality Tree* generated by this method is also compared with the *Contour Tree* generated by the previous method in chapter 12.

Chapter 4

Comparison between methods

From the discussion of the last two chapters, it can be understood that to visualize a volume data set either of the two basic methods, volume or surface rendering, is utilized to classify the voxels. Each of these methods has its advantages as well as shortcomings. In the following paragraphs, both the methods are compared against each other with respect to several important application issues.

In volume rendering, the volume primitives are directly projected into 2D pixel space and stored as a raster image in a frame buffer. Alternatively, in surface rendering, isosurfaces are extracted from the volume data and are approximated by geometric primitives, which are then projected to the screen. Surface rendering is thus, an indirect technique for visualization where the voxels are first converted into an intermediate surface representation.

Surface rendering results in a significant reduction in data storage, as only the active cells intersecting the surface need to be stored in main memory. Geometric primitives are easier to manipulate and display, and can take advantage of the existing software. However substantial preprocessing is needed to identify the active cells and to approximate a surface representation using other primitives. Most of the algorithms

described in chapter 3, are dedicated to efficient search of the active cells, and to organize them so that I/O is optimal. Moreover, surface rendering assumes that the data consists of tangible surfaces that can be extracted and visualized. Therefore, for some complex cloud and gel-like objects, it is not possible to have a surface representation. Most importantly, surface rendering techniques only concentrate on the boundary geometry of an object and do not provide any information regarding the internal structure of the object. When the surface is extracted and rendered, the interior data values are not displayed.

Volume rendering, on the other hand, requires a large memory to hold the entire data set, during every phase of visualization. Any part of the object, including interior structure can be viewed and explored, because the entire data set is maintained throughout. This process is inherently slower than its counterpart because the whole data set needs to be handled at every phase. But very importantly, this technique lets us see the inside of an object, beneath the surface. For instance, while a surface rendering of the human body might show the skin, a complete volume rendering also shows the bones and internal organs, visible from any side in proper perspective. Furthermore, volume rendering does not assume any particular boundary structure, and can work with objects of any structure. For cloud or gel-like objects, the method assigns properties, such as light attenuation, brightness, to each volume primitive, and can be displayed without any modification of the algorithm. Volume rendering does not require much preprocessing, as surface rendering does. But while displaying volume primitives, special software has to be used to project them on the screen, because most conventional software handles only geometric primitives.

Some applications (such as computational fluid dynamics) require surface construction and accurate surface normals. Methods for surface reconstruction from a volume image exist, but the presence of multiple nested surfaces can be obscured

in the volume image. On the other hand, the normals obtained from the gradient volume can often be less accurate than the geometric normals obtained from surface extraction methods.

Both approaches are concerned with rendering volume data so that the information important to the user is retained and portrayed in the image. Since the two approaches seem complementary, the choice of approach (surface, volume, or mixed rendering) depends primarily on the visualization task and the computational requirements.

Chapter 5

Challenges of Future Research: Motivation behind Digital Morse Theory

After navigating through different techniques for volume data visualization we see that most of the research was carried out in an effort to design effective and efficient visualization techniques. Future work will aim at overcoming the existing problems and extending the practical use of these techniques. Some important research challenges in volume visualization that are yet to overcome are listed below.

Meeting the Demand of Interactive and Collaborative Visualization

The need to allow interactive exploration of the data set, transmission over the networks, and the use of volume visualization with multimedia, puts tremendous performance demands on the systems of the future. These demands require better systems

with improved speed (e.g. real time performance for animation), and capable of displaying images and restoring all important information. Each step in the visualization pipeline has to be implemented efficiently. A multi-pronged approach will involve new and improved hardware, the use of multi-resolution and hierarchical models, better compression algorithms, distributed and parallel computing, and sophisticated network caching schemes.

Segmentation and Feature Extraction

Segmentation and feature extraction of the data set is the process that identifies the data points (or voxels) that belongs to a particular material, object, or anatomical feature. It is extremely difficult to develop fully automatic algorithms for segmentation because local attributes are often insufficient to make global decisions. The contextual and higher level knowledge about the data and the object to be identified, are required. Also, when using isosurface techniques to perform feature extraction, it is often difficult to identify the correct isovalue that reveals the desired structure. Designing methods for identifying the isovalues that reveal desired features is an important research challenge.

Level of Details Management

By choosing the right level of details it is possible to learn more from an image within less time and space. Properly done, this also allows one to reduce the rendering complexity in areas of the image with less variation of data values (e.g. correspondingly less interesting geometry), while retaining higher resolution for the important features in other part of the image. Thereby, it allowing one to construct the volumetric image or surfaces more efficiently. Visualization tools need to be designed which allow the

user to choose the different and correct level of details in different parts of the data set.

Volume Modeling

Surfaces and their polygon approximations form the mainstay of modeling objects in computer graphics today. There is a myriad of algorithm to deal with them, and many workstations are specifically designed to process and render them. But methods for modeling and rendering the volume are lacking. Spline and other mathematical models have been used to model surfaces, but an analogous model for volumes is missing. There is a need for volume modeling methods and subsequent discretization algorithms to drive the rendering pipeline.

Efficient Use of Memory and I/O Optimality

For a very large data set, it is often not possible to fit the entire data set in the main memory. Since the data transfer speed from a secondary storage device to the main memory is much slower than the processor speed, I/O operations can slow down the visualization process by quite an extent. More attention should be given to design I/O optimal visualization tools, so that unnecessary I/O operations can be avoided, expediting the visualization process. Related data regions for a specific object or isosurface, should be determined a priori, and only the necessary data points should be retrieved from the disk. This will reduce I/O operations significantly and avoid any unnecessarily large allocation of main memory space. The *I/O Optimal Interval Tree* of [15] are an example of a success in this regard.

Motivation behind Digital Morse Theory

The discussion above certainly suggests that the visualization field has a long way to go before all demands have been satisfied. Future concentration should be given to the development of tools which will attempt to solve these existing problems. Having this in mind, we propose to implement a suite of data set visualization tools, using a new method to organize discrete volumetric data. The method will provide a complete topological organization of the data and can be used for searching embedded geometry in extremely large data sets.

The method is called *Digital Morse Theory* [25] [26] and is based on an adaptation of the ideas of Morse theory to volume data sets. Morse theory [24] studies the change in topology of the level sets of a Morse function f at the critical points (e.g. where the first derivative vanishes). A Morse function f has the property that all its critical points are isolated and are true local maxima, minima or saddles (e.g. by non-singularity of the Hessian at critical points). In volume data sets, we cannot assume that the interpolating function is Morse. This is because clusters of identically valued readings in the data set can imply plateaus, ridges or other structures. Nevertheless, by exploiting properties satisfied by standard interpolants for volume data, *Digital Morse Theory* is able to characterize the isovalues at which the topology of the level sets and associated boundary isosurfaces changes, as the isovalue is varied. These are the values at which the isosurfaces obtained by, for example, an appropriately corrected *Marching Cubes*, or *Spider Web*, would change in topology.

An algorithm derived from the theory can quickly identify these critical isovalues and the intersected voxels. This is to be contrasted with methods that have applied Morse theory to surfaces produced at a fixed isovalue. From the identified critical isovalues, one can then construct an atlas of all topologically distinct objects in the

volume data over the entire range of isovalues.

The algorithm also segments the data into zones. Zones are the regions of space associated with each topologically distinct family of objects within the data. Each zone consists of the volume swept by the boundary isosurfaces of a single object, as the isovalue is decreased from a critical value until the topology of the object changes (e.g. another critical point is encountered). Thus each zone has an associated real interval $(\tau_1, \tau_2]$, and consists of one or more spatially connected components (e.g. one component for each boundary isosurface of the associated family of objects). Each component of the zone contains a single isosurface for any isovalue within the interval. Conversely, each level set boundary isosurface lies completely within a single zone component.

The zones also provide a representation of all topologically distinct internal structures within the volume data, up to approximate geometry, together with the range of isovalues that reveal each object.

Using the above method, one can organize the volume data set into zone files and zone component subfiles. Each of these files contains the voxels intersected by a given zone. Since each isosurface is completely contained with a single zone component, this organization reduces the I/O needed to compute object boundaries, simply because for a given isovalue, only data points within the related zones have to be scanned. Using the language above, the active cells for an isosurface are all contained in a single zone component. Note that the zone organization uses value space information, spatial proximity, and topological information in its organization of the data. Moreover, the data can be simplified (e.g. by forming larger voxels or cells) within a zone component, while preserving the topology of the associated family of isosurfaces. This allows one to dynamically choose the level of details of the rendering, and reduces unnecessary computation. A visualization tool based on this method allows one

to view all topologically distinct threshold defined objects, to dynamically vary the isovalue within the range of an associated zone, and to dynamically refine the level of geometric details of a given rendering.

Very interestingly, this method preserves all information about the internal structures to a desired accuracy, and identifies the active cells intersecting isosurfaces, which can then be used to produce boundaries. Thus, rather than taking either one of the two traditional paths (e.g. surface rendering or volume rendering), this technique represents a hybrid of both methods. This new methodology attempts to assist one in solving some of the existing challenges associated with very large data set visualization.

Chapter 6

Topological Zone Segmentation of Scalar Volume Data

Topological Zone Segmentation is a preprocessing algorithm to organize discrete scalar volume data on external storage based on the changes in topology of the level set. The preprocessed data can be used very efficiently for applications to out-of-core visualization of large data set. This method [25] [26] based on an adaptation of the ideas of Morse theory to volume data sets, can quickly identify the critical points and the associated isovalues at which the topological changes occur. Based the identified critical isovalues, the algorithm segments the data into zones which are the regions of space associated with each topologically distinct family of objects within the data. The algorithm further segments the zones into several components based on the connectivity, each one of which contains the data points of one distinct object of that family. The zones provide a representation of all topologically distinct internal structures within the volume data, up to approximate geometry, together with the range of isovalues that reveal each object. Once the zones are computed, one can then construct an atlas of all topologically distinct objects in the volume data over the

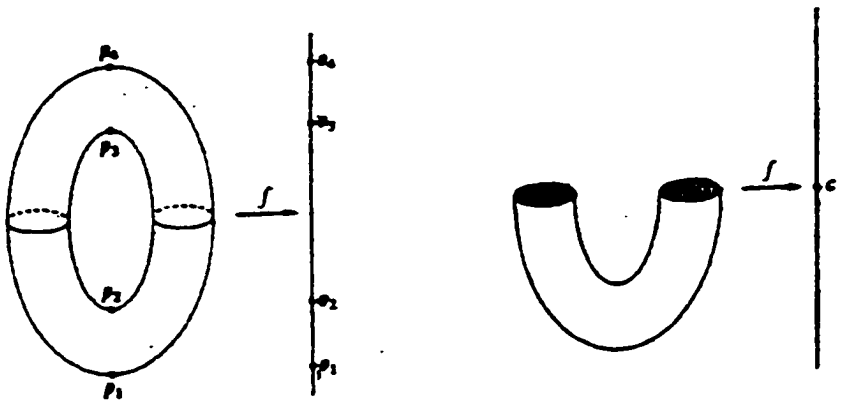

Figure 6.1: Critical points and associated topology changes

entire range of isovalues. Also, isosurface extraction and a topologically correct isosurface simplification can be efficiently implemented because the related data points lie completely within a single zone component.

6.1 Base of the Algorithm: Morse Theory

Morse theory [24] studies the change in topology of the level sets of a Morse function f as the parameter τ is decreased. This changes occur at the critical points of f, where the first derivative vanishes. A Morse function f has the property that all its critical points are isolated and the Hessian is nonsingular at each of these critical points. The critical points at which the topology of the level sets changes are local maxima, minima and saddles. At a local maxima, a level set emerges, at a local minima, a level set vanishes, and at a saddle point one level set splits into more than one level sets or one or more level sets merges into one. Figure 6.1 demonstrates critical points and associated changes in topology recorded by Morse Theory.

6.1.1 The Digital Adaptation: Digital Morse Theory

In volume data sets, one cannot assume that the interpolating function is Morse. This is because clusters of identically valued readings in the data can imply plateaus, ridges or other structures such as a maxima set at which a level set emerges. The *Digital Morse Theory* relaxes the restriction imposed by Morse theory for having isolated critical points. The identically valued connected data points are considered as a set and are treated similarly as other isolated points. Thus the results can be extended to data where the values are not unique. Topology changing criticalities are given a combinatorial characteristics in this case. Nevertheless, This adaptation does not results in an incorrect output or in any inability in computation. By exploiting properties satisfied by standard interpolants for volume data, *Digital Morse Theory* is able to characterize the isovalues at which the topology of the level sets and associated boundary manifolds of the isosurfaces changes. These are the values at which the isosurfaces obtained by, for example, an appropriately corrected *Marching Cubes* or *Spider Web*, would change in topology. Our Mathematical analysis in the following chapter demonstrates that the contours produced by the algorithm are sufficiently well behaved so that the topological changes, as the threshold is varied, are well defined and easily identifiable, without any recourse to Morse theory.

Chapter 7

Mathematical Analysis

The mathematical analysis presented here starts by defining some terminologies used in rest of the chapter. The properties satisfied by admissible interpolants for volume data are stated in subsequent section, followed by an analysis that shows that the topological changes in contours are uniquely defined by admissible interpolants for volume data and can be inferred from local information. The data obtained in many practical applications can not be extended to a Morse function, due to the restriction imposed for having only isolated critical points, without modifying the data by re-gridding and perturbation. Rather than modifying the data, we use an adaptation of Morse theory, namely *Digital Morse Theory* and analyze the behavior of the standard interpolants commonly used in isosurface extraction methods. The mathematical analysis presented here shows that the contours produced by these methods are sufficiently well-behaved and critical points are uniquely defined. The results obtained by this method are more general and conform to the isosurfaces that one extracts in practice.

7.1 Definitions

Definition 1 *We will assume the volume data set is given by a real-valued function δ defined on a discrete set of points V in R^n. Regularly gridded data is given by a function defined on Z^n, where we form unit hypercube in natural way. Irregularly gridded data is given on the vertices of a simplicial decomposition of R^n and we assume that the vertex adjacency information is represented in some form in external storage. We term both hypercube and simplices, cells. Without any loss of generality, we shall assume δ is non-negative over the entire domain, and that $\delta > 0$ on a finite subset of points. If δ does not take the same value on any two points, we say that it satisfies data uniqueness.*

For simplicity, we will describe our algorithm for volume data that satisfies data uniqueness at first. Later we will extend it for non-unique data, where identically valued spatially proximate data reading imply isovolumes.

Definition 2 *A continuous real valued function f interpolates δ if it extends δ to all of R^n and is nonzero only on the finite subset of cells that δ is nonzero on.*

We denote by $f^{-1}(> \tau)$, the set of $p \in R^n$ such that $f(p) > \tau$, for f that interpolates δ. The topologically connected components of this set are called objects. Similarly $\delta^{-1}(> \tau)$ denotes the set of $p \in V$ such that $\delta(p) > \tau$. Clearly if f interpolates δ then $\delta^{-1}(> \tau) \subseteq f^{-1}(> \tau)$. The points $p \in V$ whose function value is above or equal to (respectively, below) the threshold are termed as Highs (respectively, Lows).

Definition 3 *The isosurface construction problem is to compute the contours or the boundary of the set of $f^{-1}(> \tau)$ for specific real number τ. We use the notation $B(f^{-1}(> \tau))$ to denote the topological boundary of set $f^{-1}(> \tau)$, which encloses all the Highs of the data set for isovalue τ.*

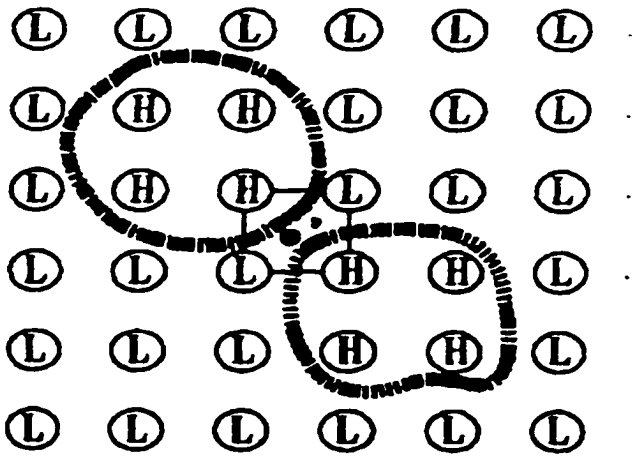

Figure 7.1: Connected sets of Highs

Commonly used isosurface extraction methods construct $B(f^{-1}(> \tau))$ with the simplest topology, consistent with the data. These methods interpolate a single contour intersection point named as hit, along a cell edge, if and only if the two end points are identified as High and Low, respectively. Objects are defined as connected set of Highs (see figure 7.1) and the isosurfaces are the contours of this objects modeled using geometric primitives. For irregularly gridded data, connectivity between the Highs are defined solely by cell edge adjacency. However for regularly gridded data, cell edge adjacency is not sufficient. It may contain the well-known ambiguity, which occurs when for a range of threshold a cube face F contains diagonally opposite Highs and diagonally opposite Lows. We call F a 4-Hit face, because there are hits on each edge. The ambiguity is resolved in a similar manner as it is done in *Asymptotic Decider* (see section 3.2.1) proposed by Neilson.

Definition 4 *The Disambiguation value c with respect to a function f is the maximum threshold for which the diagonally opposite Highs in a 4-Hit face F are locally connected through the interior of a cell sharing the face.*

Note that once the data points are path connected they will remain connected for all threshold less than c. We associate a disambiguation point with F, having the value c, which may actually be in the cube interior, and extend δ to this point. For isovalue $\tau > c$, we create a pseudo-edge connecting the two diagonally opposite Lows. For any isovalue $\tau \leq c$, a pseudo-edge connects the two diagonally opposite Highs. We extend the definitions of adjacency and connectivity of Highs and Lows to include the pseudo-edges. The coordinate and the value c of the disambiguation point of a 4-Hit face is computed by bilinear interpolation of vertex values over the face as it is done in *Asymptotic Decider* (see figure 7.2).

7.2 Properties of Admissible Interpolating Functions

We restrict the class of interpolants to those whose contours behave in a manner consistent with standard isosurface extraction methods. The properties of the admissible interpolants are described as follows.

Let τ be an isovalue not in the range of the extended δ (e.g. not a data value or disambiguation value).

- For n dimensional data the contours of $B(f^{-1}(\geq \tau))$ form a finite collection of disjoint, compact and oriented $n-1$ dimensional manifolds embedded in R^n. These contours divide R^n into disjoint n dimensional connected components, each of which contains either a single non-empty connected component of Highs or Lows. An example for 3 dimensional case is shown in figure 7.3, where the space is divided between disjoint 3 dimensional connected components of Highs and Lows, by their 2 dimensional boundary manifolds.

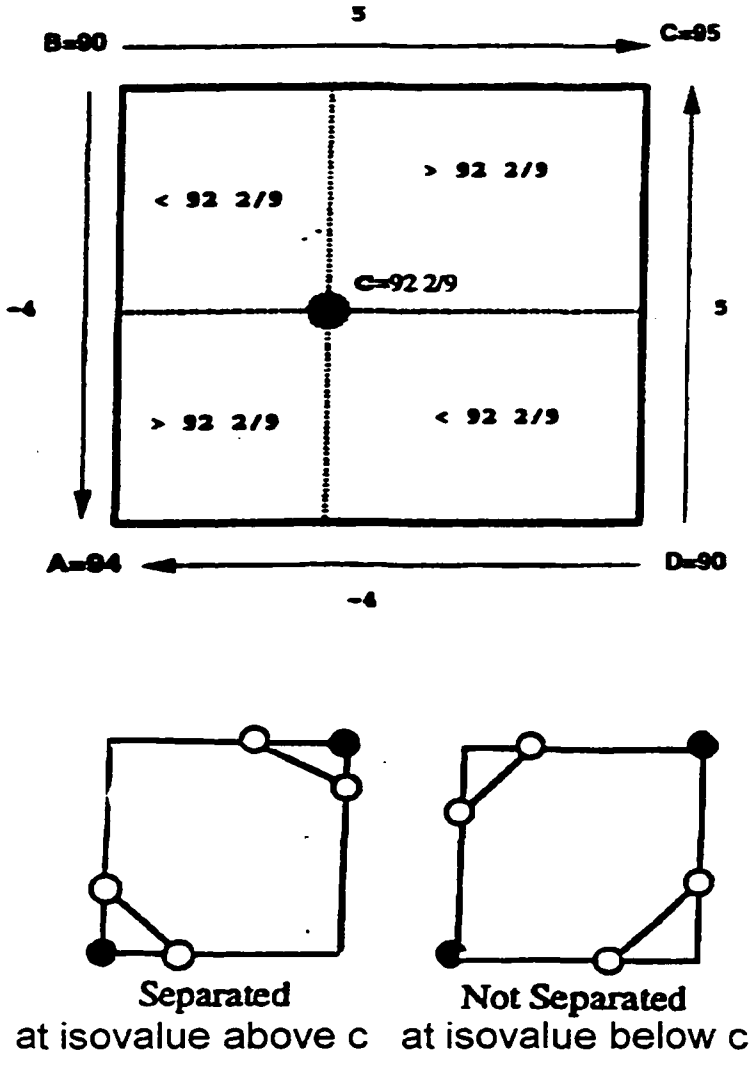

Figure 7.2: Disambiguation value c and connectivity of a 4-Hit face

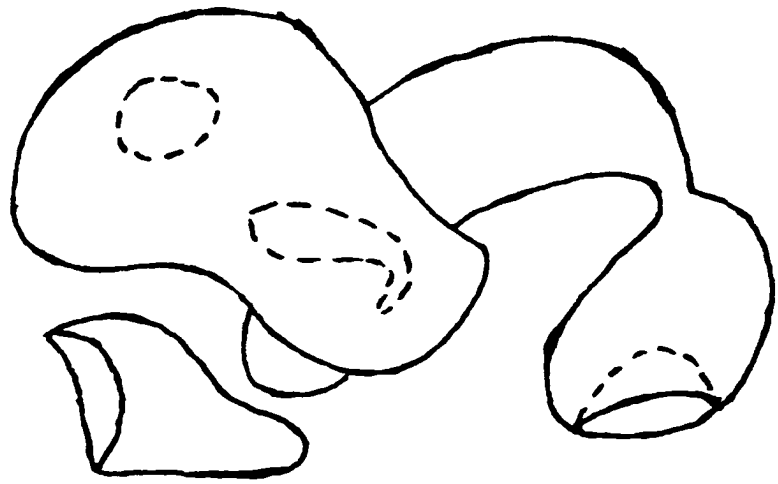

Figure 7.3: Connected components in 3D

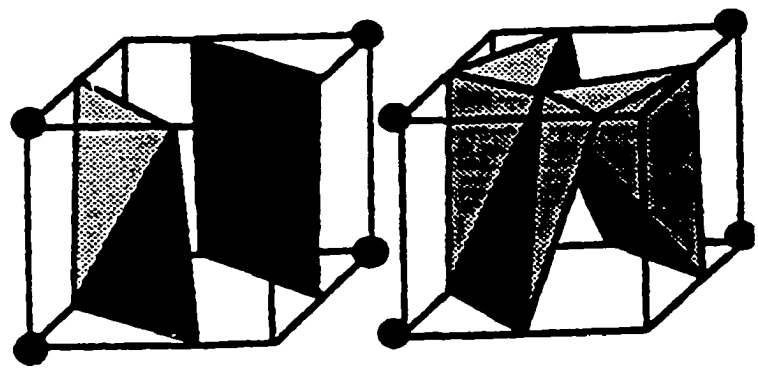

Figure 7.4: 3D Cell divided among components of Highs and Lows with 2D surface patches

- The intersection of the contours with any d-dimensional cell C, $B(f^{-1}(\geq \tau))\cap C$, where $d \leq n$, is a finite set of $(d-1)$ dimensional components called surface patches. These surface patches are bicontinuously mappable to a $(d-1)$ dimensional unit disk (ball) and the intersection divides C into disjoint d dimensional simply connected components, each of which contains precisely the vertices from a single non-empty connected components of Highs or Lows. An example for a 3 dimensional hypercube is shown is figure 7.4.

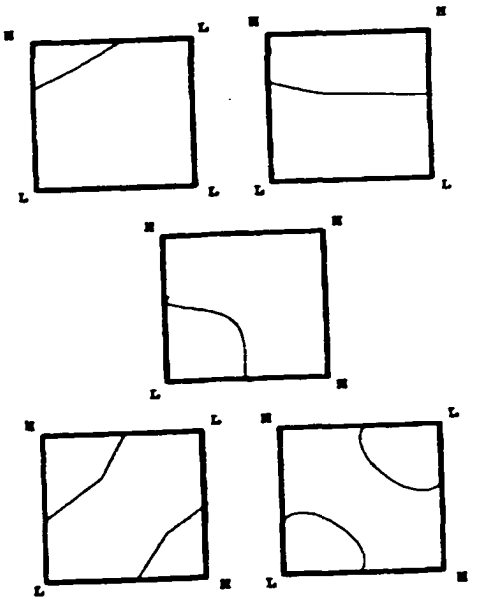

Figure 7.5: Possible intersection of the isosurface with a cubic cell face

For τ in the range of δ we require that $B(f^{-1}(\geq \tau)) = B(\cap_{x<\tau} f^{-1}(\geq x))$. This insures that the boundary encloses all the data readings precisely equal to τ.

Definition 5 *A function f is admissible if and only if it interpolates δ and satisfies the properties listed above.*

Property 1 merely expresses the goal of isosurface construction from a global point of view, and property 2 expresses the type of local non-degeneracy assumed by isosurface construction algorithms. Figure 7.5 illustrates all possible intersection of isosurface with a cubic cell face and associated division of objects. For tetrahedral cell there is always single patch that intersects the cell. For cubic cells, the number of distinct surface patches can go up to four. This situation occurs when there exists a particular threshold for which all cell faces are 4-Hit face and the given isovalue is above the disambiguation value of all faces. All the 4 High vertices will be part of disjoint components of Highs.

7.3 Admissibility Uniquely Defines Topology

We now show that these properties are sufficient to uniquely define the topology of irregularly gridded data. For regularly gridded data, the disambiguation values of 4-Hit faces are also required. More precisely,

Theorem 1 *if f and f' are admissible and additionally, for regularly gridded data both of them have the same disambiguation values, then the boundary contours (manifolds) of $B(f^{-1}(\geq \tau))$ and $B(f'^{-1}(\geq \tau))$ are homeomorphic.*

Proof: We prove this theorem by a series of lemma (see appendix A). Here is an outline of the proof. We first observe that no cell face can have an odd number of hits. The dual of data reading connectivity is the hit connectivity. We identify the hits by the cell edge on which they occur. Hits are adjacent if they are connected by a curve segment of the contour in a cell face. Property 2 ensures that pair of hits on a cell face with 2 hits is adjacent. For a 4-Hit face the hits will be divided into 2 adjacent pairs (see figure 7.2). Since both f and f' are admissible and have the same disambiguation value for all 4-Hit faces, they will have the same hit set and the same connectivity between the hits for any isovalue. Let M and M' be the manifold of $B(f^{-1}(\geq \tau))$ and $B(f'^{-1}(\geq \tau))$. We form a graph from the surface patches of a manifold where the nodes of the graph are patches and the edges connect intersecting patches. Since intersecting patches must share a common hits and the patches intersect by sharing lower dimensional patches, it follows from property 2 that we can iteratively construct a continuous bijection from M to M'. In order words, one can show that the graphs G and G', formed from M and M', are isomorphic.

Chapter 8

Critical Values

For the rest of the discussion we refer to components of $f^{-1}(\geq \tau)$ as objects and components of $f^{-1}(< \tau)$ as complementary objects. Topology changes occur when the isovalue becomes equal to a critical value. Let us first look at the possible critical values and the associated changes in topology before arguing how the admissibility uniquely defines the topological changes.

8.1 Critical Points for Volume Data Set Satisfying Data Uniqueness

Let us assume that δ satisfies data-uniqueness. A critical value c is a value at which the topology of the contours of f change as the threshold is decreased through c. Each such change has an associated critical point p for which $f(p) = c$. We use the term critical point in analogy to Morse theory, but we are not actually using any differential properties of the Morse function. We rather use the admissibility properties of the function to define the topology changes.

Let $p \in V$. Let N be the set of points in V adjacent to p by cell edge or pseudo

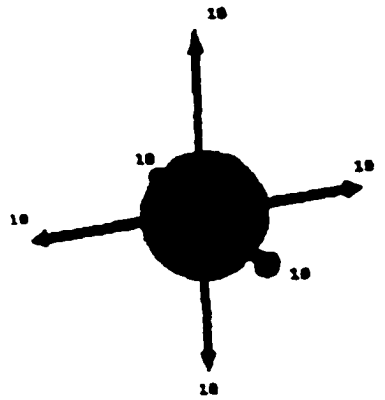

Figure 8.1: Local maximum critical point

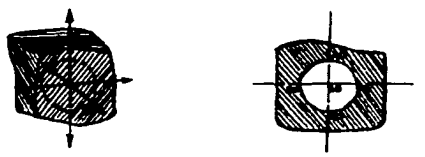

Figure 8.2: Local minimum critical point

edge. Let N_H (respectively, N_L) be the Highs in N (respectively, Lows) with respect to the value of p. Compute the connected components of N_H and N_L within the cells that share p but excluding p itself. For example, two Highs are connected if they are connected within the cells that share p by a path that does not include p.

1. If $N = N_L$, meaning p has a higher value than all its neighbors, then p is a local maximum critical point and $f(p)$ is a local maximum critical value. A new contour emerges at each of these critical points (see figure 8.1)

2. If $N = N_H$, meaning p has a lower value than all its neighbors, then p is a local minimum critical point and $f(p)$ is a local minimum critical value. An existing contour vanishes at each of these critical points (see figure 8.2)

3. If either N_L and N_H consists of more than one connected component then p is saddle critical point and $f(p)$ is a saddle critical value. Two or more contours

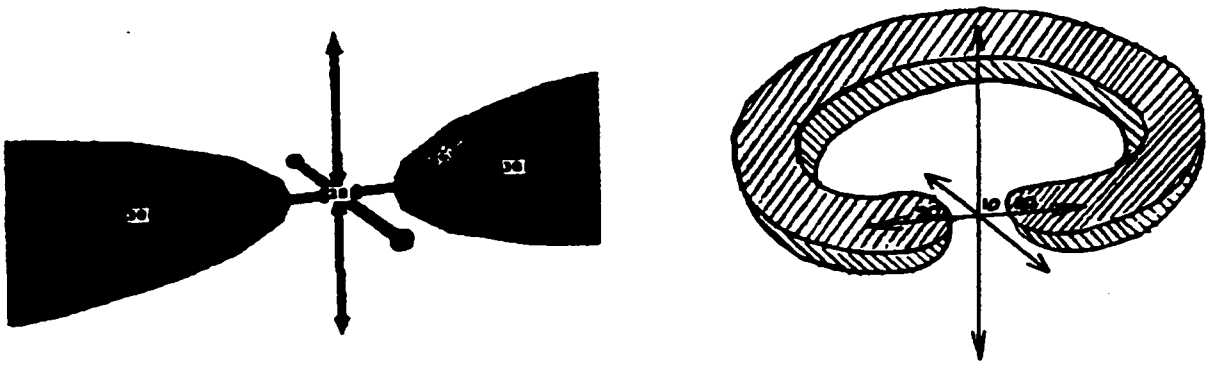

Figure 8.3: Saddle critical point

may merge into one, contours may split into two or more pieces or genus change can occur at these critical points. Figure 8.3 gives an illustration of these topological changes at saddle critical points.

4. Let p be the disambiguation point of a cell face F with disambiguation value c. If the Highs of F (respectively, Lows) are not part of the same connected component of Highs (respectively, Lows) within the cells sharing F for all threshold values $c + \epsilon$ (respectively, $c - \epsilon$) then p is saddle critical point and $c = f(p)$ is a saddle critical value. Similar topological changes as described for the previous type can occur at critical disambiguation points. However, not all 4-Hit faces produce critical disambiguation point. Figure 8.4 differentiates between a critical and a non-critical disambiguation point.

Critical points of type 1, 2 and 3 can be encountered both for simplicial and regularly gridded data, whereas critical points of type 4 can occur only for the later data type.

Topology changes to the contours can only occur at critical values (see appendix B). Hence the following theorem can be stated as,

Theorem 2 *Let f be admissible and δ satisfy data uniqueness. If $[\tau_1, \tau_2]$ is a critical*

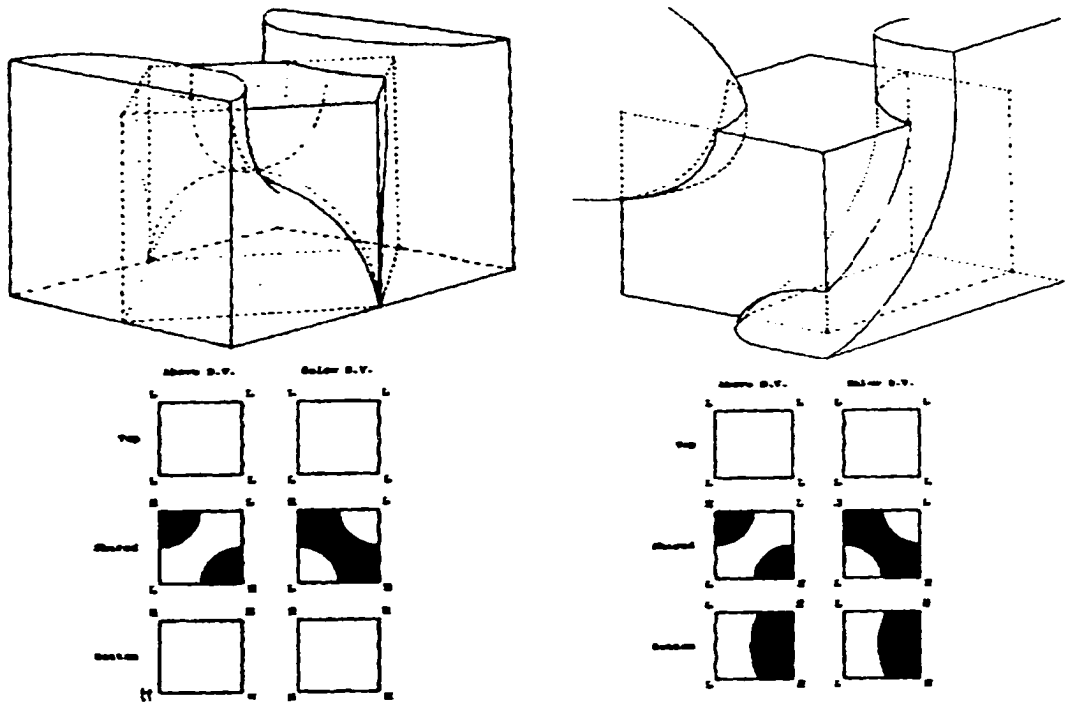

Figure 8.4: Critical and non-critical disambiguation point

value free interval then $B(f^{-1}(\geq \tau_1))$ *and* $B(f^{-1}(\geq \tau_2))$ *are homeomorphic. Conversely topological changes to* $B(f^{-1}(\geq \tau)$ *occur only at critical values.*

Proof: Refer to the appendix B for the proof.

8.2 Critical Isosets when Data Set does not Satisfy Data Uniqueness

If the data does not satisfy data uniqueness then the properties of the admissible functions imply that connected sets of data points (as Highs) of identical value c will be enclosed in isovolumes at thresholds $c-\epsilon$. As the threshold is decreased through c, an object can merge with the isovolume containing these points. Complex topology change can occur at the value c of these sets, termed isosets. Let S be an isoset

with value c and N, N_H and N_L be defined as previous section. For an admissible function f, in addition to the point criticalities described in section 8.1, there are set criticalities.

1. If $N = N_L$, then S is a maximum isoset and c is a maximum critical value. A new contour emerges at each of these critical isosets.

2. If $N = N_H$, then S is a minimum isoset and c is a minimum critical value. An existing contour vanishes at each of these critical isosets.

3. If either N_L or N_H contains more than one connected components, then S is a saddle isoset, and c is a saddle critical value. Two or more contours may merge, contours may spit into two or more pieces, or a genus change may occur at these critical isosets

4. If N_H and N_L, each is consist of a single nonempty component then S is called a regular isoset. In 3 dimensions, this can cause a genus change in an object or complimentary object O if, when it merges with O it adds or destroys a handle. This occurs if there exists a loop through N_L or N_H and a loop through S so that the loops are interlocked. For $n > 3$ dimensions the condition is that there exists a closed $n-1$ dimensional surface in N_L or N_H and a closed $n-1$ dimensional surface in S so that the two surfaces are interlocked, but non-intersecting. In this case the corresponding Betti number of a contour is changed.

Note that a new object of arbitrary topological complexity with possibly multiple boundary manifolds is created at a maximum isoset. One or several manifolds vanish and multiple objects can merge at a minimum isoset. Extremely complex merges, joins, splits, and genus change can occur at saddle isosets. At regular isoset, genus

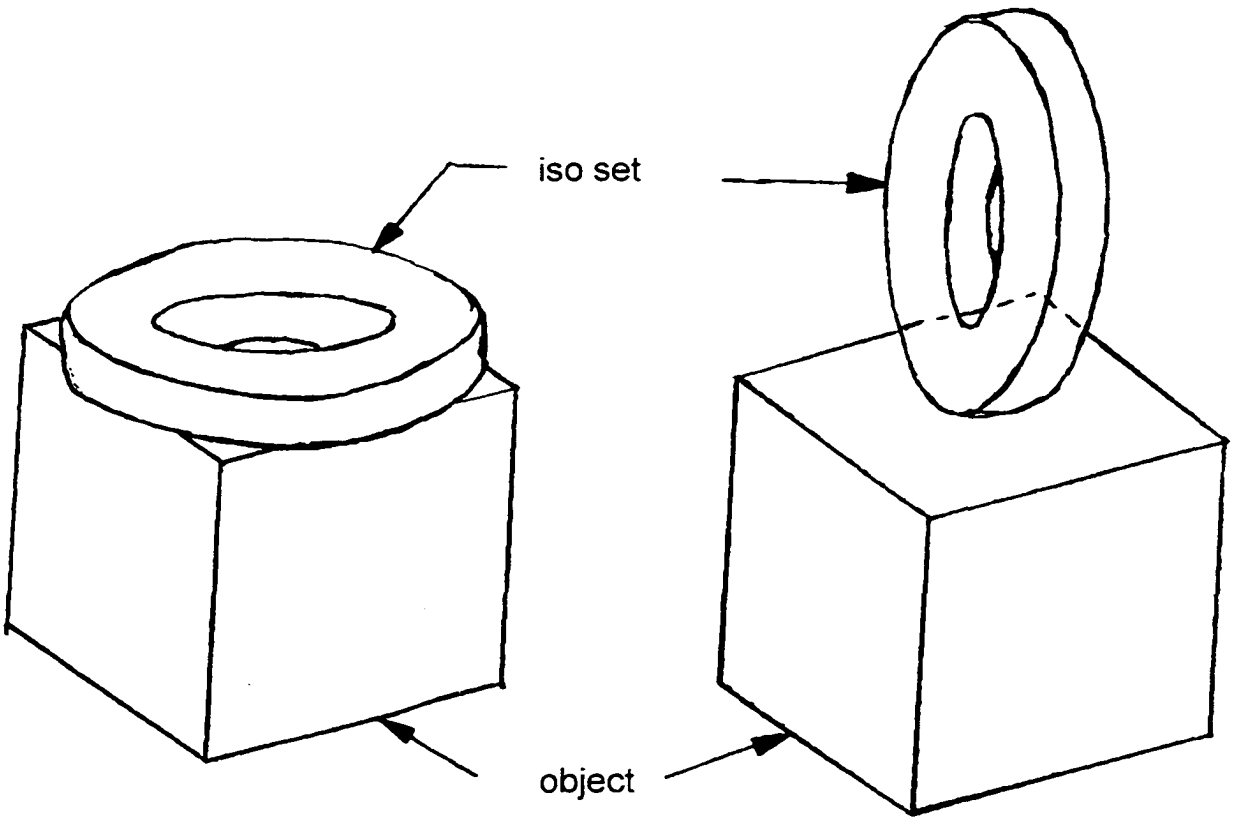

Figure 8.5: Critical and non-critical regular isoset

change can occur. For example, consider a donut-shaped critical regular isoset with flat bottom and sides. Suppose this critical set merges with a solid cube object. If the merge is such that the donut rests flat on top of the cube, then no topological change occurs because boundary of the resultant object is still homeomorphic to a sphere. But if the merge is such that the isoset is glued to the top of the cube resting on its side, then a new handle will be formed causing genus change. Both of these cases are depicted in figure 8.5. We leave it as an open problem to develop an efficient combinatorial algorithm for recognizing critical isosets in all dimensions.

With the addition of the critical isosets we can drop the data uniqueness requirement and Theorem 2 can be revised as follows.

Theorem 3 *Let f be admissible, if $[\tau_1, \tau_2]$ is a critical value free interval then $B(f^{-1}(\geq \tau_1))$ and $B(f^{-1}(\geq \tau_2))$ are homeomorphic. Conversely topological changes to $B(f^{-1}(\geq \tau)$ occur only at critical values.*

Proof: An maximum or minimum critical isoset S in 3 dimensions can induce changes in the number of contours. Consider a saddle isoset S with critical value c. Assume without any loss of generality that N_H consists of at least two connected sets of Highs, namely N_1 and N_2 (e.g. the case of Lows is symmetric). If N_1 and N_2 are part of two different global objects, then those two distinct objects will merge at S as the threshold is lowered through c. Assume that, they are globally part of the same object O above the critical threshold c. One can then find a loop L on the boundary faces of N that separates these two component N_1 and N_2, so that the loop lies completely outside of O. This is because, as in the proof of theorem 2, by the definitions of connectivity of Highs and Lows a loop consisting of cell edges and cell face diagonals connecting adjacent Lows can be found. As in the proof of theorem 2, one can find a loop L' formed interior to object O, when these two object portions merge at the threshold c. This loop L' is incompressible, as it is interlocked with the above loop L.

The proof for dimension $n > 3$ is similar. In this case the two sets of Highs can be separated by a closed $n-1$ dimensional surface Σ on the boundary of the cells bordering S (e.g. not in the interior of any cell). Again, as in the proof of theorem 2, Σ can be constructed from $n-2$ dimensional cells. In this way the class of closed surfaces formed on the contour of O when the object merges with S at the threshold c, will be incompressible. This is because each such surface will be interlocked with, but not intersect Σ, thus indicating a change in the corresponding Betti number of the contour.

An additional type of topology change can occur, one that is caused when an isoset is merged with another object and either adds a class of incompressible loops (or surfaces) to the object or destroys a class of incompressible loops (or surfaces) in the complementary object.

Hence the admissibility uniquely defines the topological changes with can only occur at critical values.

Chapter 9

Topological Zones and Criticality Tree

9.1 Topological Zones

The intuition behind the *Topological Zone* is quite simple. As the threshold is decreased from the value of a critical point p, the object containing the criticality grows and deforms with homeomorphic boundary until it encounters another criticality q, which in some manner changes the topology of the contours bounding the objects. The *Topological Zone* of criticality p is the volume thus swept by the topological boundary of this object. Each connected component of this zone is the volume swept by individual boundary manifold of the object. Figure 9.1 gives a visual representation of the zones.

We define the zones by a set difference so that no assumption that the contours vary continuously is required. In practice, if the data contains isoset, the contours will not vary continuously.

Let $O_p(\tau)$ be the object containing p at threshold τ. For any τ that is equal to

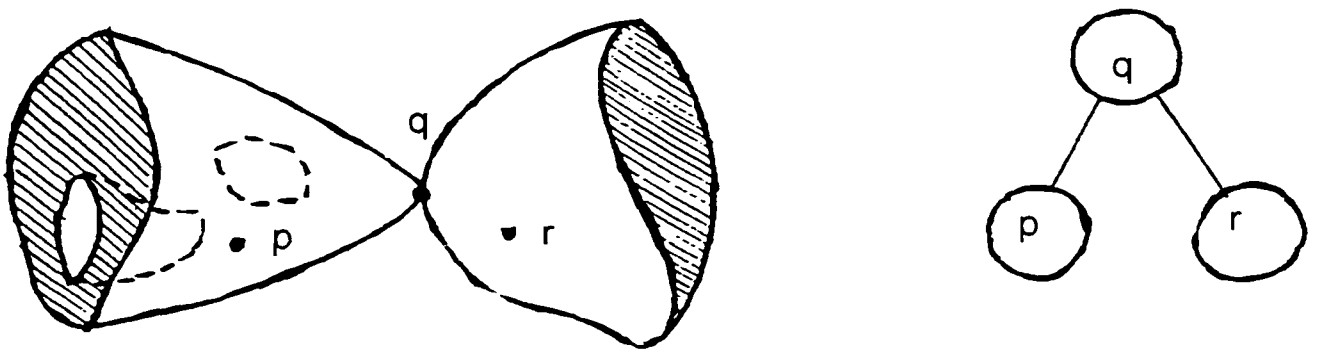

Figure 9.1: *Topological Zones* and corresponding *Criticality Tree*

a data value, define $O_p(\tau)$ to be the closure of $\cup_{c>\tau} O_p(c)$. The *Topological Zone* for criticality p can be defined as,

Definition 6 *Let p be a critical point with critical value x_p. As the threshold is decreased from x_p, suppose q is the first criticality with critical value x_q, becomes part of the same object containing p and there exists no other critical point r with critical value x_r within the same object, such that $x_p > x_r > x_q$. The Topological Zone of p, denoted as $\zeta(p)$ is the set difference between $O_p(f(q)) - O_p(f(p))$.*

Definition 7 *The Topological Zone components are the connected components of the zone of the critical point p, $\zeta(p)$.*

9.2 Criticality Tree

The *Criticality Tree* is a search structure that hierarchically organizes the zones based on values and proximity, and traces the evolution of the objects as it gets created, destroys, merges, splits or genus change occurs. This tree is used to navigate and identify the zones that includes data points of the cells intersected by the isosurface for any range of isovalue. At any critical value, a new zone is created and it keeps

growing till it meets another criticality, when this zone freezes and another new zone is formed. The nodes and the edges of the *Criticality Tree* are formed in the following way.

- Each node in the *Criticality Tree* represents a criticality.
- As the threshold is decreased from the value of a critical point p, if q is the first critical point encountered within the same object that contains p, then q is made the parent of p (see figure 9.1). For data that does not satisfy the data uniqueness, a given object containing criticality p may meet several criticalities as the same time. We just arbitrarily order the parentage of these to break ties.

9.3 Properties of the Topological Zones

Theorem 4 *Let p be a criticality with parent q, where $f(p) = a$ and $f(q) = b$.*

1. *For each τ satisfying $a \geq \tau > b$, $B(O_p(\tau))$ is contained in $\zeta(p)$ and each component M of $B(O_p(\tau))$ is contained in a single component of $\zeta(p)$. This means for any isovalue between the value of a node that represents the critical point and that of its parent's, the contour is completely contained within the zone for the criticality corresponding that node.*
2. *For an pair $[\tau_1, \tau_2]$, if $a \geq \tau_1 > \tau_2 > b$, $B(O_p(\tau_1))$ and $B(O_p(\tau_2))$ are homeomorphic.*
3. *For all τ, if O is any component of $B(f^{-1}(\geq \tau))$ that does not contain p, then $B(O)$ and $\zeta(p)$ are disjoint.*
4. *For all τ, such that $\tau > a$ and $\tau \leq b$, $B(f^{-1}(\geq \tau))$ and $\zeta(p)$ are disjoint.*

Proof:

A point $r \in \zeta(p)$ if and only if $f(q) \leq f(r) < f(p)$ and $r \in O_p(f(q))$. For any τ, $f(p) > \tau \geq f(q)$, each point $r \in B(O_p(\tau))$ satisfies $f(r) = \tau$ by continuity of f. Also, because each such $r \in O_p(f(q))$ too, $B(O_p(\tau))$ is contained in $\zeta(p)$ by monotonicity of $f^{-1}(\geq \tau)$. The connectivity of the manifolds insures that it is contained in a single component of $\zeta(p)$, establishing claim 1. By definition, $\zeta(p)$ contains no critical point between with a value between $f(p)$ and $f(q)$ which establishes the second claim. Claim 3 follows from the connectivity of the $O_p(\tau)$ and the fact that boundary manifolds are pair-wise disjoint by the property 1 of admissible interpolant functions. Claim 4 follows immediately from the proof of claim 1.

Chapter 10

Zone Segmentation Algorithm

10.1 Computing Zones when Data Uniqueness is Satisfied

The preprocessing of the volume data to compute the cells that intersect each zone component is performed in three phases, which are described below. The zone computation algorithm proceeds by assigning labels to the data points, according to the zone, where the particular point belongs. We shall assume that cell edge adjacency information is stored in the disk for irregularly gridded data.

Phase 1: Identifying the Critical Points

In the first phase the data is scanned thoroughly and the critical points are identified by computing the connected components of Highs and Lows in its neighborhood, as described in section 8.1. Because each critical point is always part of the zone that corresponds to its own criticality, they are labeled by their own coordinates, i.e. assigns $l(p) = p$, where p is a critical point. Simultaneously, it also creates the zone

file for criticality p for storing the information of all data points included in that zone. The critical points are then put into a max-heap ordered by value (with priority given to a critical point to break ties when we extend to non-unique data). Rest of the non critical points remain unlabelled at this point.

Phase 2: Zone Computation

One then iterates the following step until the max-heap is empty. Remove the maximally valued point r from the heap with label $l(r) = p$. Point r can either be a critical or non-critical point. We elaborate the actions taken in both the cases below.

Non-Critical Point:

For a non-critical point $r \neq p$, examine the neighbors of r. If a neighbor q of r is unlabelled then assign a tentative label to q from the label of r (i e. $l(q) = l(r) = p$) and place q on the heap. The label of r is now finalized and it is placed in the file for criticality p.

Critical Point:

For a critical point $r = p$, examine the neighbors of r and if there any unlabelled neighbor assign the tentative label as before. If there is a neighbor q that has a finalized label different from the label of r or in other words $l(q) = a$, such that $a \neq p$, then make the node that corresponds to criticality a, parent of the node that corresponds to the criticality r. The flooding of the zone for criticality r is now over. The points that are still waiting in the heap with a tentative label p are relabeled tentatively by a, and each one of them are also placed in the file for criticality r (see section 10.2 for discussion on efficient relabeling).

Critical disambiguation points are regarded adjacent to all the four vertices of the 4-Hit cell face.

Phase 3: Zone Component Computation & Cell List Creation

One divides each zone file into its connected components. This can be done by employing sparse matrix technology and using depth first search or the well-known set union-find data structure.

After the zone components are computed a new file is created for each zone component that stores the cell information (e.g. coordinate, values at all vertices, local maximum and local minimum) of all the cells whose one or more data vertices lie within the zone component. The cell information are also sorted by the local maximum first followed by the local minimum values to enable efficient active cell search.

Observe that the algorithm labels the entire data set by highest value first order. That ensures that a zone is expanded as much as possible by searching in lowest steepest descent direction before it is terminated. Also, at any point of time, all points greater than or equal to a particular τ have been labeled. These are precisely that points that comprise the components of $f^{-1}(\geq \tau)$. Whenever a zone labeling for a point p terminates, the points remaining in the heap with a tentative label p are the ones that lie in the cells that contain the boundary of this zone, and the data points with the finalized label of p are precisely the points contained in $\zeta(p)$. Both of these types of points are placed in the zone file for criticality p, so that the complete cell information can be fetched. The cells that are intersected by a zone boundary are included all the zones that is bounded by that boundary. These cells are part of multiple zones, so our zone segmentation is not a true partition of the data points.

10.2 Implementing Zone Segmentation Algorithm

Because our primary focus is to limit the I/O and maintain as less in-core data as possible throughout the execution, there are several techniques that are used for efficient implementation.

Relabeling

The most expensive step is the relabeling of the boundary points of a zone in phase 2. Instead of immediately relabeling the points, we use a lazy approach that results in correct but efficient relabeling. The points that have to be relabeled are left as it is with the first tentative label that was assigned to it, as long as it remains in the heap. Only when they are removed from the heap, we trace the parent pointer of the node that corresponds to the current tentative label, up to the root node in the partially built criticality tree, give them a finalized label that corresponds to the root node. While following the parent pointer, the data points are also placed in all the zone files corresponding the nodes on the path. These points occur in the cells that straddle the boundary of the zone corresponding each node.

Data Read into Main Memory

During phase 1 the maximum number of consecutive slides that are read into the main memory at any points of time are restricted to three. This is actually enough to detect the critical points in the middle slide. For instance, suppose currently slide $(i-1)$, i and $(i+1)$ are read into memory. All critical points in the slide i are detected and then the next slide $(i+2)$ is read into the same space as $(i-1)$, for detecting the critical points of slide $(i+1)$. If the data includes isoset then additional data needs to be read from arbitrary slides. In that case only the necessary data points

are brought into memory, not the whole slide.

Compute Connected Components of One Zone at a Time

We choose to compute the connected component within one zone at a time, so that only a single zone needs to be maintained at once in the memory.

Max-heap Size

In phase 2, the data readings are immediately placed into the zone files once they are removed from the heap. So, the number of in-core data is proportionate to the contour size. We can further reduce the memory complexity by sorting the criticalities and only labeling one zone at a time.

Active Cell Search

The critical value associated with each zone represents the maximum value of the zone. This is the maximum value any data point within the zone may have. The zone is not searched for active cells if the given isovalue is above this value. The critical value associated with the parent represents the minimum value of the zone. It represents the boundary value at which the zone has terminated. The zone is not searched for the active cells if this value is higher than or equal to the given isovalue.

While constructing the zone components, the program also calculates the maximum and minimum value associated with each component by searching the values of all the cells included in the component.

Results Obtained are Stored at each Phase Completion

Each phase is implemented in such a way so that the results obtained during each phase, are written in the disk at completion. Execution of any phase can be carried out independently given that all the previous phases has been carried out before. The execution of the whole procedure need not be done continuously and can be stopped after any phase and the next phase can resume later by reading the results stored from the previous phase and initializing the environment.

10.3 Zone Segmentation Algorithm when Data Uniqueness is not Satisfied

We modify the zone labeling algorithm to handle the isosets by modifying the phase 1 so that it uses depth-first search to identify all isosets as well. In addition to the critical points, we also identify the critical isosets of type 1 through 3, and the regular isosets. Select an arbitrary point in each isoset as the label, and place only this point into the heap. We call this point the *Set Representative* which is given a higher priority than a non set representative point of the same isoset.

Phase 2 is modified as follows. If a set representative point is selected then compute the isoset using depth first search and examine the neighbors of the isoset. The whole isoset is placed in the zone file for corresponding criticality. At the end of the iteration cycle, remove any point that is part of the isoset from the heap.

Since we do not have a exact procedure to identify critical regular sets, we assume all of them to be critical. However in that case, some zones may be divided by regular isosets that are not actually criticalities. This will not change the properties of the zones that we have defined, but will segment the zone unnecessarily and make

it smaller. In order to remedy the problem, one solution can be stated as follows. Consider a isoset S with value c. Use an isosurface extraction method and compute the triangle mesh that models the boundary of the objects in the neighborhood of S for isovalue $c + \epsilon$ and $c - \epsilon$. Using Euler's formula determine whether or nor a genus change has occurred as the threshold is decreased from $c + \epsilon$ to $c - \epsilon$. If so, then the regular isoset is a criticality and leave the zones as it is. If not, then it is not a critical isoset and merges the corresponding zones together.

Chapter 11

Computing Isosurfaces

11.1 Basis: The Spider Web Algorithm

The original *Spider Web* algorithm [22] constructs the isosurfaces for regularly gridded 3D data by creating the surface patches in the cells as follows.

For each connected set of hits within a given 3D cube, we interpolate an interior *Articulation Point* (AP) which is the center of mass of all the hits in that set. For each pair of adjacent hits in the set, we form a triangle consisting AP and the two adjacent face hits. We repeat this action for all cubes with hits. Figure 11.1 shows an example of isosurface construction for a 3 dimensional cubic cell.

The proof of correctness for 3D is quite simple and illustrated in detail in [22].

11.2 Implementing the Spider Web on Zone Components

The visualization tool employs *Spider Web* algorithm as the basis for constructing isosurfaces. This tool is used to construct the contour at any given isovalue τ, after

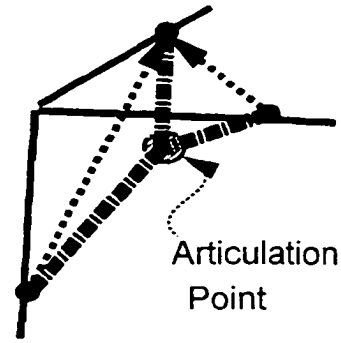

Figure 11.1: Constructing isosurfaces using Spider Web

the data is preprocessed, segmented into zone and associated components and cell list for each zone component is created and sorted by their extrema values. The tool first uses the criticality tree for navigating into the correct zone components and active cell search. It then produces the isosurface only within the active cells using *Spider Web* algorithm. The procedure works as follows.

1. The visualization toolkit at first reads the *Criticality Tree* from the disk and rebuilds it.

2. Given a isovalue τ, it travels the tree nodes in preorder, starting at the root node and conducts the search for active zone components as follows. Let $\zeta_{current}$ is the zone associated with the current node being traveled and we call a zone component active if the any cell within the component is intersected by the isosurface.

   - if $\tau > min(\zeta_{current})$ then the active zones are not yet reached. All of its children nodes are visited.
   - if $max(\zeta_{current}) \geq \tau < min(\zeta_{current})$ then the boundaries of the zone that corresponds to the current node straddle the given isovalue τ. Scan through its component list. Suppose, there are n components of this zone. For any

zone component ζ_{comp_i} of the current zone, where $i = 1, 2, \ldots, n$, that satisfies $max(\zeta_{comp_i}) \geq \tau < min(\zeta_{comp_i})$, report it as active zone component.

- if $\tau < max(\zeta_{current})$ then the active zones in this path have already been visited. So, the descendants of the subtree rooted at the current node are ignored.

3. Once the active zone components are reported the program scans through the cell list of each one of them for active cells. For any cell C, if $max(C) \geq \tau \leq min(C)$, then it is an active cell. The isosurface is constructed locally within the cell using the *Spider Web* algorithm. Because the cell lists are sorted, as soon as a cell with $max(C) < \tau$ is encountered, the search for the active cells is terminated for that zone component.

Using the procedure described above the visualization tool searches for the active cells within active zone components and applies the *Spider Web* algorithm to locally construct the isosurfaces. Since admissibility and disambiguation values uniquely define the structure of the contours and every contour is completely contained within a single zone component, it results in correct and continuous isosurfaces without any holes or tiling errors in it.

Chapter 12

Criticality Tree Vs. Contour Tree

There are some essential difference between the *Criticality Tree* and the *Contour Tree* [35], [36], search structure produced by the other topology-based decomposition algorithm (see section 3.9.1). The two search structures are contrasted and compared in table 12.1.

	Contour Tree	**Criticality Tree**
What does it trace?	Traces the evolution of individual contours.	Traces the evolution of the objects bounded by the contours.
Changes that are recorded	Does not record the change in genus since that does not effect the number of contours. It only records the creation, deletion, merge and split of components.	Records the creation, deletion, merge, split of objects as well as the genus change of the boundary manifolds.

	Contour Tree	Criticality Tree
Leaf nodes	The leaf nodes are the maxima and the minima, where the boundary manifold are created and vanished.	The leaf nodes can only be maxima, where an object is created.
Nodes with multiple children	A node with multiple children can occur when two or more distinct objects merge or two or more boundary manifolds of the same object merge/split.	A node with multiple children strictly signifies that two distinct objects have merged. Any type of genus change in the boundary manifolds of the same object would yield a node with a single child.
Search for active cells and I/O efficiency	While searching for the active cells, a seed cell known to have intersected the contour is first located and then the algorithm locally propagates from it, resulting too many I/O operations.	While searching for the active cells, only the corresponding zone component file containing all the related active cells, is read into memory, resulting optimum I/O operations.

Table 12.2: *Contour Tree* Vs. *Criticality Tree*

Let us illustrate the difference between the *Contour Tree* and the *Criticality Tree* through an example. Let us consider a previous example that is used to define the *Contour Tree* in section 3.9.1. Figure 12.2 shows the level set of a function f as the parameter τ is decreased and figure 12.1 shows the corresponding *Contour Tree* and the *Criticality Tree*. Initially there are four objects created at maxima 7 through 10.

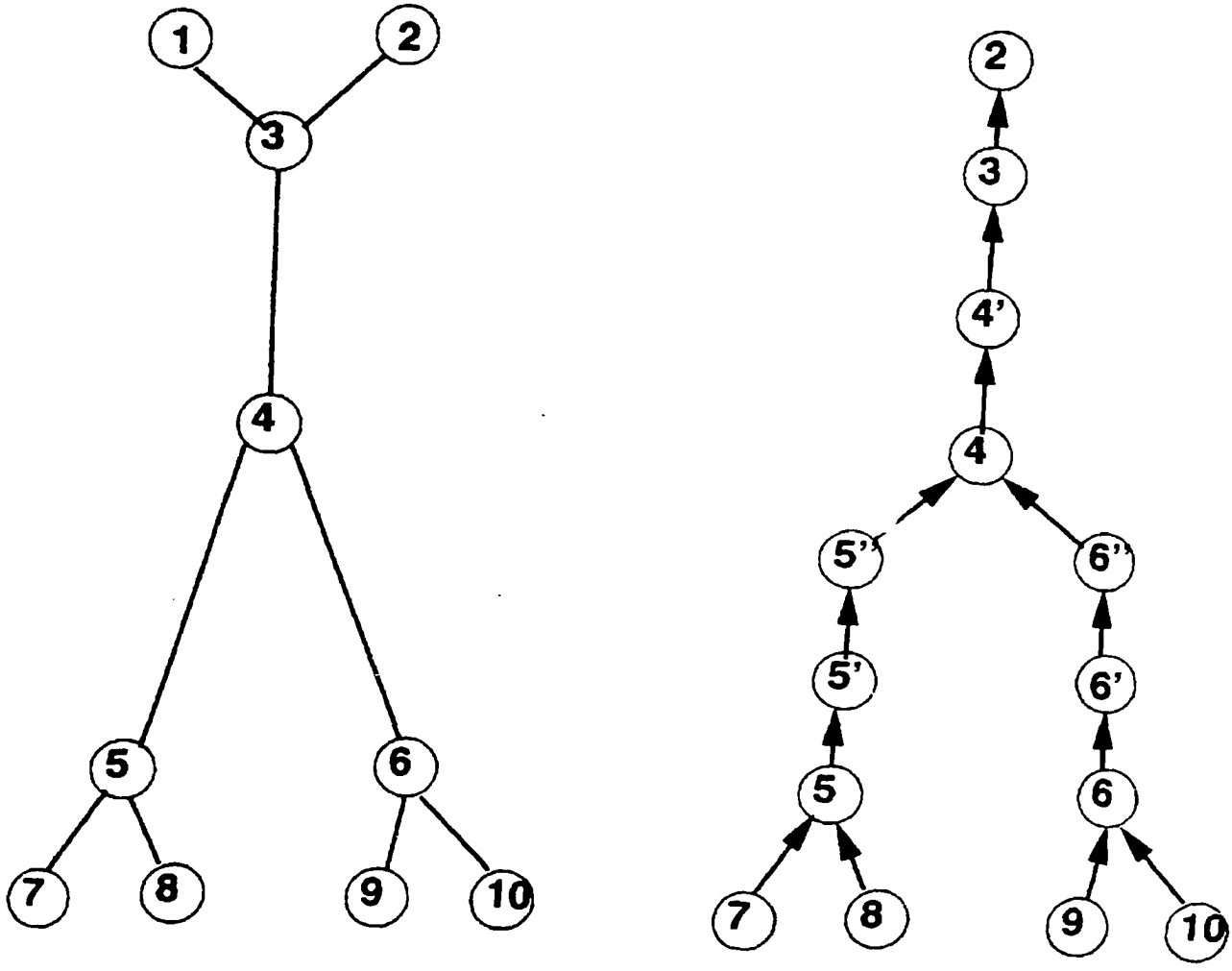

Figure 12.1: *Contour Tree* and *Criticality Tree*

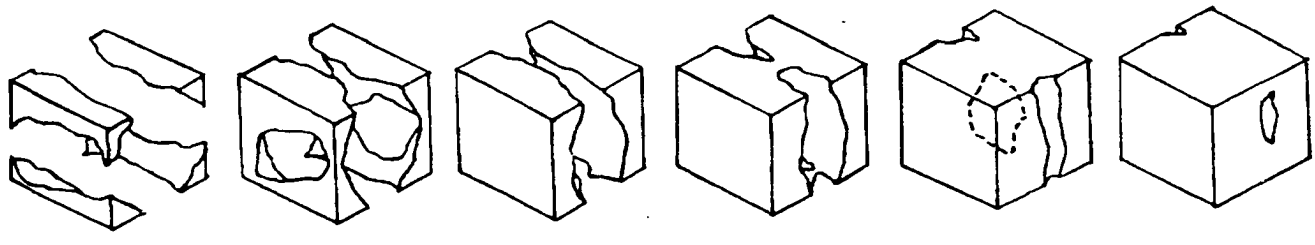

Figure 12.2: Level set of f as the parameter x decreases

These merge at saddles 5 and 6. Then the contours bounding each of the objects undergo two genus changes, changing from sphere to torus to sphere again at saddles 5' and 5" (and at saddles 6' and 6", respectively). These changes are not reflected in the *Contour Tree*. Then the two objects merge at saddle 4, the boundary of the this object becomes toroidal at saddle 4' and a inner boundary is created at saddle 3 by a split. At this point the object comes to enclose a hollow. Note that the *Contour Tree* has two edges coming out from node 3, which represents the outer and inner boundary manifolds of the object respectively. Finally the bubble is closed at minima 2 and this node represents the single object containing the entire data set.

Chapter 13

Conclusion

The *Topological Zone Segmentation* algorithm has multiple applications in out-of-core visualization of large data set and possible image registration. Some of these applications have already been implemented and presented here with the results obtained by conducting various experiments. The results shown here confirms our postulates for most of the cases, and demonstrates the benefits achieved, which includes significant improvement for active cell search. Some other applications are not yet implemented. These are mentioned here for future references and further improvements of the visualization toolkit that has been developed as the first step towards *Topological Zone Segmentation.*

13.1 Results & Applications of Zone Segmentation

Atlas of all Topologically Distinct Objects in the Data Set

The zone segmentation allows us to produce a visual atlas of all topologically distinct objects in the data set together with the range of isovalues that reveals each object, not just the individual contours . A common criticism of the isosurface extraction

methods is that they obscure the underlying structure of the scalar field. The zones reveal it. The range of isovalues associated with each zone makes it much easier to find the correct isovalue for the structure we are interested in.

Based on these ideas, we built a visualization toolkit that can be used to navigate into the data set for different isovalues. Once the data is preprocessed, all the zones and their corresponding components as well as the *Criticality Tree* is stored in the disk. The visualization toolkit reads the *Criticality Tree*, initialized the environment and inputs the user for the initial isovalue, and for the Δ value by which the isovalue will be changed at each stage, during navigation. The toolkit then travels in the tree starting at the root and searches for the zones whose boundaries contain the current isovalue and their active components. The zone component files contain a sorted list of the cells. All the cells, for which the current isovalue is in between the maximum and minimum vertex values, are considered to be the active cells. The *Spider Web* algorithm is used to construct isosurface within those cells. The search continues within a selected zone component file until a cell is found that has a maximum vertex value less than the current isovalue. The visualization tool also allows us to increment or decrement the isovalue by the Δ value entered by user. It reconstructs the isosurface every time the isovalue is changed. In other words, user can navigate into the data set for various isovalue, and find the correct isovalue for the object(s) of interest. Due to the zone preprocessing, the search space reduces to a much smaller sub space, which works as a great advantage for the toolkit. The toolkit loads and visits only a small percentage of the total cells and can generate isosurface in real time. It also allows the user to rotate the objects generated at any isovalue, incrementally, so that the objects can be examined from different points of view. It can also be used for animating the evolution of the objects as the isovalue is decreased from the global maxima to global minima. Figure 13.1 shows few snapshot

of such an animation done on a simulated data set that has a similar structure as the data set shown in figure 3.10.

Faster Isosurface Extraction

Our method constructs the isosurface without any local propagation. The active cells for any isosurface component are completely contained in a single zone component and the zone component files stores all necessary information regarding the cells within it. If the zone component is small enough (e.g. the size of the file is less than or equal to one disk block) we read this single file containing a super set of the active cells into memory reducing the seek time to $O(1)$. Otherwise, we further organize the cells within that zone component into I/O optimal interval tree and use this search structure to read the active cells from the file. This results in minimal I/O time over all known methods for isosurface extraction. Table 13.1 shows the results obtained for various data set.

Experiment 1 was conducted on simulated data that modeled similar object as figure 3.10. Experiment 2 was conducted on unfiltered medical data. Experiment 3 was conducted on the same data set as Experiment 2, but filtered. All of these data set were of size 20 X 20 x 20 (e.g. row X col X slide) and had 19 X 19 X 19 (e.g. 6859) cells (the data set is artificially bounded by an isoset of value 0). The fourth row 'Total zone created' gives an account of the number of critical points identified and the associated zones in the data set. The unfiltered medical data set resulted in too many critical points and corresponding zones, but for the other two data sets the number of critical points were reasonable. The next row shows the maximum, minimum and average number of points in the zones created. For both filtered and unfiltered medical data set, many of zones included fewer than 0.1 percent of the total

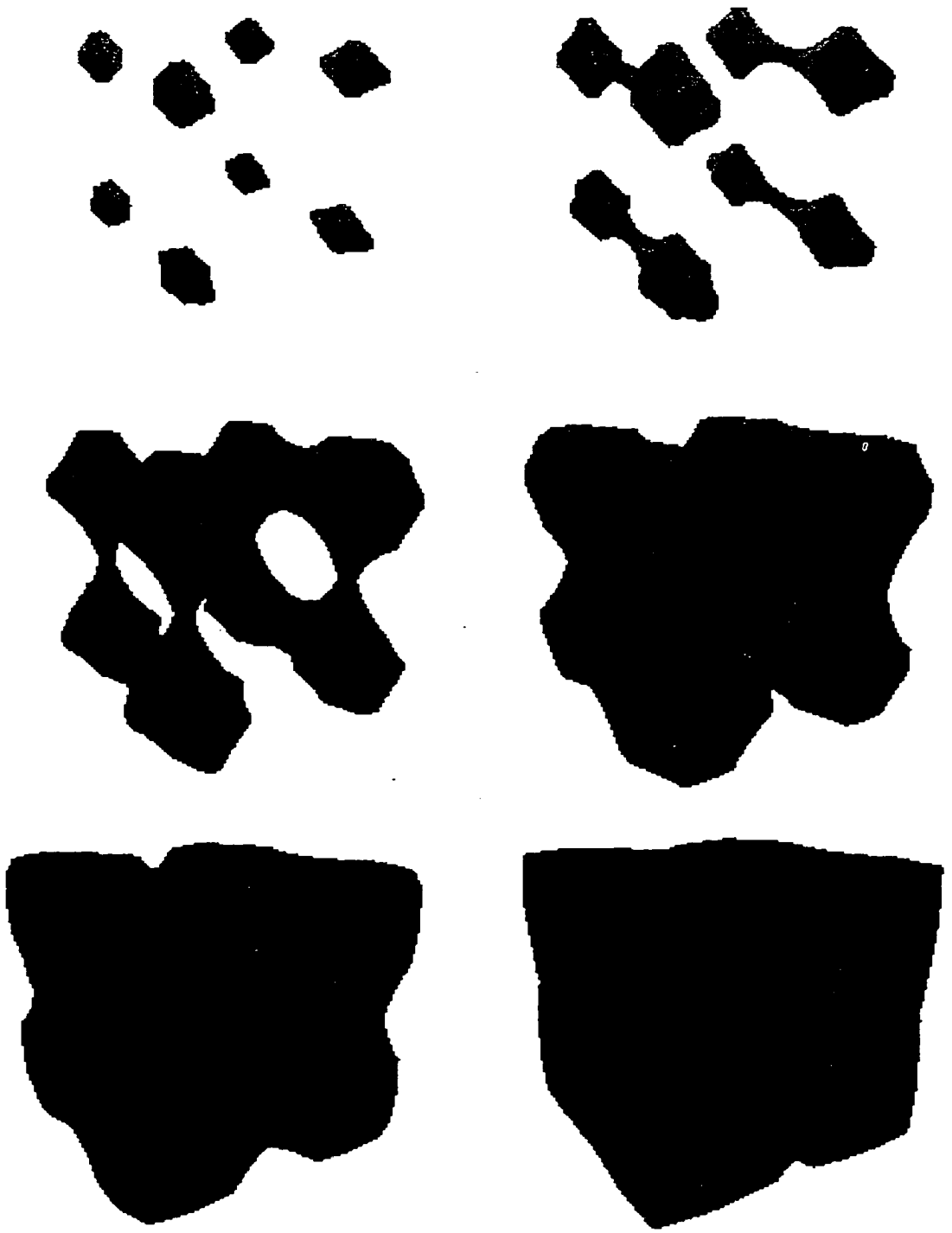

Figure 13.1: Snapshot of the level set of a function f as the parameter τ is decreased

	Exp. 1			Exp. 2			Exp. 3		
Data descrip.	Simulated			Unfiltered Medical			Filtered Medical		
Size (Row x Col x Slide)	20 x 20 x 20			20 x 20 x 20			20 x 20 x 20		
Total no. of cells	6859			6859			6859		
Total zone created	27			1345			167		
	Max.	Avg.	Min.	Max.	Avg.	Min	Max.	Avg.	Min.
Points in zone	2659	957	23	2187	1217	1	2605	1116	1
Active cells	3148	1978	64	3351	1143	8	2477	800	8
Cells visited	4563	2410	72	3356	1149	9	2733	843	9
Cells loaded in memory	4563	2923	624	4392	1459	26	2985	1063	46
Load utility (%)	100	64.7	10.3	100	71.6	30.8	100	58.7	17.0
Travel utility (%)	100	86.4	64.3	100	97.4	88.9	100	95.6	80.1
%tage of cells loaded	66.5	42.6	9.1	64	21.3	0.4	43.5	15.5	0.7
%tage of active cells	45.9	28.8	0.93	48.7	16.7	0.1	36.1	11.7	0.1

Table 13.1: Experiment Results

data points, as can be seen from the minimum column. This zones does not really reflect any significant change in the topology and can be put off the record without too much loss of information. In the next section some suggestions are made on how to classify the zones in terms of their essentiality or ability to reflect significant change in topology and how to keep only essential zones into account. The modification will further optimize the I/O overhead for the neighborhood search as discussed in the end of the next section. The next three rows, 'Active cells', 'Cells visited' and 'Cells loaded in memory' are the number of cells that actually intersected the isosurface, the number of cells that are examined in the active zone component files and the number of cells that are brought into memory (e.g. the cells in the active zone components) respectively. The *load utility* designates the percentage of active cells among the one that are brought into memory. In other words this percentage of cells were actually utilized to construct the isosurface. The *travel utility* means the percentage of active cells among the one that are examined As we can see, for all the three experiments, both the utility factor turned out to be quite high. This shows that the preprocessing of the data results in efficient load and search of active cells. Specially because the cells of an active zone component resides in the same file, the number of I/O operations will be further minimized because the cell information will be packed in one or more disk blocks. The last two rows shows the percentage of total cells that are loaded into memory and are active, respectively. The average percentage of in-core data is around 20% for medical data, which is quite reasonable, because in an average 15% of the total cells are indeed active cells that we actually need to produce the isosurface. As we can see, The zone segmentation preprocessing reduces the amount of in-core data significantly (e.g. very close to the amount that we actually need to construct the isosurface), produces correct isosurface and takes a big step forward to overcome the I/O bottle neck for visualizing very large data set.

13.2 Future Applications of Zone Segmentation

Noise and Inessential Feature Reduction

One of the limitations of our algorithm is that it is sensitive to noise. The inclusion of noise in the data may result in additional critical points with very high value and associated zones that are very small in size. Also there can be some critical points that has regular data value (e.g. not a noise) and very small zones associated with it. We can simplify the data set by eliminating smaller zones by merging them with larger ones, if we feel that the zone and associated critical point represents a noise or an inessential feature.

As the experimental results show, the unfiltered medical data set results in too many critical points. Handling too many zones for large data sets may prove to be impractical. However, many of these critical points are indeed insignificant, since for these, the parent critical point is encountered soon after the original critical point resulting very little or no flood of the corresponding zone in the neighborhood. The topology does not really change too significantly, for any isovalue τ within the range of the the two critical values. The boundaries of the parent zone just wrap that of the child zone tightly. These critical points can be eliminated as insignificant without loosing any significant change in the topology. The corresponding zone can be merged with the zone of its parent, resulting fewer critical points and associated zone files. Now the question arises, how do we determine which are the insignificant critical points. Here are some suggestions on how to determine significant critical points and lower the number of zone to a desired level.

Suggestion for Reducing Inessential Critical Points

First, let us examine, what is the end result of having too many zones and too little zones.

In one extreme, if there are too few zones, it will result in larger zones and zone component files which will probably not fit in one disk block. That means we will have to perform more than one I/O operations for transporting a single zone component file. In the other extreme, as the number of the zone increases, the size of the zone reduces. The smaller the size of the zones, the higher the possibility that single zone component will fit in one disk block resulting a single I/O operation to bring this zone component into main memory. But too many zones will also result in transporting too many zone files from the secondary memory detoriating the I/O efficiency. For the fastest results, the best choice is to create optimum size zone and optimum number of zones. Since real life medical images results in too many critical points, the first step would be to reduce these criticality to only significant ones. Some heuristics can be applied to identify the insignificant critical points as follows.

- The zone file are not created until the zone is terminated. Till then keep the points with final label of that zone stored in the memory at part of the structure. If that results in burdening the memory too much, then we can store them in a temporary file with a capabilities to move the data, and removing the file altogether, when needed. When a zone terminates, we first decide whether it is significant or not. If so, then we create the original zone file and place the points in it. Otherwise we merge the zone with its parent's. For each case, we remove the temporary file (e.g. if it exists) at the end of the operation.

- We keep a count of how many points get finalized label of a critical point before its parent critical point is encountered. This is exactly the area that is flooded

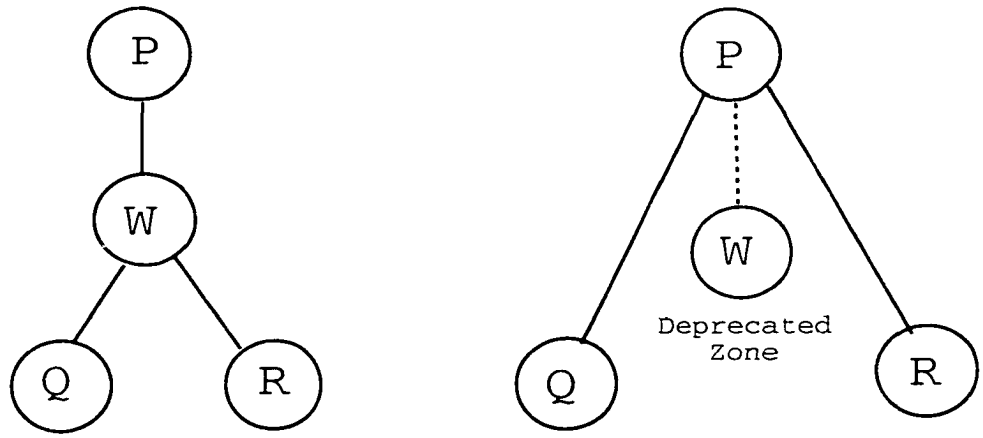

Figure 13.2: Rearrangement of the *Criticality Tree* when a zone is tagged as deprecated

by a zone before the parent critical point is reached.

- We need to set a threshold that will be minimum number of points that must have a finalized label of the zone for it to be a significant one. For example one may decide that at least 0.1% of the total data must be flooded for a zone to be accounted. Any zone that does not satisfy the criteria will be considered inessential.

- When a zone terminates we check whether or not it is essential. If not, then tree node corresponding this critical point will be tagged as deprecated in the *Criticality Tree*. We can not completely eliminate this tree node because we need to relabel the points with temporary label of this zone. All direct children of the tree node will bypass this deprecated node and direct point to its parent. The points with the finalized label of this zone will be moved to the zone of its parent. The maximum value and the number of points with finalized label of the parent zone is reseted accordingly. Figure 13.2 shows the rearrangement of the *Criticality Tree* if a zone is tagged as deprecated.

- While relabeling a point with temporary label corresponding to a deprecated zone, parent pointers are traced until a node is found with a zone that is not yet deprecated. The relabeling continues as before, by tracing the parent pointers starting from this node.

- At the end of phase 2, only the nodes corresponding essential zones in *Criticality Tree* are stored, ignoring the deprecated ones.

- The threshold value depends on the desired size of the zone and number of zone. At first we need to figure out a relationship between the number of zone and the size of the zone using different threshold. The best chosen threshold will the one, the results in optimum number of zone of optimum size of the zone.

- We can store the zone files in a database for faster access. A B Tree can be created for efficient search of the zone files.

The reduction of the zones only to essential ones should not hamper the *travel utility* to any significant extent. As we can see the filtered medical data with much fewer zones, resulted in lower but very close travel utility factor (e.g. only 2% lower *travel utility* with 87% reduction of critical points). The *load utility* suffered little bit more for filtered medical data. The primary reason for that is the data set was smoothened by filtering, resulting some big chunks of data set without any change is topology, and some chunk with still too many criticality. Zone reduction using above method restricts the minimum volume swept by a zone to a preset value without smoothing or changing the actual values. So, there will not any large chunk of smoothened data without much variation that results in big zones. Also no parent zone boundary will tightly wrap any child zone boundary reflecting no significant change in topology, increasing the range of isovalue for each zone. This will result in

very efficient isosurface construction for any small increment of isovalue, where some of all the active zone components remain to be same, and most of the cells already loaded in the memory can be used once again. No further I/O operation is needed to perform for the overlapping active zone components, further improving the I/O efficiency.

A Priori Topology Preserving Simplification and Dynamic Rendering

The topology of the boundary manifold of a zone component remains invariant for any isovalue that lies within the range of that component, and the topology changes occur only at critical points. This means as long as we retain the critical points or critical isosets, we can coalesce cells within a zone component up to the limits given by the zone structure without changing the topology of the resulting isosurface components. In this manner we can reduce the tiling complexity of the resulting isosurfaces, either prior to actually extracting them or during actual extraction based on users' demand. This allows us to have a control on the rendering complexity and manage the level of detail either statically or dynamically, preserving the topology of the contours. Thus, we can have finely rendered surfaces in the areas of interest, and coarsely rendered surfaces in other areas.

Consider the case of 3D regularly gridded data. In order to obtain coarsely rendered surfaces we can coalesce 8 adjacent small cube into a large cube and calculate the hit points and extract the surface patches based on the values of the vertices of this large cube. While refining the rendering, we can divide this large cube into 8 small cubes, by adding back the interior data points, which are anyway part of the same zone component, and extract the isosurface individually for each of these 8 small

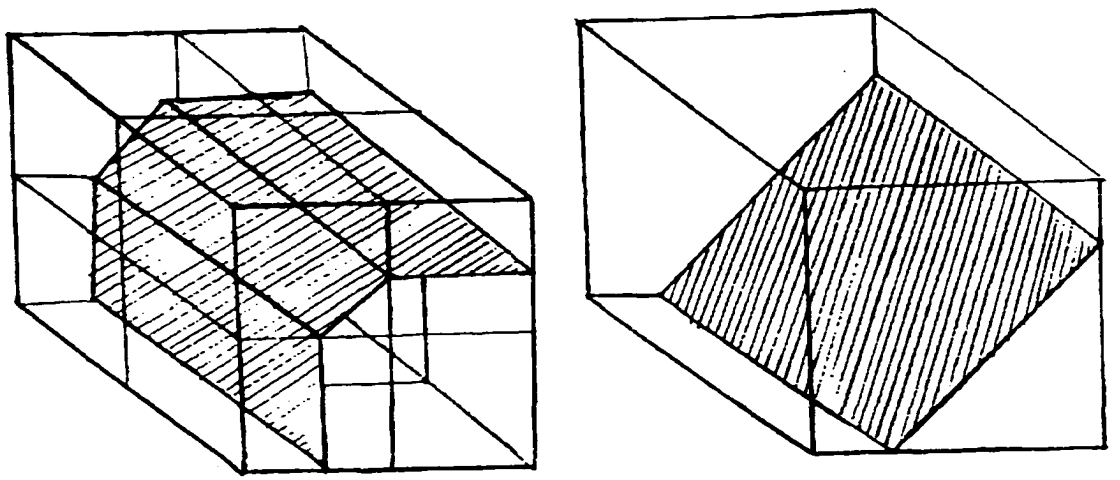

Figure 13.3: Finely and coarsely rendered surfaces without changing the topology

cubes. Figure 13.3 shows an example of different rendering complexity.

Image Registration

It will be very interesting to explore the potential use of *Topological Zones* in applications such as multi-modal image registration. Note that the criticality tree is invariant with respect to rotation, scaling, translation and uniform value translation of the data set. Two images with same contents rotated at different angle, scaled differently or translated will result in similar or partially similar *Criticality Tree.* The *Criticality Trees* can be compared to verify whether or not the images has the same contents. Once the corresponding critical points are identified, two graphs G and G' can be drawn that connect the corresponding critical points. The image registration problem can then be reduced to registration of these graphs only.

The zone segmentation provides a very natural and efficient way of organizing the data set since the organization is based on the contents of the data set itself. The whole procedure of the preprocessing is automatic and does not need any user intervention. The zone segmentation preprocessing results in optimum I/O operation

which provides a great starting point for overcoming one of the still unresolved challenges of visualization and animation of very large data set. Also because the zone segmentation traces the evolution of objects, this can be used to model the volumes and objects. We have demonstrated the feasibility, correctness and utility of zone segmentation and organization in visualizing volume data sets. We believe that we have provided a basis for exploiting this powerful technique that will ultimately result in the ability to visually navigate extremely large volume data sets in real time without any special purpose hardware.

Appendix A

Proof of theorem 1

Lemma 1 *Let e be a cell edge. Then e has a single intersection point termed as hit with* $B(f^{-1}(\geq \tau))$ *if and only if the incident end points are High and Low, respectively.*

Proof: This follows immediately from property 2.

Lemma 2 *Each 2 dimensional hypercube face F (e.g. square) has 0, 2 or 4 hit points. Each 2 dimensional simplex face F (e.g. triangle) has 0 or 2 hits.*

Proof: This follows from a simple parity argument or by examining the 4 possible arrangements of Highs and Lows on the 4 vertices of a cube face. 1, 2 or 3 edge adjacent Highs give 2 hits and a pair of diagonally opposite Highs give 4 hits (See figure 7.5).

Definition 8 *Two hits are adjacent in a face F if they are connected by a curve segment of* $B(f^{-1}(\geq \tau)) \cap F$. *Connectivity of the sets of hits is defined by the transitive closure of adjacency. Connected components are defined accordingly.*

Lemma 3 *Each hit point h of face F is connected by a curve segment of* $B(f^{-1}(\geq \tau)) \cap F$ *to a unique adjacent hit point* h' *in* F.

Proof: The results follows from the previous lemma, the disambiguation rule and property 2 of admissible interpolants (See figure 7.5).

Definition 9 *We can uniquely identify a hit point by the cell edge on which it occurs by interpolating the values of the two endpoints of the cell edge.*

Lemma 4 *$B(f^{-1}(\geq \tau))$ and $B(f'^{-1}(\geq \tau))$ have the same hit sets and the hit connectivity, if f and f' are both admissible.*

Proof: Changing the interpolation function does not change whether a vertex is High or Low. For regularly gridded data, f and f' both has the same disambiguation values resulting same hit connectivity.

Definition 10 *Two surface patches are adjacent if they intersect.*

Lemma 5 *Two $d-1$ dimensional surface patches of M, $\sigma \in C$ and $\sigma' \in C'$ are adjacent, if and only if their intersection is a set of $d-2$ dimensional surface patches of M in the $d-1$ dimensional boundary cell shared by C and C'. Further, if σ and σ' are adjacent then they share at least one hit point in each component of their intersection.*

Proof: The fact that they must intersect in some lower dimensional patch follows from the fact that the $d-1$ dimensional boundary cell is completely shared by C and C' and property 2 of the admissible interpolants. The fact that they share a hit point in each component of their intersection is due to the fact that their intersection must occur on at least one cell edge, and thus they share the hit along this cell edge.

Lemma 6 *Each connected component of hits in cell C is a member of a unique surface patch $\sigma \in B(f^{-1}(\geq \tau)) \cap C$. Conversely the hits contained in each surface patch $\sigma \in B(f^{-1}(\geq \tau)) \cap C$ form a unique connected component of hits in C.*

Proof: By definition of hit adjacency and connectivity, each pair of hit in a connected component of the hits within C is connected by a path within $B(f^{-1}(\geq \tau)) \cap C$. By the path connectivity of the surface patches, this path must be a part of the same patch. Thus the connected set of hits all lie in a single patch σ of C.

For the other direction, we must show that the hit points of σ form a connected set within C. The proof is by induction on the dimension d. The result is clearly true for $d = 1$ and $d = 2$. For inductive hypothesis, we assume that for all $j \leq d - 1$, the hits of each patch of a j dimensional cell C is a connected set within C. Let h and h' be any two hit points of σ. Since σ is homeomorphic to the unit ball, its boundary β is connected and contained in the $d-1$ dimensional boundary cells of C. Then there is a path $\pi \in \beta$ from h_1 to h_2 such that, this path π passes through a sequence, $\sigma_1, \sigma_2, \ldots, \sigma_m$, that are adjacent $d-2$ dimensional patches in the $d-1$ dimensional boundary cells of C. Let h_i be any hit point in σ_i and h_{i+1} be a hit point in σ_{i+1}, where $1 \leq i < m$. We claim that h_i and h_{i+1} are part of the same component of hits of C which is justified as follows. Since σ_i and σ_{i+1} are adjacent, by lemma 5 they share at least one hit point. By induction hypothesis, the hits of σ_i and σ_{i+1} are each connected individually within their respective boundary cells. Thus the union of the hits of σ_i and σ_{i+1} must be part of the same connected set of hits in C. This also implies that h_i and h_{i+1} are part of the same connected component of hits in C and established the claim stated before. That means any two hit points in σ are part of the same connected set. In other words, the hit points contained any surface patch σ form a unique connected component of hits within the cell C.

Lemma 7 *Each connected component of hits is contained in a unique component manifold M of $B(f^{-1}(\geq \tau))$ and conversely the hits contained in each such manifold M form a unique connected component of hits.*

Proof: A connected component of hits is contained in a single manifold M by the definition of hit adjacency and the hit connectivity. This manifold is unique, since by property 1 of admissible interpolant, the manifolds are pair-wise disjoint.

For the other direction, suppose that M contains two hit points h_1 and h_2. There is clearly a path π in M from h_1 and h_2. This path passes through some sequence of patches of M, $\sigma_1, \sigma_2, \ldots, \sigma_m$ of cells $C_1, C_2, \ldots, C_m$, where $h_1 \in \sigma_1$ and $h_2 \in \sigma_m$. For each $1 \leq i < m$, σ_i contains a unique connected component of hits within C_i by lemma 6. For each such i , σ_i and σ_{i+1} intersect in a shared boundary cell of C_i and C_{i+1} and share at least one hit point by lemma 5. Thus the hit points of σ_i and σ_{i+1} are part of the same connected component of hits, which means h_1 and h_2 are part of the same connected set of hits as well.

Lemma 8 *if f and f' are both admissible, $B(f^{-1}(\geq \tau))$ and $B(f'^{-1}(\geq \tau))$ consists of the same number of components.*

Proof: Let H be a connected component of hits. Let M and M' be the component of $B(f^{-1}(\geq \tau))$ and $B(f'^{-1}(\geq \tau))$ containing H.

Lemma 9 *Let M and M' be consists of the patches $\Sigma = \sigma_1, \sigma_2, \ldots, \sigma_k$ and $\Sigma' = \sigma'_1, \sigma'_2, \ldots, \sigma'_k$. There exists a one-to-one mapping ψ of Σ onto Σ' so that for all $1 \leq i < j \leq k$, σ_i is adjacent to σ_j, if and only if $\psi(\sigma_i)$ is also adjacent to $\psi(\sigma_j)$. In addition their respective intersection have same number of components.*

Proof:

The mapping ψ associates two patches if they share the same hit set. The mapping is one-to-one and onto by lemma 6. If σ_i is adjacent to σ_j, then from the above lemmas, they share at least one hit in common. Suppose they share a hit on cell edge e. Then $\psi(\sigma_i)$ and $\psi(\sigma_j)$ also share a common hit on the cell edge e and are adjacent.

In other words if we represent the patches and their adjacencies by graphs in the natural way, then the graphs representing M and M' are isomorphic. Using the above fact we can prove,

Lemma 10 *M and M' are homeomorphic.*

Proof: We can construct a global homeomorphism from M and M', piece-wise via their patches. From property 2 of admissible interpolants, both patches σ_i and $\psi(\sigma_i)$ are bicontinuously mappable to a unit disk. From lemma 9, we know that σ_i is adjacent to σ_j if and only if $\psi(\sigma_i)$ is also adjacent to $\psi(\sigma_j)$. Also by lemma 5 and property 2 we know that the intersections of the corresponding pair of the adjacent patches are bicontinuously mappable. We can thus bicontinuously map σ_i to $\psi(\sigma_i)$ and extend the mapping for each adjacent patches recursively. Thus we can define the homeomorphism iteratively on each σ_i and extend it to define global homeomorphism from M to M', proving the lemma.

Theorem 1 now follows immediately from lemma 7 and 10.

Appendix B

Proof of theorem 2

Proof:

Observe that objects and sets of Highs are monotonic in the sense that, as the threshold is decreased, High data readings remain High, and points that were part of an object, remain as part of the same object throughout. The Lows becomes High after a certain time and new points are added to the objects.

Topology changes to $B(f^{-1}(\geq \tau))$ can thus occur in several ways and of course, several changes can occur at the same time. First of all a new object can be created as the threshold τ is decreased through a value c, and hence a new contour is created. It is straightforward to see from that property 1 of the admissible functions that this can only happen if c is a local maximum critical value. In this case there is no object in some region of space for all thresholds $\tau > c$, but a new object exists in the same region for $\tau < c$. By admissibility and data uniqueness, this new object must contain a single data point p and must comprised of a single connected component of Highs. Thus p is the local maximum critical point. Similarly a contour can vanish at c, and by monotonicity this can happen only if c is a local minimum critical value, where the associated point p comprises a single connected component of Lows for $\tau = c + \epsilon$.

Two or more separate contours can merge at value c. From property 1 and monotonicity, this means that there exists two separate components of Highs at threshold $\tau = c + \epsilon$, that are merged into one component of Highs at $\tau = c - \epsilon$. This can only happen if c is a saddle critical value of type 3 and a point p becomes High at c and unites the two components, or c is a critical saddle point of type 4 and the change of the connectivity of Highs in a 4-Hit face unites the two locally separate component of Highs.

One or several contours can split at c. From admissibility property 1 and monotonicity, this will happen only when a component of Lows is separated by a change in connectivity at c. This can happen only if c is a saddle critical value of type 3 or 4.

In 3 dimensions a contour can change in genus at c, when a class of incompressible loops is created or destroyed. Monotonicity and homotopy imply that an increase in the genus of a contour can only occur in the following way. As the threshold is decreased through a value c, two separate portions of the same object that existed in a neighborhood of a point p at threshold $c + \epsilon$, are merged at threshold $c - \epsilon$, so that the boundary of the object can be surrounded by an incompressible loop in this region. Similarly, a decrease in the genus can occur only when there is a region of space containing the connected set of points of the complementary object (e.g. corresponding component of Lows) at threshold $c + \epsilon$, so that its boundary can be surrounded by an incompressible loop in this region. At threshold $c - \epsilon$, the complementary object splits and the boundaries are separated destroying the loop. By admissibility, this means two locally separate components of Lows exist at $c - \epsilon$, so that N_L must consist of at least two components and p is a saddle critical point of type 3 or 4.

For dimensions greater than 3, the same argument holds, with the difference being

that a class of incompressible $n-2$ dimensional surfaces (e.g. equivalent to a sphere) are created or destroyed resulting in a change of the corresponding Betti number of the contour.

We now demonstrate that topological changes occur only at critical values. By admissibility, a new object is created at a local maximum and a contour vanishes at local minimum. Multiple changes can occur at saddles, including any combination of contour merges, splits, and handles are created and destroyed. We argue that at least one of these changes occur at a saddle value. First of all, the saddle point p with value c, is part of the same global component of Lows at threshold $c+\epsilon$. At threshold $c-\epsilon$, when p becomes high, the component gets separated from p. By property 1, a boundary contour has split at c. Similarly if two globally separate component of Highs are merged at c, at least one pair of the corresponding boundary contours has merged.

For changes to topological type (e.g. genus or higher order Betti number of a contour), note that, the cells N sharing saddle point p with value c. are homeomorphic to unit ball B, with p at the center and the other data points on the spherical boundary of the ball. Suppose that there are at least two components of Highs of N_H namely, N_1 and N_2 on the boundary of B at thresholds above c (e.g. the argument for N_L is symmetric). By assumption both N_1 and N_2 are part of the same object O. In 3 dimensions one can find a loop L on the boundary of B (e.g. on the cell faces) that separates N_1 and N_2 so that L is exterior to O. The definitions of connectivity assure that we can construct L as a loop consisting of cell edges and cell face diagonals connecting adjacent Lows. O will intersect the boundary of B in at least two components O_1 and O_2 containing N_1 and N_2 respectively. By connectivity of O and its boundary contours, one can find a path π in the boundary of O from O_1 to O_2 so that π never intersects the interior of the cells B. At $c-\epsilon$, π can be

extended to a loop L' by a path from O_1 to O_2 through the interior of B. Clearly L' is incompressible as it will be interlocked with the above loop L. For $n > 3$ dimensions from connectivity of data readings and cell structure, one can similarly find a closed loop L on the boundary of B, consisting of $n-2$ dimensional surface patches. For example, in 4D the surface patches will consist of 2 dimensional cell faces. Again one can find an $n-2$ dimensional surface patch loop L' on a contour of O, extending through and out of B, so that L and L' are nonintersecting but interlocked. Thus there is a resulting change to the corresponding Betti number of the contour at the critical value.

Bibliography

[1] William E. Lorenson and H. E. Cline, **Marching Cubes: A High Resolution 3D Surface Construction Algorithm**, *SIGGRAPH Proceedings of Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 163-169.

[2] Gregory M. Nielson and Bernd Hamann, **The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes**, *Proceedings of Visualization '91*, Vol. 2, October 1991, pp. 83-91.

[3] J. Wilhelms and A. V. Gelder, **Octrees for Faster Isosurface Generation**, *ACM Transactions on Graphics*, Vol. 11, No. 3, July 1992, pp 201-227.

[4] Arie Kaufman, **Volume Visualization**, *ACM Computing Surveys*, Vol. 23, No. 1, March 1996, pp. 165-167.

[5] Christial Barillot, **Surface and Volume Rendering Techniques to Display 3-D Data**, *IEEE Engineering of Medicine and Biology*, March 1993, pp. 111-118.

[6] K.Y. Whang, J.W. Song, J.W. Chang, J.Y. Kim, W.S. Cho, C.M. Park and I.Y.Song, **Octree-R: An Adaptive Octree for Efficient Ray Tracing**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 4, July 1995, pp. 343-349.

[7] Andre Gueziec and Robert Hummel, **Exploiting Triangulated Surface Extraction using Tetrahedral Decomposition**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 4, December 1995, pp. 328-342.

[8] T. Itoh and K. Koyamada, **Automatic Isosurface Propagation using an Extrema Graph and Sorted Boundary Cell Lists**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No 4, December 1995, pp. 319-327.

[9] M. Giles and R. Haimes, **Advanced Interactive Visualization for CFD**, *Computing Systems in Engineering*, Vol. 1, No. 1, 1990, pp. 51-62.

[10] R.S. Gallagher, **Span Filter: An Efficient Scheme for Volume Visualization of Large Finite Element Models** , *Proceedings of Visualization '91*, October 1991. pp. 68-75.

[11] H. Shen and C.R. Johnson, **Sweeping Simplices: A Fast Isosurface Extraction Algorithm for Unstructured Grid**, *Proceedings on IEEE Visualization '95*, October 1995, pp. 143-150.

[12] Y.Livnat, H.W. Shen and C.R. Johnson, **A Near Optimal Isosurface Extraction Algorithm using the Span Space**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 1, March 1996, pp. 73-84.

[13] H. Edelsbrunner, **Dynamic Data Structure for Orthogonal Intersection Queries**, *Technical Report F59* Institute Informations-verarb, Tech University Graz, Graz, Austria, 1980.

[14] P. Cignoni, P. Marino, C. Montani, E. Puppo and R. Scopigno, **Speeding up Isosurface Extraction using Interval tree**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, April-June 1997.

[15] Y. Chiang and C. Silva, **I/O Optimal IsoSurface Extraction**, *Proceedings of IEEE Visualization 1997*, pp. 293-300.

[16] C. Silva and A. E. Kaufman, **PVR: High Performance Volume Rendering**, *IEEE Computational Science and Engineering*, Winter 1996, pp. 18-28.

[17] Nelson Max, **Optical Models for Direct Volume Rendering**, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 2, June 1995, pp. 99-108.

[18] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski and S. Wang, **VolVis: A Diversified Volume Visualization System**, *Proceedings of IEEE Visualization '94*. IEEE Computer Society Press, October 1994, pp. 31-38.

[19] C. Silva, J. Mitchell and A. Kaufman, **Near Optimal Rendering of Irregular Grids**, *IEEE/ACM Volume Visualization '96 Symposium Proceedings*, ACM/SIGGRAPH Press, October 1996.

[20] Gregory M. Nielson, **Challenges in Visualization Research**, *IEEE Transactions on Visualization and Computer Graphics*, Theme Issue Introduction, Vol. 2, No. 2, June 1996, pp. 97-99.

[21] G.T. Herman, **Oriented Surfaces in Digital Spaces**, *CVGIP: Graphical Models and Image Processing*, Vol. 55. No. 5, September 1993, pp. 381-396.

[22] J. Cox, D. Karron and B. Mishra, **The Spider Web Algorithm for Surface Construction from Medical Volume data: Geometric Properties of its Surface**, *Innovation et Technologie en Biologie et Medecine*, Vol. 14, No. 6, November 1993, pp. 635-656.

[23] J. Cox and D. Karron, **New Developments from Spider Web Algorithm toward Digital Morse Theory** *Proceedings of Visualization in Biomedical Computing*, October 1994, Vol. 2359, SPIE, pp. 643-657.

[24] M. Goresky and R. MacPherson, **Stratified Morse Theory** Vol 14 of Series 3, *A Series of Modern Surveys in Mathematics*, Springer-Verlag, 1998.

[25] J. Cox, D. Karron and N. Ferdous **Topological Zone Segmentation of Volume Data Set**, submitted to *Graphical Models and Image Processing*, 2001.

[26] D. Karron and J. Cox, **Digital Morse Theory for Anatomic Modeling**, *Proceedings of IEEE BMES-EMBS first joint conference*, October 1999.

[27] D. Karron, J. Cox and B. Mishra **System and Method for Surface Rendering of internal structures within the Interior of a Solid Object**, United States Patent, Patent No 5,898,793, Apr 1999.

[28] Arie E. Kaufman, **Introduction to Volume Visualization**, *IEEE Computer Society Press Tutorial*, Los Alamitos, California, August 1990, Chapter 1, pp. 1-11.

[29] P. Shirley and H. Neeman, **Volume Visualization at the Center for Supercomputing Research and Development**, *Chapel Hill Visualization Workshop Proceedings*, May 1989, pp. 17-20.

[30] H.E. Cline, W.E. Lorenson, S. Ludke and C.R. Crawford and B.C. Teeter, **Two Algorithms for three dimensional Reconstruction of Tomograms**, *Medical Physics*, Vol. 15, No. 3, May-June 1988, pp. 320-327.

[31] R.A. Drebin, L. Carpenter and P. Hanrahan, **Volume Rendering**, *Computer Graphics*, Vol. 22, No. 4, August 1988, pp. 65-74.

[32] A. Kaufman, R. Bakalash, D. Cohen and R. Yogel, **Architectures for Volume Rendering**, *IEEE Computer Society Press Tutorial*, Los Alamitos, California, pp. 311-320.

[33] Y. Zhou, B. Chen and Z. Tang, **An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes**, *Computers and Graphics*, Vol. 19, No. 3, 1995, pp. 355-364.

[34] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D.R. Schikore, **Contour trees and small seed sets for isosurface traversal**, *13th ACM Symposium on Computational Geometry (ACM)*, 1997, pp. 212-220.

[35] C. Hamish, J. Snoeyink, and U. Axen, **Computing Contour Tree in all Dimension**, *Symposium on Discrete Algorithms*, 2000.

[36] C. L. Bajaj, V. Pascucci, and D.R. Schikore, **Seed Sets and Search Structure for Optimal Isocontour Extraction**, TICAM report No. 99-35, 1999.