3

Accelerated IsoContouring of Scalar Fields

Chandrajit L. Bajaj and Valerio Pascucci *University of Texas, Austin*

Daniel R. Schikore *Center for Applied Scientific Computing, LLNL*

ABSTRACT

With the increasing size of typical 2D and 3D data, efficient computational methods are becoming increasingly crucial for achieving desired levels of interactivity. Computation of isocontours from scalar data is a particularly critical task for comprehensive visualization of volume data. In the case that the volume is discretized by a mesh of volumetric *cells*, the extraction of an isocontour consists of two primary phases: *triangulation* of a particular cell and the *search* for all intersected cells. In this chapter we will review and contrast the primary algorithmic approaches which have been suggested in the literature.

3.1 Introduction

Isocontouring is a widely used approach to the visualization of scalar data and an integral component of almost every visualization environment. Computation of isocontours has applications in visualization ranging from extraction of surfaces from medical volume data [Lor95] to computation of stream surfaces for flow visualization [van93]. Inherent in the selection of an isocontour, defined as $C(w) : \{\mathbf{x}|\mathcal{F}(\mathbf{x}) - w = 0\}$, is that only a selected subset of the data is represented in the result. In many applications, the ability to interactively modify the isovalue w while viewing the computed

DATA VISUALIZATION TECHNIQUES, Edited by C. Bajaj

result is of great value in exploring the global scalar field structure. In fact, it has been observed in user studies that the majority of the time spent interacting with a visualization is in modifying the visualization parameters, *not* in changing the viewing parameters [Hai91]. Hence there has been great interest in improving the computational efficiency of contouring algorithms.

We will focus on isocontouring of scalar fields which are defined over a piecewise cell decomposition rather than the more general case of implicit functions, although many issues cross over between the two input formats. In this situation, isocontouring algorithms can be characterized by three principal components:

- Cell Triangulation – Method of computation for determining the component of a contour which intersects a single cell.
- Cell Search – Method for finding all cells which contain components of the contour
- Cell Traversal – Order of cell visitation may be integrated with (or decided by) the cell search technique, however it nevertheless affects the performance of the isocontour extraction algorithm

In the remainder of this chapter, we will discuss several isocontouring algorithms which address one or more of these components.

3.2 Cell Triangulation

Cell triangulation concerns the approximation of the component of a contour which is interior to a given cell. Triangulation has two distinct components, *interpolation* to determine a set of points and normals, and *connectivity* to determine the local topology of the contour.

3.2.1 Interpolation

Cell-based contouring algorithms generally begin with a binary classification of each vertex of a given cell as *positive* (if greater than the isovalue) or *negative* (if less than or equal to the isovalue), which we will refer to as *black* and *white* vertices, respectively. For simplicity, most isocontouring algorithms adopt a simple interpolation approach along the edges of cells. Each edge which has one black vertex and one white vertex has exactly one intersection with the isocontour under the linear interpolation model, and this intersection point is easily computed as a linear combination of the endpoints of the edge. Any edge which has two vertices of the same color is appropriately disregarded, as the linear interpolation cannot intersect the isosurface if both endpoints are above or below the isovalue.

While linear interpolation along edges of cells is a widely used approach, interpolation is often the most compute-intensive portion of isocontour extraction. As data sizes increase and relative sizes of cells decrease, the effect of interpolating along cell edges is less noticeable. Other strategies have been developed to reduce this computational portion of isocontour approximation, such as selecting midpoints along intersected edges [MSS94]. Midpoint selection in grids of regular topology and uniform spacing has the added advantage that triangles extracted for the surface have relatively few facet orientations, resulting in large planar regions which are more easily coalesced to

produce a simplified model of the isosurface for rapid rendering and compact representation.

A primary reason for applying linear interpolation in isocontouring is the fact that gradient information is often not present in the original data. If this is the case, gradient information may be estimated for the purpose of smooth surface shading by approximating gradients at the vertices and using linear interpolation of the gradient vector components within each cell. However, the ability of higher degree interpolant and associated gradient estimators to accurately represent the underlying data are motivating work in this direction [MMMY96, BLM97].

3.2.2 Connectivity

The common approach of linear interpolation along cell edges is sufficient to obtain a sampling of points which lie on the surface, but the problem of connecting the points to form a surface still remains. A binary classification of the 8 vertices of a regular cell leads to a total of 2^8 or 256 possible configurations. Taking rotational symmetry into account, this can been reduced to 22 distinct cases [LVG80, Sri81]. Marching Cubes [LC87] further reduces the number of base cases by assigning complementary triangulation for complementary vertex configurations (*black* to *white*), resulting in 15 distinct colorings, for which connectivity information can be assigned, as shown in Figure 3.2. The full table of the 256 possible vertex configurations can easily be generated from this table of 15 cases.

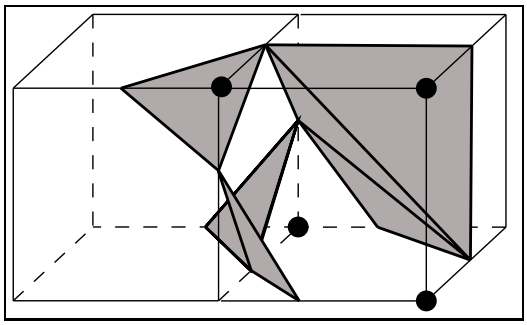

Figure 3.1 Topological inconsistency associated with the original marching cubes

The use of complementary triangulations reduces the number of base cases, but also introduces a well-know topological inconsistency on certain configurations of shared faces between cubes [Dur88], one case of which is illustrated in Figure 3.1. A number of techniques have been proposed which offer solutions to this inconsistency, which we group into two classes. The first class attempts only to provide *consistency* along cell faces, while then second class provides *correctness* with respect to a chosen model.

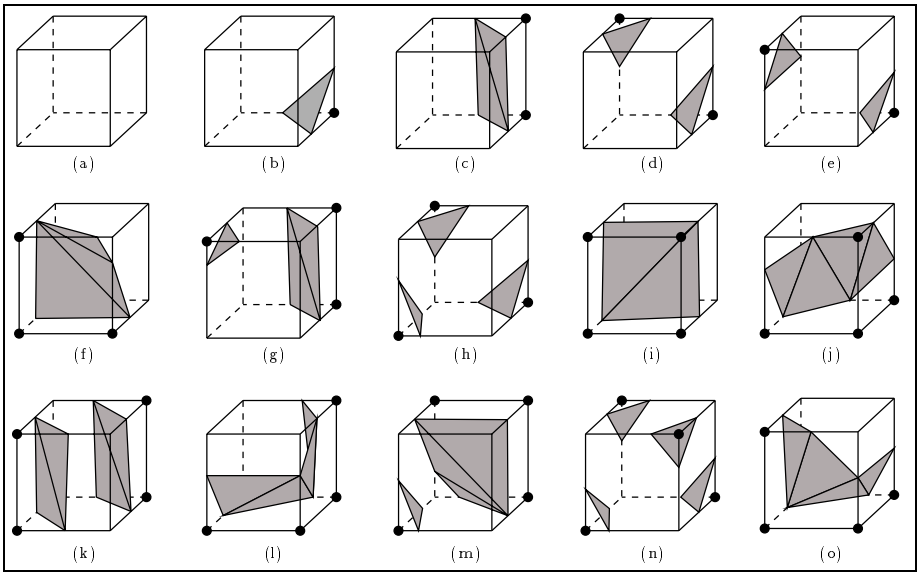

Figure 3.2 15 distinct vertex colorings

3.2.2.1 Consistent Connectivity

In cases in which the connectivity is not determined by the color of the vertices alone, continuous surfaces can be guaranteed by adopting a connectivity scheme which is consistent at a face shared by two adjacent cells.

Consistency may be achieved simply by subdividing each cell into tetrahedra and using a linear interpolant within each tetrahedron [DK91]. An efficient approach to consistency is to adopt a consistent decision rule, such as sampling the function at the center of the ambiguous face to determine the local topology [WMW86].

3.2.2.2 Correct Connectivity

The core of the problem along shared cell faces lies in determining the topological connectivity of vertices which are colored the same but which lie diagonally across a face or body of a cell. A second class of connectivity solutions guarantee consistency on a shared face by ensuring correctness with respect to a particular data model.

Nielson and Hamann propose generating a consistent decision on connectivity by enforcing a topology which is correct with respect to the bilinear interpolant along the face [NH91]. Kenwright derives a similar condition for disambiguating the connectivity on the faces in terms of the gradient of the bilinear interpolant [Ken93]. Natarajan further enforces consistency with the trilinear interpolant for the case of ambiguities which are interior to a cell, which occur when diagonal vertices across the body of the cell are similarly colored but have no edge-connected path of vertices of the same color between them [Nat94]. Karron et al. [KCM94] further discuss the proper treatment of criticalities in isocontouring, proposing a digital morse theory for describing the topological transitions of isocontours of scalar fields.

Zhou et al. [ZCT95] make the point that a tetrahedral decomposition and linear approximation change the function and may still result in incorrect, though consistent, topology. They propose that a tetrahedral decomposition may be used, provided that intersections along the introduced diagonals are computed for the cubic function which results from sampling the trilinear function across the diagonal of a cell, rather than applying linear interpolation along all edges.

Wilhelms and Van Gelder [WvG90, vGW94] provide a comprehensive review the topological considerations in extracting isosurfaces, and demonstrate that gradient heuristics applied at the vertices of a cell are necessary and sufficient to disambiguate the topology of functions which are quadratic.

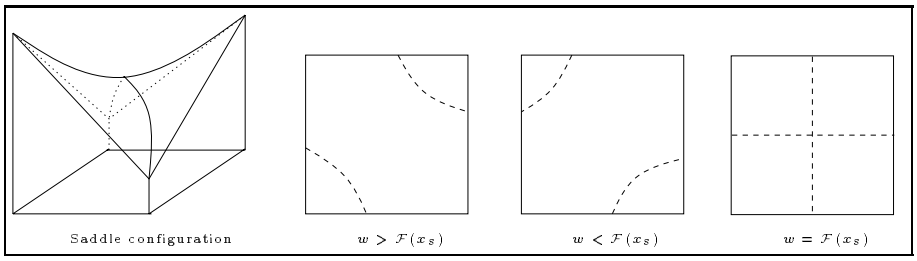

Figure 3.3 A two dimensional bilinear saddle and its contour configurations

The solution suggested by Natarajan [Nat94] is particularly attractive due to its simplicity of implementation and design to enforce consistency with the trilinear interpolant, a commonly used interpolant for 3D reconstruction and visualization. The situation on faces with colored vertices which are diagonally adjacent can be viewed in two dimensions as in Figure 3.3. The unique saddle point at coordinate $\mathbf{x_s}$ of the bilinear interpolant lies interior to the face, and the correct topology can be determined by evaluating the function at the saddle point and comparing it with the isovalue as shown. This topological consistency is carried out further by considering the unique saddle point of the full trilinear interpolant in addition to the six possible face saddles. A simple extension to a traditional case table requires sub-cases only for configurations which contain saddles. The sub-cases are indexed by the saddle point evaluations in order to determine a triangulation which is topologically consistent with the trilinear interpolant [Nat94]

Matveyev further simplifies the correctness in connectivity for the case of a regular cell by avoiding the explicit computation of the saddle points [Mat94]. With the observation that the asymptotes of a saddle in a regular cell are parallel to the coordinate axes, correct connectivity can be determined by sorting the intersections along an axial direction. The nature of the bilinear interpolant ensures that pairs in the sorting will be connected.

For the inconsistent case illustrated in Figure 3.1, several distinct topological triangulations are possible, two of which are illustrated in Figure 3.4.

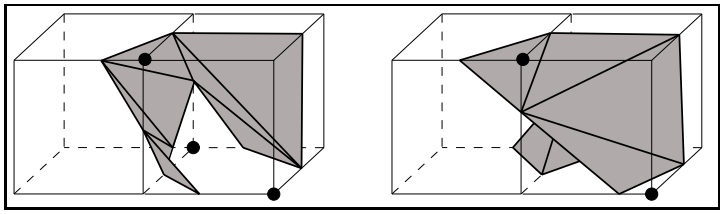

Figure 3.4 Two topologically consistent triangulations with respect to the shared face. Note that additional distinct configurations exist due to additional face saddles on the non-shared faces

3.3 Cell Search

Because a contour only passes through a fraction of the cells of a mesh on average, algorithms which perform an exhaustive covering of cells are found to be inefficient, spending a large portion of time traversing cells which do not contribute to the contour. The straightforward approach of enumerating all cells to extract a contour leads to a high overhead cost when the surface being sought intersects only a small number of the cells.

Preprocessing of the scalar field permits the construction of search structures which accelerate the repeated action of isocontouring, allowing for increased interactivity during modification of the isovalue. Many preprocessing approaches and search structures have been presented, which are conveniently classified (similar to the classification presented in [LSJ96]) based on whether the search is in *domain* space or *range* space.

3.3.1 Domain Search

A straightforward search of the domain by enumerating all cells leads to an overhead cost of $O(n_c)$. In the case when few cells are intersected, this overhead cost is a dominant factor, leading to inefficient computation. A spatial hierarchy for accelerating the search process is a natural approach which has been explored by Wilhelms and Van Gelder [WvG92, WG90]. For space efficiency considerations, a partial octree decomposition was developed which groups all cells at the highest level and adaptively approximates the data through axis-aligned subdivisions which better approximate the data. At each level in the tree, *min* and *max* values for the cells contained in the subtree are stored, providing a means to efficiently discard large spatial regions in the search phase. An analysis presented in [LSJ96] suggests a worst-case computational complexity of $O(k + k \log \frac{n_c}{k})$, where k is the size of the output and n_c is the number of cells.

3.3.2 Range Search

A large number of search techniques in the recent literature perform the search for intersected cells in the *range* space of the function. As we are dealing only with scalar-valued functions, range space search techniques have the advantage of being indepen-

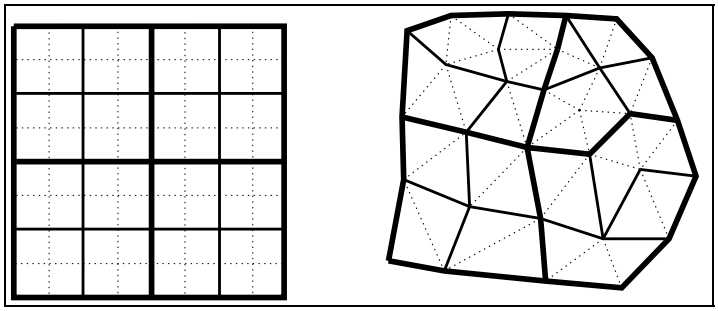

Figure 3.5 Spatial hierarchical cell decompositions for accelerating the search for isocontours.

dent of the dimension of the domain. In range space, each cell c is associated with the continuous set of values taken on by the function over the domain:

$$R(c) = [\min_{\mathbf{x} \in c} \mathcal{F}(\mathbf{x}), \max_{\mathbf{x} \in c} \mathcal{F}(\mathbf{x})]$$

There are two approaches for representing the range space, the 1D *value*-space, in which each range $R(c)$ is considered as a segment, or interval, along the real line, and the 2D *span*-space, in which each range $R(c)$ is considered as a point in 2D [LSJ96], as illustrated in Figure 3.6. While certain search structures are motivated by one geometric representation or another, others may be effectively visualized in either representation.

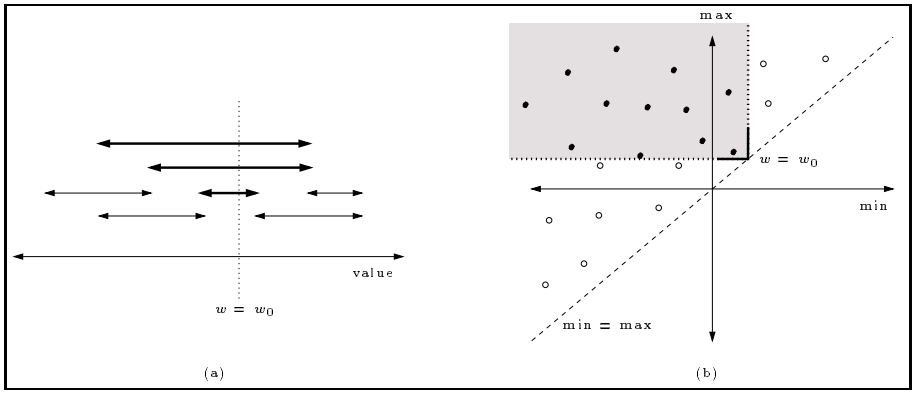

Figure 3.6 The (a) 1D value space and (b) 2D span space representations for range-space searches

Giles and Haimes introduce the use of lists of cells sorted by their minimum and maximum values to accelerate searching. In addition to forming two sorted lists of cells, the maximum cell range, Δw, is determined. Cells containing an isosurface of value w must have minimum value in the range $[w - \Delta w, w]$, which may be determined by binary search in the *min*-sorted array. This *active set* of cells is purged of cells whose

range does not contain w. For small changes in w, the active list can be updated, rather than wholly recomputed, by adding and purging new candidate cells to the active list. In the worst case, complexity remains $O(n_c)$.

Shen and Johnson [SJ95] describe the *Sweeping Simplices* algorithm, which builds on the *min-max* lists of Giles and Haimes and augments the approach with a hierarchical decomposition of the value-space. The *min*-sorted list is augmented by pointers to the associated cell in the *max*-sorted list, and the *max*-sorted list is augmented by a "dirty bit." For a given isovalue, a binary search in the *min*-sorted list determines all cells with minimum value below the isovalue. Pointers from the minimum value list to the maximum value list are followed to set the corresponding dirty bit for each candidate cell. At the same time, the candidate cell with the largest maximum value which is less than the isovalue is determined. As a result, all marked (candidate) cells to the right of this cell in the maximum list must intersect the contour, as they have minimum value below the isovalue and maximum value above the isovalue. Optimizations may be performed when the isovalue is changed by a small delta. One *min-max* list is created for each level of a hierarchical decomposition of the *min-max* search space. The overall complexity remains $O(n_c)$ in the worst case analysis.

Gallagher [Gal91] describes a span filtering algorithm, in which the entire range space of the scalar function is divided into a fixed number of *buckets*. Cells are grouped into buckets based on the minimum value taken on by the function over the cell. Within each bucket, cells are classified into one of several lists, based on the number of buckets which are *spanned* by the range of the cell. For an individual isovalue, cells which fall into a given bucket need only be examined if their span extents to the bucket which contains the isovalue. In the worst case, complexity remains $O(n_c)$.

Itoh and Koyamada [IK95] compute a graph of the extrema values in the scalar field. Every connected component of an isocontour is guaranteed to intersect at least one arc in the graph. Isocontours are generated by propagating contours from a seed point detected along these arcs. Noisy data with many extrema will reduce the performance of such a strategy. Livnat et al. [LSJ96] note that in the worst case the number of arcs will be $O(n_c)$, and hence straightforward enumeration of the arcs is equivalent in complexity to enumeration of the cells.

Livnat, Shen, and Johnson describe a new approach which operates on the 2D *min-max span space* [LSJ96]. The span-space representation of the cells is preprocessed using a *Kd-tree*, which allows $O(k+\sqrt{n_c})$ worst case query time to determine the cells which intersect the contour, where k is the size of the output. It is reported that in the average case, k is the dominant factor, providing optimal average complexity.

The same authors, with Hansen [SHLJ96], have described a technique which demonstrates improved empirical results by using an $L \times L$ lattice decomposition of the span space, in addition to allowing for parallel implementation on a distributed memory architecture. With certain assumptions on the distributions of points in the span space, the worst-case query time improves to $O(k + \frac{n_c}{L} + \frac{\sqrt{n_c}}{L})$.

Several authors have recently demonstrated improved worst-case performance bounds with the use of the *interval tree* and *segment tree* data structures [BPS96, CMPS97, vK96]. Both structures provide a search complexity of $O(k + \log n_u)$, where n_u is the number of unique extreme values of the segments which define the tree and k is the number of reported segments intersected.

3.3.2.1 Range Queries

The fundamental isocontouring query concerns the enumeration of all cells c such that $w \in R(c)$ for the input isovalue w. In this section, four data structures supporting the range query are described in more detail, including discussion of storage complexity, time complexity for creation of the structure, and query complexity for reporting cells intersected given an isovalue.

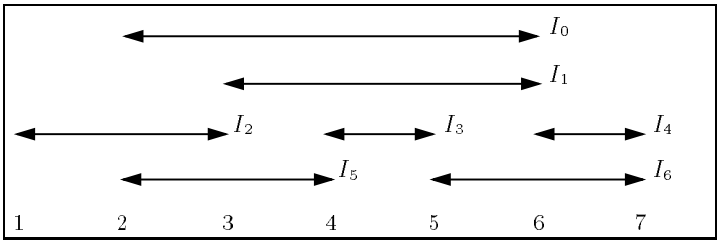

Figure 3.7 A set of segments representing cell ranges

In the following sections we review the *interval tree*, *segment tree*, and *bucket* search structures as applied to the contour query problem described. Example search structures are illustrated for the input set of intervals shown in Figure 3.7. For each search structure, the complexity measures are based on the insertion of n_s cells (value-space intervals) into the search structure. In the case that interval endpoints are taken from a small set of values (such as a limited set of the integers), the number of unique interval values is called n_u.

3.3.2.2 Interval Tree

An *interval tree* is made up of a binary tree over the set of interval *min/max* values [McC85]. Each internal node holds a *split value* s, with which intervals are compared during insertion into the tree. If the interval is entirely *less than* the split value it is inserted into the left subtree, while intervals *greater than* the split value are recursively inserted into the right subtree.

In the case that the interval spans the split value ($min < s < max$), the recursion terminates and the given interval is stored at the current node. Each nodes maintains two list of spanning cells. The first list is stored in increasing order by the *min*, the second in decreasing order by the *max* value. Because the intervals are not split in the recursive insertion, each interval is stored only twice, and the storage complexity is $O(n_s)$.

3.3.2.3 Segment Tree

A segment tree also consists of a binary search tree over the set of *min* and *max* values of all the seed cells [Meh84, Mul94]. The primary difference from the interval tree is the manner in which the segments are stored. Nodes in a segment tree form a multi-resolution hierarchy of intervals, with the root representing the infinite line, and

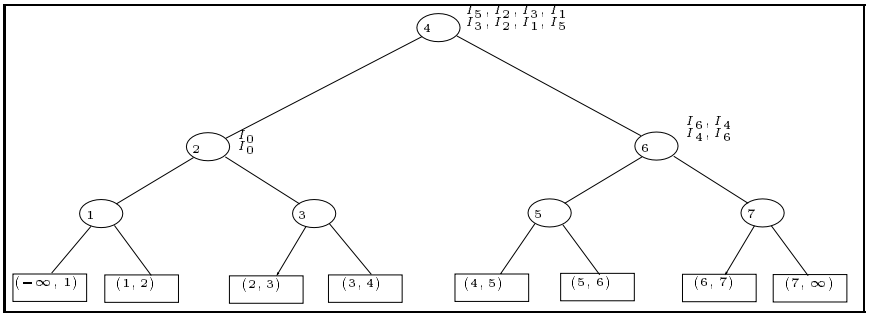

Figure 3.8 Interval tree for the intervals given in Figure 3.7

with each node dividing the parent interval at a split value (see Figure 3.9). When a segment is inserted into the tree, it is recursively split and propagated downward in the tree, to be inserted into the group of nodes whose intervals collectively sum to the entire range of the segment. Each segment identifier will be stored at most $O(\log n_u)$ times, where $\log n_u$ is the height of the tree, resulting in worst case storage complexity of $O(n_s \log n_u)$ in the improbable case that all *min-max* values are distinct, and all intervals filter all the way down to the leaves. The query complexity for reporting the k intersected cells for a given isovalue w is $O(k + \log n_u)$.

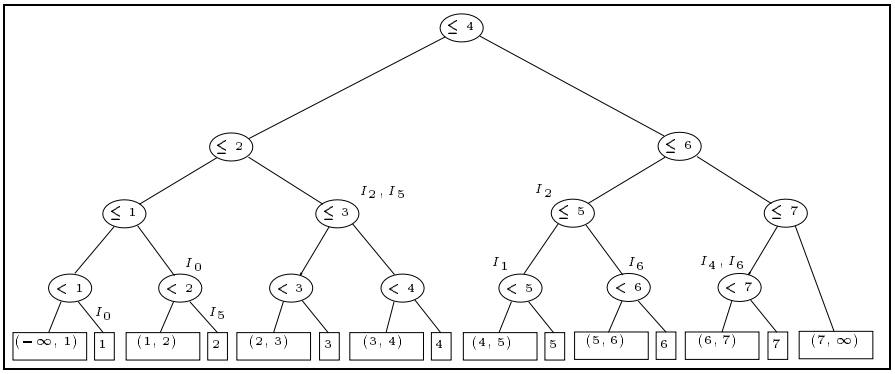

Figure 3.9 Segment tree for the segments given in Figure 3.7

3.3.2.4 Bucket Search

Much of the scientific data that one are concerned with comes in the form of integer values in a small range. For example, Computed Tomography (CT) data generally have a 12-bit integer range of values. This regular subdivision allows a simple bucket search strategy with $n_u - 1$ buckets each representing a unit interval $(h, h + 1)$. For each cell, an identifier is stored in each bucket which is spanned by the cell. Clearly, the worst case storage complexity of this strategy is $O(n_s n_u)$, which may be infeasible

in the case in which all cells are stored. Given the approach of forming a small set of seed cells, such a technique may prove feasible, with the added benefit of allowing intersected cells to be reported in $O(k)$ time, linear in the number of reported cells.

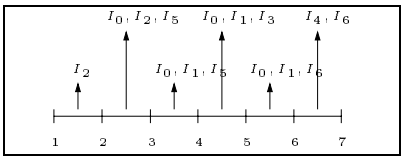

Figure 3.10 Bucket search structure for the intervals given in Figure 3.7

3.3.2.5 Search Structure Discussion

In this section we discuss the storage cost of each of the three presented search structures. Table 3.1 summarizes the theoretical space and query complexities.

Search Structure	Storage Complexity	Query Complexity
Interval Tree	$O(n_s)$	$O(k + \log n_u)$
Segment Tree	$O(n_s \log n_u)$	$O(k + \log n_u)$
Bucket	$O(n_s n_u)$	$O(k)$

Table 3.1 Comparison of the theoretical complexities of the three search structures for performing an interval query.

3.4 Cell Traversal

The order in which cells are visited can impact the efficiency of contouring algorithms in several ways. Coherent traversal algorithms, such as a regular traversal scheme or contour propagation (breadth first traversal of a connected component), can potentially be implemented more efficiently than a random cell visitation order. One issue is the efficiency of avoiding re-computation (recomputing intersection along shared edges of cells). Through regular traversal and contour propagation, information can be saved more efficiently than in a random order visitation which is required by some cell search techniques.

3.4.1 Contour Propagation

Contour propagation [AFH80, HB94, IK95, BPS96] is a surface tracking method which is based on continuity of the scalar field, and hence of the isocontours derived from the field. Given a single *seed cell* on a connected component of a contour, the entire component is traced by breadth-first traversal through the face-adjacencies. The traversal is terminated when a cell which has already been processed is met again, which is usually determined by a set of *mark* bits, which indicate for each cell whether processing has taken place. The procedure is illustrated in Figure 3.11. In a contour propagation

framework, as in a marching order traversal, optimization can be performed based on the fact that with each step, information from adjacent cells is available which can be used to avoid re-computation. In addition, the extracted contours are more easily transformed into representations such as triangle strips for efficient storage and rendering.

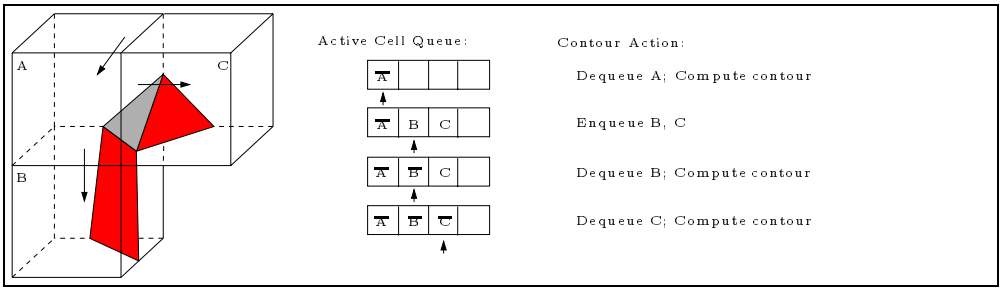

Figure 3.11 Illustration of contour propagation. The active surface is traced through adjacent cells.

Cignoni et al. introduce a limited propagation scheme for regular grids based on a "checkerboard" seed set, as illustrated in Figure 3.12. By selecting a regular pattern of cells, it is guaranteed that all contours will intersect a black or grey cell. Modified contour propagation rules are applied to reach white cells from the selected black or grey cells. Determining the seed set requires very little computation, thus preprocessing is essentially limited to building the range search structure, in this case an interval tree.

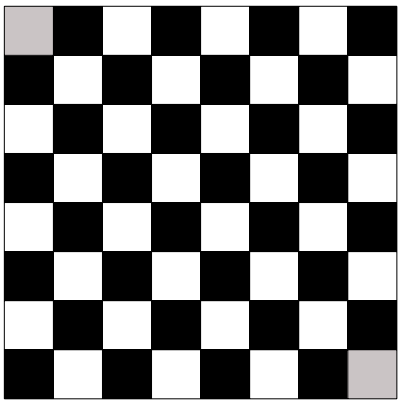

Figure 3.12 Illustration of the "checkerboard" approach to sufficient seed sampling. Black cells are on the checkerboard, while a number of grey cells are also required in the seed set.

3.4.2 Seed Set Construction

Several of the cell search techniques presented above depend upon a subsequent cell traversal algorithm such as contour propagation. The use of a subsequent cell traversal algorithm allows a reduction in the size of the search structure, because a cell which will be processed by *traversal* need not be entered into the primary search structure. The traversal stage can thus be considered a secondary search phase.

Propagation from seed cells requires a sufficient subset of cells which guarantee that every connected component for any arbitrary isovalue intersects at least one cell in the subset, called a *seed set*. A general definition of seed sets and framework for construction of seed sets is presented in [BPS96].

3.4.2.1 Optimal Seed Sets

In [vvB+97], the theory of optimal seed sets is discussed, which suggests that optimal (minimal) seed sets can be constructed in time which is polynomial in the number of cells, though the cost for minimal seed sets remains prohibitive for most cases.

The cost of implementation and computation for optimal seed sets is generally restrictive for all but the smallest of input meshes. Therefore, considerable work has been devoted to approximation algorithms. Algorithms for computing "good" seed sets are free to balance the desire for small (close to optimal) seed sets with the competing desire for low space/time complexity. As a result, seed set approaches can be tailored to suit a wide variety of settings and applications, depending on the available resources which can be dedicated to the computation. We review a selected subset of seed set construction algorithms.

3.4.2.2 Extrema Graphs

Itoh and Koyamada [IK95] introduce the use of *extrema graphs* for accelerating the search for isocontours. They observe that any closed contour must enclose an extremum of the scalar field, or be constant (or empty) within. By combining a search along a graph of the extreme points with a search of the boundary cells of the mesh, it is assured that at least one cell for each connected component of an isocontour is found. Cells extracted in this search are used as seed cells for an isocontour tracking algorithm, similar to the slicing algorithm described by Speray and Kennon [SK90].

3.4.2.3 Volume Thinning

Extending the extrema graph approach, Itoh et al. [IYK96] have applied image thinning techniques to progressively remove cells from a volume mesh which are not necessary in the "skeleton" of the function. Cells which are on the current boundary are iteratively visited, and may be removed subject to conditions on the connectivity of the neighboring cells which remain in the mesh. They report that the number of cells extracted by volume thinning are significantly fewer than those extracted using extrema graphs, partially due to the fact that boundaries are no longer considered as a special case. Furthermore, the computational complexity of volume thinning is virtu-

ally independent of the number of extrema, and thus the thinning approach results in faster preprocessing in many cases.

3.4.2.4 Greedy Climbing

For computation of a nearly optimal seed sets Bajaj et al. [BPS97b] develop a greedy technique which progressively covers the domain with seed cells by explicitly computing the coverage of each seed cell introduced. This *climbing* algorithm can be applied to both regular and unstructured grids of any cell type provided that the appropriate function R is given which computes the range of a cell or face.

The algorithm begins by considering the universal seed set S consisting of all cells c. Processing continues by iteratively selecting a cell in the seed set and tracing the set of all contours from the selected cell, effectively performing contour propagation for an interval of values. During the interval propagation, cells which are found to be unnecessary can be removed from the seed set. Figure 3.13 illustrates the selection and removal process.

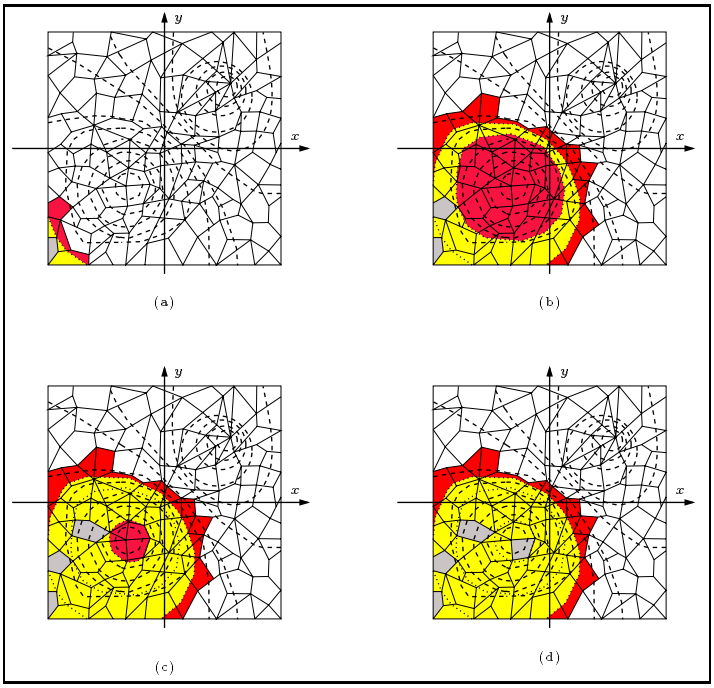

Figure 3.13 Greedy climbing approach to seed cell selection. Grey cells represent the selected seed cells. Yellow cells have been processed and removed from consideration, while red cells represent the current *front* of cells from which the next seed cell will be chosen.

3.4.2.5 Sweep Filtering

Bajaj et al. [BPS97b] present a seed selection algorithm with simple selection criteria, motivated by practical considerations when dealing with extremely large data. The seed selection is conceptually easiest to understand as a sweep of the cells in a particular direction. The algorithm has the property that selected seeds fall on the extrema of the contours in the given sweep direction. Detection of contour extrema is based on a simple comparison of the gradient within each cell and its immediate neighbors. With such a seed set, contouring may be performed coherently and efficiently by executing a contouring sweep, with only a slice of data required to be resident in memory at any given time, resulting in efficient computation for visualization of large out-of-core datasets.

The seed selection stage is illustrated as a left-to-right sweep in Figure 3.14. Conceptually, the sweep line l is moved from left to right to determine the order in which cells are processed. Note that this ordering is not required by the selection algorithm, and so cells which are stored in main memory can be processed in any order, or even in parallel. When a cell c is met which contains a local maximum of an isocontour along the sweep direction $\vec{l}_\perp$ the cell c is added to the seed set.

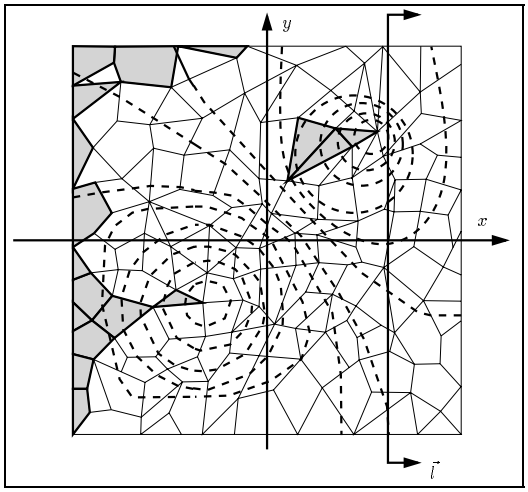

Figure 3.14 One-pass seed selection by forward sweep

Sweep filtering requires $O(n_c)$ time for considering each cell, and no additional storage beyond that of the extracted seed set (and the portion of the mesh kept in memory). In addition to facilitating out-of-core computation, the sweep filtering approach provides an extremely efficient method for computing a small seed set. Moreover, due to the local criteria for seed selection, cells may be considered in any order, allowing for parallel implementation with little or no communication overhead during the preprocessing.

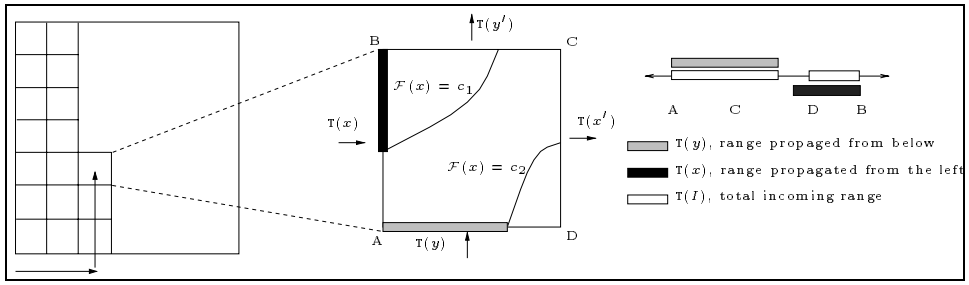

Figure 3.15 Illustration of responsibility propagation. Each cell processes input responsibilities and produces output responsibilities

3.4.2.6 Responsibility Propagation

In earlier work, Bajaj et al. [BPS96] described a plane sweep approach to seed set computation for regular grids. Results from the *responsibility propagation* algorithm fall between the general sweep filtering and the contour climbing. Processing of cells is performed in a regular traversal order, with ranges of responsibility propagated along the directions of the traversal. Illustrated in Figure 3.15, the traversal order is left-to-right and bottom-to-top. *Incoming* responsibility ranges are denoted $\mathtt{T}(x)$ and $\mathtt{T}(y)$, and computed propagated responsibilities are $\mathtt{T}(x')$ and $\mathtt{T}(y')$. Propagated responsibilities are computed using set arithmetic on the incoming responsibilities.

The *range propagation* method for selecting seed cells requires $O(n^{(d-1)/d})$ storage to maintain the propagated ranges for a sweep line or plane, where d is the dimension of the regular grid.

3.5 Hierarchical and Out-of-core Processing

Extremely large data often require special care and processing. Two approaches in particular have been explored with respect to contouring of extremely large scalar fields.

Zhou et al. [ZCK97] describe a hierarchical tetrahedral representation for volume data and discuss an approach to adaptive isocontouring from the multi-resolution volume representation. The hierarchy is constructed by recursive application of a set of three tetrahedron "splitting" rules. Isosurfaces can be extracted from the at a user-defined level of detail.

Chiang and Silva [CS97] use an I/O-optimal interval tree to perform efficient contouring of data which cannot be stored in primary memory. Results have indicated that isocontouring can be performed on data residing on secondary storage such that the I/O operations required are not the limiting factor of the computation. The only primary memory required includes a small constant amount to store a portion of the mesh and storage for the isosurface being constructed.

3.6 Summary

A key to efficient computation is in exploiting coherence. The isocontouring approaches described above can be loosely classified and analyzed based on the coherence which is exploited.

Spatial Coherence – We assume a minimum of C^0 continuity in our scalar field. Continuity along shared cell faces is exploited by many contouring approaches described above. The octree decomposition exploits spatial coherence in a hierarchical manner. As should be expected, the analysis in [LSJ96] reveals that the complexity gain breaks down when the spatial frequency is high, forcing large portions of the octree to be traversed.

Range-Space Coherence – Searches in range-space have demonstrated improved worst-case query complexity with performance which is independent of spatial frequency. Such advances, however, come at the cost of decreased ability to exploit spatial coherence. Assuming a continuous scalar field over a cell representation, cells which are spatially adjacent also overlap in the value space for the range of the shared face. However, the construction of value-space search structures such as the interval tree and segment tree are completely independent of assumptions such as scalar field continuity. While this may be an advantage in the case that discontinuous fields or disjoint groups of cells are considered, for most purposes it means that spatial coherence is under-utilized.

In general, domain-space and range-space searches exploit coherence in one sense by sacrificing coherence in another. The seed set approaches are best understood as a hybrid of spatial and value-space approaches, with the goal and result of exploiting both range-space and domain-space coherence.

The approach is based on a fragmentation of the search for intersected cells into *range-space* and *domain-space* phases, taking advantage of coherence in both. Range-space searches exhibit improved worst-case complexity bounds due to their independence from the spatial frequencies of the input data. By adopting *contour propagation* to compute each connected component, full advantage of spatial coherence during cell traversal is realized. Contour propagation also has the advantage of requiring only one *seed cell* for each connected component from which to begin tracing the contour, allowing for a much smaller search structure compared to algorithms which must search over the entire set of cells.

3.7 Future Directions

The use of interval tree and segment tree data structures has reduced the cost of searching for intersected cells to the point that contouring cost is highly dominated by the triangulation phase, and principally by the interpolation along mesh edges. Future avenues for interactive isocontouring will include improved approximate and hierarchical contouring algorithms. Hierarchical algorithms which are progressive will be developed, allowing computation to proceed at a specified rate for effective interaction use in real-time environments.

Another promising direction in making isocontouring more useful is through automated isovalue selection processes. Quantitative user interfaces such as the Contour Spectrum [BPS97a] both aid the user in selecting relevant isovalues while also providing a framework within which the relevance of isovalues can be directly computed, removing from the user much of the need for blindly exploring the space of isosurfaces.

4

Surface Interrogation: Visualization Techniques for Surface Analysis

DATA VISUALIZATION TECHNIQUES, Edited by C. Bajaj

5

Vector Field Visualization Techniques

DATA VISUALIZATION TECHNIQUES, Edited by C. Bajaj

6

Applications of Texture Mapping to Volume and Flow Visualization

DATA VISUALIZATION TECHNIQUES, Edited by C. Bajaj

7

Continuous Bayesian Tissue Classification for Visualization

DATA VISUALIZATION TECHNIQUES, Edited by C. Bajaj

References

[AFH80] E. Artzy, G. Frieder, and G. T. Herman. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 2–9, 1980.

[BLM97] M. J. Bentum, B. B. A. Lichtenbelt, and T. Malzbender. Frequency analysis of gradient estimators in volume rendering. *IEEE Transactions on Visualization Computer Graphics*, 1997.

[BPS96] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Fast isocontouring for improved interactivity. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 39–46, October 1996.

[BPS97a] ———. The contour spectrum. In *Proceedings of IEEE Visualization '97*, pages 167–173, October 1997.

[BPS97b] ———. Fast isocontouring for structured and unstructured meshes in any dimension. In *Proceedings of Late Breaking Hot Topics (IEEE Visualization 1997)*, pages 25–28, October 1997.

[CMPS97] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

[CS97] Y.-J. Chiang and C. T. Silva. I/o optimal isosurface extraction. In *Proceedings of IEEE Visualization '97*, pages 293–300, October 1997.

[DK91] A. Doi and A. Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans. Commun. Elec. Inf. Syst.*, E-74(1):214–224, 1991.

[Dur88] M. J. Durst. Additional reference to marching cubes. *Computer Graphics*, 22(2):72–73, 1988.

[Gal91] R. S. Gallagher. Span filtering: An efficient scheme for volume visualization of large finite element models. In G. M. Nielson and L. Rosenblum, editors, *Proceedings of IEEE Visualization '91*, pages 68–75, 1991.

[Hai91] R. Haimes. Techniques for interactive and interrogative scientific volumetric visualization, 1991.

[HB94] C. T. Howie and E. H. Blake. The mesh propagation algorithm for isosurface construction. *Computer Graphics Forum*, 13(3):65–74, 1994. Eurographics '94 Conference issue.

[IK95] T. Itoh and K. Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.

[IYK96] T. Itoh, Y. Yamaguchi, and K. Koyamada. Volume thinning for automatic isosurface propagation. In *Proceedings of IEEE Visualization '96*, pages 303–310, 1996.

[KCM94] D. B. Karron, J. Cox, and B. Mishra. New findings from the SpiderWeb algorithm: Toward a digital Morse theory. In *Visualization in Biomedical Computing*, volume 2359, pages 643–657, 1994.

[Ken93] D. Kenwright. Dual stream function methods for generating three-dimensional streamlines, 1993.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, 1987.

[Lor95] W. E. Lorensen. Marching through the visible man. In G. M. Nielson and D. Silver, editors, *Proceedings of IEEE Visualization '95*, pages 368–373, 1995.

[LSJ96] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm for unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[LVG80] S. Lobregt, P. W. Verbeek, and F. C. A. Groen. Three-dimensional skeletonization: Principle and algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):75–77, 1980.

[Mat94] S. V. Matveyev. Approximating of isosurface in the marching cube: Ambiguity problem. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of Visualization '94*, pages 288–292, 1994.

[McC85] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14:257–276, 1985.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1984.

[MMMY96] T. Moller, R. Machiraju, K. Mueller, and R. Yagel. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 71–78, 1996.

[MSS94] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of IEEE Visualization '94*, pages 281–287, 1994.

[Mul94] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[Nat94] B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11(1):52–62, 1994.

[NH91] G. M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity of marching cubes. In G. M. Nielson and L. Rosenblum, editors, *Visualization '91 Proceedings*, pages 83–91, 1991.

[SHLJ96] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency. In *Visualization '96 Proceedings*, pages 287–294, 1996.

[SJ95] H.-W. Shen and C. R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In G. M. Nielson and D. Silver, editors, *Proceedings of IEEE Visualization '95*, pages 143–150, 1995.

[SK90] D. Speray and S. Kennon. Volume probes: Interactive data exploration on arbitrary grids. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):5–12, November 1990.

[Sri81] S. N. Srihari. Representation of three-dimensional digital images. *Computing Surveys*, 13(4):399–424, 1981.

[van93] J. J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization '93*, pages 245–252, October 1993.

[vGW94] A. van Gelder and J. Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, 1994.

[vK96] M. van Kreveld. Efficient methods for isoline extraction from a digital elevation model based on triangulated irregular networks. *International Journal of Geographical Information Systems*, 10:523–540, 1996. Also appeared as Technical Report UU-CS-1994-21, University of Utrecht, the Netherlands.

[vvB+97] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore. Contour trees and small seed sets for isosurface traversal. In *13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.

[WG90] J. Wilhelms and A. V. Gelder. Octrees for faster isosurface generation extended abstract. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):57–62, 1990.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.

[WvG90] J. Wilhelms and A. van Gelder. Topological considerations in isosurface generation. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, pages 79–86, 1990. Published as Computer Graphics (San Diego Workshop on Volume Visualization), volume 24, number 5.

[WvG92] ———. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[ZCK97] Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing volume data. In *Proceedings of IEEE Visualization '97*, pages 135–142, October 1997.

[ZCT95] Y. Zhou, W. Chen, and Z. Tang. An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. *Computers and Graphics*, 19(3):355–364, 1995.