A Unified Infrastructure for Parallel Out-Of-Core Isosurface Extraction and Volume Rendering of Unstructured Grids

Yi-Jen Chiang* Polytechnic University
Ricardo Farias† University at Stony Brook
Cláudio T. Silva‡ AT&T
Bin Wei§ AT&T

Abstract

In this paper, we present a unified infrastructure for *parallel out-of-core* isosurface extraction and volume rendering of large unstructured grids on distributed-memory parallel machines. We parallelize the out-of-core isosurface extraction algorithm of [9] and the out-of-core ZSweep technique [17] for direct volume rendering, using the *meta-cell* technique as a unified underlying building block.

Our one-time preprocessing first partitions the dataset into *meta-cells* that are stored in disk. From the meta-cells, we build a *BBIO tree* in disk, which can be used to speed up isosurface extraction, and a *bounding-box file* in disk, which is used for direct volume rendering. At run-time, we use a simple *self-scheduling scheme* [39] to achieve *load balancing* among the processors.

We perform several experiments on a sixteen-node cluster of PCs connected by a gigabit ethernet, using datasets as large as 6.6 million cells. For the larger datasets, we have found that both our isosurface extraction and direct volume rendering approaches are perfectly scalable up to sixteen nodes.

Keywords: Isosurface Extraction, Volume Rendering, Parallel Computation, Out-Of-Core Techniques, Scientific Visualization.

1 Introduction

In recent years, new challenges for scientific visualization emerged as the size of data generated from simulations grew exponentially [4]. The sheer size of data often makes the task of interactive visualization impossible, as only a small portion of data can fit into main memory at a time, and the computation cost is often too high for an algorithm to run in real-time. In this paper, we address both issues of limited main memory size and insufficient computing speed of the current graphics workstations for processing large visualization applications, by proposing a unified infrastructure for *parallel out-of-core* isosurface extraction and direct volume rendering. Our methods focus on the class of *unstructured-grid* volume datasets, which is the most general class of volumetric data and has been proposed as an effective means of representing disparate field data that arises in a broad spectrum of applications including structural mechanics, computational fluid dynamics, partial differential equation solvers, and shock physics.

Isosurface extraction and direct volume rendering are two of the most important classes of visualization techniques for volume datasets. Although these techniques have been developed to a high degree of sophistication, most of the algorithms require the entire dataset to be kept in main memory, which is a severe limitation on their applicability, especially for large scientific applications.

In this paper, we present a unified infrastructure that supports both isosurface extraction and direct volume rendering for large unstructured grids, using out-of-core techniques that are parallelized for distributed-memory parallel machines with a local disk and a limited-size main memory available for each node. We parallelize the out-of-core isosurface extraction algorithm of Chiang *et al.* [9] and the out-of-core ZSweep technique of Farias and Silva [17] for direct volume rendering, using the *meta-cell* technique first developed in [9] as a unified underlying building block.

In the preprocessing phase, we partition the original dataset into clusters of cells, called *meta-cells*, which are kept in disk and each occupying roughly the same number of disk blocks. Similar to the out-of-core isosurface extraction technique [9], we build an indexing data structure, namely the *binary-blocked I/O interval tree* (BBIO tree), which is a B-tree-like interval tree and is entirely kept in disk, to index the meta-cells, so that given an isovalue the *active* meta-cells that are intersected by the isosurface can be located in an I/O-optimal way. In addition, similar to the out-of-core ZSweep algorithm [17], we build a bounding-box file, which is also kept in disk and contains the bounding box and other auxiliary information for each meta-cell, to facilitate the (in-core) ZSweep algorithm [16] (which is based on sweeping a plane in the z-direction) in an I/O-efficient way. The meta-cells, the BBIO tree, and the bounding-box file only need to be constructed once, and this construction process can be performed efficiently even without the need of keeping the entire dataset in main memory; the running time for this preprocessing is the same as running external sorting a few times.

In the run-time phase, we parallelize the bottleneck operations in the out-of-core isosurface extraction technique of [9] and in the out-of-core ZSweep algorithm [17]: for the former, these include reading the active meta-cells from disk and scanning through the active meta-cells to generate isosurface triangles; for the latter, these include reading the meta-cells from disk in a desired front-to-back order and performing in-core ZSweep on the meta-cells read. We develop *self-scheduling schemes* [39] for both our parallel algorithms, which ensure *load balancing* among processor nodes and are very simple to implement.

2 Previous Related Work

In this section we review the previous related work in the areas of isosurface extraction, direct volume rendering for unstructured grids, and out-of-core techniques; related parallel visualization work is reviewed in the context of isosurface extraction and direct volume rendering.

2.1 Isosurface Extraction

There is a very rich literature on isosurface extraction; we refer to [31] for an excellent and thorough review. In Marching

* Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn, NY 11201; yjc@poly.edu. Supported in part by NSF CAREER Grant CCR-0093373 and by CATT, a New York Office of Science, Technology and Academic Research (NYSTAR) designated Center for Advanced Technology.

† Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, NY 11794-3600; rfarias@ams.sunysb.edu.

‡ AT&T Labs-Research, 180 Park Ave., Room D265, Florham Park, NJ 07932; csilva@research.att.com.

§ AT&T Labs-Research, 180 Park Ave., Room D286, Florham Park, NJ 07932; bw@research.att.com.

Cubes [32], all cells in the volume dataset are searched for isosurface intersection. Techniques avoiding exhaustive scanning include using an octree [56], identifying a collection of *seed cells* and performing contour propagation from the seed cells [3, 26, 53], NOISE [31], and other nearly optimal isosurface extraction methods [41, 42]. The first *optimal* algorithm was given by Cignoni *et al.* [11]. The first *out-of-core* isosurface technique was given by Chiang and Silva [7]. They follow the ideas of Cignoni *et al.* [11], but use the I/O-optimal interval tree of [1] to solve the interval search problem. Later, Chiang *et al.* [9] further improved the disk space overhead and the preprocessing time of [7], at the cost of slightly increasing the isosurface query time, by developing a *two-level* indexing scheme, the *meta-cell* technique, and the *BBIO* tree which is used to index the meta-cells. We will review this approach in more detail in Section 3. As for parallel isosurface extraction, Hansen and Hinker [22] described parallel methods on SIMD machines, Ellsiepen [15] gave a parallel algorithm for FEM data by distributing working blocks to a number of connected workstations, Shen *et al.* [41] developed a sequential and parallel approach called ISSUE, and Parker *et al.* [37] presented a parallel technique using ray tracing. In addition, Bajaj *et al.* [2] proposed a parallel and out-of-core approach based on contour propagation from seed cells. Sequential isosurface techniques for time-varying data were given by Shen [43] and Sutton and Hansen [49].

2.2 Volume Rendering for Unstructured Grids

A number of efficient algorithms for rendering unstructured grids have been developed. One class is based on adapting ray tracing techniques [20, 52, 5]. In general, these techniques require random access to the cells, connectivity information, and in some cases extra memory (such as [5]) to optimize the computation of intersections of rays with faces of the cell complex. Other techniques use scan-line algorithms, which sweep the data with a plane perpendicular to the image plane [58, 6]. Some of them (e.g., [48]) are designed to be memory efficient, but still use the connectivity of the mesh. Others (e.g., [21, 55]) use discrete buffers in z to determine the order of compositing, and completely avoid the need for connectivity information. However, using discrete buffers has the potential to lower the accuracy, and can require a substantial amount of main memory. The technique of [61] samples the irregular grid with a fixed number (e.g., 50 or 100) of planes which are later composited together. This algorithm does not use connectivity, but the space to keep the planes can be quite substantial. Farias *et al.* [16] developed ZSweep, also based on sweeping the data with a plane in the z direction. Very recently, Farias and Silva parallelized the ZSweep algorithm for distributed shared-memory architectures [18], and also developed an out-of-core version of ZSweep [17]. In [18] a *tiling* idea and an octree partition scheme were proposed; in [17] meta-cells and a similar tiling idea were employed. We will discuss ZSweep and out-of-core ZSweep in more detail in Section 3.3.

Another approach for rendering irregular grids is the use of projection ("feed-forward") methods (e.g., [57, 35, 45, 10], in which the cells are projected onto the screen, one-by-one. Most of these techniques exploit graphics hardware to compute the volumetric lighting models [45], by first computing a *visibility ordering* [35, 60, 12], and incrementally accumulating their contributions to the final image. As for parallel volume rendering of irregular grids, Ma and Crockett [33] used the approach of computing all intersections between each cast ray with all the cells, and performing a post-sorting to compute the image. Their technique distributes the cells among processors in a round-robin fashion. To avoid storing a very large number of ray intersections, they cleverly schedule the computation using a k-d tree. As shown in [33, 34], their algorithm is very scalable on message-passing machines. Recently, Hofsetz and Ma [23] have developed an efficient shared-memory version of this algorithm. Hong and Kaufman [24] proposed a very efficient ray-casting based rendering algorithm for curvilinear grids, and parallelized their ray caster using an image-based task scheduling scheme. The parallelization of a ray casting technique has also been studied by Uselton [52] with very good results. Challinger [6] and Wilhelms *et al.* [58] proposed scan-line rendering algorithms. Both papers report on parallelization, which is the main focus of [6]. Challinger [6] also used an image tiling scheme for parallelization with very good results. Other parallel techniques on shared-memory machines include the results by Williams [59], Nieh and Levoy [36], and Lacroute [27, 28].

2.3 Out-Of-Core Techniques

We now briefly review the work on out-of-core techniques. For theoretical results on out-of-core algorithms for graphs and for computational geometry, we refer to the recent survey by Vitter [54]. Teller *et al.* [50] described a system to compute radiosity solutions for polygonal environments larger than main memory, and Funkhouser *et al.* [19] presented prefetching techniques for interactive walk-throughs in large architectural virtual environments. More recently, Pharr *et al.* [38] gave memory-coherent ray-tracing algorithms, Cox and Ellsworth [13] presented application-controlled demand paging methods, and Ueng *el al.* [51] proposed out-of-core streamline techniques. As mentioned, Chiang and Silva [7, 8] and Chiang *et al.* [9] gave a series of out-of-core algorithms for isosurface extraction, and Bajaj *et al.* [2] presented parallel and out-of-core isosurface techniques. Leutenegger and Ma [29] and Farias and Silva [17] developed out-of-core volume rendering approaches. Shen *et al.* [44] and Sutton and Hansen [49] reported out-of-core visualization for time-varying datasets. For surface simplification, Hoppe [25] proposed view-dependent simplification method for terrains larger than main memory, and Lindstrom [30] gave an out-of-core technique to simplify large polygonal models, which can perform simplification very efficiently but does not produce any hierarchical structure for level-of-detail rendering. Concurrently, El-Sana and Chiang [14] developed out-of-core view-dependent simplification and rendering techniques that can efficiently perform both simplification and view-dependent rendering for polygonal models larger than main memory.

3 Parallel Out-of-Core Isosurface and Volume Rendering

We now present our unified infrastructure that supports both isosurface extraction and direct volume rendering using parallel out-of-core techniques. Our method is based on parallelizing the out-of-core isosurface extraction algorithm of Chiang *et al.* [9] and the out-of-core ZSweep technique of Farias and Silva [17] for direct volume rendering, with the *meta-cells* as the underlying building block.

Our target computing architecture is very simple, and is basically a cluster of machines, which contain the following components:

- one *host* machine, where a *client* application drives the computation to be performed by the processing nodes;
- one or more file servers or a directly attached storage-area network (SAN), where the relevant data files reside and are dynamically fetched by the processing nodes as needed;
- one or more processing nodes, which perform computation by receiving assignments from the client application running on the host machine.

Compared to the PVR system of Silva et al [47, 46], our current system has extra functionalities of supporting out-of-core volume rendering and isosurface extraction, but does not have the PVR's high-performance compositing back-end [40] and thus is not designed to be as flexible. We believe our extensions could be integrated into PVR.

Local storage Often, we assume large temporary files (e.g., intermediate isosurface results) can be stored on the processing nodes. This can either be accomplished by providing the cluster machines with local disk (which is usually the case), or configuring a directly attached storage-area network (SAN).

3.1 Preprocessing

In our preprocessing phase, we construct three files in disk:
(1) the file containing all meta-cells,
(2) the file of BBIO tree, and
(3) the bounding-box file.

The meta-cell file is essentially the data file, while the other two files are auxiliary files and are much smaller in size. The BBIO tree is used to index the meta-cells for isosurface extraction, and the bounding-box file is used to facilitate the traversal of meta-cells for direct volume rendering. All three files can be constructed efficiently without the need of keeping the entire input in main memory; the entire preprocessing time is the same as running external sorting a few times. We refer the interested reader to [9] and [17] for details on the algorithms used for preprocessing.

We describe the purpose of meta-cells, which is a critical feature of our work. Typical input of an unstructured-grid dataset has a list of vertices, where each vertex entry contains its x-, y-, z- and scalar values, and a list of cells, where each cell entry contains pointers to its vertices in the vertex list. We refer to this as the "index cell set" (ICS) format. While this format is very compact and very useful for in-core algorithms, it is not suitable for out-of-core accesses. Observe that random accesses in disk by following pointers to the vertex list is very inefficient, since this involves a large amount of disk head movement (the disk "seeks"). Moreover, since each I/O operation reads/writes an entire disk block, this also results in reading an entire disk block into main memory in order to just access a single item in that block (which usually contains hundreds of items). One way to avoid such random pointer accesses is to replace each vertex pointer by the vertex x-, y-, z- and scalar values in the cell list, but this causes many duplicated vertex information and is very inefficient in disk space.

The meta-cell technique is used to optimize both disk access cost and disk space requirement. We cluster spatially neighboring cells together to form a meta-cell. Each meta-cell is roughly of the same storage size, usually in a multiple of disk blocks and always able to fit in main memory. Each meta-cell has *self-contained* information and is always read as a whole from disk to main memory. Therefore, we can use the compact ICS representation for each meta-cell, namely a local vertex list and a local cell list which contains pointers to the local vertex list. In this way, a vertex shared by many cells in the same meta-cell is stored just *once* in that meta-cell; the only duplications of vertex information occur when a vertex belongs to two cells in different meta-cells; in this case we let both meta-cells include that vertex in their vertex lists to make each meta-cell self-contained.

During meta-cell construction, a cell may lie between different meta-cells. For each such cell, we use a voting scheme to assign the cell to a single meta-cell, and also duplicate the cell to the other meta-cells that it intersects, with these additional duplicated cells marked. During isosurface extraction, we scan the cells in each active meta-cell but skip the marked cells, so that each potential active cell is examined exactly once to avoid producing duplicated isosurface triangles. During volume rendering, all the cells in the current meta-cell, including the marked cells, participate the rendering procedure, so that each cell makes its correct contribution to the final image.

3.2 Parallel Out-of-Core Isosurface Extraction

In the run-time phase, we support isosurface queries by parallelizing the bottleneck operations in the out-of-core isosurface technique of [9]. In [9], isosurface queries are performed as follows. Given an isovalue q, we first search via the BBIO tree in disk to find all $[\min, \max]$ meta-intervals containing q. These meta-intervals correspond to active meta-cells that are intersected by the isosurface of q. For each such meta-interval, we use its meta-cell ID to read the corresponding active meta-cell from disk to main memory, and go through the cells in this meta-cell to find the isosurface triangles using the Marching Cubes algorithm (actually, Marching Tetrahedra in our case). Finding meta-intervals containing q with the BBIO tree is relatively fast, since the interval search is performed I/O-optimally, and the record size of the reported meta-intervals is very small. The computational bottleneck comes from two sources of operations: reading the active meta-cells from disk (an I/O operation), and scanning through active meta-cells to compute isosurface triangles (a CPU operation). We reduce the bottleneck by parallelizing these operations.

Our overall algorithm is described as follows.

1. Given an isovalue q, the host uses the BBIO tree in the host disk to find all meta-intervals containing q. The host then creates a process queue Q that collects all active meta-cell IDs from these meta-intervals.

2. The host acts as a process dispatcher to handle job requests from all other processor nodes. When a processor node finishes its current job (or is idle initially), it sends a job request to the host; the host, upon receiving a job request, removes the next available active meta-cell ID from Q and gives this meta-cell ID to the corresponding processor node making the job request. This procedure is repeated until all active meta-cell IDs are removed from Q.

   Each processor node, after getting an active meta-cell ID from the host, performs the following operations.

   (a) Read the corresponding meta-cell M from the meta-cell file in disk to its local main memory.

   (b) Perform Marching Cubes/Marching Tetrahedra on the meta-cell M to compute the subset of the isosurface triangles that is contributed by M. As described in Section 3.1, when scanning through the cells of M, skip the marked cells to avoid generating duplicated isosurface triangles.

   (c) Save the generated isosurface triangles in local storage, deallocate the main memory space for M, and send a job request and a job completion message to the host.

3. When the host receives job-completion messages from all processors, it knows the isosurface computation is finished.

   Post-processing is required for actual rendering — in certain systems, e.g. PVR, the data could be streamed directly from the processing nodes to the rendering nodes.

Our algorithm parallelizes the original bottleneck operations very well. In particular, the scheme of using the process queue Q to dynamically dispatch the jobs to available processor nodes ensures a *load balancing* among all nodes: a node that finishes its

current job earlier will continue to process another job, so that all nodes are kept busy. Moreover, this parallel scheme is very simple to implement.

Similar to the original out-of-core isosurface technique [9], we only need a very small amount of main memory for each processor node. For the host node, this includes the main memory space to hold one disk block that is the current node of the BBIO tree being visited for the interval search, the space to keep the reported meta-intervals and the active meta-cell IDs in Q, and finally the space to retain the complete set of isosurface triangles. For other processor nodes, we only need the main memory space for the current meta-cell being processed. The isosurface triangles generated from this meta-cell are directly dumped to disk (either local, NFS mounted file system, or SAN).

3.3 Parallel Out-of-Core Volume Rendering

In the run-time phase, we support direct volume rendering by parallelizing the bottleneck operations in the out-of-core ZSweep algorithm [17]. We first give an overview for the in-core ZSweep technique [16], which is the basic building block. ZSweep is based on sweeping the dataset with a plane parallel to the viewing plane, in the order of increasing z (i.e., front-to-back). As the vertices are encountered by the sweep plane, the faces of cells that are incident to the encountered vertices are *projected*. During face projection, one simply computes the intersection between the face and the ray emanating from each pixel, and stores its z-value and other auxiliary information, in sorted order, into a list of ray-face intersections for the given pixel. In order to avoid the lists getting arbitrarily large, a scheme called *target-z* is employed to enable early compositing. Figure 3 demonstrates these ideas.

We now briefly review the out-of-core ZSweep algorithm [17], which uses meta-cells and the bounding boxes of the meta-cells. First, the image plane is divided into rectangular *tiles*. The meta-cells, after viewing transformation, are assigned to the buckets of the tiles onto which the meta-cells project (calculated using the bounding box of each meta-cell). For each tile, its final image is rendered independently from other tiles. The meta-cells assigned to the same tile are sorted in front-to-back order (using the bounding boxes). They are then read from disk to main memory, one by one as they are encountered by the sweep plane of the tile in front-to-back order, and rendered and composited using the ZSweep algorithm [16].

We observe that in the above out-of-core ZSweep approach, the computations involving bounding boxes are relatively fast, since the record of each bounding box is very small and the only I/O operations needed on the bounding-box file is linearly scanning through the file once. The actual computational bottleneck comes from two sources: reading the meta-cells from disk in a desired front-to-back order (I/O operations), and performing in-core ZSweep on the meta-cells read (CPU operations). We reduce the bottleneck by parallelizing these operations.

A complete description of our parallel out-of-core volume rendering algorithm is given below.

1. The host performs the following operations.
   (a) Partition the final image into a number of rectangular tiles, and create a bucket for each tile.
   (b) Create a process queue P, in which each process corresponds to a tile.
2. The host acts as a process dispatcher to handle job requests from all other processor nodes. When a processor node finishes its current job (or is idle initially), it sends a job request to the host; the host, upon receiving a job request, removes the next available tile T from P and gives T to the processor node making the job request. This procedure is repeated until all tiles are removed from P.

   As part of a start-up initialization, each processor node scans the bounding-box file, projects each bounding box to the image space, and puts the projected bounding box to the bucket of a tile if the projected bounding box intersects this tile. If the projected bounding box intersects several tiles, put it to *all* the corresponding buckets. Doing this once avoids having to recompute such information every time a new rendering job is received. An alternative design (which also works fine in most cases) would be to have the host do this one-time computation.

   Each processor node, after getting a tile T from the host, performs the following operations.
   (a) Sort the projected bounding boxes stored in the bucket of T into front-to-back order. Let the corresponding meta-cells in the sorted order be $M_1, M_2, \cdots, M_\ell$. These are all the meta-cells that may potentially have contributions to the tile T in the final image, in front-to-back order.
   (b) Process meta-cells $M_1, M_2, \cdots, M_\ell$ one by one in this order as their bounding boxes are encountered by the sweep plane of the tile T. Usually there is only one or two such *working* meta-cells at a time, but sometimes there are a few more working meta-cells. For each such working meta-cell M_t, perform the following steps.
      i. Read M_t (using its meta-cell ID that is stored with its bounding box) from the meta-cell file in disk to the local main memory.
      ii. Transform the vertices of M_t into image space, and sort them in the z-direction. Merge these vertices with the sorted vertices of the original working meta-cells already in main memory. Continue to run ZSweep (or start to run ZSweep if M_t is the first working meta-cell for T) on the sorted vertices and the cells of the current working meta-cells, now including M_t. As described in Section 3.1, all cells in M_t, marked or not, participate the process, so that each cell makes its correct contribution to the final image. When M_t is completely processed, deallocate the main memory space for M_t to make room for a next working meta-cell.
   (c) When meta-cells $M_1, M_2, \cdots, M_\ell$ are all processed, the rendering of the tile T is complete. Send the image of T as well as a job request to the host.
3. When P is empty and all processor nodes finish their jobs, the host combines the completed images of all tiles together to make up the final image.

The above algorithm effectively parallelizes the original bottleneck operations. The self-scheduling scheme is similar to the one in our parallel out-of-core isosurface extraction algorithm described in Section 3.2, and again is very simple to implement. Observe that different tiles may have very different number of meta-cells projected to them, and hence the time to render them may differ significantly. With our scheme, the processor nodes are kept busy and thus an effective load balancing among the nodes is achieved. Also, each tile is completely rendered by a single processor node using the front-to-back ordering, and thus optimizations such as early ray termination could potentially be used to speed up the computation.

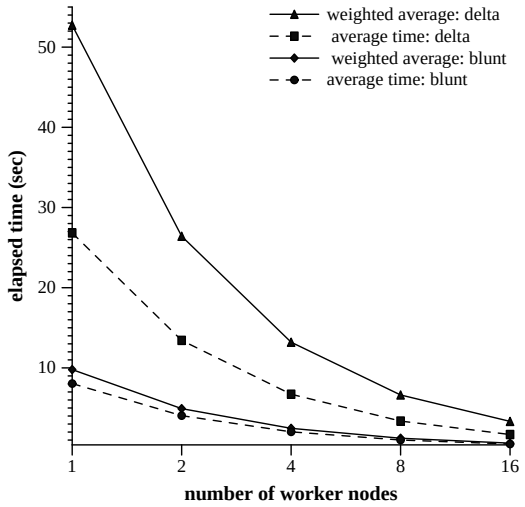

Figure 1: Summary of performance of parallel isosurface computation. Each data point is an average of several isosurfaces. For the Delta dataset we computed ten different isosurfaces; for the Blunt dataset, we computed five different isosurfaces.

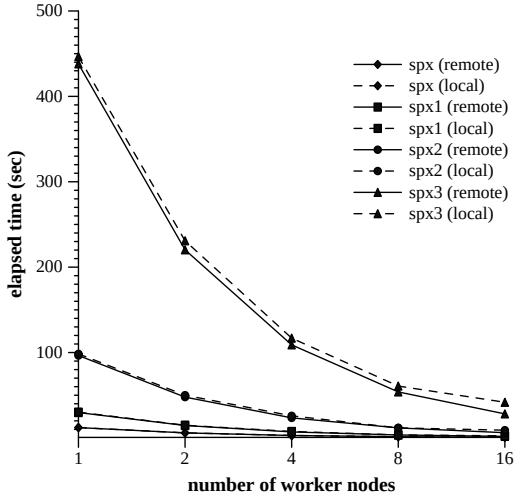

Figure 2: Summary of performance of the volume rendering computation. Each data point is an average of six views, each of which corresponds to an additional $\frac{\pi}{3}$ rotation of the object from the previous view. The images were computed with a 512-by-512 resolution using 256 tiles for task subdivision. A curve labeled "(remote)" means the centralized-data model is used, while a curve labeled "(local)" means the local-data model is used.

Similar to the original out-of-core ZSweep method, our algorithm essentially uses a small, fixed amount of local main memory space for each processor node. This includes the space for holding a few working meta-cells being processed, the space for small lists of ray-face intersections for each pixel in a tile, and the space for the image footprint for a tile.

4 Experimental Results and Analysis

We implemented the techniques presented in this paper using a combination of C/C++, Vtk for the isosurface computation, and MPI for the communication between processors. The task dispatching code was implemented using a portion of the PVR code described in [46].

Our experiments were performed on a small PC cluster, composed of 16 client machines and one server machine. All the machines are equipped with an AMD Athlon running at 900 MHz and 512 MB of main memory (the server has 784 MB of main memory). The clients have IDE hard disks, while the server has a 400 GB disk array composed of eight SCSI disks, but configured as two 200 GB striped disks. Our communication layer is a switched gigabit ethernet. All the machines are running vanilla RedHat 7.0. We used Vtk 3.2 and MPI/Pro 1.6.3 (running on top of TCP/IP) for our experiments.

The server machine was used for two purposes: (1) it was the host node which served up work and collected information for the client machines, and (2) it was also an NFS server, which served large files to the client machines. Our setup is very simple to replicate, and is fairly inexpensive since our cluster is put together completely out of commodity parts and commodity software (as far as possible — clearly, we wrote the query part of the isosurface computation and the volume rendering algorithm).

Table 1 shows the datasets we used in our experiments. For the purpose of comparison, we also experimented with two models of data access: (1) centralized-data model, in which we only have one copy of the meta-cell file residing in the server disk, and (2) local-data model, in which there is one copy of the meta-cell file in the local disk of each processor node, so that each node can read the desired data from its local disk.

Isosurface Computation

We have computed isosurfaces for both the Blunt and Delta datasets. A sample image of our isosurfaces extracted from the SPX dataset is given in Figure 4. We computed five isosurfaces for the Blunt dataset, and ten isosurfaces for the Delta dataset. We spread the isovalues in a wide range that has an isosurface in each case. Figure 1 summarizes our findings. Our I/O-efficient querying technique coupled with the self-scheduling scheme worked together to generate an excellent performance.

We report results in two different ways. One way is to simply average the time of each isosurface computation. This does not account for the fact that different isosurfaces have different complexities and thus take different running times. Measuring performance this way gives us an overall speedup of 15.77 for the Blunt dataset and 15.86 for the Delta dataset. Another way to measure would be to *weight* the measurements by taking the number of active cells into account. We define the *weighted time* as the usual running time multiplied by the number of active cells in each step, and then divided by the total number of active cells. For this, we measured a *perfect* speedup of 16 for the Blunt dataset, and an almost perfect 15.8 for the Delta dataset.

We would like to point out that each processor node fetches only the data it needs, and only once. In fact, the amount of data movement is quite low. This is the main reason we did not care to experiment with copying the data to the local disk of each node, as we would not expect any change in performance.

Volume Rendering

We computed volume renderings for all the datasets. A summary of the performance is given in Figure 2 and Table 2; Figure 5 shows a sample image of our results. In general, volume rendering requires a considerable amount of data movement. In fact, for each image, the whole dataset has to be moved to the processor nodes. This is in sharp contrast to isosurface computations, where only a small portion of the dataset has to move.

Dataset Information					
Dataset	# of vertices	# of cells	meta-cell data	BBIO tree	bounding-box file
Blunt Fin	41K	187K	27 MB	31.8 KB	40 KB
Delta Wing	212K	1005K	212 MB	224 KB	254 KB
SPX	2.9K	13K	1 MB	4.1 KB	1 KB
SPX1	20K	103K	12 MB	17.7 KB	23 KB
SPX2	150K	830K	65 MB	18.1 KB	27 KB
SPX3	1150K	6620K	453 MB	44 KB	56 KB

Table 1: Main datasets used for benchmarking. The first two are tetrahedralized versions of the well-known NASA datasets. SPX is an unstructured grid composed of tetrahedra. For the last three versions of SPX, each version is obtained from the previous version by subdividing each tetrahedron into 8, that is, SPX3 is 512 times as large as SPX. We list the number of vertices (in thousands), number of tetrahedra (in thousands), the size of the meta-cell data file (in megabytes), the size of the BBIO tree (in kilobytes), and the size of the bounding-box file (in kilobytes).

Parallel Out-Of-Core Volume Rendering Times (sec)								
Num. Processors.	SPX2		SPX3		Blunt Fin		Delta Wing	
	Local	Remote	Local	Remote	Local	Remote	Local	Remote
1	98.4	96.5	446.6	438.1	43	44.7	177	177
2	50.0	47.8	230.9	220.3	20	21	88	86
4	25.9	23.5	116.9	109.0	10.5	10.9	44	42
8	11.8	11.7	60.7	53.9	5.5	5.6	30	26
16	9.1	6.1	41.7	28.1	3.5	3.7	28	28

Table 2: The times shown are the average over the running times for six rotational views. See Figure 2 for details. "Local" means the local-data model is used, and "Remote" means the centralized-data model is used.

Comparing the single processor times, we see that our new parallel code is about twice as slow as the sequential out-of-core ZSweep algorithm [17].

In any case, our parallel volume rendering results are quite good. As data in Table 2 indicates, our speedups were excellent for all versions of SPX — for SPX3, with over 6.6 million cells, we got a speedup of 15.6 (a parallel efficiency of 98%). We also see that the speedups were very good for Blunt (over 12), and only about 6 or so for Delta. We believe the limited image size is the main cause of the problem for the Delta dataset. We had similar poor results for our shared-memory version of ZSweep when the resolution was low (see [18]). More interestingly, and somewhat surprisingly, we see that reading data from local disks resulted in no better or sometimes even worse performance than reading data from the remote disk served by a single machine. This might be due to the fact that the local disks are just IDE disks, while the server has faster SCSI striped disks. In addition, the gigabit ethernet seems to have plenty of capacity to handle our traffic, and the amount of data movement is not enough to overload the server.

Even for the Delta dataset, we would like to point out that our approach is quite competitive in the sense that each node only needs to have a small amount of main memory, thus allowing for rendering very large dataset efficiently. In addition, since each node only reads the small portion of the dataset that is needed to render its current tile, this is quite efficient. Using the original ZSweep code [16], it would take over 20 seconds to simply read the dataset. In that time, our approach would be close to finishing the rendering.

5 Conclusions

We have presented a unified infrastructure that supports both isosurface extraction and direct volume rendering for large unstructured grids, using out-of-core techniques that are parallelized for distributed-memory parallel machines with a local disk and a limited-size main memory available for each node. Our experiments demonstrated a perfect or near perfect speedup for both isosurface extraction and direct volume rendering.

Our self-scheduling scheme is especially advantageous. It is very simple to implement, and it enables our algorithms to achieve load balancing very effectively. Although our current experiments were performed only on a cluster of machines that are of the same platform, we believe that this self-scheduling scheme can achieve a similar load balancing in a heterogeneous environment. This would enable us to fully utilize the computing power of the entire site.

Our out-of-core techniques are also of special interest. A central theme of the techniques is that each processor node only fetches the small portion of the dataset that is needed for the current computation. This not only minimizes the data movement between disk and main memory (I/O communications), but also minimizes the data movement in the network (network communications). It is clear that minimizing the data distribution cost is the key to designing efficient visualization algorithms, especially for large datasets. In addition, our efficient client-server visualization model implies an efficient *remote* visualization: our *meta-cell* data representation seems to be able to live out of the visualization machines with little disk space overhead, in addition to the large visualization speedup it brings to us.

In conclusion, our work of integrating out-of-core techniques with parallel approaches decouples the size of a visualization task from the amount of computational resources available, and indicates a promising direction towards resolving the big challenges posed by large-scale visualization problems.

Acknowledgments

We thank Peter Williams (LLNL) and NASA for the datasets used in our experiments. This work was made possible with the gener-

ous support of Sandia National Labs and the Dept of Energy Mathematics, Information and Computer Science Office. Y.-J. Chiang acknowledges partial support by NSF CAREER Grant CCR-0093373 and by CATT, a New York Office of Science, Technology and Academic Research (NYSTAR) designated Center for Advanced Technology. R. Farias acknowledges partial support from CNPq-Brazil under a PhD fellowship.

References

[1] L. Arge and J. S. Vitter. Optimal interval management in external memory. In *Proc. IEEE Foundations of Comp. Sci.*, pages 560–569, 1996.

[2] C. Bajaj, V. Pascucci, D.Thompson, and X.Y. Zhang. Parallel accelerated isocontouring for out-of-core visualization. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium*, pages 97–104, 1999.

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

[4] S. Bryson, D. Kenwright, and M. Cox. *Exploring gigabyte datasets in real time: algorithms, data Management, and time-critical Design*. ACM SIGGRAPH course note, 1997.

[5] P. Bunyk, A. Kaufman, and C. Silva. Simple, fast, and robust ray casting of irregular grids. In *Scientific Visualization, Proceedings of Dagstuhl '97*, pages 30–36, 2000.

[6] J. Challinger. Scalable parallel volume raycasting for nonrectilinear computational grids. *ACM SIGGRAPH Symposium on Parallel Rendering*, pages 81–88, November 1993.

[7] Y.-J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In *Proc. IEEE Visualization*, pages 293–300, 1997.

[8] Y.-J. Chiang and C. T. Silva. External memory techniques for isosurface extraction in scientific visualization. *External Memory Algorithms and Visualization (DIMACS Book Series, American Mathematical Society)*, 50:247–277, 1999.

[9] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In *Proc. IEEE Visualization*, pages 167–174, 1998.

[10] P. Cignoni, C. Montani, D. Sarti, and R. Scopigno. On the optimization of projective volume rendering. In *Visualization in Scientific Computing '95*, pages 58–71. Springer Verlag, 1995.

[11] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), April - June 1997.

[12] J. Comba, J. T. Klosowski, N. Max, J. S. B. Mitchell, C. T. Silva, and P. L. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. *Computer Graphics Forum*, 18(3):369–376, September 1999.

[13] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proc. IEEE Visualization*, pages 235–244, 1997.

[14] J. El-Sana and Y.-J. Chiang. External memory view-dependent simplification. *Computer Graphics Forum (EUROGRAPHICS 2000 Proceedings)*, 19(3):139–150, August 2000.

[15] P. Ellsiepen. Parallel isosurfacing in large unstructured datasets. In *Visualization in Scientific Computing '95*, pages 9–23. Springer Verlag, 1995.

[16] R. Farias, J. Mitchell, and C. Silva. ZSweep: An efficient and exact projection algorithm for unstructured volume rendering. In *Proc. Volume Visualization Symposium*, pages 91–99. ACM Press, October 2000.

[17] R. Farias and C. Silva. Out-of-core rendering of large unstructured grids. *IEEE Computer Graphics & Applications*, 21(4):42–50, July 2001.

[18] R. Farias and C. Silva. Parallelizing the ZSweep algorithm for distributed-shared memory architectures. In *Proc. International Workshop on Volume Graphics*, 2001.

[19] T. A. Funkhouser, S. Teller, C. H. Séquin, and D. Khorramabadi. Database management for models larger than main memory. In *Interactive Walkthrough of Large Geometric Databases, Course Notes 32, SIGGRAPH '95*, pages E15–E60, 1995. Appeared as "The UC Berkeley System for Interactive Visualization of Large Architectural Models", in *Presence: Teleoperators and Virtual Environments*, Vol.5, No.1, Winter 1996.

[20] M. P. Garrity. Raytracing irregular volume data. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):35–40, November 1990.

[21] C. Giertsen. Volume visualization of sparse irregular meshes. *IEEE Computer Graphics & Applications*, 12(2):40–48, March 1992.

[22] C. Hansen and P. Hinker. Massively parallel isosurface extraction. In *Proc. IEEE Visualization*, 1992.

[23] C. Hofsetz and K.-L. Ma. Multi-threaded rendering unstructured-grid volume data on the sgi origin 2000. In *Third Eurographics Workshop on Parallel Graphics and Visualization*, 2000.

[24] L. Hong and A. E. Kaufman. Accelerated ray-casting for curvilinear volumes. *IEEE Visualization '98*, pages 247–254, October 1998.

[25] H. H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *IEEE Visualization '98*, pages 35–42, October 1998.

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

[27] P. Lacroute. Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization. *IEEE Parallel Rendering Symposium*, pages 15–22, October 1995.

[28] P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3), September 1996.

[29] S. Leutenegger and K.-L. Ma. Fast retrieval of disk-resident unstructured volume data for visualization. *External Memory Algorithms and Visualization (DIMACS Book Series, American Mathematical Society)*, 50, 1999.

[30] P. Lindstrom. Out-of-core simplification of large polygonal models. *Proceedings of SIGGRAPH 2000*, pages 259–262, July 2000.

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

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

[33] K.-L. Ma and T. W. Crockett. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. *IEEE Parallel Rendering Symposium*, pages 95–104, November 1997.

[34] K.-L. Ma and T. W. Crockett. Parallel visualization of large-scale aerodynamics calculations: A case study on the cray t3e. *Symposium on Parallel Visualization and Graphics*, pages 15–20, October 1999.

[35] N. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, November 1990.

[36] J. Nieh and M. Levoy. Volume rendering on scalable shared-memory MIMD architectures. In *Workshop on Volume Visualization*, pages 17–24. ACM Press, October 1992.

[37] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. *IEEE Visualization '98*, pages 233–238, October 1998.

[38] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of SIGGRAPH 97*, pages 101–108, August 1997.

[39] M. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill, 1987.

[40] C. R. Ramakrishnan and C. Silva. Optimal processor allocation for sort-last compositing under BSP-tree ordering. In *SPIE Electronic Imaging, Visual Data Exploration and Analysis IV*, 1999.

[41] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In *IEEE Visualization '96*, October 1996.

[42] H.-W. Shen and C.R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization '95*, pages 143–150, October 1995.

[43] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. *IEEE Visualization '98*, pages 159–166, October 1998.

[44] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. *IEEE Visualization '99*, pages 371–378, October 1999.

[45] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):63–70, November 1990.

[46] C. Silva. Parallel volume rendering of irregular grids, Ph.D. thesis, Department of Computer Science, State University of New York at Stony Brook, 1996.

[47] C. Silva, A. Kaufman, and C. Pavlakos. PVR: High performance volume rendering. *IEEE Computational Science and Engineering (Special Issue on Visual Supercomputing)*, pages 18–28, Winter 1996.

[48] C. T. Silva and J. S. B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), April - June 1997.

[49] P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, April - June 2000.

[50] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. Partitioning and ordering large radiosity computations. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 443–450. ACM SIGGRAPH, ACM Press, July 1994.

[51] S. K. Ueng, K. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructed meshes. *IEEE Transations on Visualization and Computer Graphics*, 3(4):370–380, 1997.

[52] S. Uselton. Volume rendering for computational fluid dynamics: Initial results. Tech Report RNR-91-026, Nasa Ames Research Center, 1991.

[53] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. ACM Symp. on Comput. Geom.*, pages 212–220, 1997.

[54] J. S. Vitter. External memory algorithms and data structures. *External Memory Algorithms and Visualization (DIMACS Book Series, American Mathematical Society)*, 50, 1999.

[55] R. Westermann and T. Ertl. The VSbuffer: Visibility ordering of unstructured volume primitives by polygon drawing. *IEEE Visualization '97*, pages 35–42, November 1997.

[56] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 57–62, November 1990.

[57] J. Wilhelms and A. Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):275–284, July 1991.

[58] J. P. Wilhelms, A. Van Gelder, P. Tarantino, and J. Gibbs. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. *IEEE Visualization '96*, pages 57–64, October 1996.

[59] P. Williams. Parallel volume rendering finite element data. In *Proceedings of Computer Graphics International*, 1993.

[60] P. L. Williams. Visibility-ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, April 1992.

[61] R. Yagel, D. M. Reed, A. Law, P.-W. Shih, and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. *1996 Volume Visualization Symposium*, pages 55–62, October 1996.

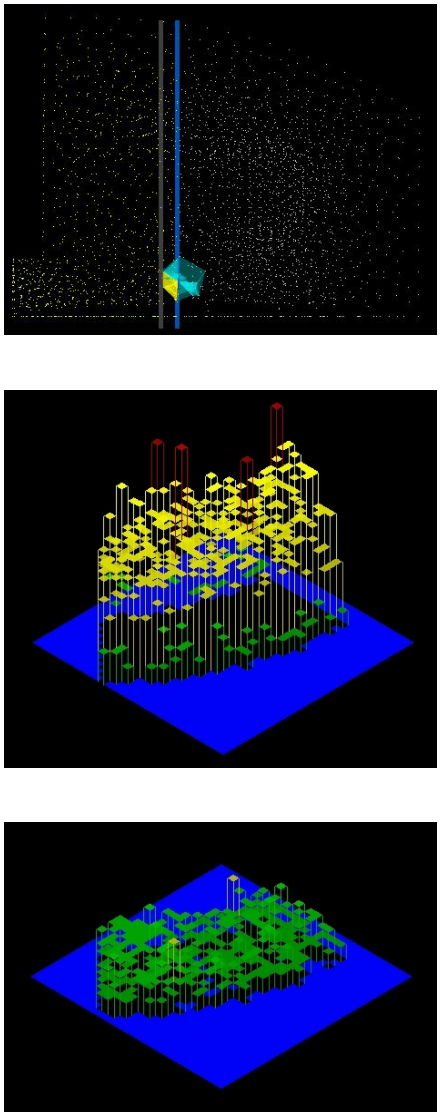

Figure 3: Illustration of the ZSweep algorithm. In the top figure, the sweeping plane is shown in blue and the plane determined by the *target-z* is shown in light gray. The sweeping direction is from right to left. Faces to be projected are shown in yellow, which lie ahead of the sweeping plane. The middle and the bottom figures show the snapshots before and after the sweeping plane hits the *target-z*, and the image plane is shown in blue. The length of the intersection lists over each pixel is represented by the height of the columns, colored with the following scheme: green is used for lists with fewer than six intersections, yellow from seven to 12, and red from 13 to 18.

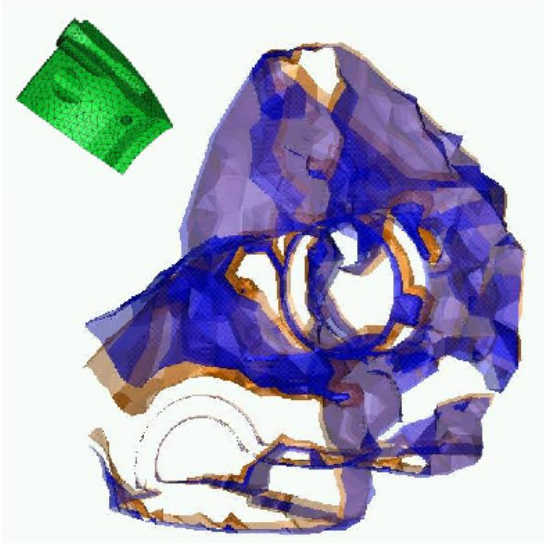

Figure 4: A few isosurfaces extracted from the SPX dataset, each rendered with a distinct semi-transparent color. The upper-left corner shows the bounding surface of the dataset.

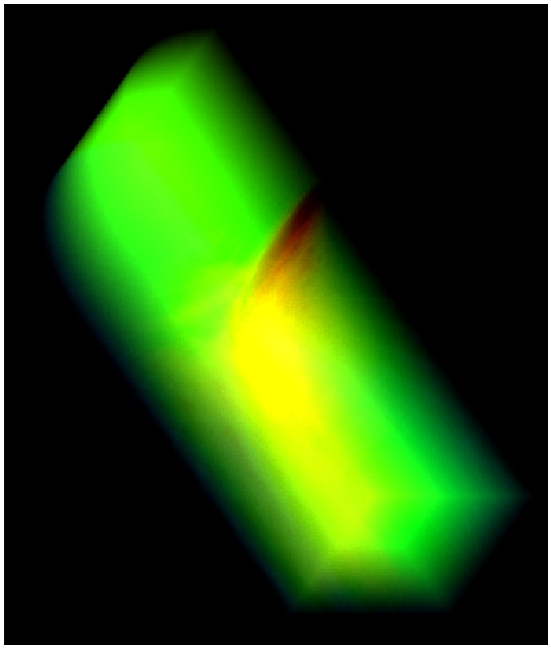

Figure 5: A 512 $\times$ 512 image of Blunt Fin.