Parallel Algorithms for Volumetric Surface Construction

Joseph JaJa, Qingmin Shi, and Amitabh Varshney

Institute for Advanced Computer Studies

University of Maryland, College Park

Summary

Large scale scientific data sets are appearing at an increasing rate whose sizes can range from hundreds of gigabytes to tens of terabytes. Isosurface extraction and rendering is an important visualization technique that enables the visual exploration of such data sets using surfaces. However the computational requirements of this approach are substantial which in general prevent the interactive rendering of isosurfaces for large data sets. Therefore, parallel and distributed computing techniques offer a promising direction to deal with the corresponding computational challenges. In this chapter, we give a brief historical perspective of the isosurface visualization approach, and describe the basic sequential and parallel techniques used to extract and render isosurfaces with a particular focus on out-of-core techniques. For parallel algorithms, we assume a distributed memory model in which each processor has its own local disk, and processors communicate and exchange data through an interconnection network. We present a general framework for evaluating parallel isosurface extraction algorithms and describe the related best known parallel algorithms. We also describe the main parallel strategies used to handle isosurface rendering, pointing out the limitations of these strategies.

1. Introduction

Isosurfaces have long been used as a meaningful way to represent feature boundaries in volumetric datasets. Surface representations of volumetric datasets facilitate visual comprehension, computational analysis, and manipulation. The earliest beginnings in this field can be traced to microscopic examination of tissues. For viewing opaque specimen it was considered desirable to slice the sample at regular intervals and to view each slice under a microscope. The outlines of key structures in such cross-sectional slices were traced on to transparent sheets that were then sequentially stacked with transparent spacers in between them [WC71, LW72]. A user holding such a translucent stack would then be able to mentally reconstruct the location of the surface interpolating the contours on successive slices. It is therefore only natural that one of the first algorithms for surface reconstruction from volumes proceeded by interpolation between planar contours [FKU77]. Fuchs *et al.*'s method proceeded by casting the problem of optimal surface reconstruction between planar contours as one of finding minimum cost cycles in a directed toroidal graph. Their method computed the minimum-area interpolating surface between the given planar contours. The advent of X-ray computed tomography (CT) in 1973 through theoretical advances by Cormack [Cormack63, Cormack64] and their practical realization by Hounsfield [Hounsfield73] made it significantly easier to acquire highly-detailed cross-sectional images. The use of magnetic fields and radio waves to create images of physical objects by Lauterbur [Lauterbur73] and Mansfield [GGM74] led to the development of Magnetic Resonance Imaging (MRI). The obvious implications of such volumetric imaging technologies in medical diagnosis and therapy sparked a lively interest amongst radiologists and scientific visualization researchers and greatly accelerated research in a variety of volume visualization and analysis tools and techniques [Levoy88, Kaufman91]. The Visible Human Project (VHP) of the National Library of Medicine (NLM) has greatly spurred the development of fast volume visualization techniques, including fast techniques for isosurface extraction by making available 1971 digital anatomical images (15 GBytes) for the VHP male dataset and 5189 digital images (39 GBytes) for the VHP female dataset [Yoo04]. Some isosurfaces extracted from the VHP male dataset appear below [Lorensen01].

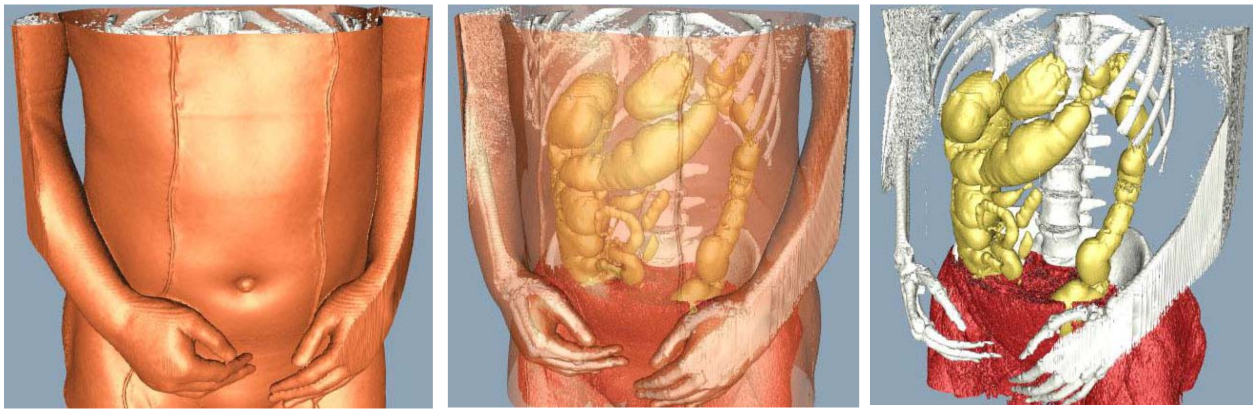


Figure 1: These images show various isosurfaces corresponding to different anatomical organs extracted and rendered using the Marching-Cubes algorithm. These images are reproduced from [Lorensen01].

Isosurface extraction also became important in another area – modeling of 3D objects with implicit functions. The basic idea in this research is to model 3D objects as zero crossings of a scalar field induced by an implicit function. The basis functions used in such implicit functions could be Gaussians (blobby models) [Blinn82], quadratic polynomials (metaballs) [NHK+86], or higher-degree polynomials (soft objects) [WMW86]. Subsequent work on level set methods by Osher and Sethian [OS88] added a temporal dimension to implicit surfaces and enabled them to evolve through time using numerical solutions of a time-dependent convection equation. Techniques based on level set methods have been extensively used in modeling a variety of time-evolving phenomena in scientific computing, CAD/CAM, computational fluid dynamics, and computational physics [OF03].

Simplicity of shape control, smooth blending, and smooth dynamics characteristics have led to the rapid adoption of isosurfaces for modeling a number of objects and phenomenon in digital entertainment media. For instance metaballs have been used in modeling of musculature and skin as isosurfaces of implicit radial basis functions.

Figure 2: An animation still that depicts battle between two battling Tyrannosauruses's. The mesh was taken from New Riders IPAS Plugin Reference CD. The muscular structure is created using the Metareyes plug-in. This involves creating hundreds of Metaballs to gradually build up a solid mesh. This image was created by Spencer Arts and is reproduced here from the website *http://www.siggraph.org/education/materials/HyperGraph/modeling/metaballs/trex.htm*

In dynamic settings morphing and fluid flow simulations have used isosurfaces computed as level sets of a scalar field. For instance, level-set-based morphing techniques developed by Whitaker and Breen [BW01] have been used in movies such as Scooby Doo 2 (screen shots below) and have been extended to work directly from a compressed format in a hierarchical runlength-encoded scheme [HNB+06].

Scooby Doo 2 Image © 2004 Warner Bros. Studios. Image provided by Frantic Films

(a) (b)

Figure 3: Tar Monster simulation in Scooby-Doo2 (Warner Brothers): (a) shows an early test sequence of the Tar Monster morphing, (b) shows the final rendered scene (Images copyright Frantic Films).

Although fairly sophisticated volume visualization techniques have evolved over the last decade, isosurface-based visualization of volumetric features continues to occupy an important cornerstone of modern-day volume visualization. The main reasons behind the continued interest in isosurface-based representations for volumes include:

- simple algorithms for extraction of the isosurface,
- triangle-mesh-based isosurfaces map well to the graphics hardware that is optimized for rendering triangles,
- isosurface-based depiction of volumetric features is intuitive and greatly facilitates their visual comprehension

2. Isosurface Extraction

We start by introducing a general framework for addressing the isosurface extraction problem. A scalar field function f defined over a domain $D \subset R^3$ is sampled at a set of points $\{v_i \in D\}_{i=1}^{n}$, where D has been decomposed into a set Σ of polyhedral cells $\{C_j\}$ whose vertices are the points $\{v_i\}$. The decomposition will be referred to as *structured* grid if the cells in Σ form a rectilinear or curvilinear mesh. Otherwise, we have an *unstructured* grid. In a structured grid, the connectivity of the cells is implicitly defined and Σ can be represented as a three-dimensional array of scalar field values of its vertices. In particular, cells in a rectilinear grid are axis-aligned but their vertices can be spaced arbitrarily in each of the dimensions. On the other hand, the connectivity of the cells in an unstructured grid has to be defined explicitly and such a grid will typically be represented by a list of all the cells (where each cell is specified by a list of its vertices) and another list consisting of the coordinates of the vertices. Given a scalar value λ, the isosurface corresponding to λ consists of the set of all points $p \in D$ such that $f(p) = \lambda$. Under fairly general conditions on *f*, it can be shown that the isosurface is a manifold that separates space into the surface itself and two connected open sets: "above" (or "outside") and "below" (or "inside") the isosurface. In particular, a point $p \in D$ is above the isosurface if $f(p) > \lambda$ and below the isosurface if $f(p) < \lambda$. The isosurface extraction problem is to determine, for any given isovalue, a piecewise linear approximation of the isosurface from the grid sampling of the scalar field.

The popular Marching Cubes (MC) algorithm, developed by Lorensen and Cline in 1987 [LC87], computes a triangular mesh approximation of the isosurface based on the following strategy. A cell C is cut by the isosurface for λ if, and only if, $f_{\min} \le \lambda \le f_{\max}$ where $f_{\min}$ and $f_{\max}$ are respectively the minimum and the maximum values of the scalar field over the vertices of C. Such a cell is called *active*; otherwise the cell is called *inactive*. The MC algorithm scans the entire cell set, one cell at a time, and determines whether a cell is active and in the affirmative generates a local triangulation of the isosurface intersection with the cell. More specifically, each vertex of an active cell is marked as “above” or “below” the isosurface depending on whether its scalar field value is larger or smaller than the isovalue, followed by a triangulation based on linear interpolation performed through a table look-up. This algorithm provides an effective solution to the isosurface extraction problem but can be extremely slow for moderate to large size grids. Since its introduction, many improvements have appeared which attempt to reduce the exhaustive search through the grid by indexing the data during a preprocessing phase. There are two general approaches to index the data and speed up the search. The first approach, primarily applicable to structured grids, amounts to developing a hierarchical spatial decomposition of the grid using an *octree* as the indexing structure. Each node of the octree is tagged by the minimum and maximum values of the scalar field over the vertices lying within the spatial region represented by the node. Such a structure restricts in general the search to smaller subsets of the data, and hence can lead to a much faster search process than the MC algorithm. Moreover, it can be effective in dealing with large scale data that does not fit in main memory. This approach was initially introduced by Wilhelms and Van Gelder in [WG92]. We should note that it is possible that, for some datasets, the octree search could be quite inefficient as many subtrees may need to be explored to determine relatively very few active cells.

The second approach focuses on the value ranges $[f_{\min}, f_{\max}]$ of all the cells, and how they can be organized to speed up the search. These ranges can be viewed either: (i) as points of the *span space* defined by a coordinate system in which one axis corresponds to the minimum value and the other axis corresponds to the maximum value, or (ii) as intervals over the real line. The problem of determining the active cells amounts to determining the points in the span space lying in the upper rectangle as indicated in Figure 4 or equivalently determining all the intervals that contain the isovalue λ. Algorithms based on the span space involve partitioning the span space into tiles and the encoding of the partition by some indexing structure. For example, we can partition the span space using equal-size or equal-frequency rectangular tiles [SHL+96] or using a *kd*-tree [LSJ96]. Theoretically optimal algorithms organize the intervals $[f_{\min}, f_{\max}]$ into an *interval tree* defined recursively as follows. The root of the tree stores the median of the endpoints of the intervals as the *splitting value*, and is associated with two lists of the intervals $[f_{\min}, f_{\max}]$ that contain the splitting value, one list in increasing order of $f_{\min}$ while the second is in decreasing order of $f_{\max}$. The size of this indexing structure is proportional to the number N of intervals and the search complexity is $O(\log N + k)$, where N is the total number of intervals and k is the number of intervals that contain the isovalue. It is easy to see that this search complexity is asymptotically the best possible for main memory algorithms.

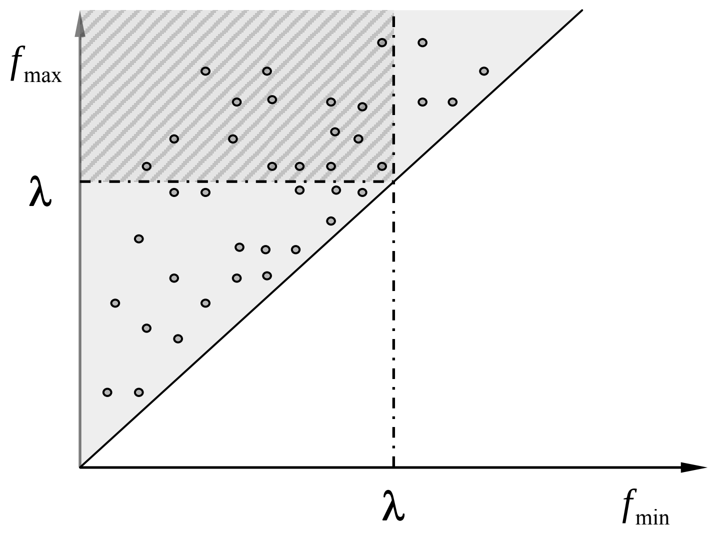


Figure 4: Span space and the subset corresponding to the isovalue.

It is worth noting that a hybrid technique was developed in [BPT+99] and involves the construction of a set of *seed cells,* such that, for any given isovalue, this set contains at least one cell from each connected component of the corresponding isosurface. The search for active cells involves contour propagation starting from some seed cells pertaining to the isosurface. Note that such a technique requires a structured grid to effectively explore neighboring cells as the contour is being propagated. A main issue with this technique is how to determine and organize the set of seed cells whose size may be quite significant.

In this chapter, we will primarily focus on isosurface extraction for large scale data, that is, the corresponding datasets are at least an order of magnitude larger than the size of the main memory of a processor and hence have to reside on disk storage. Due to their electromechanical components, disks have two to three orders of magnitude longer access time than random access main memory. A single disk access reads or writes a block of contiguous data at once. The performance of an external memory (or out-of-core) algorithm is typically dominated by the number of I/O operations, each such operation involving the reading or writing of a single disk block. Chiang and Silva [CS97] present the first out-of-core isosurface extraction algorithm based on an I/O optimal interval tree, which was later improved in [CSS98] using a variation of the interval tree called the binary-blocked I/O interval tree and the metacell notion to be introduced next. A *metacell* consists of a cluster of neighboring cells that are stored contiguously on a disk such that each metacell is about the same size that is a small multiple of the disk block size. For structured grids, a metacell consists of a subcube that can be represented by a sequence of the scalar values appearing in a predefined order. The metacell technique turns out to be quite useful when dealing with out-of-core algorithms.

Next we introduce a very efficient out-of-core isosurface extraction algorithm that will be easy to implement on a multiprocessor system with optimal performance as we will see later.

3. A Simple Optimal Out-of-Core Isosurface Extraction Algorithm

Introduced in [WJV06], the compact interval tree is a simple indexing structure that enables the extraction of isosurfaces by accessing only active metacells. It is a variation of the interval tree but makes use of the concepts of span space and metacells, and can be used for both structured and unstructured grids. We start by associating with each metacell an interval $[f_{\min}, f_{\max}]$ corresponding respectively to the minimum and maximum values of the scalar field over the metacell. Each node of the indexing tree contains a splitting value as in the case of the standard interval tree, except that no sorted lists of intervals are stored at each node. Instead, we store the distinct values of the $f_{\max}$ endpoints of these intervals in decreasing order, and associate with each such value a list of the left endpoints of the corresponding intervals sorted in increasing order. We present the details next.

Consider the span space consisting of all possible combinations of the $(f_{\min}, f_{\max})$ values of the scalar field. With each such pair we associate a list that contains all the metacells whose minimum scalar field value is $f_{\min}$ and whose maximum scalar field value is $f_{\max}$. The essence of the scheme for our compact interval tree is illustrated through Figure 5 representing the span space, and Figure 6 representing the compact interval tree built upon the *n* distinct values of the endpoints of the intervals corresponding to the metacells.

Let f_0 be the median of all the endpoints. The root of the interval tree corresponds to all the intervals containing f_0. Such intervals are represented as points in the square of Figure 5 whose bottom right corner is located at (f_0, f_0). We group together all the metacells having the same $f_{\max}$ value in this square, and store them consecutively on disk from left to right in increasing order of their $f_{\min}$ values. We will refer to this contiguous arrangement of all the metacells having the same $f_{\max}$ value within a square as a *brick*. The bricks within the square are in turn stored consecutively on disk in decreasing order of the $f_{\max}$ values. The root will contain the value f_0, the number of non-empty bricks in the corresponding square, and an index list of the corresponding bricks. This index list consists of at most *n/2* entries corresponding to the non-empty bricks, each entry containing three fields: the $f_{\max}$ value of the brick, the smallest $f_{\min}$ value of the metacells in the brick, and a pointer that indicates the start position of the brick on the disk (see Figure 6). Each brick contains contiguous metacells in increasing order of $f_{\min}$ values. We recursively repeat the process for the left and right children of the root. We will then obtain two smaller squares whose bottom right corners are located respectively at (f_{10}, f_{10}) and (f_{11}, f_{11}) in the span space, where f_{10} and f_{11} are the median values of the endpoints of the intervals associated respectively with the left and right subtrees of the root. In this case, each child will have at most *n/4* non-empty index entries associated with its corresponding bricks on the disk. This recursive process is continued until all the intervals are exhausted. At this point we have captured all possible pairs $(f_{\min}, f_{\max})$ and their associated metacell lists.

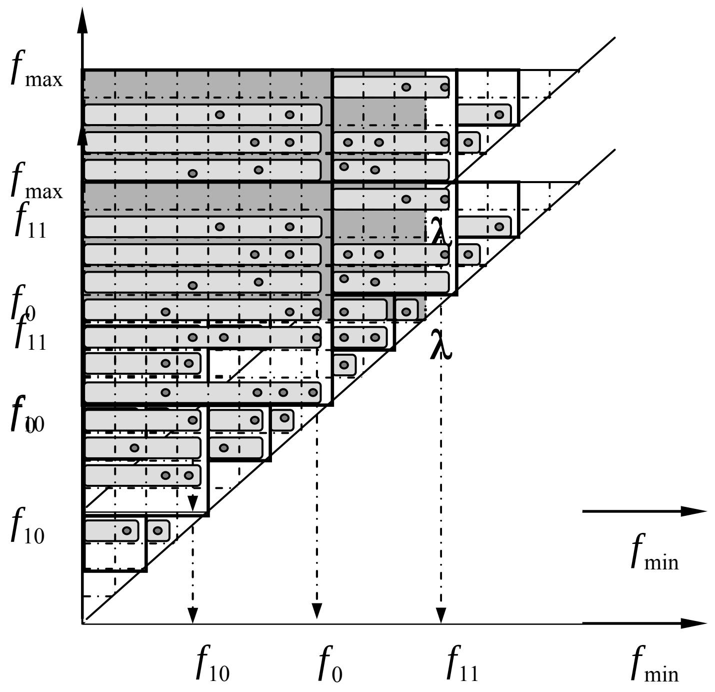


Figure 5: Span space and its use in the construction of the compact interval

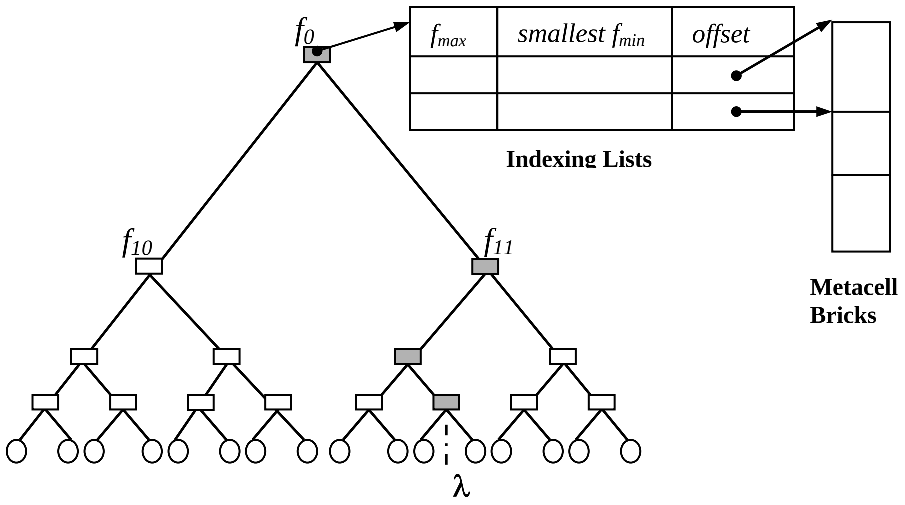


Figure 6: Structure of the compact interval tree.

Note that the size of the standard interval tree is in general much larger than the compact interval tree. We can upper bound the size of the compact interval tree as follows. There are at most *n/2* index entries at each level of the compact interval tree and the height of the tree is no more than $\log_2 n$. Hence our compact interval tree is of size *O(n*log *n),* while the size of the standard interval tree is $\Omega(N)$, where N is the total number of intervals and hence can be as large as $\Omega(n^2)$. Note that the size of the preprocessed dataset will be about the same size as the original input.

In Table 1 below we compare the sizes of the two indexing structures for some well-known data sets, including the datasets Bunny, MRBrain, and CTHead from the Stanford Volume Data Archive [Stanford]. As can be seen from the table, our indexing structure is substantially smaller than the standard interval tree, even when $N \approx n$.

Data Set	Scalar Field Size	N	n	Standard Interval Tree	Compact Interval Tree
Bunny	2 bytes	2,502103	5,469	38.2MB	42.7KB
MRBrain	2 bytes	756,982	2,894	11.6MB	22.6KB
CTHead	2 bytes	817,642	3,238	12.5MB	25.3KB
Pressure	4 bytes	24,507,104	20,748,433	374MB	158.3MB
Velocity	4 bytes	27,723,399	22,108,501	423MB	168.7MB

Table 1

We now address the algorithm to extract the isosurface using the compact interval tree. Given a query isovalue λ, consider the unique path from the leaf node labeled with the largest value $\leq \lambda$ to the root. Each internal node on this path contains an index list with pointers to some bricks.

For each such node, two cases can happen depending on whether λ belongs to the right or left subtree of the node.

- *Case1*: λ falls within the range covered by the node's right subtree. In this case, the active metacells associated with this node can be retrieved from the disk sequentially starting from the first brick until we reach the brick with the smallest value $f_{\max}$ larger than λ.
- *Case2:* λ falls within the range covered by the node's left subtree. The active metacells are those whose $f_{\min}$ values satisfy $f_{\min} \le \lambda$, from each of the bricks on the index list of the node. These metacells can be retrieved from the disk starting from the first metacell on each brick until a metacell is encountered with a $f_{\min} > \lambda$.

Note that since each entry of the index list contains the $f_{\min}$ of the corresponding brick, no I/O access will be performed if the brick contains no active metacells. It is easy to verify that the performance of our algorithm is I/O optimal in the sense that the active metacells are the only blocks of data which are moved into main memory for processing.

4. Strategies for Parallel Isosurface Extraction

To describe the basic strategies used for extracting isosurfaces on multiprocessor systems, we need to establish some general framework for discussing and evaluating parallel algorithms. We start by noting that a number of multiprocessor systems with different architectures are in use today. Unfortunately the different architectures may require different implementations of a given algorithm in order to achieve *effective and scalable performance*. By effective and scalable performance, we mean a performance time that *scales linearly relative to the best known sequential algorithm, without increasing the size of the problem*. In particular, such parallel algorithm should match the best sequential algorithm when run on a single processor. The development of such an algorithm is in general a challenging task unless the problem is embarrassingly parallel (that is, the input can be partitioned easily among the processors, and each processor will run the sequential algorithm almost unchanged).

A major architectural difference between current multiprocessors relates to the interconnection of the main memory to the processors. There are two basic models. The first is the *shared memory model* in which all the processors share a large common main memory and communicate implicitly by reading from or writing into the shared memory. The second model is a *distributed memory* architecture in which each processor has its own local memory and processors communicate through an interconnection network using message passing. In general, distributed memory systems tend to have a better cost/performance ratio and tend to be more scalable, while shared memory systems can have an edge in terms of ease of programmability. Many current systems use a cluster of symmetric multiprocessors (SMPs), for which all processors within an SMP share the main memory, and communication between the SMPs is handled by message passing through an interconnection network.

Another architectural difference, which is quite important for us, is the interconnection between the disk storage system and the processors. For a shared storage system, the processors are connected to a large shared pool of disks and a processor either has very small or no local disk. Another possibility is to assume that each processor has a its own local disk storage, and processors can exchange the related data through an interconnection network using message passing. These two models are the most basic and other variations have been suggested in the literature.

In this chapter, we will primarily focus on distributed memory algorithms (not involving SMPs) such that each processor has its own disk storage. Given that we are primarily interested in isosurface extraction for large scale data of size ranging from hundreds of gigabytes to several terabytes, we will make the assumption that our data cannot fit in main memory, and hence it resides on the local disks of the different processors. Under these assumptions, a parallel isosurface extraction algorithm has to deal with the following issues:

1. What is the input data layout on the disks of the processors performed during the preprocessing step? And what is the overhead in arranging the data in this format from the initial input format?
2. How is the overall dataset indexed and where is the index stored? What is the size of the index set?
3. What is the role of each processor in identifying the active cells and generating the triangles?

Critical factors that influence the performance include the amount of work required to index and organize the data during the initial preprocessing phase, the relative computational loads on the different processors to identify active cells and generate corresponding triangles for any isovalue query, and the amount of work required for coordination and communication among the processors. More specifically, the performance of a parallel isosurface extraction algorithm depends on the following factors:

1. *Preprocessing Overhead,* which include the time required to layout the dataset on the

processors' disks and to generate the indexing structure.

2. *Query Execution Time,* which depends on the following three measures:
    a. *Total Disk Access Time*, which can be captured by the maximum number of disk blocks accessed by any processor from its local disk.
    b. *Processor Computation Time*, which amounts to the maximum time spent by any processor to search through the local metacells (when in memory) and generate the corresponding triangles.
    c. *Interprocessor Communication*, which can be captured by the number of communication steps and the maximum amount of data exchanged by any processor during each communication step.

We focus here on the query execution time while taking note of the complexities of the preprocessing phase. Regarding the (parallel) execution time, the relative importance of the three measures depend on the multiprocessor system used and on their relative magnitudes. For example, interprocessor communication tends to be significantly slower for clusters assembled using off-the-shelf interconnects than for vendor-supplied multiprocessors with a proprietary interconnection network, and hence minimizing the interprocessor communication should be given a high priority in this case.

Having set a context for evaluating parallel isosurface extraction algorithms, we group the recently published parallel algorithms under the following two general categories.

- *Tree-Based Algorithms*

We group under this category the algorithms that make use of a single tree structure for indexing the whole dataset using either an octree or an external version of the interval tree. For example, using a shared-memory model, a parallel construction of an octree is described in [BGE+98], after which the isosurface extraction algorithm amounts to a sequential traversal of the octree until the appropriate active blocks (at the leaves) are identified. These blocks are assigned in a round-robin fashion to the available processors (or different threads on a shared memory). Each thread then excutes the MC algorithm on its local data. The preprocessing step described in [CS99, CFS+01, CSS98] involves partitioning the dataset into metacells, followed by building a B-tree like interval tree, called Binary-Blocked I/O interval tree (BBIO tree). The computational cost of this step is similar to an external sort, which is not insignificant. The isosurface generation requires that a host traverses the BBIO tree to determine the indices of the active metacells, after which jobs are dispatched on demand to the available processors. This class of algorithms suffers from the sequential access to the global indexing structure, the amount of coordination and communication required to dynamically allocate the corresponding jobs to the available processors, and the overhead in accessing the data required by each processor.

- *Range Partitioning Based Algorithms*

We group under this category all the algorithms that partition the range of scalar field values and then allocate the metacells pertaining to each partition to a different processor. In this case, the strategy is an attempt to achieve load balancing among the processors that use the local data to extract active cells. For example, the algorithm described in [ZBB01] follows this strategy. Blocks are assigned to different processors based upon which partitions a block spans (in the hope that for any isovalue, the loads among the processors will be even). An external interval tree (BBIO tree) is then built separately for the local data available on each processor. The preprocessing algorithm described in [ZN03] is based on partitioning the range of scalar values into equal-sized subranges, creating afterwards a file of subcubes (assuming a structured grid) for each subrange. The blocks in each range file are then distributed across the different processors, based on a work estimate of each block. In both cases, the preprocessing is computationally demanding and there is no guarantee that the loads on the different processors will be about the same in general. In fact one can easily come up with cases in which the processors will have very different loads, and hence the performance will degrade significantly in such cases.

We next describe an effective and scalable algorithm based on the compact interval tree indexing structure reported in [WJV06].

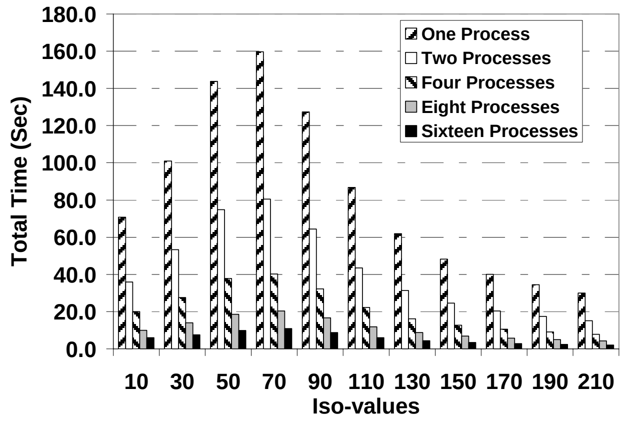
5. A Scalable and Efficient Parallel Algorithm for Isosurface Extraction

The isosurface extraction algorithm based on the compact interval tree, described in Section 3, can be mapped onto a distributed memory multiprocessor system in such a way as to provably achieve load balancing and a linear scaling of the I/O complexity without increasing the total amount of work relative to the sequential algorithm. Recall that the indexing structure is a tree such that each node contains a list of $f_{\max}$ values and pointers to bricks, each brick consisting of a contiguous arrangement of metacells with increasing $f_{\min}$ (but with the same $f_{\max}$ value). Let p be the number of processors available on our multiprocessor system, each with its own local disk. We now show how to distribute the metacells among the local disks in such a way that the active metacells corresponding to any isovalue are spread evenly among the processors.

For each index in the structure, we stripe the metacells stored in a brick across the p disks, that is, the first metacell on the first brick is stored on the disk of the first processor, the second on the disk on the second processor, and so on wrapping around as necessary. For each processor, the indexing structure will be the same except that each entry contains the $f_{\min}$ of the metacells in the corresponding local brick and a pointer to the local brick. It is clear that, *for any given isovalue, the active metacells are spread evenly among the local disks of the processors*.

Given a query, we start by broadcasting the isovalue to all the processors. Each processor will then proceed to use its own version of the indexing structure to determine the active metacells and extract the triangles as before. Given that each processor will have roughly the same number of active metacells, approximately the same number of triangles will be generated by each processor simultaneously. Note that it is easy to verify that we have split the work almost equally among the processors, without incurring any significant overhead relative to the sequential algorithm.

We have performed extensive experimental tests on this algorithm, all of which confirmed the scalability and efficiency of our parallel implementation. Here we only cite experimental results achieved on a 16-node cluster, each node is a 2-way Dual-CPU running at 3.0 GHz with 8 GB of memory and 60 GB local disk, interconnected with a 10Gbps Topspin InfiniBand network. We used the Richtmyer-Meshkov instability datatset produced by the ASCI team at Lawrence Livermore National Labs [LLNL]. This dataset represents a simulation in which two gases, initially separated by a membrane, are pushed against a wire mesh, and then perturbed with a superposition of long wavelength and short wavelength disturbances and a strong shock wave. The simulation data is broken up over 270 time steps, each consisting of a $2048 \times 2048 \times 1920$ grid of one-byte scalar field. Figures 7 and 8 illustrate the performance and scalability of the algorithm described in this section using 1, 2, 4, 8, and 16 processors. Note that the one-processor algorithm is exactly the same as the one described in Section 3 and hence the scalability is relative to the fastest sequential algorithm.



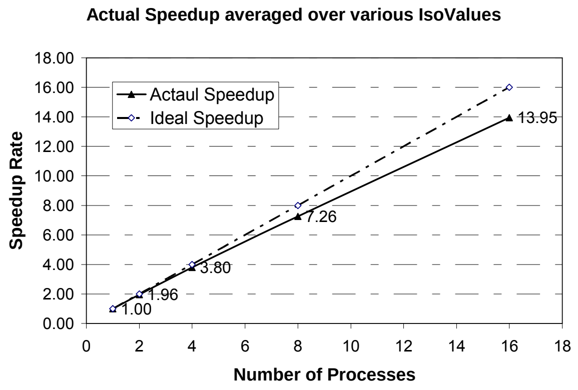


Figure 7

Figure 8

6. View-dependent Isosurface Extraction and Rendering

The isosurface extraction techniques described thus far aim at extracting a triangular mesh that closely approximates the complete isosurface. However, as the sizes of the data sets increase, the corresponding isosurfaces often become too complex to be computed and rendered at interactive speed. There are several ways to handle this growth in complexity. If the focus is on interactive rendering alone, one solution is to extract the isosurface as discussed in previous sections and then render it in a view-dependent manner using ideas developed by Xia and Varshney [XV96], Hoppe [Hoppe97], and Luebke and Erikson [LE97]. However full-resolution isosurface extraction is very time-consuming and such a solution can result in a very expensive pre-processing and an unnecessarily large collection of isosurface triangles. An alternative is to extract view-dependent isosurfaces from volumes that have been pre-processed into an octree

[FGH+03] or a tetrahedral hierarchy of detail [DDL+02, GDL+02]. This is a better solution, but it still involves traversing the multiresolution volume hierarchy at run time and does not directly consider visibility-based simplification. The most aggressive approaches compute and render only the portion of the isosurface that is visible from a given arbitrary viewpoint. As illustrated in Figure 9, a 2-D isocontour consists of three sections: A, B, and C. Only the visible portion, shown as bold curves, will be computed and rendered given the indicated viewpoint and screen.

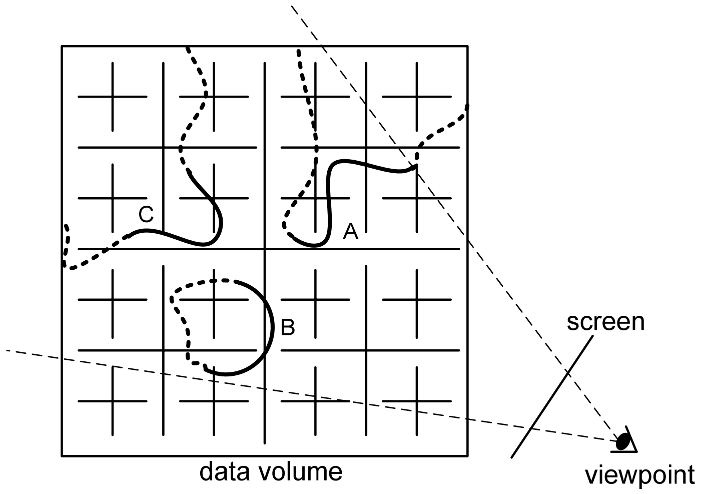


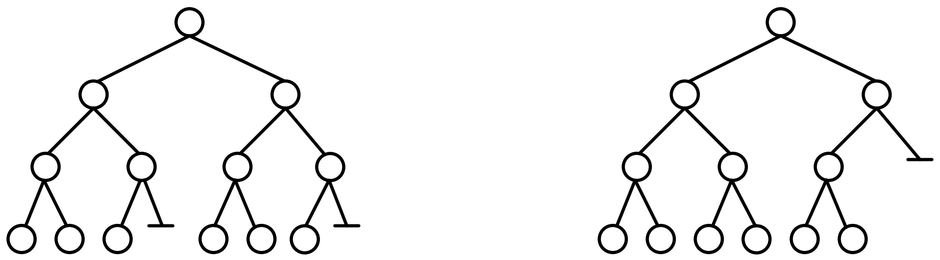
Figure 9: Illustration of the visibility of isosurface patches with respected to a viewpoint.

Experiments have shown that the view dependent approach may reduce the complexity of the rendered surfaces significantly [LH98]. In this section, we will describe a typical sequential view-dependent isosurface extraction algorithm and then introduce two existing parallel algorithms that more or less are based on this approach. We will also briefly discuss another view-dependent approach based on *ray tracing*.

A sequential algorithm for structured grids

The view-dependent isosurface extraction algorithm requires that cells be examined in a front-to-back order with respect to the viewpoint in order to guarantee that the isosurface patches on the front are generated first and are used to test the visibility of the remaining cells. Because the visibility test is in the volume space, an indexing structure based on volume space partitioning rather than value space partitioning is more suitable. One such data structure is the octree mentioned earlier. An octree recursively subdivides the 3-D rectangular volume into eight subvolumes. For the purpose of isosurface generation, each node v is augmented with a pair of *extreme values*, namely the minimum and maximum scalar values $f_{\min}(v)$ and $f_{\max}(v)$ of the points within the corresponding subvolume. When traversing the octree, we can check at each node v visited whether $f_{\min}(v) \leq \lambda \leq f_{\max}(v)$ for a given isovalue λ. If this is the case, then the subtree rooted at v is recursively traversed. Otherwise, the subvolume corresponding to v does not intersect the isosurface and the entire subtree can be skipped.

A particular version of the octree called *Branch-On-Need Octree (BONO)* [WG92] has been shown to be space efficient when the resolution of the volume is not a power of two in all three dimensions. Unlike the standard octree, where the subvolume corresponding to an internal node is partitioned evenly, a BONO recursively partitions the volume as if its resolution is a power of two in each dimension. However, it allocates a node only if its corresponding spatial region actually “covers’’ some portion of the data. An illustration of a standard octree and a BONO is given in Figure 10 for a one-dimensional case.



(a) A standard octree *(b) A Branch-On-Need Octree*

Figure 10: A 1D illustration of the standard octree and BONO, both having 6 leaf nodes.

The visibility test of a region is performed using the *coverage mask*. A coverage mask consists of one bit for each pixel on the screen. This mask is incrementally "covered" as we extract the visible pieces of the isosurface. A bit is set to 1 if it is covered by the part of the isosurface that has already been drawn and 0 otherwise. The visibility of a subvolume is determined as follows.

The subvolume under consideration is first projected onto the screen. If every pixel in the projection is set to one in the coverage mask, then the corresponding subvolume is not visible and can therefore be safely pruned. Otherwise, that subvolume needs to be recursively examined. Typically, to reduce the cost of identifying the bits to be tested, the visibility of the *bounding box* of the projection instead of the projection itself is tested. This is a conservative test but guarantees that no visible portion of the isosurface will be falsely classified as invisible. Once an active cell is determined to be visible, a triangulation of the isosurface patches inside it is generated as before. The triangles thus generated are used to update the coverage mask. Figure 11 gives the pseudo-code of the overall algorithm.

```
procedure traverse()
begin
   Set all bits in the coverage mask to 0;
   traverse(root, λ);
end

procedure traverse(node, λ)
begin
    if $f_{min}(node) > \lambda$ or $f_{max}(node) < \lambda$ then
        return;
    else
        Compute the bounding box b of the subvolume corresponding to node.
        if b is invisible then
            return;
        else
            if node is a leaf then
                Compute the triangulation of node;
                Update the coverage mask using the triangles just generated;
                return;
            else
                for each child c of node do
                    traverse(c, λ);
                endfor
            endif
    endif
End
```

Figure11: The framework of a sequential view-dependent isosurface extraction algorithm.

Mapping a sequential view-dependent algorithm to a distributed memory multiprocessor system presents several challenges. First, the set of visible active cells are not known beforehand, which makes it difficult to partition the triangulation workload almost equally among the available processors. Second, once the processors have finished triangulating a batch of active cells, the coverage mask needs to be updated using the triangles just computed, a process that requires significant inter-processor communication. And finally, in the case where data set cannot be duplicated at each node, careful data placement is required so that each processor is able to access most of the data it needs locally. We next describe two parallel algorithms that attempt to address these problems.

The multi-pass occlusion culling algorithm

The *multi-pass occlusion culling* algorithm was proposed by Gao and Shen in [GS01]. In this algorithm, one of the processors is chosen as the *host* and the other processors serve as the *slave*s. *Blocks*, each consisting of $m \times n \times l$ cells, are the units of the visibility test. The host is responsible for identifying active and visible blocks and for distributing them to the slaves on the fly. The slaves are responsible for generating the isosurface patches within the active blocks they receive, and for updating the corresponding portions of the coverage mask.

`

To improve the workload balance of the slave nodes, the entire set of active blocks is extracted at once at the host using a front-to-back traversal of an octree and is stored in an *active block list*. This list is then processed in multiple passes. Each pass picks some active blocks for triangulation and discard those determined as invisible. This process continues until the active cell list becomes empty.

A modified coverage mask is used to perform the visibility test. Each bit of the coverage mask has three states: *covered*, *pending*, and *vacant*. A covered bit indicates that the corresponding pixel has been drawn already; a vacant bit indicates a pixel that has not been drawn; and a pending bit represents a pixel that *may be* drawn in the current pass.

In each pass, the host first scans the active block list and classifies each block as *visible*, *invisible*, or *pending*. An active block is visible if its projection covers at least one vacant bit in the coverage mask. An active block is determined to be invisible if every bit in it is covered. Otherwise, this active block is classified as pending. A visible block is put into the *visible block list* and will be rendered in the current pass. Its projection is also used to update the corresponding invisible bits to pending**.** An invisible block is discarded. A pending block may be visible depending on the triangulation result of the current pass and therefore will remain in the active blocks list and will be processed in the next pass.

Once the visible blocks are identified, they are distributed to the slaves for triangulation. In order to keep the workloads at the slaves as balanced as possible, the number of active cells in each active and visible block is estimated. A recursive image space partitioning algorithm is used to partition the image space into *k* tiles, where *k* is the number of slaves, such that each tile has approximately the same number of estimated active cells. The active blocks, whose projected bounding boxes have centers that lie in the same tile, are sent to the same slave. Each slave maintains a local coverage mask. Once the current batch of active blocks is processed, the slave updates its local coverage mask. The local coverage masks are then merged to update the global coverage mask, which resides at the host. A technique called *binary swap* [MPH+94] is used to reduce the amount communication needed to update the global coverage mask.

A single pass occlusion culling algorithm with random data partitioning

Another parallel view-dependent isosurface extraction algorithm is due to Zhang *et al*. [ZBR02]. Unlike the algorithm of [GS01], which assumes that each node has a complete copy of the original data set, this algorithm statically partitions the data and distributes the corresponding partitions across all the processors. Deterministic data partitioning techniques based on scalar value ranges of the cells such as the one discussed in the previous section and the one in [ZBB01] can be used for this partitioning task. Zhang et al. show that random data distribution could be equally effective in terms of maintaining workload balance. In their scheme, the data set is partitioned into blocks of equal size and each block is randomly assigned to a processor with equal probability. They show that in such a scheme it is very unlikely that the workload of a particular processor will greatly exceed the expected average workload. The blocks assigned to a processor are stored on its local disk and an external interval tree is built to facilitate the identification of active blocks among them.

To address the problem of the high cost in updating the coverage mask, they opt to create a high quality coverage mask at the beginning and use it throughout the entire extraction process. This coverage mask is constructed using ray casting. A set of rays are shot from the viewpoint towards the centers of each boundary face of the boundary blocks. Some additional randomly rays are also used. For each ray, the first active block *b* it intersects is added to the *visible block set V*. To further improve the quality of the coverage mask, the nearest neighboring blocks of *b* is also added to *V*. The ray casting step can be done in parallel with each processor responsible for the same number of rays. The range information of all the blocks is made available at each processor for this purpose. The occluding blocks computed during the ray casting step are collected from all the processors and broadcast so that every processor gets the complete set of occluding blocks.

Using the local external interval tree, each processor obtains the set of local active blocks. Those local active blocks that are also occluding blocks are processed immediately and the isosurface patches generated are used to create the local coverage mask. The local coverage mask produced by each processor is then merged into a global coverage mask, which is broadcast to all the processors. Notice that, unlike the multi-pass algorithm, the computation of the global coverage mask in this case happens only once.

Once the global coverage mask is available to all the processors, each processor uses it to decide which of the remaining active blocks are visible and process only these blocks. The images created by different processors are then combined to yield the final image.

View-dependent isosurface extraction by ray tracing

Finally, we describe briefly another parallel scheme for extracting and rendering isosurface in a view-dependent fashion, namely *distributed ray tracing*. It was first proposed by Parker et al. [PSL+98], and was implemented on a shared-memory multiprocessor machine and later ported to distributed memory systems [DPH+03].

Unlike the view-dependent isosurface extraction algorithm we have just discussed, the ray tracing method does not actually compute the linear approximation of the isosurface. Instead, it directly renders the image pixel by pixel. For each pixel p, a ray r_p is created which shoots from the viewpoint through p towards the volume. The first active cell r_p intersects can be identified using an indexing structure such as the k-D tree [WFM+05] or the octree [WG92]. The location and orientation of the isosurface at its intersection point with r_p is then analytically computed and is used to draw the pixel p. To find the intersection point, we need to be able to determine the function value $f(p)$ of any point p within the active cell. This can be done using trilinear interpolation of the function values sampled at the vertices of the active cell.

One benefit of this scheme is that the tracing of a ray is totally independent of the tracing of other rays. Thus the parallelization of this scheme is quite straightforward. However, one needs to be aware that tracing thousands of rays can be very expensive compared to finding and rendering a linear approximation of the isosurface. Also, the tracing of each ray may involve any portion of the volume. Therefore, a shared-memory system is more suitable for this scheme. Nevertheless, as is demonstrated in [DPH+03], by carefully exploiting the locality of the search and by caching data blocks residing at remote processors, the same scheme can still achieve good performance in a distributed memory system.

To avoid tracing thousands of rays in the above scheme, Liu *et al.* [LFL02] have used ray-casting to only identify the active cells for a given view. These active cells then serve as seeds from which the visible isosurfaces are computed by outward propagation as suggested by Itoh and Koyamada [IK95].

7. Conclusion

We presented in this chapter an overview of the best known techniques for extracting and rendering isosurfaces. For large scale data sets, the interactive rendering of isosurfaces is quite challenging and requires some form of extra hardware support either through parallel processing on a multiprocessor or through effective programming of graphics processors. We have focused on the best known parallel techniques to handle isosurface extraction and rendering, and examined their performance in terms of scalability and efficiency. We believe that the rendering techniques do not achieve the desired performance and more research is required to improve on these techniques.

References

[Stanford] http://graphics.stanford.edu/data/voldata/

[LLNL] http://www.llnl.gov/CASC/asciturb/

[BPT+99] C. L. Bajaj, V. Pascucci, D. Thompson and X. Zhang. Parallel accelerated isocontouring for out-of-core visualization. In Proceedings of 1999 IEEE Parallel Vis. and Graphics Symp., pp. 97–104, 1999.

[BGE+98] D. Bartz, R. Grosso, T. Ertl, and W. Straßer. Parallel Construction and Isosurface Extraction of Recursive Tree Structures. In Proc. of WSCG'98, volume III, pp. 479–486, 1998.

[Blinn82] J. Blinn. A Generalization of Algebraic Surface Drawing, In ACM Transactions on Graphics, Vol. 1, No. 3, pp. 235–256, July, 1982.

[BW01] D. Breen and R. Whitaker. A Level-Set Approach for the Metamorphosis of Solid Models, In IEEE Transactions on Visualization and Computer Graphics, Vol. 7, No. 2, pp. 173–192, 2001.

[CFS+01] Y–J. Chiang, R. Farias, C. Silva and B. Wei. A unified infrastructure for parallel out-of-core isosurface and volume rendering of unstructured grids. In Proc. IEEE Symp. on Parallel and Large-Data Visualization and Graphics, pp. 59–66, 2001.

[CS97] Y–J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In Proceedings IEEE Visualization, pp. 293–300, 1997.

[CS99] Y. Chiang and C. Silva. External memory techniques for isosurface extraction in scientific visualization. In

External Memory Algorithms and Visualization, Vol. 50, pp. 247–277, DIMACS Book Series, American Mathematical Society, 1999.

[CSS98] Y–J. Chiang, C. T. Silva and W. J. Schroeder. Interactive out-of-core isosurface extraction. In Proceedings IEEE Visualization, pp. 167–174, 1998.

[Cormack63] A. M. Cormack. Representation of a function by its line integrals, with some radiological applications, In Journal of Applied Physics, Vol. 34, pp. 2722–2727, 1963.

[Cormack64] A. M. Cormack. Representation of a function by its line integrals, with some radiological applications. II, In Journal of Applied Physics, Vol. 35, pp. 2908–2913, 1964.

[DDL+02] E. Danovaro, L. De Floriani, M. Lee, H. Samet. Multiresolution Tetrahedral Meshes: an Analysis and a Comparison. In Proceedings of the International Conference on Shape Modeling, Banff (Canada), pp. 83–91, 2002.

[DPH+03] D. DeMarle, S. Parker, M. Hartner, C. Gribble, and Charles Hansen. Distributed Interactive Ray Tracing for Large Volume Visualization. In Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG'03), pp. 87–94, 2003.

[FGH+03] D. Fang, J. Gray, B. Hamann, and K. I. Joy. Real-Time View-Dependent Extraction of Isosurfaces from Adaptively Refined Octrees and Tetrahedral Meshes. In SPIE Visualization and Data Analysis, 2003.

[FKU77] H. Fuchs, Z. Kedem and S. P. Uselton. Optimal Surface Reconstruction from Planar Contours, In Communications of the ACM, Vol. 20, No. 10, pp. 693–702, October 1977.

[GS01] Jinzhu Gao and Han-Wei Shen. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In Parallel and LargeData Visualization and Graphics, pages 67–74. IEEE Computer Society Press, Oct. 2001.

[GGM74] A. N. Garroway, P. K. Grannell, P. Mansfield. Image formation in NMR by a selective irradiative process. In Journal of Physics C: Solid State Physics, Vol 7, pp. L457–462, 1974.

[GDL+02] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. I. Joy. Interactive View-Dependent Rendering of Large IsoSurfaces. In Proc. IEEE Visualization 2002 (VIS'02), pp. 475–482, 2002.

[Hounsfield73] G. N. Hounsfield. Computerized transverse axial scanning (tomography): part 1. Description of system, In The British Journal of Radiology, Vol. 46, pp. 1016–1022, 1973.

[Hoppe97] H. Hoppe. View-dependent refinement of progressive meshes. In Proceedings of SIGGRAPH 97, pp. 189–198, 1997.

[HNB+06] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation, In ACM Transactions on Graphics, Vol. 25, No. 1, pp. 1–24, 2006.

[IK95] T. Itoh and K. Koyamada. Automatic isosurface propagation using an extreme graph and sorted boundary cell lists. IEEE Transactions on Visualization and Computer Graphics, Vol. 1, No. 4, pp. 319–327, Dec 1995.

[Kaufman91] A. Kaufman. Volume Visualization, IEEE Computer Society Press, ISBN 0-8186-9020-8, 1991.

[Lauterbur73] P. Lauterbur. Image Formation by Induced Local Interactions: Examples Employing Nuclear Magnetic Resonance, In Nature, Vol. 242, pp. 190–191, 1973.

[Levoy88] M. Levoy. Display of Surfaces from Volume Data, IEEE Computer Graphics and Applications, Vol. 8, No. 3, pp. 29–37, May 1988.

[LW72] C. Levinthal and R. Ware. Three-dimensional reconstruction from serial sections. In Nature 236, pp. 207–210, March 1972.

[LFL02] Z. Liu, A. Finkelstein, and K. Li. Improving progressive view-dependent isosurface propagation. In Computers and Graphics, Vol. 26, pp. 209–218, 2002.

[LH98] Y. Livnat and C. Hansen. View dependent isosurface extraction. In Visualization '98, pp. 175–180, October 1998.

[LSJ96] Y. Livnat, H. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space". In IEEE Transactions on Visualization and Computer Graphics, Vol. 2, No. 1, pp. 73–84, 1996.

[Lorensen01] W. Lorensen. Marching Through the Visible Man, at http://www.crd.ge.com/esl/cgsp/projects/vm/, August 2001.

[LC87] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In Maureen C.Stone, editor. Computer Graphics (SIGGRAPH'87 Proceedings), Vol. 21, pp. 161–169, July, 1987.

[LE97] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In Proceedings of SIGGRAPH 97, pp. 199–208, 1997.

[MPH+94] K. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. Parallel volume rendering using binary-swap composition. In IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp. 59–68, 1994.

[NHK+86] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, K. Omura. Object Modeling by Distribution Function and a Method of Image Generation, In Electronics Communication Conference 85 J68-D(4), pp. 718–725, 1986 (in Japanese).

[OF03] S. Osher and R. Fedkiw. Level Set Methods and Dynamic Implicit Surfaces, Springer, NY, 2003.

[OS88] S. Osher and J. Sethian. Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations, In Journal of Computational Physics, Vol. 79, pp. 12–49, 1988.

[PSL+98] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In Visualization 98, pp. 233–238. IEEE Computer Society Press, October 1998.

[SHL+96] H. W. Shen, C. D. Hansen, Y. Livnat and C. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In IEEE Visualization'96, pp. 281–294, Oct.1996.

[WC71] M. Weinstein and K. R. Castleman. Reconstructing 3-D specimens from 2-D section images, In Proc. SPIE 26, pp. 131–138, May 1971.

[WFM+05] I. Wald, H. Friedrich, G. Marmitt, P. Slusallek, and H.-P. Seidel. Faster Isosurface Ray Tracing using Implicit KD-Trees. In IEEE Transactions on Visualization and Computer Graphics, Vol. 11, No. 5, pp. 562–572, 2005.

[WJV06] Q. Wang, J. JaJa, and A. Varshney. An efficient and scalable parallel algorithm for out-of-core isosurface extraction and rendering. To appear in IEEE International Parallel and Distributed Processing Symposium, 2006.

[WG92] J.Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. ACM Transactions on Graphics, Vol. 11, No. 3, pp. 201–227, July 1992.

[WMW86] G. Wyvill, C. McPhetters, B. Wyvill. Data Structure for Soft Objects, The Visual Computer, Vol. 2, pp. 227–234, 1986.

[XV96] J. C. Xia and A. Varshney. Dynamic View-dependent Simplification of Polygonal Models. In Proceedings of IEEE Visualization 96, pp. 327–34, 1996.

[Yoo04] T. Yoo. Insight into Images: Principles and Practice for Segmentation, Registration, and Image Analysis, A. K. Peters Ltd, MA, 2004.

[ZBB01] X. Zhang, C. L. Bajaj and W. Blanke. Scalable isosurface visualization of massive datasets on cots clusters. In Proc. IEEE Symposuim on Parallel and Large-data Visualization and Graphics, pp. 51–58, 2001.

[ZBR02] X. Zhang, C. L. Bajaj and V. Ramachandran. Parallel and out-of-core view-dependent isocontour visualization using random data distribution. In Proc. Joint Eurographics-IEEE TCVG Symp. on Visualization and Graphics, pp. 9–18, 2002.

[ZN03] H. Zhang and T. S. Newman. Efficient Parallel Out-of-core Isosurface Extraction. In Proc. IEEE Symposium on parallel and large-data visualization and graphics (PVG) ’03, pp. 9–16, Oct. 2003.