Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)
D. Bartz, X. Pueyo, E. Reinhard (Editors)

Interactive Ray Tracing of Time Varying Data

Erik Reinhard, Charles Hansen and Steve Parker

School of Computing, University of Utah

Abstract

We present a simple and effective algorithm for ray tracing iso-surfaces of time varying data sets. Each time step is partitioned into separate ranges of potentional iso-surface values. This creates a large number of relatively small files. Out-of-core rendering is implemented by reading for each time step the relevant iso-surface file, which contains its own spatial subdivision as well as the volumetric data. Since any of these data partitions is smaller than a single time step, the I/O bottleneck is overcome. Our method capitalizes on the ability of modern architectures to stream data off disk without interference of the operating system. Additionally, only a fraction of a time-step is held in memory at any moment during the visualization, which significantly reduces the required amount of internal memory.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Hardware Architectures]: Parallel Processing I.3.7 [Three-Dimensional Graphics and Realism]: Ray tracing

1. Introduction

Researchers in many science and engineering fields rely on insight gained from instruments and simulations that produce discrete samplings of three-dimensional scalar fields. Visualization methods allow for more efficient data analysis to guide researchers. Iso-surface extraction is an important technique for visualizing three-dimensional scalar fields by exposing contours of constant value[10]. These contours isolate surfaces of interest, focusing attention on important features in the data such as material boundaries and shock waves while suppressing extraneous information. Several disciplines, including medicine[12, 22], computational fluid dynamics (CFD)[5, 6], and molecular dynamics[9, 14], have used this method effectively.

Understanding the dynamic behavior of a data set requires the visualization of its changes with respect to time. However, most high performance computers possess neither the disk space nor the amount of memory necessary to store and manipulate large time-varying data sets efficiently. While visualization research has begun to address this problem[2, 3, 21, 20, 17, 24], data sets from both computational and measurement sources have continued to increase in size, putting pressure on storage systems. Simulations that compute and store multiple time steps further increase the demand for storage space, commonly producing data sets on the order of one half to one gigabyte per time step with hundreds of time steps. With this vast amount of data to process, the interactive iso-surface visualization problem is impacted by consuming large amounts of time reading a multitude of huge files from disk and potentially performing swapping due to limited physical memory. Without a high degree of interactivity, the user loses the visual cues necessary to understand the structure of the field, reducing the effectiveness of the visualization.

We present an algorithm for the interactive visualization of iso-surfaces in time-varying fields that minimizes the impact of the I/O bottleneck. By preprocessing the data into effective iso-contour ranges, the amount of data read is limited. By streaming the data from disk, the potential overall size of a time-varying simulation is bounded only by disk capacity, not by the I/O rate. Coupled with a parallel ray tracing engine, we achieve interactive time varying data visualization.

In the following sections, we first discuss related work and then present our algorithm for streaming data for iso-surface visualization of time-varying fields. We then provide experimental results, demonstrating the performance of the algorithm on several large time-varying data sets. Finally, we draw conclusions and suggest directions for future work.

2. Background

A number of different techniques have been introduced to increase the efficiency of iso-surface extraction over the linear search proposed in the Marching Cubes algorithm[13, 26]. Wilhelms and van Gelder[25] describe the branch-on-need octree (BONO), a space-efficient variation of the traditional octree. This data structure partitions the cells in the data based on their geometric positions. Extreme values (minima and maxima) are propagated up the tree during construction such that only those nodes that span the iso-surface, i.e. those with $minvalue < isovalue < maxvalue$, are traversed during the extraction phase.

Other recent methods have focused on partitioning the cells based on their extreme values. Livnat et al.[11] introduced the span space, where each cell is represented as a point in 2D space. The point's x-coordinate is defined by the cell's minimum value, and the y-coordinate by the maximum value. The NOISE algorithm described in [11] uses a kd-tree to organize the points. Shen et al.[18] use a lattice subdivision of span space in their ISSUE algorithm. This simplifies and accelerates the search phase of the extraction, as only one element in the lattice requires a full min-max search of its cells. This acceleration comes at the cost of a less efficient memory footprint than the kd-tree.

The Interval Tree technique introduced by Cignoni et al.[4] guarantees worst-case optimal efficiency. Cells, represented by the intervals defined by their extreme values, are grouped at the nodes of a balanced binary tree. For any iso-value query, at most one branch from a node is traversed.

An alternate technique is to propagate the iso-surface from a set of seed cells. Itoh et al.[7, 8], Bajaj et al.[1], and van Kreveld et al.[23] construct seed sets that contain at least one cell per connected component of each iso-surface. The iso-surface construction begins at a seed and is traced through neighboring cells using adjacency and intersection information.

An algorithm to improve I/O performance and allow efficient iso-surface extraction on data sets larger than physical memory was described by Chiang et al.[2, 3]. An interval tree is built on disk using a two-level hierarchy. Cells are first grouped into meta-cells and meta-intervals are defined. These meta-intervals are then composed into an interval tree, which is divided into disk block-sized groups to allow efficient transfer from disk.

Weigle and Banks[24] consider time-varying scalar data as a four-dimensional field. They construct an "iso-volume" for each iso-value, representing the volume swept by the iso-surface over time. Imposing a time constraint on the iso-volume yields an instantaneous surface. This method elegantly captures temporal coherence, but its high execution time makes it impractical for large data sets.

Shen[17] proposed the Temporal Hierarchical Index Tree to perform iso-surface extraction on time-varying data sets. This method classifies the data cells by their extreme values over time. Temporal variation of cells is defined using lattice subdivision, extending the ISSUE algorithm. Nodes in the tree contain cells with differing temporal variation and are paged in from disk as needed to extract an iso-surface at a particular time step. At every time step, an ISSUE search[18] is performed at each node. In order to accelerate the full min-max search, an Interval Tree is constructed in those lattice elements that may require such a search. The Temporal Hierarchical Index Tree shows significant improvement in storage requirements over construction of a span-space search structure which treats each time step as an independent data set. This is achieved while retaining an efficient search strategy for iso-surface extraction.

Shen's work clearly accelerates the search for iso-surfaces in time dependent data. However, at each time step the entire data domain (time step) is loaded into physical memory. The iso-surface extraction process potentially needs to access all of the time steps in a time-varying data set. If all time steps do not simultaneously fit into physical memory, I/O can become a bottle neck. As noted by Wilhelms and Van Gelder[25], for a particular iso-value, large portions of the data not containing the iso-value need not be examined. Similarly these same large portions of the data need not be read from disk when constructing an iso-surface. For time dependent data sets, these savings can be significant and has led us to develop a method aimed at exploiting this observation.

Sutton and Hansen overcome these limitations with their T-BON method for iso-surface extraction[21, 20]. The Temporal Branch-on-Need Octree (T-BON) extends the three-dimensional branch-on-need octree for time-varying iso-surface extraction. This minimizes the impact of the I/O bottleneck by reading from disk only those portions of the search structure and data necessary to construct the current iso-surface. By performing a minimum of I/O and exploiting the hierarchical memory found in modern CPUs, the T-BON algorithm achieves high performance iso-surface extraction in time-varying fields.

All of these methods still require rendering the resulting geometry. For complex geometry, a faster approach is to visualize iso-surfaces without explicitly extracting geometry. Such a method for interactive iso-surface visualization of very large datasets was introduced by Parker *et al.*[16]. Their ray tracing algorithm rendered iso-surfaces for static scalar fields by intersecting viewing rays with the data volume and displaying the iso-surface without generating an intermediate polygonal representation. The parallel nature of the ray tracing algorithm maps well onto the architecture of massively parallel computers. A 32 processor SGI Origin 2000 can generate images of an iso-surface at interactive rates even for large datasets. However, for time-varying datasets, it requires the entire time-series to be memory resident for interactive applications.

3. Approach

Our approach to ray tracing iso-surfaces of large time varying data sets, is to partition each time step into a number of small files. Each file contains the data for one time step and a small range of iso-values. During visualization, only one file containing the given iso-surface value and time step needs to be loaded into memory. This method of partitioning data reduces the amount of traffic between the disk sub-system and internal memory, making this a viable out-of-core rendering technique.

The partitioning of the time varying data into small chunks, is performed during a preprocessing step. Here, one time step is read into memory at a time and voxels are distributed over a number of iso-surface files for that particular time step. We typically split the range of iso-surface values into 256 non-overlapping subranges, creating a maximum of 256 files per time step. If the iso-surface value that needs to be visualized is known in advance, we only create one file per time step containing just that iso-value thereby saving a large amount of disk space.

Because we split the data into separate files, the regular grid structure of the input data is lost: we write separate voxels to file, which each consist of an (x,y,z) triplet specifying spatial location, as well as the eight iso-surface values associated with the vertices of the voxel. This data can be stored in various ways, dependent on whether storage space is at a premium or whether the rendering time needs to be optimized. If the data is stored as three integers for the coordinates and eight shorts for the iso-surface values, this amounts to 28 bytes of storage space per voxel. To optimize cache performance, this data may be padded to 32 bytes so that exactly four voxels may be stored in a single cache line. If disk storage needs to be minimized, the (x,y,z) triplet may be encoded as a single integer, provided that these coordinates do not take more than 10 bits each. Such encoding would reduce storage space to 20 bytes per voxel. In Section 4, we evaluate the performance of all three voxel storage mechanisms.

We note that an iso-value of 0 is a special case, indicating that no data is present. During data partitioning, we remove voxels that have an iso-value of zero for all eight vertices. For many data sets, this significantly reduces the amount of data stored.

Because ray tracing an unstructured set of voxels is inefficient without a spatial subdivision structure, for each of the iso-surface files, we create a grid spatial subdivision. The grid is created in traditional fashion, using the rule of thumb that the number of cells in this data structure is roughly equal to the number of voxels for the current time step and iso-value range. Note that in this paper we use the term "voxel" to indicate a volumetric element in the original data, whereas we use the term "cell" to indicate one element of the spatial subdivision that is superimposed on the data. The grid data structure is appended to the relevant iso-surface file. Because we would like to avoid any further processing of data during rendering, this spatial subdivision uses indices (stored as integers) instead of pointers. Although we have chosen a grid spatial subdivision for simplicity, our approach does not preclude the use of other spatial subdivisions.

Currently, the preprocessing stage determines the maximum size of the data sets that can be rendered, since one time step needs to fit into memory. Note that this is already a much less severe constraint than requiring that all time steps fit into memory, although data could also be read in small chucks. This would remove any size constraints due to the preprocessing.

During rendering, one processor is responsible for reading the next time step and all other processors trace rays through the data that is currently in memory. This overlapping of processes allows us to hide the latency of the data reads. The ray tracing engine is based on the interactive ray tracer developed by Parker et al.[15]. The processor responsible for displaying the image, the display thread, is also responsible for reading the data. The distribution of data over a large number of small files reduces the amount of data that needs to be read for each time step. However, we obtain a further reduction in reading time by using direct I/O, which is available on the SGI platform. This mechanism bypasses the operating system, allowing each set of data and spatial subdivision to be read quickly with a single read operation. Because one processor reads the data into memory while all others access data already in memory, we use double buffering of the data to avoid artifacts and race conditions.

The processors in charge of tracing rays operate in a conventional manner. Ray traversal through the grid data structure is unaltered, and the voxels that represent the volumetric data are treated as separate boxes. Ray-box intersections are straightforward and implementations are readily available in the literature[19]. Our approach therefore leverages existing infrastructures, while at the same time allowing out-of-core iso-surface rendering of large time varying data sets.

4. Results

Our system is implemented using the interactive ray tracer of Parker et al.[15], which runs on an SGI Origin 2000 with 32 processors, 12 GB of memory, and direct I/O capability. Data access from disk using direct I/O operates at 200 MB/s.

To evaluate our approach, we use the Jet and H300 datasets. The Jet dataset models the Kelvin-Helmholtz Instability in a 3D jet and consists of 100 time steps occupying 33 MB of disk space each. The H300 data set models a heptane pool fire and consists of 170 time steps of 55 MB each. A number of frames of each of these datasets is shown in Figure 1.

For each time step, one small iso-surface range (1/256th of the full range of iso values) is extracted and stored on disk. This process takes about 15 seconds per time-step for

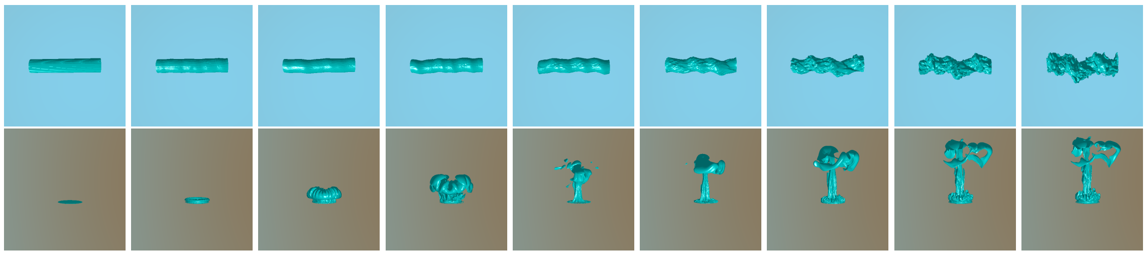

Figure 1: *Images from the Jet dataset (top) and H300 dataset (bottom).*

the Jet dataset and 25 seconds per time step for the H300 dataset. The voxels are stored either as three integers for the position of the voxel and eight shorts for its iso-values (28 bytes, called “standard”), one integer encoding voxel coordinates plus eight shorts (20 bytes, called “encoded”), or three integers, eight shorts and four bytes of padding for cache aligned operation (32 bytes, called “padded”). The resulting files sizes for each of these encoding schemes and for both data sets are presented in Figure 2. The numbers in this figure include the grid spatial subdivision that is appended to each file. Because reading the time-steps for each frame re-uses previously allocated memory, the amount of main memory used was never greater than 224MB. The frame-rates for these encoding schemes as function of time-step are presented in Figure 3. This data was collected without tracing of shadow rays. Note that the first few frames of the H300 dataset do not contain any data, which accounts for the very high frame-rates observed in this figure.

Speed-up figures are shown in Figure 4 for runs with and without shadow rays. The frame-rates shown were averaged over 500 frames, each rendered at 450x450 pixels. The reason to show results with and without ray tracing shadow rays is that for certain applications and datasets, shadows may be less relevant for analyzing the data.

We observe a distinct drop in frame-rate for both datasets when tracing shadow rays, which is according to expectation (Figure 4). In both cases the frame-rate appears to scale almost linearly with the number of processors. However, for the H300 dataset, the frame-rates scale better when the compact voxel coding scheme is used. Our hypothesis was that cache aligned data would perform best, but this does not appear to be the case in the current implementation. We speculate that the data coherence of our dataset is too low to measure an effect of such cache management. Presumably, if a bricking scheme were introduced[16, 21], alignment of voxel data with cache boundaries would become more beneficial.

Figures 2 and 3 show that for both datasets, the geometric complexity significantly increases over time. Later time steps therefore produce larger files and are slower to render. However, regardless of the complexity of each time step, the display thread which streams the data off disk, never becomes the bottleneck. For a typical frame, the read overhead

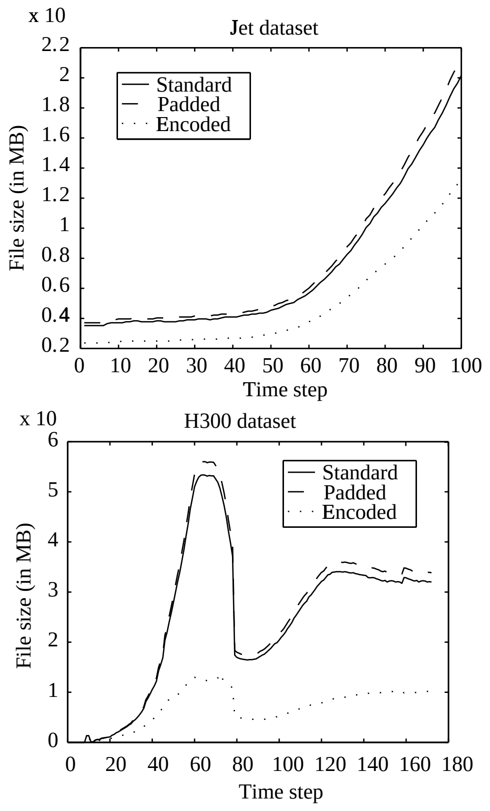

Figure 2: *Iso-surface file sizes as function of time step for each of three types of data storage.*

for the Jet dataset is shown in red in Figure 5. This figure also shows that the load is well balanced between processors, as all processors have rendering tasks to execute until the end of the frame.

5. Discussion

By splitting time varying datasets into separate iso-surface values, only a small amount of data needs to be read for each time step. This amount is small enough to allow out-

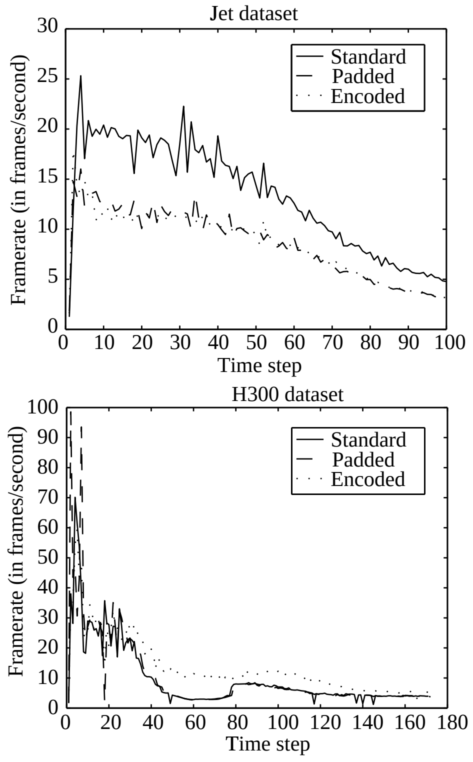

Figure 3: *Frame-rate as function of time step for each of three types of data storage (using 30 processors).*

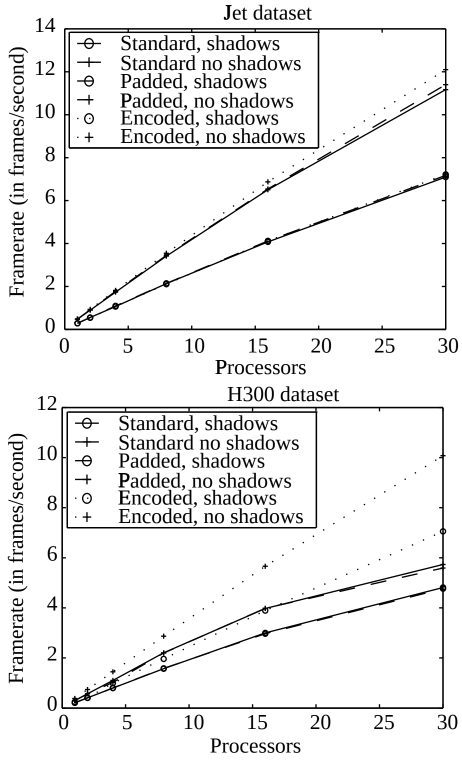

Figure 4: *Frame-rate as function of the number of processors for the Jet data set. Plotted are results with and without shadows and with and without padding of voxel data to fit cache lines.*

of-core rendering, especially when the direct I/O facility of SGI computers can be used. Appending a spatial subdivision to each of these iso-surface files produces extra data, but for the datasets that we tested, this never caused data reads to become the bottleneck. As such, our data partitioning approach produces an iso-surface rendering solution that is effective, while being algorithmically simple.

The work presented in this paper is intended as a proof of concept. The grid spatial subdivision was chosen for simplicity. Because our method does not preclude the use of more sophisticated spatial subdivisions, we anticipate that more advanced spatial sorting schemes may further improve the system's performance. Finally, because the voxels in our approach are independent (i.e. their position in space is explicitly stored, rather than inferred from their place in the file), it is possible to freely reorder them. This feature could be used to group them and so regain cache coherence.

Acknowledgments

This work was supported by NSF grants 97-96136, 97-31859, 98-18344, 99-77218, 99-78099 and by the DOE AVTC/VIEWS.

References

1. BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. Fast isocontouring for improved interactivity. In *1996 Symposium on Volume Visualization* (Oct. 1996), pp. 39–46.

2. CHIANG, Y., AND SILVA, C. T. I/O optimal isosurface extraction. In *Proceedings of Visualization 1997* (Oct. 1997), pp. 293–300.

3. CHIANG, Y., SILVA, C. T., AND SCHROEDER, W. J. Interactive out-of-core isosurface extraction. In *Proceedings of Visualization 1998* (Oct. 1998), pp. 167–174.

4. CIGNONI, P., MARINO, P., MONTANI, C., PUPPO, E., AND SCOPIGNO, R. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (Apr. 1997), 158–170.

5. FAVRE, J. M. Towards efficient visualization support for single–block and multi–block datasets. In *Proceedings of Visualization'97* (October 1997), pp. 425–428.

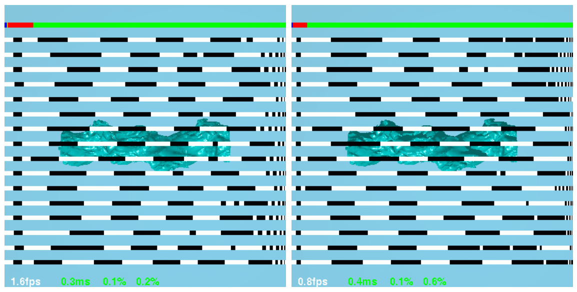

Figure 5: *A typical frame from the Jet dataset is shown. Overlaid is the fraction of time spent for each processor in various tasks. At the top is the task distribution for the display thread. The red portion shows the time required for reading data from disk. Green is idle time. The black and white blocks indicate rendering tasks for each of the 16 renderers. Left is a frame rendered without shadows, and to the right is a frame rendered with shadows.*

6. Heermann, P. D. Production visualization for the asci one teraflops machine. In *Proceedings of Visualization'98* (October 1998), pp. 459–462.

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

8. Itoh, T., Yamaguchi, Y., and Koyamada, K. Volume thinning for automatic isosurface propagation. In *Proceedings of Visualization 1996* (Oct. 1996), pp. 303–310.

9. Lanzagorta, M., Kral, M. V., Swan II, J., Spanos, G., Rosenberg, R., and Kuo, E. Metallurgical application of three dimensional visualization techniques. In *Proceedings of Visualization'98* (October 1998), pp. 487–490.

10. Livnat, Y., Hansen, C., and Johnson, C. R. Isosurface extraction for large–scale data sets. In *Proceedings of Scientific Visualization* (June 2000).

11. Livnat, Y., Shen, H.-W., and Johnson, C. R. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (1996), 73–84.

12. Lorensen, W. E. Marching through the visible man. In *Proceedings of Visualization'95* (October 1995), pp. 368–373.

13. Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics 21*, 4 (1987), 163–169.

14. Monks, C. R., Crossno, P. J., Davidson, G., Pavlakos, C., Kupfer, A., Silva, C., and Wylie, B. Three dimensional visualization of proteins in cellular interactions. In *Proceedings of Visualization'96* (October 1996), pp. 363–366.

15. Parker, S., Martin, W., Sloan, P.-P., Shirley, P., Smits, B., and Hansen, C. Interactive ray tracing. In *Symposium on Interactive 3D Computer Graphics* (April 1999).

16. Parker, S., Parker, M., Livnat, Y., Sloan, P., Hansen, C., and Shirley, P. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (July 1999), 238–250.

17. Shen, H.-W. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization 1998* (Oct. 1998), pp. 159–164.

18. Shen, H.-W., Hansen, C., Livnat, Y., and Johnson, C. R. Isosurfacing in span space with utmost efficiency (issue). In *Proceedings of Visualization 1996* (Oct. 1996), pp. 287–294.

19. Shirley, P. *Realistic Ray Tracing*. A K Peters, Natick, Massachusetts, 2000.

20. Sutton, P., and Hansen, C. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of Visualization 1999* (Oct. 1999), pp. 147–153.

21. Sutton, P., and Hansen, C. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 98–107.

22. Tiede, U., Schiemann, T., and Höhne, K. H. High quality rendering of attributed volume data. In *Proceedings of Visualization'98* (October 1998), pp. 255–262.

23. van Kreveld, M., van Oostrum, R., Bajaj, C., Pascucci, V., and Schikore, D. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry* (June 1997), pp. 212–220.

24. Weigle, C., and Banks, D. C. Extracting isovalued features in 4-dimensional scalar fields. In *1998 Symposium on Volume Visualization* (Oct. 1998), pp. 103–110.

25. Wilhelms, J., and Van Gelder, A. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (1992), 201–227.

26. Wyvill, G., McPheeters, C., and Wyvill, B. Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234.