Visualizing Very Large-Scale Earthquake Simulations

Kwan-Liu Ma Aleksander Stompel
Department of Computer Science
University of California at Davis
{ma,stompel}@cs.ucdavis.edu

Jacobo Bielak Omar Ghattas Eui Joong Kim
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
{bielak,oghattas,ejk}@cs.cmu.edu

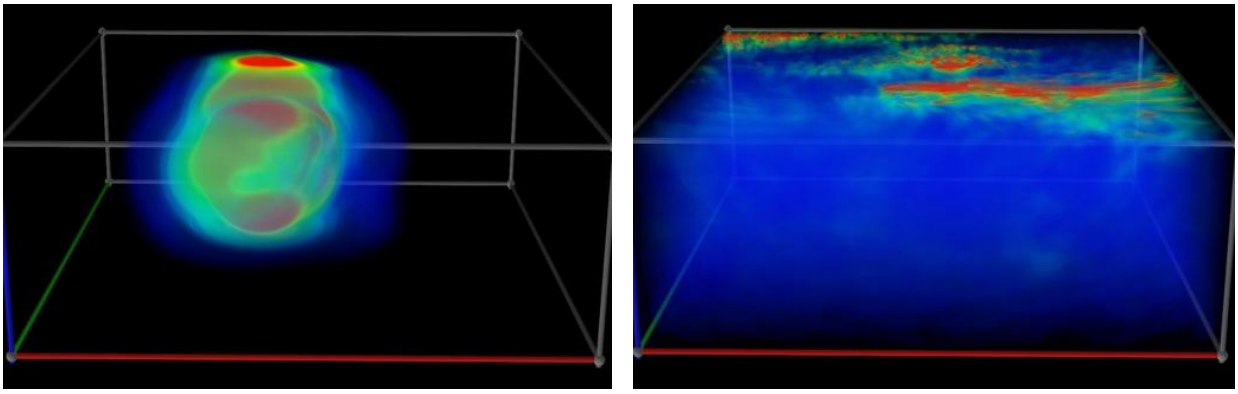

Figure 1: Volume visualization of simulation of earthquake-induced ground motion with 11.5 millions hexahedral elements. Left: time step 50 (actual time = 4 seconds). Right: time step 500 (actual time = 40 seconds).

Abstract

This paper presents a parallel adaptive rendering algorithm and its performance for visualizing time-varying unstructured volume data generated from large-scale earthquake simulations. The objective is to visualize 3D seismic wave propagation generated from a 0.5 Hz simulation of the Northridge earthquake, which is the highest resolution volume visualization of an earthquake simulation performed to date. This scalable high-fidelity visualization solution we provide to the scientists allows them to explore in the temporal, spatial, and visualization domain of their data at high resolution. This new high resolution explorability, likely not presently available to most computational science groups, will help lead to many new insights. The performance study we have conducted on a massively parallel computer operated at the Pittsburgh Supercomputing Center helps direct our design of a simulation-time visualization strategy for the higher-resolution, 1Hz and 2 Hz, simulations.

Keywords: earthquake modeling, high-performance computing, massively parallel supercomputing, scientific visualization, parallel rendering, time-varying data, unstructured grids, volume rendering, wave propagation

SC'03, November 15-21, 2003, Phoenix, Arizona, USA Copyright 2003 ACM 1-58113-695-1/03/0011...$5.00

1 Introduction

We present a new parallel rendering algorithm for the visualization of time-varying unstructured volume data generated from very large-scale earthquake simulations. The algorithm is used to visualize 3D seismic wave propagation generated from a 0.5 Hz simulation of the Northridge earthquake, which is the highest resolution volume visualization of an earthquake simulation performed to date.

Reduction of the earthquake risk to the general population can be achieved through understanding gained by computer modeling of earthquake-induced ground

motion in large heterogeneous basins and analysis of structural performance resulting from the soil-structure interaction. The simulation results can guide development of more rational seismic provisions for building codes, leading to safer, more efficient, and economical structures in earthquake-prone regions. However, a complete quantitative understanding of strong ground motion in large basins requires a simultaneous consideration of 3D effects of earthquake source, propagation path, and local site conditions. The large scale associated with the modeling places enormous demands on computational resources.

The Quake Project [21, 3] comprises a multidisciplinary team of researchers with the goal of developing the tools to model ground motion and structural response in large heterogeneous basins, and apply these tools to characterize the seismic response of large populated basins such as Los Angeles. To model at the needed scale and accuracy, the Quake group has created some of the largest unstructured finite element simulations ever performed by utilizing massively parallel supercomputers. Consequently, a serious challenge the Project faces is in visualization of the output of these very large, highly unstructured simulations. An important component of understanding earthquake wave propagation is the ability to volume render the time history of displacement and velocity fields. However, interactive rendering of time-dependent unstructured hexahedral datasets with $10^7 - 10^8$ elements (anticipated to grow to 10^9 over the next several years) is a major challenge. Past visualizations were limited to downsized versions of the data on a regular grid. The development of algorithms and software for parallel rendering of unstructured hexahedral datasets that scale to the very large grid sizes required will significantly assist the Quake group's ability to interpret and understand earthquake simulations.

In this paper, we discuss plausible approaches to high-resolution visualization of large-scale parallel earthquake simulations, describe a new parallel rendering algorithm we have developed to enable interactive visualization of time-varying unstructured data, and present our test results from rendering an 11.5 million hexahedral element simulation of the 1994 Northridge earthquake in the greater LA basin. Our present implementation takes about 2-10 seconds to directly render one time step using up to 128 processors of a parallel supercomputer operated at the Pittsburgh Supercomputing Center. If adaptive rendering is used, 3-4 times improvement can be obtained while maintaining visually indistinguishable results. With the test results obtained thus far, we are able to identify places for further optimization. Our goal is to achieve without employing adaptive rendering minimally 4-5 frames per second for 512×512 pixels, and about 1 frame per second for 1024×1024 pixels when using 128 processors. Such an interactive, high-resolution visualization capability, which was not previously available to the Quake project team, will greatly improve their understanding of the modeled phenomena.

2 Earthquake Ground Motion Modeling

Modeling and forecasting earthquake ground motion in large basins is a highly challenging and complex task. The complexity arises from several sources. First, multiple spatial scales characterize the basin response: the shortest wavelengths are measured in tens of meters, whereas the longest measure in kilometers, and basin dimensions are on the order of tens of kilometers. Second, temporal scales vary from the hundredths of a second necessary to resolve the highest frequencies of the earthquake source up to a couple of minutes of shaking within the basin. Third, many basins have highly irregular geometry. Fourth, the soils' material properties are highly heterogeneous. Fifth, strong earthquakes give rise to nonlinear material behavior. And sixth, geology and source parameters are only indirectly observable, and thus introduce uncertainty into the modeling process.

Simulating the earthquake response of a large basin is accomplished by numerically solving the partial differential equations (PDEs) of elastic wave propagation [2]. An unstructured mesh finite element method is used for spatial approximation, and an explicit central difference scheme is used in time. The mesh size is tailored to the local wavelength of propagating waves via an octree-based mesh generator [24]. Even though using an unstructured mesh may yield three orders of magnitude fewer equations than with structured grids, a massively parallel computer still must be employed to solve the resulting dynamic equations.

The Quake group is currently running earthquake simulations in the greater LA basin to 10 meters finest resolution with 100 million unstructured hexahedral finite elements, which is a factor of 4000 smaller than a regular grid would require. These include simulations of the 1994 Northridge mainshock to 1 Hz resolution, the highest resolution obtained to date. Despite the large degree of irregularity of the meshes, their codes are highly efficient: they regularly obtain close to 90% parallel efficiency in scaling up from 1 to 2048 processors on the HP/Compaq AlphaServer-based parallel system at the Pittsburgh Supercomputing Center. Node performance is also excellent for an unstructured mesh code, permitting sustained throughputs of nearly one teraflop per second on 2048 processors. A typical simulation requires 25,000 time steps to simulate 40 seconds

of ground shaking, and requires wall-clock time on the order of several hours, depending on the material damping model used, size of the region considered, number of processors (between 512 and 2048), and output statistics required.

3 Visualization Challenges and Solutions

A typical dataset generated by the ground motion simulation may consist of thousands of time steps and the spatial domain is composed of 10-100 million elements. Each mesh node outputs six values, three displacement components and three velocity components. The corresponding visualization challenges include:

- Large data
- Time varying data
- Unstructured mesh
- Multiple variables
- Vector and displacement fields

The work reported in this paper addresses the first three challenges. Visualizations of multivariate and multidimensional data are left as future work.

3.1 Large data

Our approach to the large data problem is to distribute both the data and visualization calculations to multiple processors of a parallel computer. In this way, we can not only visualize the dataset at its highest resolution but also achieve interactive rendering rates. The parallel rendering algorithm used thus must be highly efficient and scalable to a large number of processors because of the size of the dataset. Ma and Crockett [16] demonstrate a highly efficient, cell-projection volume rendering algorithm using up to 512 T3E processors for rendering 18 millions tetrahedral elements from an aerodynamic flow simulation. They achieve over 75% parallel efficiency by amortizing the communication cost as much as possible and using a fine-grain image space load partitioning strategy. Parker et al. [20] use ray tracing techniques to render images of isosurfaces. Although ray tracing is a computationally expensive process, it is highly parallelizable and scalable on shared-memory multiprocessor computers. By incorporating a set of optimization techniques and advanced lighting, they demonstrate very interactive, high quality isosurface visualization of the Visible Woman dataset using up to 124 nodes of an SGI Reality Monster with 80%-95% parallel efficiency. Wylie et al. [25] show how to achieve scalable rendering of large isosurfaces (7-469 million triangles) and a rendering performance of 300 million triangles per second using a 64-node PC cluster with a commodity graphics card on each node. The two key optimizations they use are lowering the size of the image data that must be transferred among nodes by employing compression, and performing compositing directly on compressed data. Bethel et al. [4] introduce a very unique remote and distributed visualization architecture as a promising solution to very large scale data visualization.

3.2 Time-varying data

Visualizing time-varying data presents two challenges. The first is the need to periodically transfer sequences of time steps to the processors from disk through a data server. The second is the need of an exploration mechanism accompanied by an appropriate user interface for tracking and correct interpretation of the temporal aspects of the data. We have mainly looked into the I/O issues and aim to hide the I/O cost to reduce interframe delay. For interactive browsing in both the spatial and temporal domains of the data, a minimum of 2-5 frames per second is needed. McPherson and Maltrud [18] develop a visualization system capable of delivering realtime viewing of large time-varying ocean flow data by exploiting the high performance volume rendering of texture mapping hardware of four InfiniteReality pipes attached to an SGI Origin 2000 with enough memory to hold thousands of time steps of the data. The ParVox system [9] is designed to achieve interactive visualization of time-varying volume data in a high-performance computing environment. Highly interactive splatting-based rendering is achieved by overlapping rendering and compositing, and by using compression.

A survey of time-varying data visualization strategies developed more recently is given in [13]. One very effective strategy is based on a hardware decoding technique such that data stay compressed until reaching the video memory for rendering [10]. Even though encoding methods can significantly reduce the data size, we cannot afford the cost of encoding the raw data since our ultimate goal is to support simulation-time visualization. In the absence of high-speed network and parallel I/O support, a particularly promising strategy for achieving interactive visualization is to perform pipelined rendering. Ma and Camp [14] show that by properly grouping processors according to the rendering loads, compressing images before delivering, and completely overlapping uploading each time step of the data, rendering, and delivering the images, interframe delay can be kept to a minimum. Garcia and Shen [6] develop a dynamic load balancing strategy based on asynchronous communication for more efficiently rendering time-varying volume data on a PC cluster. Improved load balancing is achieved by cleverly and dynamically distributing the

image compositing job.

3.3 Unstructured-grid data

To efficiently visualize unstructured data additional information about the structure of the mesh needs to be computed and stored, which incur considerable memory and computational overhead. For example, ray tracing rendering needs explicit connectivity information for each ray to march from one element to the next [11]. Ma and Crockett [15] present a parallel cell projection rendering algorithm which requires no connectivity information. Since each tetrahedral element is rendered completely independent of other elements, data distribution can be done in a more flexible manner facilitating load balancing. A similar approach is also used for the rendering of AMR data [12]. Chen, Fujishiro, and Nakajima [5] present a hybrid parallel rendering algorithm for large-scale unstructured data visualization on SMP clusters such as the Hitachi SR8000. The three-level hybrid parallelization employed consists of message passing for inter-SMP node communication, loop directives by OpenMP for intra-SMP node parallelization, and vectorization for each processor. A set of optimization techniques are used to achieve maximum parallel efficiency. In particular, due to their use of an SMP machine, dynamic load balancing can be done effectively. However, their work does not address the problem of rendering time-varying data.

3.4 Visualization strategies for large-scale earthquake simulations

In summary, the following common strategies have been used to successfully achieve high performance rendering of large time-varying unstructured datasets using parallel computers:

- interleaving load distribution to achieve better load balancing;
- avoiding per-time-step preprocessing calculations as much as possible;
- overlapping communication and computation to hide data transfer overhead;
- buffering of intermediate results to amortize communication overheads; and
- compressing data to lower communication cost.

We design our visualization solutions by closely following these guidelines.

Our ultimate goal is to perform simulation-time visualization allowing scientists to monitor the simulation, make immediate decision on data archiving and visualization production, and even steer the simulation. To achieve such an ambitious goal, we start by first developing a highly efficient parallel visualization algorithm that is capable of delivering interactive rendering of 10-100 millions data elements, scalable to large MPP systems, and easily coupled with extended capabilities such as vector field rendering. At this stage, batch mode rendering is used to produce animations for the scientists for playback. The subsequent task is to design appropriate user interface and interaction techniques for interactive browsing in both the spatial and temporal domains of the data. Scientists therefore can conduct interactive data exploration on their desktop. Finally, the parallel simulation and renderer will run simultaneously on either the same machine or two different machines connected with high-speed network interconnect such as the Quadrics network which has a bandwidth over 300 megabytes per second, permitting remote interaction with the simulation and visualization. Simulation data and image data are stored on demand.

In this paper, we report the performance of our parallel renderer design on LeMieux, an HP/Compaq AlphaServer with 3,000 processors operated at the Pittsburgh Supercomputing System, and visualizations of time-varying ground motion simulation data consisting of 11.5 million hexahedral elements. The rest of the paper describers the rendering algorithm, its performance, and our simulation-time visualization strategy for higher-resolution simulations.

4 The Parallel Rendering Algorithm

Our new parallel rendering algorithm performs a sequence of tasks as shown in Figure 2. Each simulation run generates a set of data files. While the number of files is the same as the number of processors (hundreds to thousands) used to run the simulation, the number of processors (tens to hundreds) used for visualization calculations is selected based on the rendering performance requirements. Since the mesh structure never changes throughout the simulation, for each resolution level a preprocessing step is done to generate a spatial (octree) encoding of the raw data. When performing the rendering, the host loads the raw data from disk and uses this octree to distribute block of hexahedral elements among processors. Each block of elements is associated with a subtree of the global octree. This subtree is delivered to the assigned processor for the corresponding block of data only once at the beginning since all time steps data use the same subtree structure. This centralized data distribution method is not ideal. Hardware parallel I/O support using multiple file servers should be used for both the postprocessing visualization and simulation-time visualization.

After blocks of data are distributed and before rendering begins, each processor conducts a view-dependent

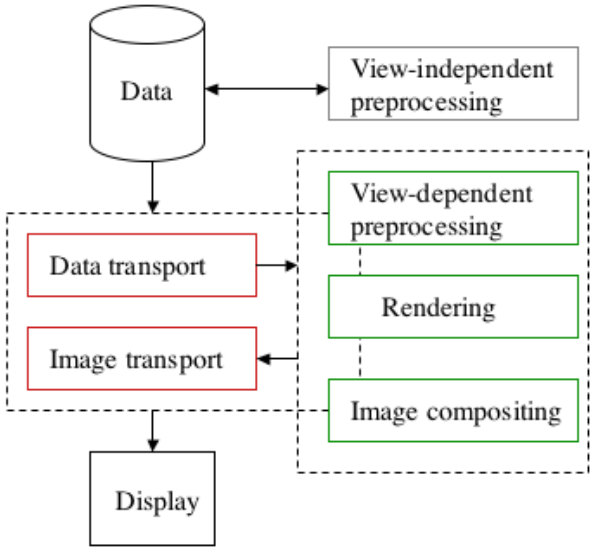

Figure 2: The parallel rendering pipeline.

preprocessing step whose cost is very small and thus negligible. Rendering can be broken into a ray-casting block rendering step and an image compositing step while data blocks for subsequent time steps are continuously transferred from disk to each processor in the background. Overlapping data transport and rendering helps lower interframe delay.

4.1 Adaptive rendering

Rendering cost can be cut significantly by moving up the octree and rendering at coarser level blocks instead. This is done for maintaining the needed interactivity for exploring in the visualization parameter space and the data space. A good approach is to render adaptively by matching the data resolution to the image resolution while taking into account the desired rendering rates. For example, when rendering tens of millions elements to a 512×512 pixels image, unless a close-up view is selected, rendering at the highest resolution level would not reveal more details. One of the calculations that the view-dependent preprocessing step performs is to choose the appropriate octree level. The saving from such an adaptive approach can be tremendous and there is virtually very little impact on the level of information presented in the resulting images as shown in Figure 3. Presently the appropriate level to use is computed based on the image resolution, data resolution, and a user-specified number that limits the number of elements allowed to be projected into a pixel.

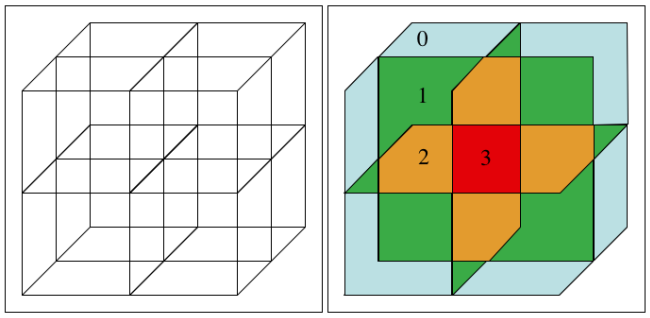

Figure 4: Left: Projection of the bounding boxes of eight subvolumes. Right: The image space is partitioned into areas according to the number of overlaps. In addition to the background, there are areas with no overlap (blue), one overlap (green), two overlaps (orange), three overlaps (red), etc.

4.2 Parallel image compositing

The parallel rendering algorithm is sort-last which thus requires a final compositing step involving interprocessor communication. Several parallel image compositing algorithms are available [17, 8, 1] but their efficiency is mostly limited to the use of specific network topology or number of processors. We have developed an optimized version of the *direct send* compositing method, which offers maximum flexibility and performance. The direct send method has each processor send pixels directly to the processor responsible for compositing them. This approach has been used in [7, 19, 15] because it is easy to implement and does not require a special network topology. With direct send compositing, in the worst case there are $n(n-1)$ messages to be exchanged among n compositing nodes. For low-bandwidth networks care should be taken to avoid that many nodes try to send messages to the same node at the same time.

Our new image compositing algorithm, which we call SLIC [23], uses a minimal number of messages to complete the parallel compositing task. The optimizations are achieved by refining the direct send method based on the following observation. After local rendering is done by each processor, there are three types of pixels: background pixels, pixels in the nonoverlapping areas, and pixels in the overlapping areas. Background pixels can be ignored. Pixels in the nonoverlapping areas can be delivered directly to the host or display device. Only the pixels in the overlapping areas need to be sent to the processors responsible for compositing the corresponding areas. Figure 4 shows pixels classification as a result of a particular projection. Once pixels are classified, an optimized compositing schedule for all processors and respective assignments can be computed. Note that each processor is assigned within the image space

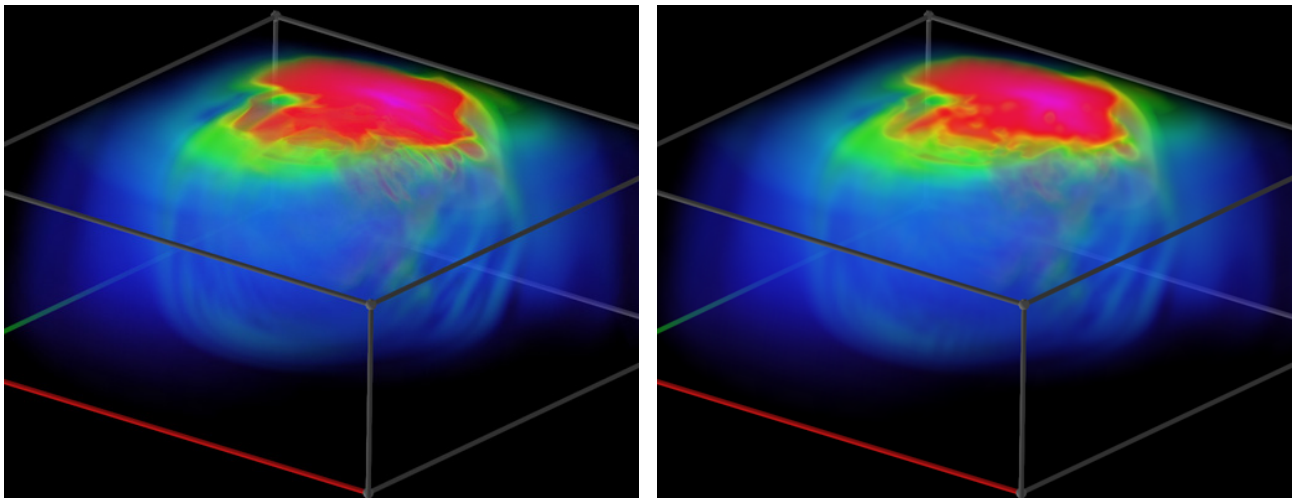

Figure 3: Left: high-resolution rendering (level 11). Right: Adaptive rendering (level 6). The image on the right provides enough high level information about the data while it can be generated about 3 times faster than the high resolution one.

it rendered into. With direct send or binary swap, a processor could be assigned compositing regions that it was not involved with in rendering, which results in additional sends. Reducing the number of messages that must be exchanged among processors should be beneficial since it is generally true that communication is more expensive than computation.

Whenever view is changed each processor computes the compositing schedule independent of other processors. The schedule is determined based on the overlapping relations between the projection of the local blocks of volume and the projection of other blocks. Because a rectangular-block data partitioning based on the octree is used, each processor also knows the exact projection of nonlocal blocks. Specifically, each node performs the following steps:

1. projecting corner vertices of each block based on the current view and constructing its convex hull,
2. traversing through the overlapped convex hulls in scanline order to identify compositing tasks in terms of spans, and
3. assigning each span to a node in an interleaving fashion.

The convex hull defines the exact projected area of the block in the image space. A scanline algorithm similar to polygon scan-conversion is then used to process the edges of the overlapped convex hulls. Note that each node only needs to scan the projected bounding edges of the local blocks. The projected bounding edges of nonlocal blocks are used to determine the number of overlaps.

The edges that each scanline intersects break the scanline into multiple spans, which can be classified into: background spans, no-overlap spans, one-overlap spans, two-overlap spans, etc. Background spans are never generated. No-overlap spans are sent directly to the host processor. The rest of spans are either kept locally or delivered to other processors by following Step 3. In our current implementation, the processor assignment is determined by $((x+y)\times p)\texttt{mod}n$ where x and y are the coordinates of the starting position of the span, p is a large prime number, and n is the total number of compositing nodes used. This interleaving approach gives us good compositing load balancing.

The scanline-based algorithm works because the bounding edges break the scanlines in exactly the same way across all processors. As a result, if two spans created by different processors would overlap, they must completely overlap; that is, the two spans have the same starting and end screen positions. Because all blocks are presorted in depth order, each span can be assigned a compositing order, which simplifies the actual compositing calculations. Other information stored with each span includes a sequence of RGBA values and the starting and end screen coordinates of the span.

This preprocessing step to compute a compositing schedule for each new view introduces very low overhead, generally under 10 milliseconds [23]. With the resulting schedule, the total amount of data that must be sent over the entire network to accomplish the compositing task is minimized. According to our test results, SLIC outperforms previous algorithms, especially when rendering high-resolution images, like 1024×1024 pixels or larger. Since image compositing contributes to the parallelization overheads, reducing its cost helps improve parallel efficiency.

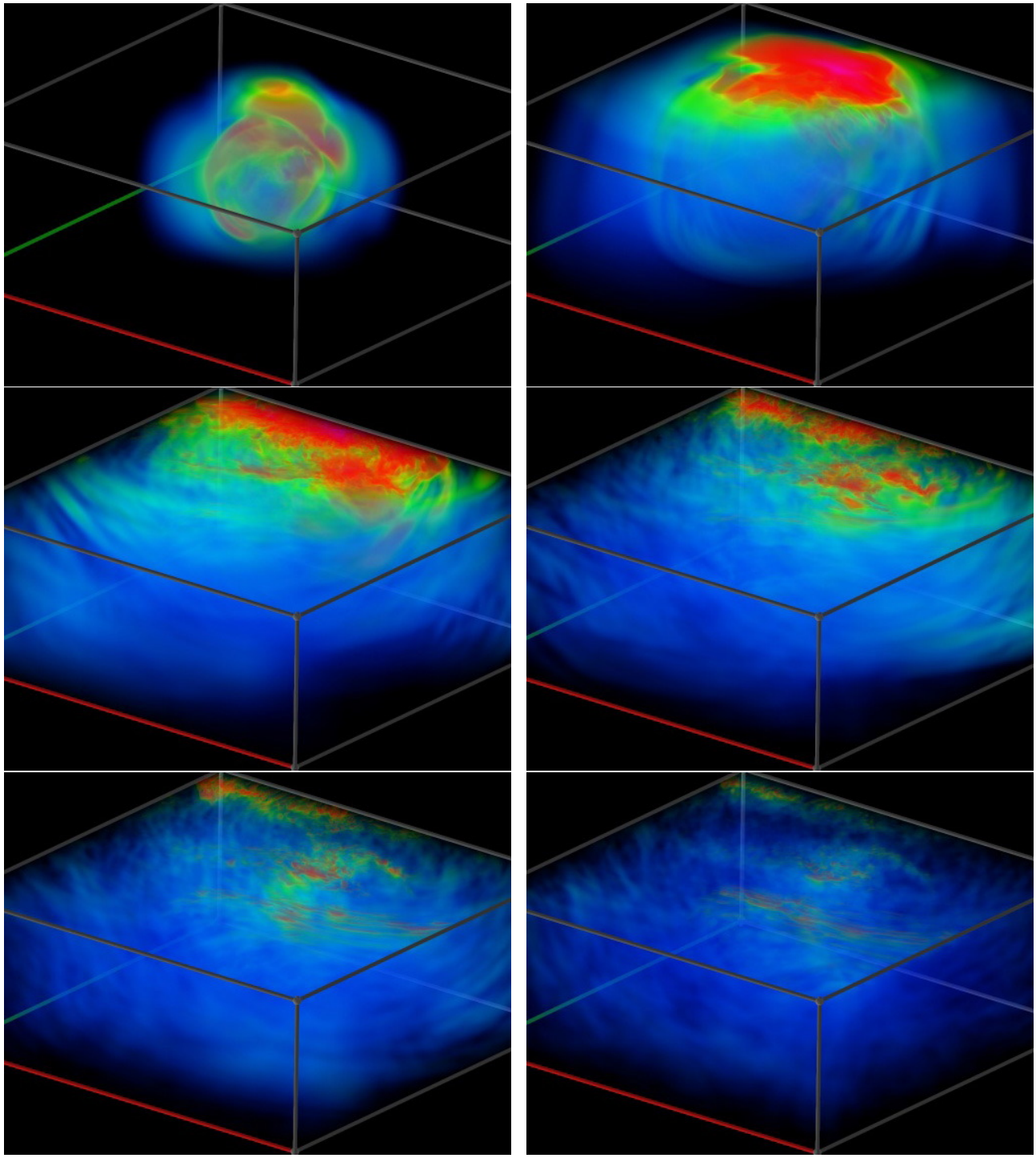

Figure 5: Selected frames from an animation of the ground motion simulation. At beginning, the seismic wave is traveling towards the free surface. The top left image shows the motion once the seismic waves have hit that surface. And subsequent images show the evolution of the seismic motion, primarily concentrated near the surface (surface waves) until it eventually dies out.

5 Test Results

We have studied the performance of our rendering algorithm using up to 128 processors of an HP/Compaq AlphaServer which is an SMP massively parallel supercomputer. The performance study results provide the Quake project team and PSC directions for configuring hardware systems for the large-scale simulation-time visualization we intend to conduct next. Figure 5 displays selected frames from an animation of a simulation of the Northridge earthquake with 11.5 million elements. At this resolution, scientists are able to observe fine scale volumetric details that they have been unable to visualize before.

We first present the performance of the renderer without counting the cost of uploading the volume data. This was done by rendering a selected time step to 512×512 pixels from 120 different view angles and computing the average rendering time. Figure 6 shows plots of the time each processor took to complete the rendering when using 8, 16, 32, and 64 processors. Overall, each chart also displays how load distribution was done for the number of processors used. As the charts reveal, our current implementation does not scale beyond 32 processors. Parallel efficiency drops under 60 percent when 64 or more processors are used. This is mainly due to the load distribution scheme used and the constant compositing cost. Nevertheless, we are able to achieve a rendering rate at two frames per second when using 64 processors.

According our test tests, the image compositing time varies according to view and image resolution rather than the number of processors used. For most of the cases, the image compositing time is under 100 milliseconds. Figure 7 shows the compositing costs for 60 different view points and eight different numbers of processors used to render to 2048×2048 pixels.

Figure 8 shows how the rendering cost varies as viewpoint changes. The difference can be quite dramatic. Certain view points would make load balancing more difficult, resulting in longer rendering time. Figure 9 shows the time for rendering to 1024×1024 pixels, and exhibit a similar performance trend.

Next we present the time for performing temporal animation using nonlocal disk. That is, each time step of volume data is subsequently transmitted from a nonlocal disk to the parallel computer for rendering. Figure 10 shows the time for rendering a sequence of 500 time steps. The image resolution is 512×512 pixels. At the beginning, the cost is much lower than later time steps because a relatively small image area is drawn as a result of the modeled earthquake phenomena. To alleviate the data loading cost, 32 data servers were used. Plus overlapping the data loading with the rendering calculations as much as possible, we are able to achieve satisfactory rendering rates, near one frame per

Figure 6: Breakdowns of average rendering time when using 8, 16, 32, and 64 processors to generate a spatial-domain animation for a selected time step. Image size is 512×512. Rendering calculations time is colored red. All others contribute to compositing.

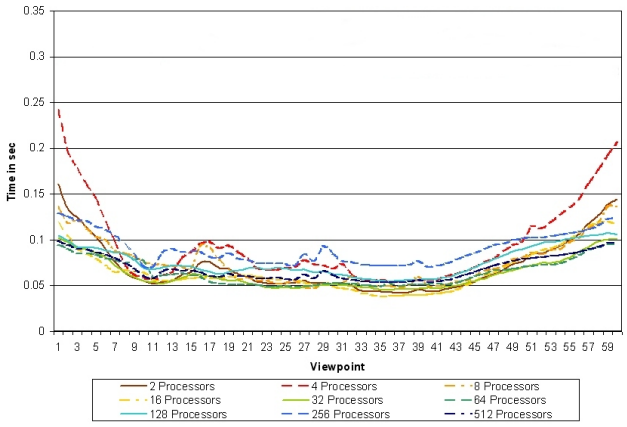

Figure 7: Compositing costs for 60 different view points and eight different number of processors used to render to 2048×2048 pixels.

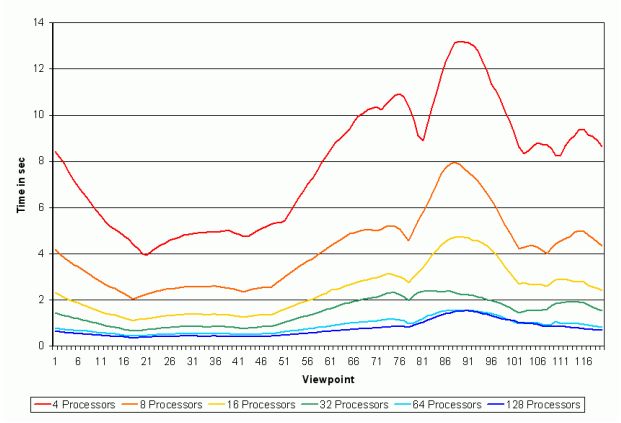

Figure 8: Rendering to 512×512 pixels from different view points with different numbers of processors.

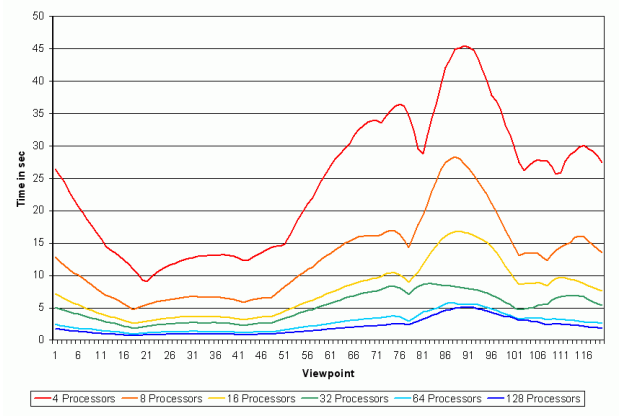

Figure 9: Rendering to 1024×1024 pixels from different view points with different numbers of processors.

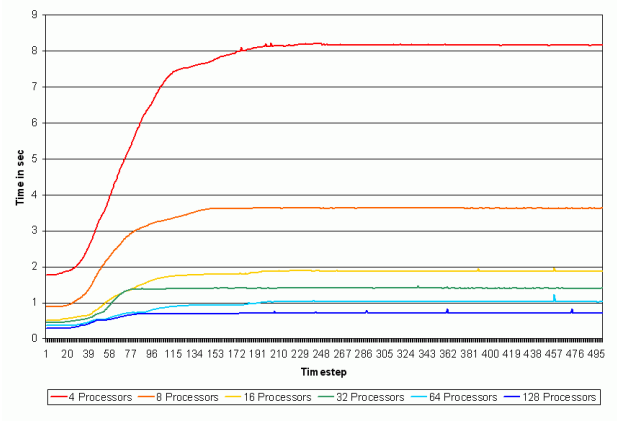

Figure 10: Rendering time for temporal domain animation using a non-local disk. Multiple data servers were used to alleviate the I/O bottleneck. Image size is 512×512 pixels.

second using 64 processors. Finally, Figure 11 shows inter-frame delay. When rendering to 1024×1024 pixels, using 64 processors it takes about 3-9 seconds to complete each frame.

Because of the large dynamic range of the data, it often becomes difficult to follow time-varying phenomena. For example, in Figure 5, only early time steps are displayed because after the 400th time step, direct volume rendering reveals very little variation in the domain without modifying the opacity mapping used. Figure 12 displays the results of employing a new temporal domain filtering method to enhance the wave propagation throughout the whole time period. Time steps between 50 and 600 are shown with an interval=50. The enhancement is done locally by using values in either previous or next time step, or both. As a result, both large-scale and small-scale wave propagation are captured in the picture. The cost of this enhancement is small and negligible. The user can turn the enhancement on and off during interactive viewing to ensure a correct interpretation of the data.

6 Conclusions

We have experimentally studied a new parallel rendering algorithm for the visualization of large-scale earthquake simulations. This parallel visualization solution incorporates adaptive rendering, a new parallel image compositing algorithm, and a data transferring scheme to make possible efficient rendering of large-scale time-varying data. Our performance study using up to 128 processors of LeMieux at the PSC shows promising results, and also reveals the interplay between data transport strategy used and interframe delay.

Based on our test results, the two main subjects of

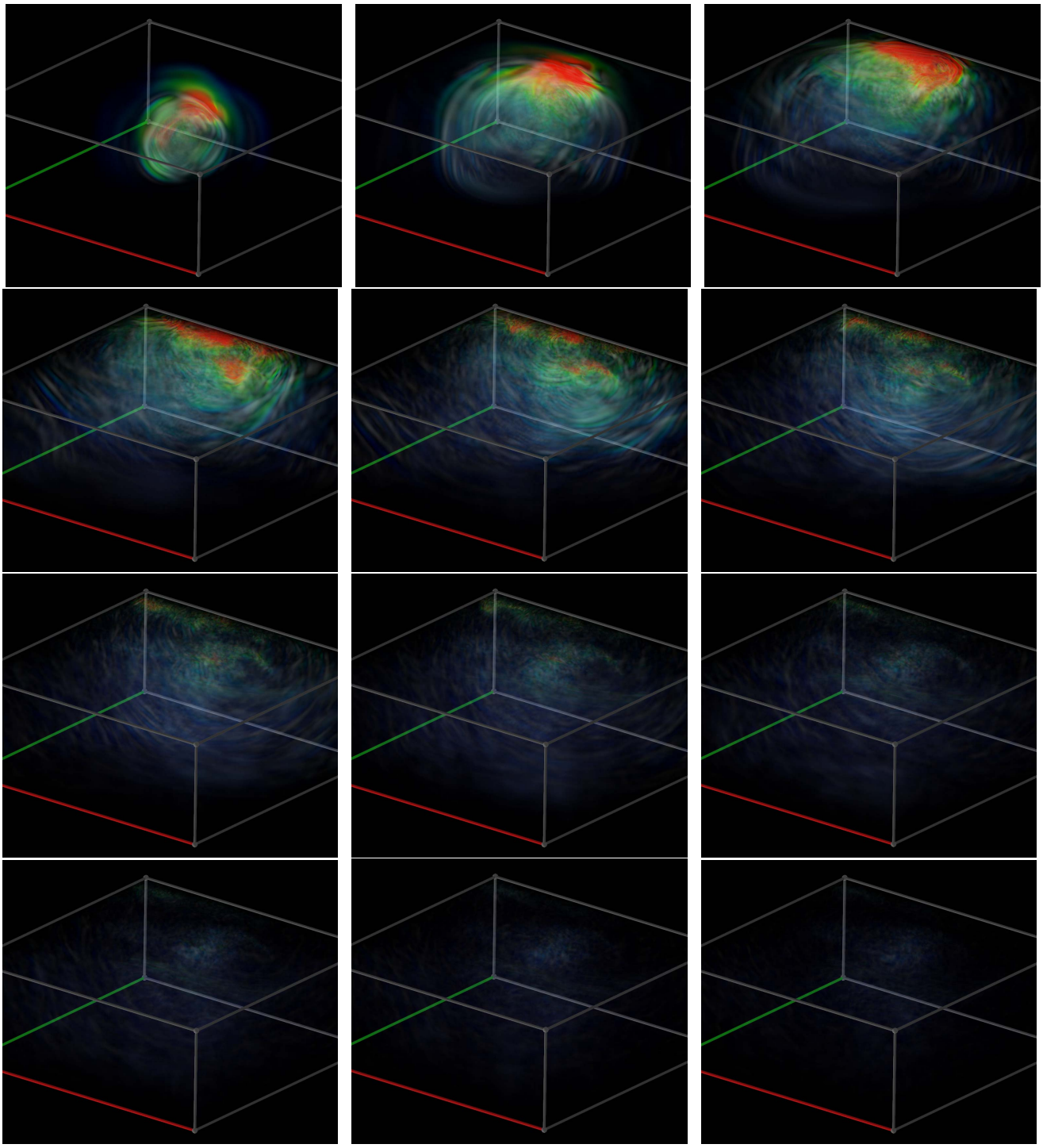

Figure 12: Selected frames from an animation of the earthquake simulation. Enhancement was used to bring out the wave propagation at different scales. At beginning, the seismic wave is traveling toward the free surface. The second row of images show the motion once the seismic waves have hit that surface. And subsequent images show the evolution of the seismic motion, primarily concentrated near the surface (surface waves) until it eventually dies out. Because of the enhancement, even at the later time steps, the images capture the remaining scattering of the waves.

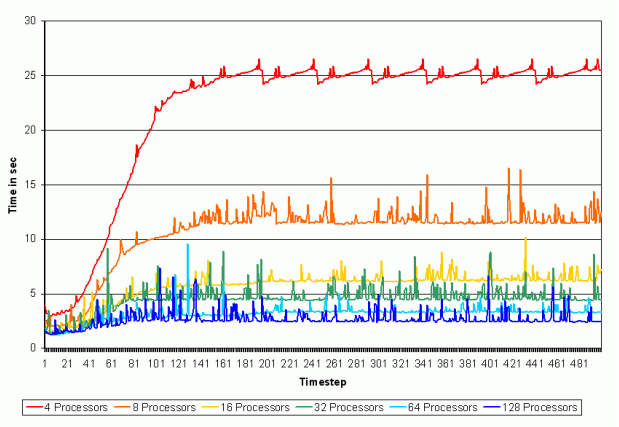

Figure 11: Interframe delay which includes both I/O and rendering cost for temporal domain animation using a non-local disk. Multiple data servers were used to alleviate the I/O bottleneck. Image size is 1024×1024 pixels.

study are load balancing and I/O. We plan to investigate a fine grain load redistribution method and study how to reduce its overhead as much as possible. We have demonstrated that using multiple data servers helps alleviate, but not remove, the I/O bottleneck. The PSC will configure a testbed for our project especially addressing the I/O problem, which will allow us to focus our effort more on other aspects of the overall visualization problem such as load balancing, image compositing, user interfaces, and others. Presently the image compositing cost is about constant. We believe compression can help lower communication cost to possibly make the overall compositing scalable to large machine size. Our preliminary test results show a 50% reduction in the overall image compositing time with compression.

We have not exploited the SMP features of LeMieux, which we believe could allow us to accelerate the rendering calculations while reducing communication cost. The result will be a more scalable renderer offering higher frame rates.

Adaptive rendering will play a major role in our subsequent work. As shown previously, the full rendering and adaptive rendering can result in visually indistinguishable results but the saving in rendering cost can be tremendous. Our study in this direction will focus on how adaptive rendering can be done with minimal user intervention and perception of level switching.

To perform simulation-time visualization, we anticipate no change in the core rendering code. Rather, a coordination between data streaming, rendering, image delivery, and user feedback must be established. A buffering mechanism is likely needed for the user to conduct spatial domain exploration of a selected time step, which would defer the rendering of incoming time steps.

To complicate the problem further, it could be desirable to create a single visualization by making use of multiple variables and/or multiple time steps [22]. Other important problems which we cannot ignore include user interface, extracting temporal features of the data, and vector data visualization.

This paper contributes to the supercomputing community in the following two ways. First, we have demonstrate visualization of large-scale time-varying unstructured volume data sets at near interactive rates. Ma and Crockett show rendering of 18 million tetrahedral elements while Chen, Fujishiro, and Nakajima of the Japan's Earth Simulator project perform rendering of 7.9 million mix-type elements, neither of which address time-varying aspects of the visualization problem. Second, the adaptive high-fidelity visualization solution we provide to the scientists will allow them to explore in the temporal, spatial, and visualization domain of their data at high resolution. This new high resolution explorability, likely not presently available to most computational science groups, will help lead to many new insights. Finally, the new temporal enhancement technique we introduce provides scientists an alternative way to understand the time-varying phenomena they model.

Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE award) and CMS-9980063, and the U.S. Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 and No. B523578. Pittsburgh Supercomputing Center (PSC) provided time on their parallel computers through AAB grant BCS020001P. Thanks especially go to John Urbanic for his assistance on setting up the needed system support at PSC.

References

[1] Ahrens, J., and Painter, J. Efficient sort-last rendering using compression-based image compositing. In *Proceedings of the 2nd Eurographics Workshop on Parallel Graphics and Visualization* (1998), pp. 145–151.

[2] Bao, H., Bielak, J., Ghattas, O., Kallivokas, L. F., O'Hallaron, D. R., Shewchuk, J. R., and Xu, J. Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers. *Computer Methods in Applied Mechanics and Engineering 152*, 1–2 (Jan. 1998), 85–102.

[3] Bao, H., Bielak, J., Ghattas, O., O'Hallaron, D. R., Kallivokas, L. F., Shewchuk, J. R., and Xu, J. Earthquake ground motion modeling on parallel computers. In *Supercomputing '96* (Pittsburgh, Pennsylvania, Nov. 1996).

[4] Bethel, W., Tierney, B., Lee, J., Gunter, D., and Lau, S. Using high-speed wans and network data caches to enable remote and distributed visualization. In *Proceedings of Supercomputing 2C00* (November 2000).

[5] Chen, L., Fujishiro, I., and Nakajima, K. Parallel performance optimization of large-scale unstructured data visualization for the earth simulator. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (2002), pp. 133–140.

[6] Garcia, A., and Shen, H.-W. Asynchronous rendering for time-varying volume datasets on pc clusters. In *Proceedings of the IEEE Visualization 2003 Conference (to appear)* (October 2003).

[7] Hsu, W. M. Segmented ray casting for data parallel volume rendering. In *Proceedings of 1993 Parallel Rendering Symposium* (1993), pp. 7–14.

[8] Lee, T.-Y., Raghavendra, C. S., and Nicholas, J. B. Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. *IEEE Transactions on Visualization and Computer Graphics 2*, 3 (1996), 202–217.

[9] Li, P., Whitman, S., Mendoza, R., and Tsiao, J. ParVox – a parallel spaltting volume rendering system for distributed visualization. In *Proceedings of 1997 Symposium on Parallel Rendering* (1997), pp. 7–14.

[10] Lum, E., Ma, K.-L., and Clyne, J. A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 286–301.

[11] Ma, K.-L. Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. In *Proceedings of the Parallel Rendering '95 Symposium* (1995), pp. 23–30. Atlanta, Georgia, October 30-31.

[12] Ma, K.-L. Parallel rendering of 3D AMR data on the SGI/Cray T3E. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation* (1999), pp. ?–?

[13] Ma, K.-L. Visualizing time-varying volume data. *IEEE Computing in Science & Engineering 5*, 2 (2003), 34–42.

[14] Ma, K.-L., and Camp, D. High performance visualization of time-varying volume data over a wide-area network. In *Proceedings of Supercomputing 2000 Conference* (November 2000).

[15] Ma, K.-L., and Crockett, T. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In *Proceedings of 1997 Symposium on Parallel Rendering* (1997), pp. 95–104.

[16] Ma, K.-L., and Crockett, T. Parallel visualization of large-scale aerodynamics calculations: A case study on the cray t3e. In *Proceedings of 1999 IEEE Parallel Visualization and Graphics Symposium* (1999), pp. 15–20.

[17] Ma, K.-L., Painter, J. S., Hansen, C., and Krogh, M. Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics Applications 14*, 4 (July 1994), 59–67.

[18] McPherson, A., and Maltrud, M. Poptex: Interactive ocean model visualization using texture mapping hardware. In *Proceedings of the Visualization '98 Conference* (October 18-23 1998), pp. 471–474.

[19] Neumann, U. Communication costs for parallel volume-rendering algorithms. *IEEE Computer Graphics and Applications 14*, 4 (July 1994), 49–58.

[20] Parker, S., Parker, M., Livnat, Y., Sloan, P., and Hansen, C. Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (July-September 1999), 1–13.

[21] The Quake project, Carnegie Mellon University and San Diego State University. http://www.cs.cmu.edu/~quake.

[22] Stompel, A., Lum, E., and Ma, K.-L. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In *Proceedings of Pacific Graphics 2002 Conference* (2002), pp. 394–402.

[23] Stompel, A., Ma, K.-L., Lum, E., Ahrens, J., and Patchett, J. SLIC: scheduled linear image compositing for parallel volume rendering. In *Proceedings of IEEE Sympoisum on Parallel and Large-Data Visualization and Graphics (to appear)* (October 2003).

[24] Tu, T., O'Hallaron, D., and Lopez, J. Etree: A database-oriented method for generating large octree meshes. In *Proceedings of the Eleventh International Meshing Roundtable* (September 2002), pp. 127–138.

[25] Wylie, B., Pavlakos, C., Lewis, V., and Moreland, K. Scalable rendering on PC clusters. *IEEE Computer Graphics and Applications 21*, 4 (July/August 2001), 62–70.