A Case Study of Isosurface Extraction Algorithm Performance

Philip M. Sutton[1], Charles D. Hansen[1], Han-Wei Shen[2], and Dan Schikore[3]

[1] Department of Computer Science
University of Utah
Salt Lake City, UT 84112
[psutton | hansen] @cs.utah.edu

[2] Computer and Information Science Department
The Ohio State University
Columbus, OH 43210

[3] CASC
Lawrence Livermore Laboratories
Livermore, CA 94550

Abstract. Isosurface extraction is an important and useful visualization method. Over the past ten years, the field has seen numerous isosurface techniques published, leaving the user in a quandary about which one should be used. Some papers have published complexity analysis of the techniques, yet empirical evidence comparing different methods is lacking. This case study presents a comparative study of several representative isosurface extraction algorithms. It reports and analyzes empirical measurements of execution times and memory behavior for each algorithm. The results show that asymptotically optimal techniques may not be the best choice when implemented on modern computer architectures.

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 and can guide researchers to new insights. Isosurface extraction is an important technique for visualizing three-dimensional scalar fields. By exposing contours of constant value, isosurfaces provide a mechanism for understanding the structure of the scalar field. 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 [10, 17], computational fluid dynamics (CFD) [4, 5], and molecular dynamics [8, 12], have used this method effectively.

The original Marching Cubes algorithm [11, 20] for isosurface extraction examined all cells in the data set. A tremendous amount of research has focused on reducing the number of cells visited while constructing an isosurface [1, 2, 9, 15, 19]. These methods utilize auxiliary data structures to examine only those cells that contain a portion of the isosurface. While the search structures introduced by many of these methods increase the storage requirements, the acceleration gained by the isosurfacing technique offsets this overhead.

Algorithms that use data structures to accelerate isosurface extraction generally provide lower latency than simple marching methods in visualization applications. In the context of isosurface extraction, latency is defined as the elapsed time between receiving a query and returning a complete isosurface. Reducing latency greatly improves interactivity, providing researchers with a better understanding of their data. The visualization literature lacks studies or surveys comparing the latency and overall performance of the many different three-dimensional isosurface extraction algorithms. Authors of isosurfacing papers usually compare their algorithm's performance only with that of Marching Cubes. Analysis of theoretical average- and worst-case efficiency also plays a large role in the literature. Unfortunately, different implementations and different platforms make objective, empirical comparisons between algorithms difficult. Memory behavior on modern computer architectures, for example, plays a crucial role in an application's performance, but an analysis of this important factor rarely appears.

This paper presents a comparative study of several representative isosurface extraction algorithms. Each algorithm uses a different data structure to accelerate the search for cells containing an isosurface, then computes the surface using a Marching Cubes-style interpolation. This paper reports and analyzes empirical measurements of execution times and memory behavior for each of these algorithms. Section 2 describes the algorithms tested, along with implementation details of each. Section 3 describes the experiments and presents the results. Section 4 summarizes the paper and draws conclusions.

2 Isosurface Extraction Techniques

Visualization applications in many fields [5, 8, 10] use the Marching Cubes [11, 20] algorithm to extract isosurfaces from volumetric data. Marching Cubes and other algorithms use a voxel representation of the volume, considering each data point as the vertex of some geometric primitive, such as a cube or tetrahedron. These primitives, or cells, subdivide the volume and provide a useful abstraction for computing isosurfaces. The Marching Cubes algorithm tests each cell in the volume for intersection with the isosurface. By visiting cells in an order based on their position, this method can exploit the spatial coherence of the isosurface by reusing interpolation calculations along edges shared by two or more cells. However, the Marching Cubes method spends a high percentage of time visiting cells that do not contain portions of the isosurface.

Researchers have introduced a number of techniques to increase the efficiency of isosurface extraction over the linear search proposed in the Marching Cubes algorithm. These methods fall into two general categories, characterized by the criteria used to partition the cells. Geometric techniques retain the original representation of the volume and partition along divisions in the geometric mesh. Span space decomposition techniques create and manipulate abstract representations of the cells. Both geometric and span space methods can extract isosurfaces from unstructured grids as well as regular grids (see [13, 16] for geometric decomposition techniques for unstructured grids). Sections 2.1 and 2.2 describe representative methods from these two categories.

2.1 Geometric Decomposition Techniques

Wilhelms and van Gelder [19] describe the branch-on-need octree (BONO), a space-efficient variation on the traditional octree. Their data structure partitions the cells in the data based on their geometric positions. Extreme values (minimums and maximums) propagate up the tree during construction, enabling the extraction phase to prune branches of the tree. The extraction algorithm traverses only those nodes whose values span the isovalue, i.e. those with $minvalue < isovalue < maxvalue$. The branch-on-need strategy partitions the volume such that the "lower" subdivision in each direction covers the largest possible power of two cells. This results in fewer nodes when the dimensions of the volume do not equal powers of two, making the tree traversal more efficient. Leaf nodes in the branch-on-need octree generally reference eight cells (nodes may reference fewer cells along the edges of the volume). This greatly reduces the memory required, as one pair of extreme values covers eight cells. In the original paper, a hash table of edges was used to exploit spatial coherence. After the initial interpolation of a point along an edge, cells that share that edge access its hash entry to avoid recomputing the interpolation.

Another technique involves propagating the isosurface from a set of seed cells. This method combines aspects of both geometric decomposition techniques and span space algorithms[1]. A seed set must contain at least one cell per connected component of each isosurface. The algorithm groups seed cells into a hierarchical search structure, then traverses that structure to find all seeds that intersect the current isosurface. Construction of the isosurface begins at these seeds and propagates through neighboring cells using adjacency and intersection information. The difficult portion of the surface propagation algorithm lies in locating and selecting the seed cells. Itoh et al. [6, 7] find the local extremum points in the data and connect them with a graph in the spatial domain. The seed set consists of the cells containing extremum points, plus all cells intersected by the arcs of the graph and some cells along the boundaries if the volume has "holes". A thinning algorithm, commonly used in image processing, can then generate a skeleton of the seed set that connects all extremum points, yet contains fewer cells. van Kreveld et al. [18] also use a graph of local extremum points, but add saddle points to create a contour tree. Bajaj et al. [1] use set theory to find seed cells and a segment tree to organize and traverse them. Both structured and unstructured meshes can utilize these techniques, which theoretically provide near-optimal worst-case time complexity. However, noisy data may disturb the complicated seed set construction process. Measurement data such as MRI and CT scans can cause these algorithms to produce large numbers of seed cells, causing slower preprocessing time.

[1] Seed sets contain aspects of span space techniques, but surface propagation requires information about the structure of the volume, establishing it as an inherently geometric technique.

2.2 Span Space Decomposition Techniques

Span space techniques partition cells based on their extreme values. Livnat et al. [9] introduce the span space, where each cell maps to a point in 2D space. The cell's minimum value defines the x-coordinate of the point, and the maximum value determines the y-coordinate. All points in span space lie above the $y = x$ line. For a given isovalue v, the points representing cells which intersect the isosurface have $y \geq v$ and $x \leq v$. The NOISE algorithm described in [9] overlays a kd-tree on the points. This structure organizes the points such that during traversal, the algorithm needs to test only one of the two extreme values at each node in the tree. The authors use a pointerless representation of the kd-tree to avoid the additional overhead of pointer traversal. Constructing this array involves sorting the cells based on their extreme values. Sorting in a preprocess minimizes the effect on isosurface extraction performance.

Shen et al. [15] use a lattice subdivision of span space in their ISSUE algorithm. The user defines a lattice resolution L and the algorithm divides the span space points into one of the $L \times L$ lattice elements. Given an isovalue v, the ISSUE method assigns each lattice element to one of five categories. The algorithm trivially excludes cells in region 1 and trivially includes those in region 2. Cells in region 3 require a test against their maximum value, those in region 4 require a test against their minimum. Only region 5 requires a full min-max search of its cells. Shen proposed a modified version of the ISSUE algorithm [14] which creates search structures only at the lattice elements along the diagonal region 5 and coalesces lattice elements at other regions into sorted linear arrays. These modifications are implemented by building a kd-tree for each lattice element along the $y = x$ line, since only these elements may fall into the region 5 classification. This method simplifies and accelerates the search phase of isosurface extraction, but this acceleration comes at the cost of a higher memory requirement than the NOISE algorithm. The lattice structure itself requires additional memory and by creating a search structure in lattice elements that may require a full min-max search, further memory overhead is introduced. However, simple division of lattice elements among parallel processors makes this algorithm easily parallelizable. The authors report near-linear speedups using this parallel algorithm [15].

The Interval Tree technique introduced by Cignoni et al. [2, 3] guarantees that the worst-case efficiency is asymptotically optimal. This algorithm groups cells, represented by the intervals defined by their extreme values, at the nodes of a balanced binary tree. Each node contains two lists, one sorted in ascending order of cell minima, the other sorted in descending order of cell maxima. For any isovalue query, the algorithm traverses at most one branch from a node after scanning through one of the lists. The number of nodes created depends on the number of distinct interval extremes, usually much smaller than the number of cells in the volume. The memory requirements for representing intervals and for the two lists at each node dominate. The authors propose improvements specific to the underlying geometry (structured or unstructured mesh). A hash table of edges exploits spatial coherence in unstructured meshes, while regular grids can utilize a form of local surface propagation.

3 Experimental Results

Both geometric and span space techniques accelerate isosurface extraction by limiting the number of cells examined. This acceleration is usually described in terms of average- and worst-case algorithm complexity. The analysis of asymptotic complexity given by various authors [1, 2, 9, 15] shows that in the limit, the Interval Tree [2, 3] and seed set [1] algorithms guarantee worse-case optimal efficiency while the NOISE [9] algorithm provides near-optimal complexity. However, no quantitative performance comparison between the different algorithms exists, since most authors compare their technique only with Marching Cubes. This section describes the comparative performance of various three-dimensional isosurface extraction techniques, each implemented and tested using the same hardware and software framework. Marching Cubes [11, 20], the branch-on-need octree (BONO) [19], and a surface propagation algorithm using seed sets [1] represent the geometric decomposition techniques, while NOISE [9], ISSUE [15], and the Interval Tree [2, 3] represent span space algorithms. The NOISE implementation uses a pointerless representation of the kd-tree. A similar data structure could be used by the other span space algorithms, but since span space algorithms must always index cells, a certain amount of memory overhead is unavoidable. Each implementation includes the optimizations given in the paper, with the exception of techniques explicitly designed to exploit spatial coherence, such as those given in [19] and [3]. Every algorithm would benefit from these improvements, so fairness dictates their omission. Each algorithm performs both the isosurface query and triangle construction and thus are representative of execution times for the isosurface generation process.

The test data consists of both a noisy, measurement data set and a simulation data set that contains a continuous scalar field. The Head128 data set contains results from a CT scan of a human head and consists of $128 \times 128 \times 128$ points. The Rage256 data set represents a CFD simulation of the classic Rayleigh-Taylor hydrodynamic instability, in which two fluids of differing densities mix. This data set contains $256 \times 256 \times 256$ points. Figure 4 (see color plate) shows a sample isosurface from each data set. The strikingly different characteristics of these data sets can be seen in Figure 5 (see color plate). These images show the density of points in span space for each data set. The Head128 data set shows a nonuniform density, characteristic of noisy measurement, with large empty areas gradually coalescing into two hot spots. The Rage256 data set contains much more uniform data spread over the entire range of values, with very localized hot spots at the extremes.

Experimental results allow comparison of both execution time and memory system behavior from execution on a single dedicated processor of an SGI Origin 2000 with 8GB of memory. The Origin 2000 is a commonly used visualization system with a memory hierarchy typical among high end systems. Table 1 shows execution times for each algorithm using the Head128 data set. Table 2 displays results from the Rage256 data set. Results in Table 1 represent averages from ten repeated executions of ten representative isovalue queries. Table 2 represents averages from ten executions of five representative isovalue queries.

All algorithms exhibit significant speedup over Marching Cubes. Most algorithms perform similarly for the Head128 data set, as shown in Table 1. However, for the

Table 1. Experimental results from the Head128 data set.

Type	Algorithm	Average execution time (s)
Standard	Marching Cubes	2.13
Geometric	BONO	0.58
	Seed Set	0.66
Span Space	ISSUE	0.57
	Interval Tree	0.56
	NOISE	0.51

Table 2. Experimental results from the Rage256 data set.

Type	Algorithm	Average execution time (s)
Standard	Marching Cubes	17.31
Geometric	BONO	4.92
	Seed Set	6.67
Span Space	ISSUE	5.60
	Interval Tree	8.84
	NOISE	5.04

Rage256 data set, large disparities in performance exist. Table 2 shows that BONO, ISSUE, and NOISE present the largest speedups, with the Branch-on-Need Octree providing the best performance. The surface propagation code used for the seed set technique contains a large number of branches. Since the processor cannot readily predict the outcome of these branches, this algorithm performs poorly. The Interval Tree technique, although provably optimal in the limit, actually executes slower than every algorithm but Marching Cubes for the Rage256 data set. To discover the causes of this result, each implementation used the performance counters on the R10000 to track the number of clock cycles, TLB misses, and L1 and L2 cache misses during execution. Figure 6 shows the experimental results for the Head128 data set. Figure 7 shows the results obtained from the Rage256 data set. The clock cycles charted in Figures 6(a) and 7(a) correspond closely to the execution times given in Tables 1 and 2. Figure 6(b) shows that the span space techniques have a high TLB miss rate for the Head128 data set, but Figures 6(c) and 6(d) demonstrate no such distinctions in L1 and L2 cache behavior. Figures 7(b), 7(c), and 7(d) uncover the reason for the Interval Tree's low performance in the Rage256 data set — the algorithm's poor memory behavior. The large number of TLB and cache misses imply that this algorithm visits data in a different order than that adhered to by the data in memory. The severity of this difference in marching order requires processing to stall repeatedly as the operating system swaps information in and out of these hardware structures. In contrast, the BONO and Marching Cubes algorithms visit data in an order similar to that of the data in memory, since they traverse the geometric volume. These two algorithms have low instances of TLB and L2 cache misses, which incur high penalties.

To demonstrate the tradeoff between performance and storage space, Figure 8 shows the amount of memory overhead required by each algorithm for both test data sets. These figures do not include the memory required to store the data set, nor the mem-

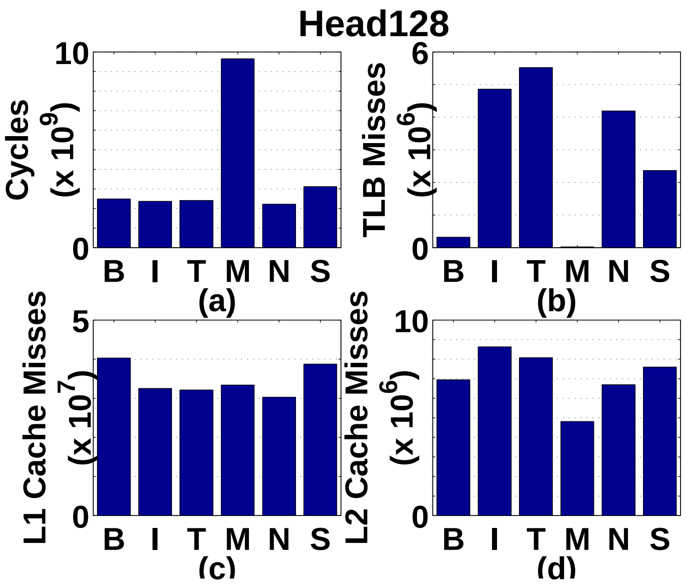

Fig. 6. Experimental results from the Head128 data set. B = BONO, I = ISSUE, T = Interval Tree, M = Marching Cubes, N = NOISE, S = Seed Set/Surface Propagation.

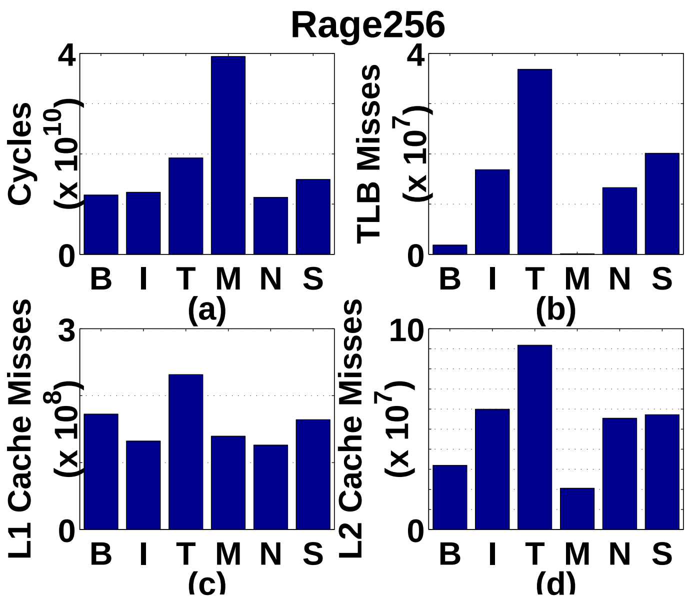

Fig. 7. Experimental results from the Rage256 data set. B = BONO, I = ISSUE, T = Interval Tree, M = Marching Cubes, N = NOISE, S = Seed Set/Surface Propagation.

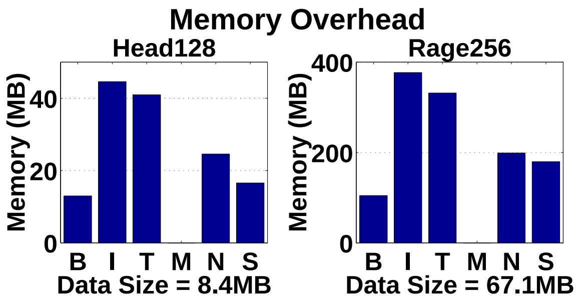

Fig. 8. Memory overhead requirements for the Head128 and Rage256 data sets. B = BONO, I = ISSUE, T = Interval Tree, M = Marching Cubes, N = NOISE, S = Seed Set/Surface Propagation.

ory consumed by storing the triangles that compose the isosurface, since all algorithms require this memory. BONO uses the least amount of additional memory, which is to be expected since it retains the original representation of the data volume. The seed set method has nearly as much memory overhead as the NOISE algorithm because a extrema data structure is used. In these implementations, the ISSUE and Interval Tree algorithms consume large amounts of memory, approximately five times the amount required to store the original data. In fairness, this is due to implementation details. Both ISSUE and the Interval tree could use a pointerless data structure which would reduce the amount of memory overhead. Also, the span space methods as implemented use indices to individual cells while BONO indexes eight cell simultaneously. An interesting comparison would be to raise the index to groups of cells rather than individual cells.

These experiments show that data structures increase the memory required to extract isosurfaces, but allow the computation to execute more quickly than in a simple marching method. However, algorithms that consume larger amounts of memory do not necessarily yield better performance.

4 Conclusion

The comparison of multiple isosurface extraction algorithms has not been previously available. This case study performs such a comparison with several data sets and attempts to show the empirical performance differences on both measurement data and simulation data. Based on the data structures required, the large constant for provably optimal algorithms is amplified by modern computer architectures where cache and page misses induce large performance penalties. With such architectures being the prevalent compute platform, the theoretical gains do not appear in practice.

While this case study relates a relatively fair comparison of different isosurface techniques, several enhancements can be made. Most importantly, this comparison should be repeated for more data sets and on different compute platforms. This would better demonstrate the variations caused by diverse data sets and by other memory hierarchy designs.

The span space methods could benefit from indexing to groups of cells, as the BONO algorithm does, rather than to single voxels. The implementation of the NOISE algorithm uses a pointerless data structure while the BONO, ISSUE and Interval Tree implementations do not. These algorithms could use pointerless data structures which would improve their memory overhead and performance. The memory overhead for ISSUE and the Interval Tree would still be higher than NOISE due to the required data structures.

This case study points out the value of performing empirical comparisons of published algorithms. Such comparisons should not be limited to isosurface techniques but should also be performed for other visualization algorithms to examine the practicality of such techniques.

5 Acknowledgements

The authors would like to thank the reviewers for their insightful comments which improved the final version of this paper. This work was supported in part by the C-SAFE DOE ASCI Alliance Center, the DOE Advanced Visualization Technology Center (AVTC), and a grant from LLNL. A portion of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48. The Rage data set is courtesy of Robert Weaver (Los Alamos National Laboratory).

References

1. Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. Fast isocontouring for improved interactivity. In *1996 Symposium on Volume Visualization*, pages 39–46. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1996.
2. P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *1996 Symposium on Volume Visualization*, pages 31–38. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1996.
3. Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.
4. Jean M. Favre. Towards efficient visualization support for single-block and multi-block datasets. In Roni Yagel and Hans Hagen, editors, *Proceedings of Visualization 1997*, pages 423–428. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1997.
5. Philip D. Heermann. Production visualization for the asci one teraflops machine. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization 1998*, pages 459–462. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1998.
6. Takayuki Itoh and Koji Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, December 1995.
7. Takayuki Itoh, Yasushi Yamaguchi, and Koji Koyamada. Volume thinning for automatic isosurface propagation. In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of Visualization 1996*, pages 303–310. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1996.

8. Marco Lanzagorta, Milo V. Kral, J.Edward Swan II, George Spanos, Rob Rosenberg, and Eddy Kuo. Three-dimensional visualization of microstructures. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization 1998*, pages 487–490. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1998.
9. Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.
10. William E. Lorensen. Marching through the visible man. In Gregory M. Nielson and Deborah Silver, editors, *Proceedings of Visualization 1995*, pages 368–373. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1995.
11. William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics*, volume 21, pages 163–169. ACM SIGGRAPH, ACM SIGGRAPH, July 1987.
12. Colin R. F. Monks, Patricia J. Crossno, George Davidson, Constantine Pavlakos, Abraham Kupfer, Cláudio Silva, and Brian Wylie. Three dimensional visualization of proteins in cellular interactions. In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of Visualization 1996*, pages 363–366. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1996.
13. S. Parker, M. Parker, Y. Livnat, P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, July 1999.
14. Han-Wei Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization 1998*, pages 159–164. IEEE Computer Society, ACM Press, New York, NY, October 1998.
15. Han-Wei Shen, Charles D. Hansen, Yarden Livnat, and Christopher R. Johnson. Isosurfacing in span space with utmost efficiency (issue). In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of Visualization 1996*, pages 287–294. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1996.
16. Philip M. Sutton and Charles D. Hansen. Accelerated isosurface extraction in time-varying fields. *Submitted to IEEE Transactions on Visualization and Computer Graphics*, 2000.
17. Ulf Tiede, Thomas Schiemann, and Karl Heinz Höhne. High quality rendering of attributed volume data. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization 1998*, pages 255–262. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, October 1998.
18. Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 212–220. ACM Special Interest Groups for Graphics and Algorithms and Computation Theory, ACM Press, New York, NY, June 1997.
19. Jane Wilhelms and Allen van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.
20. Geoff Wyvill, Craig McPheeters, and Brain Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

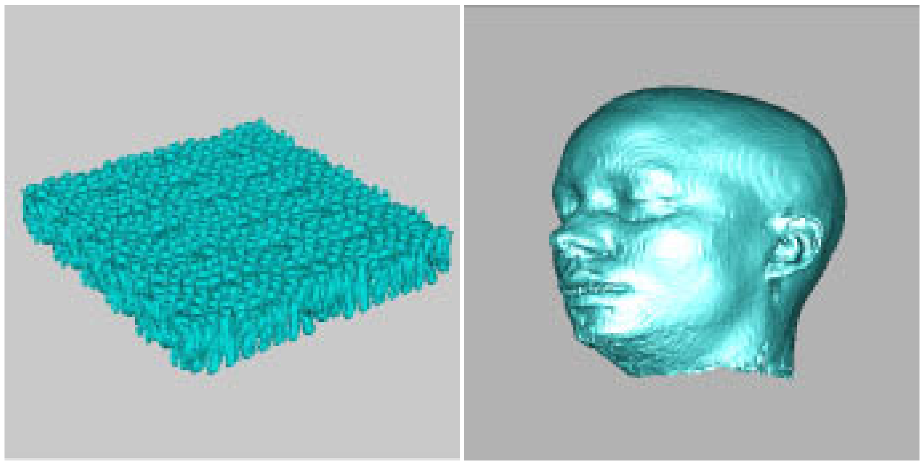

Fig. 4. Isosurfaces from simulation and measurement sources. (a) depicts an isosurface from the RAGE computational fluid dynamics simulation, showing the bubbles formed by Rayleigh-Taylor instability. (b) shows the skin isosurface from the volumetric data set produced by a CT scan.

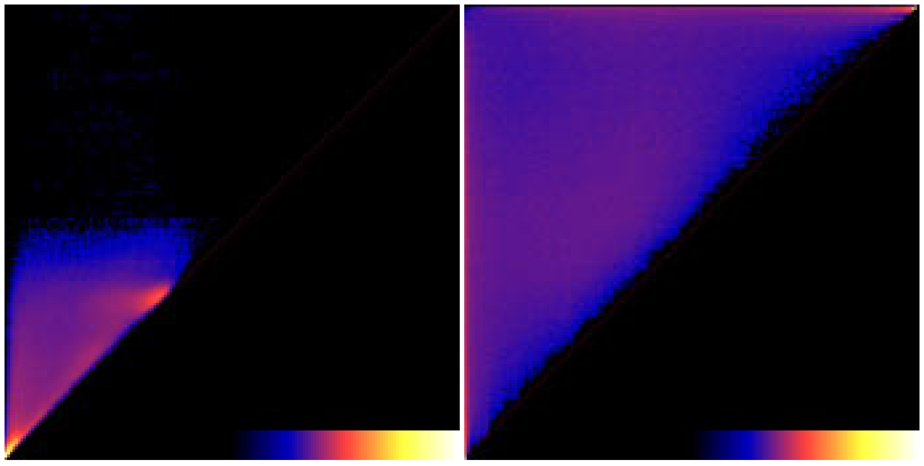

Fig. 5. Span space representations of the (a) Head128 and (b) Rage256 data sets. Blue represents low density of points, white represents high density.