Scalable Isosurface Visualization of Massive Datasets on COTS* Clusters

Xiaoyu Zhang† Chandrajit Bajaj† William Blanke‡

† Department of Computer Sciences and Center for Computational Visualization, TICAM
‡ Department of Electrical and Computer Engineering
University of Texas at Austin
http://www.ticam.utexas.edu/CCV

Abstract

Our scalable isosurface visualization solution on a commodity off-the-shelf cluster is an end-to-end parallel and progressive platform, from the initial data access to the final display. In this paper we focus on the back end scalability by introducing a fully parallel and out-of-core isosurface extraction algorithm. It partitions the volume data according to its workload spectrum for load balancing and creates an I/O-optimal external interval tree to minimize the number of I/O operations of loading large data from disk. It achieves scalability by using both parallel processing and parallel disks. Interactive browsing of extracted isosurfaces is made possible by using parallel isosurface extraction and rendering in conjunction with a new specialized piece of image compositing hardware called the Metabuffer. We also describe an isosurface compression scheme that is efficient for isosurface processing.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Parallel Processing; I.3.8 [Computer Graphics]: Applications

Keywords: Parallel Rendering, Metabuffer, Multi-resolution, Progressive mesh, Parallel and Out-of-core Isocontouring

1 Introduction

Today tomographic imaging and computer simulations are increasingly yielding massive datasets. Interactive and exploratory visualization has rapidly become an indispensable tool to determine and browse regions of interest within volumetric imaging data, and verify and validate the results of computer simulations. One paradigm of exploratory visualization is to extract multiple 2-dimensional surfaces satisfying $w(\mathbf{x}) = const$ from a given scalar field $w(\mathbf{x})$, $x \in \mathbf{R}^3$, and render it at interactive frame rate ($30HZ$). This interactive and exploratory visualization technique s popularly known as isocontour visualization.

Isocontour visualization for extremely large datasets poses challenging problems for both computation and rendering with guaranteed frame rates. First, large isosurfaces are to be extracted in time-critical manner from those large datasets, whose sizes are from multi-gigabytes to terabytes. As the size of the input data increases, isocontouring algorithms necessarily need to be executed out-of-core and/or on parallel machines for both efficiency and data accessibility. Second, the interactive aspect of the isocontour visualization demands that the scene is rendered quickly in order to provide responsive feedback to the user. In some cases, the detail allowed by a single high performance monitor may not be adequate for the resolution required. An even more common problem is that the dataset itself may be too large to store and render on a single machine. Third, the extracted isosurface may need to be transmitted from the computational servers to the rendering servers via the network, if the computational and rendering servers do not coexist on the same machines. It may also need to be saved on disk for future studies. Compact representation of the isosurfaces should be used in order to meet the time limit of data transmission and save disk space.

Related Work: Hansen and Hinker describe parallel methods for isosurface extraction on SIMD machines [17]. Ellsiepen describes a parallel isosurfacing method for FEM data by dynamically distributing working blocks to a number of connected workstations [13]. Shen, Hansen, Livnat and Johnson implement a parallel algorithm by partitioning load in the span space [28]. Parker et al. present a parallel isosurface rendering algorithm using ray tracing [24]. Chiang and Silva give an implementation of out-of-core isocontouring using the I/O optimal external interval tree on a single processor [8,9]. Bajaj et al. use range partition to reduce the size of data that are loaded for given isocontour queries and balance the load within a range partition [3]. In this paper, we propose and implement a parallel and out-of-core isocontouring algorithm using parallel processors and parallel I/O, which would be fully scalable to arbitrarily large datasets.

Many research groups have recently studied the problem of using fast improving PC graphics cards for parallel rendering [12,18,20,26,27]. Schneider analyzes the suitability of PCs for parallel rendering for four parallel polygon rendering scenarios: rendering of single and multiple frames on symmetric multiprocessors and clusters [27]. Samanta et al. discuss various load balancing schemes for a multi-projector rendering system driven by multiple PCs [26]. Heirich and Moll demonstrate how to build a scalable image composition system using off-the-shelf components [18]. In general, most parallel rendering methods can be classified based on where data is sorted from object-space to image-space [22].

In the sort-first approach, the display space is broken into a number of non-overlapping display regions, which can vary in size and shape. Sort-first methods may suffer from load imbalance in both the geometric processing and rasterization if polygons are not evenly distributed across the screen partitions, because polygons are assigned to the rendering process before geometric processing. The Princeton University SHRIMP project [26] uses the sort-first approach to balance the load of multiple PC graphical workstations. The sort-middle approach distributes transformed primitives

*Commodity off-the-shelf

instead of polygons to the graphics pipes responsible for the screen partitions.

The sort-last approach is also known as image composition. Each rendering process performs both geometric processing and rasterization independent of all other rendering processes. Local images rendered on the rendering processes are composited together to form the final image. The sort-last method makes the load balancing problem easier since screen space constraints are removed. However, compositing hardware is needed to combine the output of the various processors into a single correct picture. Such approaches have been used since the 60's in single-display systems [15, 23], and more recent work includes [14, 18].

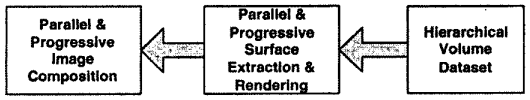


Figure 1: A schematic drawing showing the three stages of scalable isosurface visualization pipeline.

Our solution to the image compositing problem is the Metabuffer, whose architecture is shown in figure 3. This is a sort-last multi-display image compositing system with several unique features such as multi-resolution and antialiasing [6]. A very similar project, though currently without stressing multi-resolution support, exists at Stanford University and is called Lightening-2 [19]. The Metabuffer hardware supports a scalable number of PCs and an independently scalable number of displays–there is no *a priori* correspondence between the number of renderers and the number of displays to be used. It also allows any renderer to be responsible for any axis-aligned rectangular viewport within the global display space at each frame. Such viewports can be modified on a frame-by-frame basis, can overlap the boundaries of display tiles and each other arbitrarily, and can vary in size up to the size of the global display space. Thus each machine in the network is given equal access to all parts of the display space, and the overall screen is treated as a uniform display space, that is, as though it were driven via a single, large framebuffer, hence the name Metabuffer.

While compression is important for handling large datasets, one possible approach to the isosurface compression problem is to first extract the isosurface into triangular meshes and then apply to it one of the surface compression algorithms [4, 10, 11, 16, 29, 30]. Although it is conceptually simple, this method has several disadvantages. First the rendering servers have to wait until the computational servers finish both isosurface extraction and compression. The compression of the surface often takes a very long time, especially true for the very large surfaces extracted from large datasets, which contradicts the goal of using compression for real-time rendering. Furthermore, isosurfaces have the property that each vertex is an intersection point with one unique edge of the 3D volume. An algorithm designed specifically for an isosurface may get a better compression ratio. In this paper we describe an index and function value encoding scheme that compresses the isosurface and allows streaming the compressed data to the rendering servers incrementally either during the surface extraction or from cache on disk. We also show the method achieves a better compression ratio than some general purpose surface compression algorithms.

Main Results: With all three aforementioned techniques combined, parallel and out-of-core computation, parallel rendering and compression, it is possible to obtain a fully scalable system for interactive isosurface visualization across multiple isovalues and from different viewpoints. In this paper we focus on the back end parallel and out-of-core isosurface extraction, leaving the details of progressive image composition using the Metabuffer and its performance results to a separate paper [6]. The rest of our paper is organized as follows: Section 2 briefly describes the architecture of our framework for scalable isosurface visualization. Section 3 gives the details of our parallel and out-of-core isocontouring algorithm. Section 3.2 provides the detail of our isosurface compression method and compares its results to that of another surface compression algorithm. Section 4 gives the performance of our parallel implementation on a COTS cluster.

2 Framework

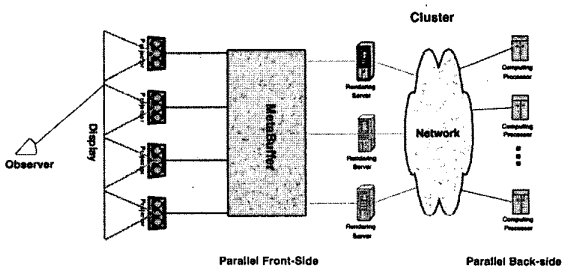


Figure 2: Our system architecture for scalable isosurface visualization. The parallel back-side accomplishes progressive surface extraction and rendering while the parallel front-side composites and displays progressively.

2.1 Pipelined Stages

We can think of the process of scalable time-critical isosurface visualization of massive data as a parallel and progressive stream from back to front as shown in figure 1. Triangle streams are generated by the back-end nodes by progressively extracting the isosurfaces. The triangle stream from the extraction node will be rendered by the middle parallel rendering servers. All rendered images will then be composited by the Metabuffer to display on a multi-tiled screen. The process of compositing images from multiple rendering servers to multiple displays using the Metabuffer is called parallel image composition. Given the required frame rate, the refresh time between two frames needs to be shared among these three stages: triangle extraction, rendering and image composition. While the image composition time taken by the Metabuffer is constant, how much external time left in the frame interval determines what resolution of triangles will be extracted and rendered.

2.2 Hierarchical Volumetric Data

Due to the large size of the massive dataset, it is extremely time-consuming or even impossible to do isosurface extraction on a single processor. In order to scale to very large datasets, we use a computational back end consisting of both parallel processors and parallel disks. Large datasets are partitioned among the parallel processors in a load-balanced way and stored hierarchically on disk for efficient I/O access. The hierarchical volume data can be stored on disk in compressed form [5]. Figure 2 illustrates the parallel end to end framework for scalable isosurface visualization on a commodity off-the-shelf cluster. The parallel back-side provides multi-resolution isosurfaces extracted from the volume dataset to satisfy the time limit. This stage is governed by the parallel and progressive triangle extraction algorithms. We will describe in detail our parallel triangle extraction algorithms in section 3.1. Producing multi-resolution representation of the data at the back-side is essential for the time critical rendering of massive data. When the user changes

viewing parameters frequently, coarser representations of the data are rendered in order to give the user responsive feedback. Only when the user chooses a certain viewing position and some interesting isovalue, are the details of the progressive mesh or isocontour streamed for rendering in order to produce higher resolution image. To reduce the time of data transmission over the network, the extracted mesh may be communicated to rendering servers in compressed format. Although the rendering servers might be on the same set of machines as the triangle extraction processes, the progressive triangle mesh extraction processes can in general scale independently of the number of parallel rendering servers.

2.3 The Metabuffer

One novel feature of the framework is the parallel rendering and image composition system that is able to render the given scene in the least latency. The parallel front-side is built around the Metabuffer [6], which is custom hardware built from commodity PC components. The Metabuffer hardware provides several unique advantages to assist in rendering large surface in parallel, such as arbitrarily located and overlapped viewports and multi-resolution. Each rendering server in figure 2 is mapped to a viewport on the screen space. A very important problem in the parallel rendering is how to position those viewports and partition the mesh such that each rendering server has approximately an equal amount of work.

The Metabuffer allows the number of rendering servers to scale independently from the the number of display tiles. Since the Metabuffer allows the viewports to be located anywhere within the total display space and overlap each other, it is possible to achieve a much higher degree of load balancing. Since the viewports can vary in size, the system supports multi-resolution rendering, for instance allowing a single machine to render a background at low resolution while other machines render foreground objects at much higher resolution.

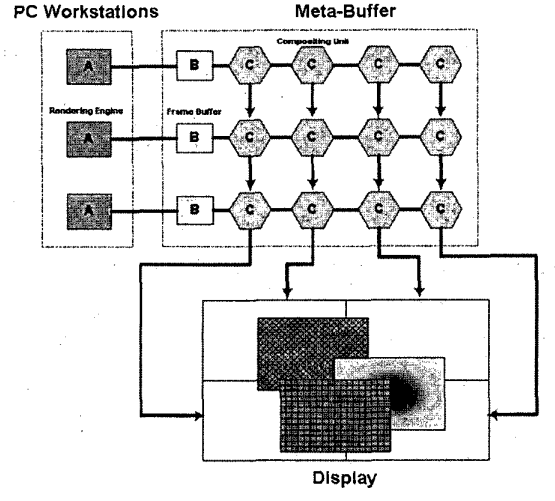


Figure 3: Our Metabuffer architecture, where A represents a rendering engine, B is an on-board frame buffer, and C represents a composing unit.

Given the progressivity from the triangle extraction stage to final image composition stage in our framework and the fact that each stage is fully parallelizable, we can achieve a truly scalable rendering of large isosurfaces.

3 Parallel Algorithms

In this section we will discuss in more detail the parallel and out-of-core isocontouring algorithm and the isosurface compression algorithm mentioned in section 2 that enables the framework for scalable time-critical visualization of massive datasets. First we discuss the scalable algorithm of extracting progressive isosurfaces from large volume datasets.

3.1 Scalable Isosurface Extraction

A scalable data analysis and visualization application must take data processing, I/O, network and rendering all into consideration. Specifically for the isocontour extraction of large volume data, it should have load balanced parallel computation for fast surface extraction, out-of-core computation to scale to datasets bigger than the size of total main memory, and parallel I/O to avoid the bottleneck of accessing massive data on disk.

We can model such a system that combines parallel processing and parallel disk access with a model called the BSP-Disk model [25]. The BSP-Disk model consists of P interconnected processors, each of which may have a local memory and disk. The BSP-Disk model combines the features of the BSP parallel computation model [31] and the PDM [32] parallel disk model. It is a distributed memory parallel computation model, while each processor can access its local disk in parallel. The BSP-Disk model very closely describes the characteristics of a PC cluster, where each node has its own CPU, local memory and local disk. The parameters of the BSP-Disk model are as follows:

- N : Total number of atomic units of the problem.
- M : Total number of atomic units that can fit in the one processor's main memory.
- P : Number of processors.
- D : Number of Disks.
- B : Number of atomic units that fits in one disk block.
- A : Time to read or write one disk block on the local disk.
- L : minimal synchronization time of the BSP model.
- g : gap parameter of the BSP model which characterizes the communication bandwidth.

In the case of large datasets, we have the design conditions $N > P \cdot M$ and $M \gg B$. In contrast to the PDM model where a single processor has equal access to all parallel disks, the disks in the BSP-Disk model are associated to different processors as their local disks. One very important example is that every processor has one associated local disk ($P = D$), as for the case of a PC cluster. More generally $\frac{P}{D}$ processors can be assigned to every disk. Extra communication time is required when one processor needs to access the data on a remote disk. In the BSP-Disk model one disk block can be loaded from every disk into memory in one parallel I/O because each disk can be accessed independently. Thus up to D disk blocks can be read into main memory in one parallel I/O. In other words, the data access time is shared among the D disks.

The algorithms designed for the BSP-Disk model would consider all three parts of the time for a real parallel and out-of-core algorithm, local computation, local disk I/O and communication. Therefore the time of the isosurface extraction can be written as $T = \max_P(T_w + T_{io} + T_c)$, where T_w is the time for local computation, T_{io} is the time for local disk access and T_c is the time for communication. T_w and T_c are to be measured using the BSP

model and T_{io} is measured as $A \times N_d$, where N_d is the number of parallel I/O operations. The objective of our isocontouring algorithm for large datasets is to speedup the computation by distributing the load to multiple processors, minimize the number of parallel I/Os, and minimize inter-processor communication (such as the remote disk accesses). These factors do not always play together for each other. We must make tradeoffs according to the real system parameters. To minimize the number of parallel I/O's, we need to load only those portions of dataset that contribute to the final result, and distribute data to the disks such that data is loaded evenly from the D local disks. Minimizing the local computation time requires good load balance among the processors. Finally minimizing communication means minimal data redistribution and remote data access during computation. We choose the approach of static data partitioning that minimizes the communication during isocontour extraction because data communication is currently the least scalable factor when data size gets larger and new processors and disks can be easily added. The methods of allowing processors to dynamically steal data or work from remote disks are for future study.

In order to achieve good performance for a static workload allocation method for parallel computation, data must be partitioned carefully such that each processor has approximately the same amount of work. Furthermore data must be distributed among the D disks such that the number of parallel I/O is minimal. Fortunately the work load of isocontouring computation on a processor is proportional to the size of data it loads from its local disk. We use a data partition method similar to that in [3] to distribute the data according the contour spectrum [2]. The contour spectrum provides a work load histogram for different isovalue queries. We use the number of triangles in the isosurface as the Y axis of the contour spectrum. In the ideal data partition, each processor does the same amount of work for every value of the parameter v. While [3] tries to break data into range partitions that can each fit into main memory, we here partition the whole range of data according to the workload histogram. The partitioning of the whole range avoids the problem of data duplication among different range partitions. After data is partitioned, an I/O-optimal interval tree [1] is built as index structure for the data on each disk in a way similar to [9]. The external memory interval tree has optimal $O(\log N + K)$ I/O operations for stabbing queries, where K is the number of active cells.

Cell is the minimum unit of the volume dataset. Function values are defined on the *vertices* of the cells and usually tri-linearly interpolated inside cell. Ideally we can use the granularity of cell for the data partition and build the external interval tree for all the cells. However as shown in [9], it is very storage inefficient to build an external index data structure using the unit of cell because of the high overhead of data duplication among cells. Instead we use the atomic unit of block, which is usually a $3D$ rectangular slab of adjacent cells. Although some extra cells maybe be loaded because of the larger granularity, block provides the possibility of tradeoff between disk space and I/O efficiency. In our implementation, we choose the size of block as the disk block size such that one block can be loaded in one I/O operation. Since we use a block as the unit of data partition and accessing, it is possible to extract isosurface progressively at different resolution to give user the basic shape of the surface with minimum delay. Figure 4 shows an isosurface (isovalue 1200) of the Male MRI data is extracted and rendered at three different resolutions.

At the first stage of our algorithm, we partition the data among multiple computational nodes and build the external interval tree as the index structure for the blocks on each disk. Here we have assumed each node has one processor and associated local disk. The parallel and out-of-core isocontouring algorithm consists of following steps.

1. **Break the data into blocks.** At the start of the data distribution, there is an initial distribution of data among those disks, which is not load balanced for isosurface query. For example, we can just break the big volume into slabs and each disk contains one slab of the data. First we divide the dataset into blocks, each of which is in the same order of the disk block size. With each block, we store all the information for doing isocontour extraction on the block, including the function values of all vertices of the block and the geometric information such as the dimensions, the origin and the span of the block. We also store the range interval of the block explicitly. We construct a triangular matrix in range space to help the data partition. One axis of the matrix represents the entire range of function values which is divided into a specified number of buckets. The second axis is the number of buckets that the range of a block spans. The minimum function value of a block and the number of buckets spanned by its range determine which matrix elements it belongs to. For instance, if a block has range $[x, y]$ and the bucket interval size is a, the block would belong to the $[\lceil \frac{(x-min)}{a} \rceil, \lceil \frac{(y-x)}{a} \rceil]$ matrix element. Thus blocks falling into the same matrix element have similar span in the range space. We categorize the blocks according to which range space triangular matrix element it falls into.

2. **Redistribution of data blocks.** The blocks falling into the same matrix element are then assigned equally to the processors. This process is repeated for all the matrix elements. At the beginning of redistribution, the processors broadcast to each other the number of blocks in the matrix element and the statistical information about the blocks. Then each processor run the same algorithm to determine to which processors it needs to send blocks and from which processors it will receive blocks. Here first we try to make each processor have the same number of blocks of the matrix element. Further consideration is given to minimize the number of blocks to be communicated.

3. **Build External Interval Tree as the search data structure.** During the redistribution of blocks, every block assigned to one processor is given a unique ID and added to a single file that contains all the blocks assigned to the processor. Every block is stored from the disk block boundary and its location is easily decided from its ID. While the block is written to the block file, its range interval associated with its ID is stored in an interval file, which will be used to build the index structure to the data blocks. Since even the index structure may not fit into the main memory while the data size gets larger, we choose the external interval tree [1, 8] for the full scalability of our algorithm.

4. **Isocontour Query Processing.** For an isovalue query, we first search the external interval tree to find all blocks whose range intersects the isovalue. Such stabbing query on external interval tree is simple and I/O optimal [1]. The isosurface is then extracted from those intersected blocks. The surface can be extracted in compressed format, as we describe later. To give the user quick response, the extracted surface is streamed to the parallel rendering servers which may reside on the same machine. The rendered images from the parallel rendering engines are then composed by the Metabuffer. Since we use the block as the atomic unit of data distribution and accessing, the surface can be extracted at different resolution to meet the different time requirement. The user can get a quick view of the shape of the isosurface before he can pick an interesting isovalue and study it in more detail.

This algorithm provides a load balanced and fully out-of-core method for isocontour extraction, which would be scalable to arbi-

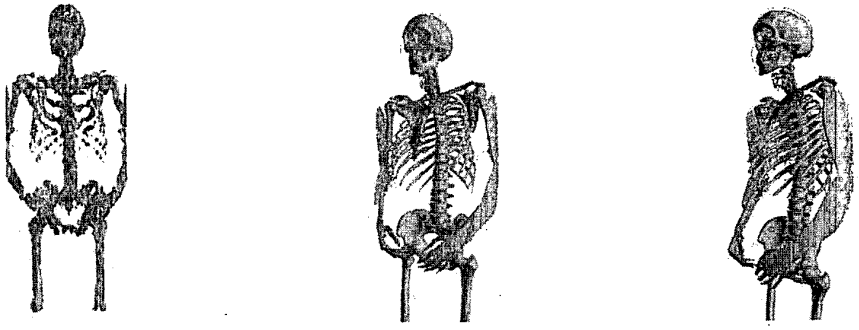


Figure 4: An isosurface for the Male MRI data (isovalue = 1200) is progressively extracted and rendered at different resolutions and from various viewpoints. The leftmost isosurface has 24,084 triangles; the center isosurface has 651,234 triangles, and the rightmost isosurface has 6,442,810 triangles.

trary large dataset. The extracted surface are rendered by the parallel rendering servers and finally composited by the Metabuffer. The surface extraction process and rending process can be run in parallel. Figure 5 shows one isosurface extracted and rendered by eight processors. Since we have used the diagram of triangle numbers to determine the partition of dataset, each processor will generate approximately equal number of triangles and balance the load of rendering.

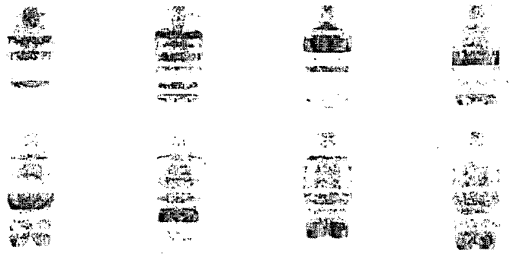


Figure 5: Individual portions of an isosurface (isovalue = 800) extracted from the visible male MRI data with eight computing processors and rendered by eight rendering servers. The full resolution isosurface has 9,128,798 triangles.

3.2 Isosurface Compression

The isosurfaces extracted on the computational servers need to be rendered by the rendering servers and composited by the Metabuffer. The computational processes and rendering processes might not be present on the same machines. Therefore isosurfaces need to be transmitted from the back-end to rendering processes via the network. Furthermore we need to save and cache those large isosurfaces extracted from large dataset to examine them from different viewing directions. Compact representation of the isosurface should be used in the transmission in order to meet the time limit. We employ a method of edge index encoding to compress the isosurface. Given the function values at its 8 vertices greater or less than the isovalue p, a cell has its inside isosurface topology determined as one of the $2^8 = 256$ possible configurations, which can be further reduced according to rotational symmetry and vertex complement [21]. We can easily derive what edges of a cell are intersected by the isosurface from its index and configuration. A cell intersected by the isosurface is called a *valid cell*. Given function values on the two endpoints of the intersected edge, the point on this edge intersected by the isosurface can be computed. A vertex that is one end point of an edge intersected by the isosurface is called a *relevant vertex*. Hence the representation of the isosurface can be reduced as encoding the configurations of valid cells and the function values on the relevant vertices. We further notice that a cell configuration can be determined if we know which vertices of its 8 corners are relevant vertices. Therefore all necessary information for reconstructing the isosurface is known: the relevant vertices and valid cells, and the function values on the relevant vertices. The steps of the edge index compression algorithm are:

1. For every vertex on a slice, we set a bit $r = 1$ if it is a relevant vertex, $r = 0$ otherwise. Similarly for each cell in a layer between two slices, we set a bit $i = 1$ if it is a valid cell, $i = 0$ otherwise.
2. Encode the vertex bitmap of a slice using an entropy encoding method, such as adaptive run-length coder [7] or arithmetic coder [33].
3. Encode the function values of the relevant vertices on the slice using the second-order difference of their quantized values.
4. Encode the cell bitmap of a layer similarly.
5. Repeat upper steps until there are no more slices and layers.

Model	Value	Data Type	Orig. Size	Gen. Alg.	Edge Index
Hipip	0.0	float	1,465,862	129,068	88,804
Foot	600.	u_short	7,536,227	587,344	407,658
Foot	1500.	u_short	8,645,243	761,413	386,506
Engine	180.	u_char	3,601,040	224,893	121,504
Engine	89.	u_char	15,888,473	1,012,491	618,492

Table 1: Comparison of compressed isosurface sizes in bytes using edge index coding with a general surface compression algorithm. The general compression algorithm uses 8 bits for each coordinate per vertex. For unsigned short and char data type, we encode them directly using the predictive coder. Float values are normalized and quantized using 32 bits for the first point and 14 bits for difference.

This method has yielded a very good compression ratio to isosurface of regular 3D mesh compared to the general purpose triangular surface compression algorithms as shown in table 1, where the general algorithm is that of [4]. Furthermore, the edge index

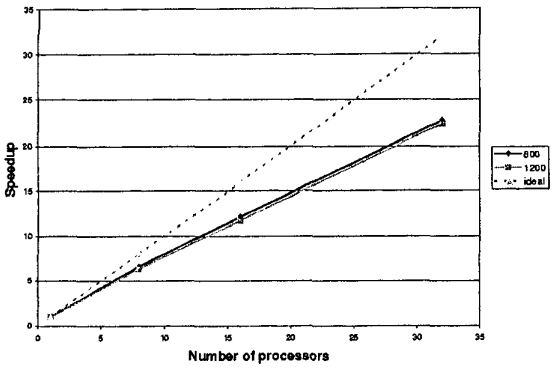


Figure 6: Speedup of isosurface extraction and rendering for two isovalues (800 and 1200) compared to the ideal case. The data used is the Visible Male MRI data .

method has the advantage of incremental encoding and decoding because two slices and one layer are necessary in main memory at any time for the encoding and decoding processes. Thus both the compression and decompression of the isosurface using edge index encoding need only a small amount of main memory and the reconstruction of the surface can start far before the whole surface transmission is finished. Furthermore the encoding of indices and function values is done during isosurface extraction, such that no expensive post-extraction compression process is necessary.

4 Implementation and Results

Here we present some experimental results of the parallel and out-of-core isocontouring algorithm and the parallel visualization framework. Our primary interest in those tests is to show the scalability of our algorithms to the size of the data set and the number of computers. Our implementation platform is a cluster of Compaq SP750 PC workstations, each of which has a 800MHZ Pentium III processor with 256M of Rambus memory, a 9GB system disk and a 18GB data disk, and an nVdia Geforce II graphics card. These machines are connected by 100Mb/s ethernet and Servernet II. Our tests use the 100Mb/s ethernet for communication. These machines run Linux (kernel 2.2.18) as the operating system and each disk block size is 4,096 bytes.

We use several datasets of different size, shown in table 2, and a different number of machines to test the performance of the algorithm. The three test datasets, whose size ranges are from $656MB$

Name	Dimension	Size
Male MRI	512 x 512 x 1252	656MB
Male cryosection	1800 x 1000 x 1878	6.6GB
Female cryosection	1600 x 1000 x 5186	16.5GB

Table 2: The sizes of our test imaging datasets.

to more than $16GB$, are all from the visible human project of the National Library of Medicine. Each of those datasets is too large for a single PC to handle in its main memory. While all those datasets are from medical imaging, our algorithm can be certainly applied to other types of data, for example those from large scale simulation.

The first test data is the Male MRI dataset. The surface extracted on one machine is rendered by the same machine and the images are then composited according their z values. Every rendering server in this configuration has the viewport of the whole display space. In this configuration every rendering server renders at the same resolution. We measure its isosurface extraction and rendering time by using from one processor to 32 processors. Being able to run efficiently on a single processor demonstrates the out-of-core property of our method. Figure 6 shows the speedup of isosurface extraction and rendering for two typical isovalues 800 and 1200, corresponding to the skin and bone respectively, for the MRI dataset. Figure 7 shows the actual time of each processor for the isovalue 800 with a different number of processors.

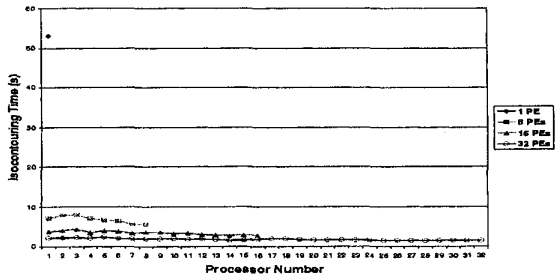


Figure 7: Individual processor time for extracting and rendering an isosurface (isovalue = 800) from the Male MRI data with 1, 8, 16 and 32 processors.

The slopes of the two speedup curves are very similar, which demonstrates the the isocontour extraction algorithm applies equally well to different isovalues.

The contour spectrums of data partitioning for the cases of 8, 16 and 32 processors are shown in figure 8. The first row of figures show the diagrams of the triangle numbers extracted by each processor for the range of isovalue from 0 to 1900, where the curves of the real experimental result in solid thin lines are compared to the ideal case in thick dashed line. The second row of figures are the actual surface extraction and rendering time for the whole range of isovalues. The similarity of the two set of curves justifies our use of the spectrum of triangle numbers as the work load diagram.

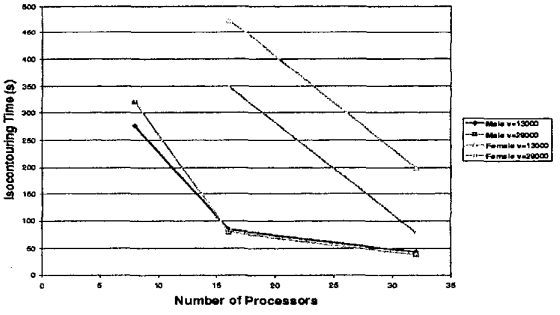
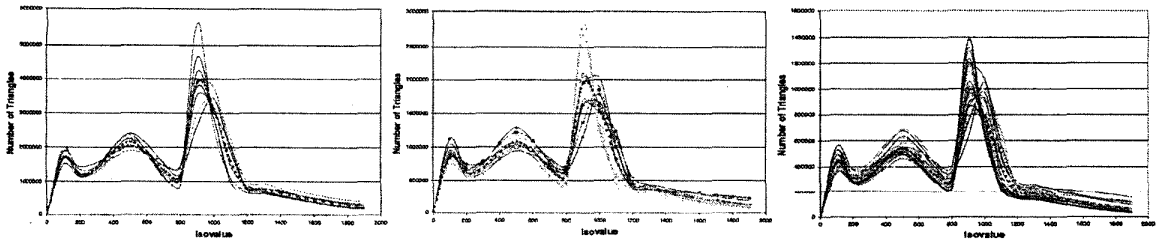


Figure 9: Isosurface extraction and rendering time for the Visible Male cryosection and Female cryosection data at two different isovalues(13000 and 29000).

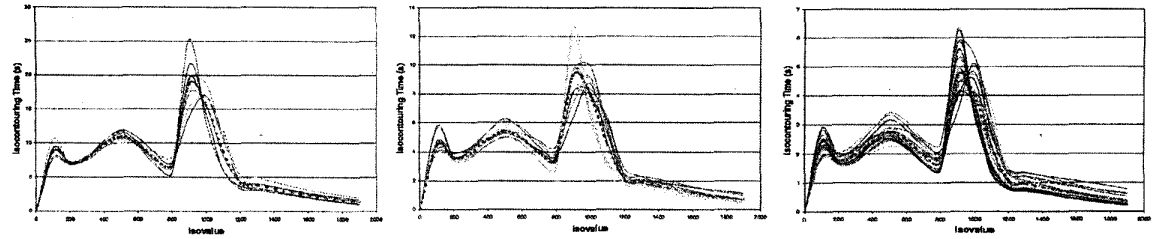
We also test our implementation on the much larger Male cryosection and Female cryosection datasets. We start at partitioning the Male cryosection data into 8 pieces and the Female cryosection data into 16 pieces, because of the $2GB$ file size limit of Linux. Figure 9 shows the time of isosurface extraction and rendering for the two datasets for two different isovalues 13000 and 29000. It gives very good speedup when the number of processors and disks



(a) Triangle distribution for 8 processors

(b) Triangle distribution for 16 processors

(c) Triangle distribution for 32 processors



(d) Isosurface extraction and rendering time for 8 processors

(e) Isosurface extraction and rendering time for 16 processors

(f) Isosurface extraction and rendering time for 32 processors

Figure 8: The histogram of triangle distribution and isosurface extraction and rendering time for the data partitioning of the Male MRI data, where the thick dashed lines represent the averaged ideal case.

increases. The sharp drop of computational time from 8 processors to 16 processors for the Male cryosection data and from 16 processors to 32 processors for the Female cryosection data is due to better operating system disk cache performance for the smaller partitions. A very large isosurface (487, 635, 342 triangles) extracted from the Female cryosection data is shown in Figure 10. While the surface is noisy due to the dataset, it demonstrates the scalability of our system to very large data and very large surface.

5 Conclusion and Future Work

In this paper we propose a scalable time-critical rendering framework of massive datastreams based around the Metabuffer image composition hardware.The time-critical rendering process can be thought of as a chain from parallel and progressive mesh generation, parallel rendering to parallel image composition. We describe in detail a scalable isocontouring algorithm by taking advantage of the parallel processors and parallel disks. It partitions the volume data according to its workload spectrum for load balancing and creates an I/O-optimal external interval tree to minimize the number of I/O operations of loading large data from disk.

There are improvements necessary for real interactivity for very large datasets. There are also the remaining problems of introducing dynamic load balancing by replicating some data on different disks and accessing data from remote disks at runtime. It is an interesting problem to see how much improvement we can achieve by choosing the right replication factor and data distribution.

Acknowledgments: This research is supported in part by grants ACI-9982297, DMS-9873326 from the National Science Foundation, a DOE-ASCI grant BD4485-MOID from Sandia National Laboratory, Lawrence Livermore National Laboratory, from grant UCSD 1018140 as part of the National Partnership for Advanced

Figure 10: Two views of an isosurface(isovalue = 29000) extracted and rendered from the full resolution Visible Female cryosection data. The isosurface contains 487, 635, 342 triangles

Computational Infrastructure, a grant BD 781 from the Texas Board of Higher Education, and a gift and cluster support from Compaq Computer Corporation.

References

[1] ARGE, L., AND VITTER, J. S. Optimal interval management in external memory. In *Proc. IEEE Foundations of Computer Science* (1996), pp. 560–569.

[2] BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. The contour spectrum. In *Proceedings of the 1997 IEEE Visualization Conference* (October 1997).

[3] BAJAJ, C., PASCUCCI, V., THOMPSON, D., AND ZHANG, X. Parallel accelerated isocontouring for out-of-core visualization. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium* (October 1999), pp. 97–104.

[4] BAJAJ, C., PASCUCCI, V., AND ZHUANG, G. Single resolution compression of arbitrary triangular meshes with prop erties. In *IEEE Data Compression Conference* (1999), pp. 247–256.

[5] BAJAJ, C., AND ZHANG, X. Streaming compressed surfaces extracted from compressed volumes. Tech. rep., University of Texas at Austin, 2000.

[6] BLANKE, W. J., FUSSELL, D. S., BAJAJ, C., AND ZHANG, X. The metabuffer: A scalable multiresolution multidisplay 3-d graphics system using commodity rendering engines. Tr2000-16, University of Texas at Austin, February 2000.

[7] BOTTOU, L., HOWARD, P., AND BENGIO, Y. The z-coder adaptive binary coder. In *Proceedings of IEEE Data Compression Conference DCC'98* (March 1998), pp. 13–22.

[8] CHIANG, Y., AND SILVA, C. T. I/O optimal isosurface extraction. In *IEEE Visualization 97* (Nov. 1997), R. Yagel and H. Hagen, Eds., IEEE, pp. 293–300.

[9] CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. Interactive out-of-core isosurface extraction. In *Proceedings of the 9th Annual IEEE Conference on Visualization (VIS-98)* (Oct. 18–23 1998), ACM Press, pp. 167–174.

[10] CHOW, M. M. Optimized geometry compression for real-time rendering. In *Proceedings of IEEE Visualization '97* (Phoenix, 1997), pp. 347–354.

[11] DEERING, M. Geometry compression. *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), 13–20.

[12] ELDRIDGE, M., IGEHY, H., AND HANRAHAN, P. Pomegranate: A fully scalable graphics architecture. *Computer Graphics (SIGGRAPH 2000 Proceedings)* (2000), 443–454.

[13] ELLSIEPEN, P. Parallel isosurfacing in large unstructed datasets. In *Visualization in Scientific Computing* (1995), Springer-Verlag, pp. 9–23.

[14] EYLES, J., MOLNAR, S., POULTON, J., GREER, T., LASTRA, A., ENGLAND, N., AND WESTOVER, L. Pixelflow: The realization. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware* (August 1997), pp. 57–68.

[15] FUSSELL, D. S., AND RATHI, B. D. A vlsi-oriented architecture for real-time raster display of shaded polygons. In *Graphics Interface '82* (May 1982).

[16] GUEZIEC, A., TAUBIN, G., LAZARUS, F., AND HORN, W. Cutting and stitching: Efficient conversion of a non-manifold polygonal surface to a manifold. Tech. Rep. RC-20935, IBM T.J. Watson Research Center, 1997.

[17] HANSEN, C., AND HINKER, P. Massively parallel isosurface extraction. In *Visualization '92* (September 1992).

[18] HEIRICH, A., AND MOLL, L. Scalable distributed visualization using off-the-shelf components. In *Parallel Visualization and Graphics Symposium – 1999* (San Francisco, California, October 1999), J. Ahrens, A. Chalmers, and H.-W. Shen, Eds.

[19] HUMPHREYS, G., BUCK, I., ELDRIDGE, M., AND HANRAHAN, P. Distributed rendering for scalable displays. In *Proceedings of Supercomputing 2000* (October 2000).

[20] HUMPHREYS, G., AND HANRAHAN, P. A distributed graphics system for large tiled displays. In *Proceedings of IEEE Visualization Conference* (1999), pp. 215–223.

[21] LORENSEN, W., AND CLINE, H. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics 21*, 4 (July 1987), 163–169.

[22] MOLNAR, S., COX, M., ELLSWORTH, D., AND FUCHS, H. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (July 1994).

[23] MOLNAR, S. E. Image composition architectures for real-time image generation. Ph.d. dissertation, technical report tr91-046, University of North Carolina, 1991.

[24] PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P. Interactive ray tracing for isosurface rendering. In *Visualization '98* (October 1998).

[25] RAMACHANDRAN, V. Personal communication, November 1999.

[26] SAMANTA, R., ZHENG, J., FUNKHOUSER, T., LI, K., AND SINGH, J. P. Load balancing for multi-projector rendering systems. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware* (August 1999).

[27] SCHNEIDER, B.-O. Parallel rendering on pc workstations. In *Parallel and Distributed Processing Techniques and Applications* (July 1998), pp. 1281–1288.

[28] SHEN, H., HANSEN, C., LIVNAT, Y., AND JOHNSON, C. Isosurfacing in span space with utmost efficiency (issue). In *Visualization '96* (1996), pp. 287–294.

[29] TAUBIN, G., AND ROSSIGNAC, J. Geometric compression through topological surgery. *ACM Trans. Gr. 17*, 2 (1996), 84–115.

[30] TOUMA, C., AND GOTSMAN, C. Triangle mesh compression. In *Proceedings of the 24th Conference on Graphics Interface (GI-98)* (1998), W. Davis, K. Booth, and A. Fourier, Eds., Morgan Kaufmann Publishers, pp. 26–34.

[31] VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM 33* (September 1990), 103–111.

[32] VITTER, J. S. External memory algorithms and data structure. In *DIMACS Series in Discrete Mathematics and Theoretical Computer S cience* (1999).

[33] WITTEN, I., NEAL, R., AND CLEARY, J. Arithmetic coding for data compression. *Commun. ACM 30* (June 1987), 520–540.