Ashish Sharma
sharmaa@usc.edu
Collaboratory for Advanced Computing and Simulations
Dept. of Computer Science

Aiichiro Nakano
anakano@usc.edu
Dept. of Computer Science

Rajiv K. Kalia
rkalia@usc.edu
Dept. of Physics & Astronomy

Priya Vashishta
priyav@usc.edu
Dept. of Materials Science & Engineering
University of Southern California
Los Angeles, California 90089

Sanjay Kodiyalam
sanjay@rouge.phys.lsu.edu

Paul Miller
pmiller@bit.csc.lsu.edu

Wei Zhao
wzhao@ee.lsu.edu

Xinlian Liu
Concurrent Computing Laboratory for Material Simulations
Biological Computing and Visualization Center
Dept. of Computer Science
Dept. of Physics & Astronomy
Louisiana State University
Louisiana 70803, USA

Timothy J. Campbell
tjcamp@navo.tipc.mil

Andy Haas
haas@navo.tipc.mil
Naval Oceanographic Office Major Shared Resourse Center
Stennis Space Center
Mississippi 39529, USA

Presence, *Vol. 12, No. 1, February 2003,* 85–95

Immersive and Interactive Exploration of Billion-Atom Systems

Abstract

We have developed a visualization system, named *Atomsviewer,* to render a billion atoms from the results of a molecular dynamics simulation. This system uses a hierarchical view frustum culling algorithm based on the octree data structure to efficiently remove atoms that are outside of the field of view. A novel occlusion culling algorithm, using a probability function, then selects atoms with a high probability of being visible. These selected atoms are further tested with a traditional occlusion culling algorithm before being rendered as spheres at varying levels of detail. To achieve scalability, *Atomsviewer* is distributed over a cluster of PCs that execute a parallelized version of the hierarchical view frustum culling and the probabilistic occlusion culling, and a graphics workstation that renders the atoms. We have used *Atomsviewer* to render a billion-atom data set on a dual processor SGI Onyx2 with an InfiniteReality2 graphics pipeline connected to a four-node PC cluster.

1 Introduction

A molecular dynamics (MD) simulation follows the trajectories of atoms to study the behavior of materials. In the past, we have interactively explored simulated materials in an immersive environment to better understand and track atomic features responsible for macroscopic phenomena (Nakano et al., 2001a). For example, Figure 1 shows a scientist walking through a simulated fractured ceramic fiber composite material to investigate atomistic processes that make the material tough. However, our recent MD simulation involving one billion atoms (Nakano et al., 2001b), produced 100 GB of data per frame to record atomic species, positions, velocities, and stresses (Omeltchenko et al., 2000). Visualizing such a large data set interactively is a nontrivial task.

Considerable progress has been made in visualizing large data sets. One of the earliest works by Clark (1976) introduced a hierarchical representation of models, that provides a fast mechanism to minimize the polygons used in a frame through the use of varying level of detail (LOD). Most of the previous research on large data set visualization has focused either on volumetric data or architectural data. The Center for Computational Visualization at UT Austin led by Bajaj has visualized large volumetric data sets on stereoscopic display systems using data compression and parallel processing (Bajaj & Cutchin, 1999; Bajaj, Ihm, & Park, 2000). Hamann and Ma (Ma & Camp, 2000; Pinsky et al., 2000) at UC–Davis have developed multiresolution algorithms as well as parallel and distributed systems to accomplish large data set visualiza-

Figure 1. *A scientist investigating a fracture in a ceramic fiber composite material rendered on an* ImmersaDesk *virtual environment.*

tion. The Scientific Computing and Imaging Institute at the University of Utah led by Johnson (Johnson, Parker, Hansen, Kindlmann, & Livnat, 1999; Parker et al., 1999) have developed tools to visualize large computational fields with applications in areas such as biomedicine and oil exploration.

Another area that utilizes very large data sets in virtual environments is architectural modeling. Several projects (Airey, Rohlf, & Brooks, 1999; Funkhouser, Teller, Sequin, & Khorramabad, 1996; Teller & Sequin, 1992) subdivide the architectural model into cells that correctly map onto the rooms and other physical domains. For example, the UC–Berkley walkthrough of a building (SODA Hall) uses a hierarchical representation of the model and incorporates both visibility culling and LOD. More recent work at UNC–Chapel Hill (Aliaga et al., 1999) has used culling, data management, and rendering enhancements to achieve an interactive walkthrough for data sets involving millions of triangles. Funkhouser, Sequin, and Teller (1992) have proposed numerous techniques for the management of large data sets using adaptive display algorithms.

All these efforts have, however, focused on volume rendering and 3D modeling, and the optimization techniques developed for them do not easily apply to particle rendering. For example, in 3D modeling, the structure of the model can be used in culling and LOD control. This is not possible in data sets from molecular dynamics (MD) simulations, which are an irregular distribution of atomic position and attributes.

Previously, we developed a visualization system designed specifically for atomistic data sets. The system used octree-based visibility culling and multiresolution rendering to render atoms at varying LOD and interactively visualize a few million atoms, but not a billion atoms (Sharma et al., 2001). The primary reason for this shortcoming was a lack of computing power.

In this paper, we present our solution for visualizing a billion-atom dataset by using a hierarchical view frustum culling algorithm and a new occlusion culling algorithm that uses a visibility probability function. These algorithms are distributed over a PC cluster, and the resulting reduced data set is rendered on the graphics server. An overview of the newly developed system—called *Atomsviewer*—is given in section 2. Section 3, 4, and 5 provide detailed descriptions of its three major components, and the parallel and distributed architecture is discussed in section 6. Finally, numerical results and a summary are presented in section 7 and 8, respectively.

2 System Overview

Atomsviewer has three key components that are distributed over a networked environment. The three components are as follows.

- Data extraction module: DEM implements an octree-based hierarchical view frustum culling algorithm to efficiently remove atoms outside the frustum.
- Probabilistic occlusion module: POM implements a new occlusion culling algorithm that uses a probability function to select atoms with a high probability of being visible.
- Rendering and visualization module: RVM implements a traditional occlusion culling algorithm and

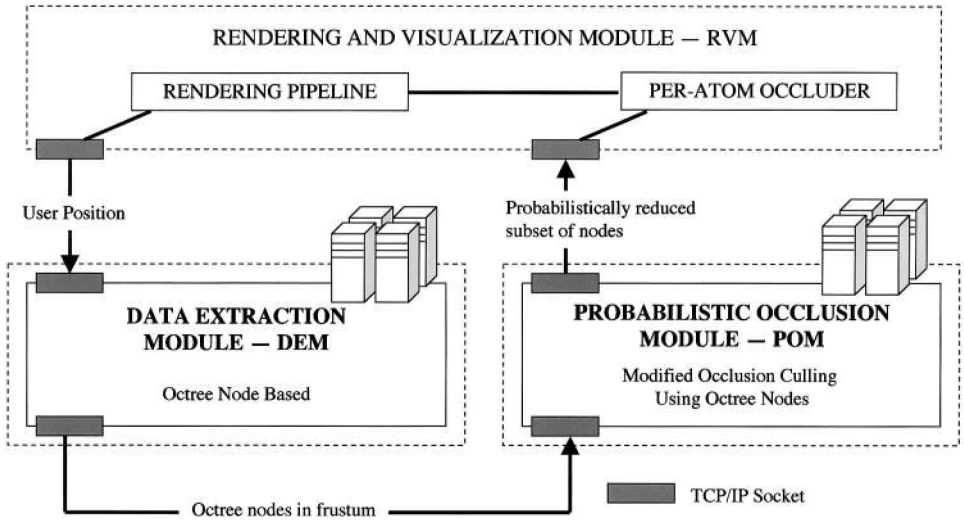

Figure 2. *Flow diagram of* Atomsviewer.

renders the visible atoms as spheres at varying LOD.

Figure 2 shows a schematic of the system in which the three components and their mutual interactions are represented by dotted rectangles and arrows, respectively. At runtime, the user's position and orientation are tracked and are sent to the DEM, which extracts a data subset that conservatively approximates the view frustum. This subset is forwarded to the POM to remove atoms that are likely to be occluded by other atoms. The result of the POM operation is forwarded to the RVM for rendering. It should be noted that the DEM and the POM use octree nodes as an abstraction of the atomic data, and only the RVM deals with individual atomic data. This allows all atomic data to reside on the RVM, thereby reducing network traffic between the three modules.

3 Data Extraction Module

The DEM is responsible for removing atoms that lie outside the view frustum. However, it is computationally expensive to perform frustum tests on individual atoms for a billion-atom data set. Instead, we create an octree by recursive subdivision of the space occupied by the atomic data set. Each terminal node of this octree (region) abstracts those atoms that lie in the corresponding subspace. Therefore, the view frustum testing of atoms is replaced by testing if a region is present in the frustum.

The number of subdivisions is determined by minimizing the overall computation time of the three modules. Whereas the computation involved in the DEM and the POM is an increasing function of the number of subdivisions, that of the RVM is a decreasing function. This tradeoff results in an optimal number of approximately 500 atoms per region.

The octree-based abstraction provides scalability to Atomsviewer, which may be understood as follows. The total computation time is composed of the time taken to cull data that is not in the frustum, the time to perform probabilistic occlusion culling, and the time to render the remaining data. In most cases, the number of visible atoms is largely independent of the viewpoint, and consequently the rendering time is nearly constant. However, without the octree abstraction, the culling time scales linearly with the number of atoms. The oc-

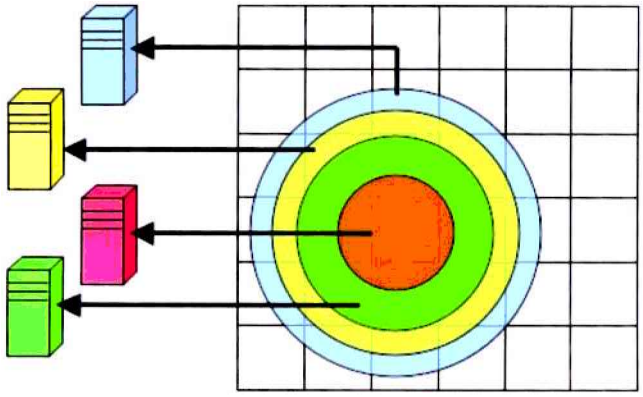

Figure 3. *The parallel implementation of the octree-based view frustum culling. The concentric shells, each of equal volume, is distributed over a PC cluster to perform a coarse extraction of octree regions.*

tree abstraction allows a larger number of atoms to be culled at once at higher octree levels through recursive traversal of the tree structure.

The view frustum culling in the DEM uses a series of bounding shapes. The first and foremost of these is a sphere, S_1, (Sharma, 2001) that fully encloses the frustum. A coarse extraction is done by approximating all octree nodes as spheres and selecting those spheres that intersect with S_1. This process is implemented through a traversal of the octree wherein a node is tested only if the sphere of the parent node intersects S_1. The end result of the process is a set of terminal nodes—regions—all of which intersect S_1. These regions are then pruned by testing them against a cylinder and subsequently a cone, each of which improves the approximation of the frustum.

The coarse extraction process can be parallelized by decomposing S_1 into a set of concentric shells and an inner sphere, all of which have equal volume. (See figure 3.) The number of shells is one less than the number of available processors.[1] The sequence of tests used to extract the regions is similar to those used in the serial

1. One processor is allocated to the innermost sphere, that is, the core.

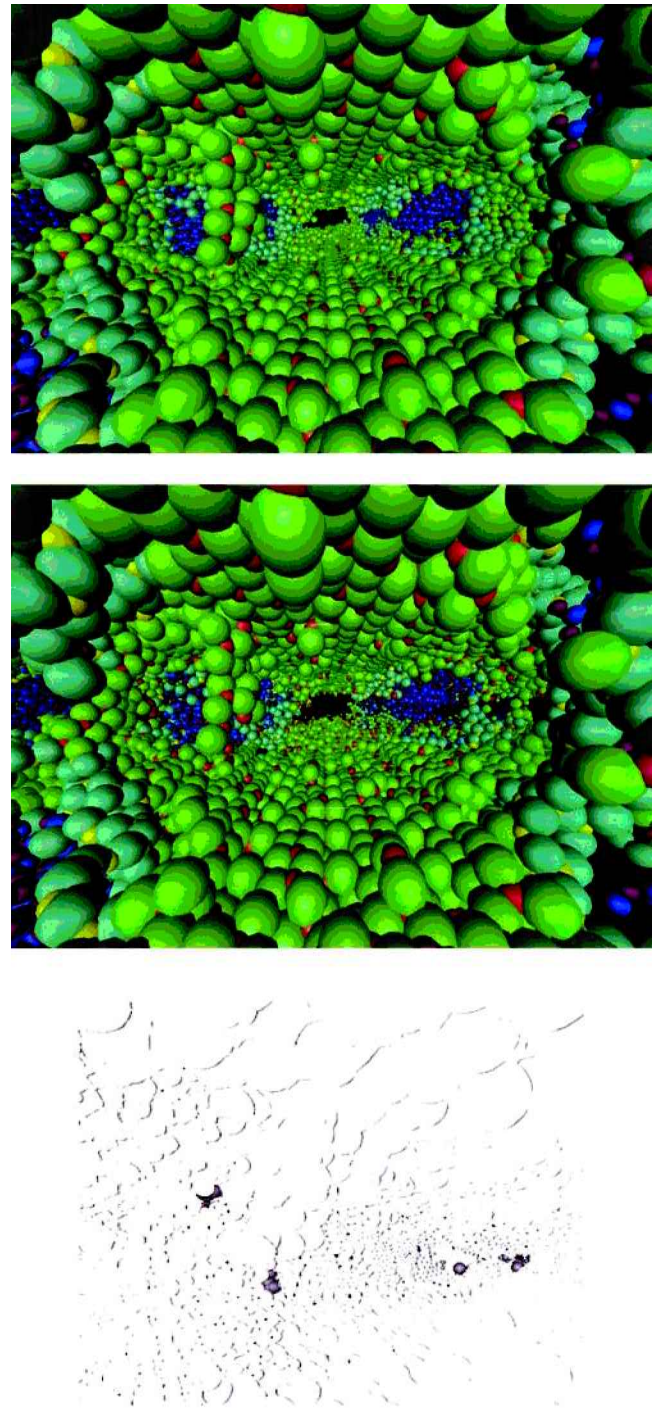

Figure 4. Atomsviewer *without (top) and with (middle) the POM enabled. The image on the bottom shows the difference between the two images.*

implementation with the modification that a node is visited only if its parent lies in the bounding shell.

Because all the tests in the DEM are performed on a per-region basis and because the parallelization uses spatial decomposition into equal volumes, the computational load involved in the extraction of regions is nearly balanced across all nodes. Each processor keeps a copy of the complete octree to reduce communication overhead. This also allows for fault tolerance because, in the event of a disturbance such as the loss of a node, the module will have to recalculate only the radii of its bounding shells.

4 Probabilistic Occlusion Module

The regions output by the DEM still contain many atoms that are not visible to the viewer due to occlusion. A traditional occlusion culling algorithm working on a per-atom level would be computationally expensive. Therefore, we have developed an algorithm that exploits the octree-based abstraction and employs a probabilistic approach to perform occlusion culling without per-atom testing.

In this probabilistic approach, each region is assigned a visibility value, which represents the fraction of visible atoms in the region. To determine this value, we assume that the region, R_0, closest to the viewer has 100% visibility. Now a region, R_1, directly behind R_0 has a proportionately lower visibility due to the high probability of atoms in R_0 occluding those contained in R_1. Likewise, as we analyze regions farther away from the viewer (yet in the view frustum), we begin to see a progressive decline in visibility. Therefore, it is natural to define a visibility function for a region, R_x, as a recursive function that is dependent upon the visibility of the nearest neighboring region that lies along the shortest path between R_x and the viewer. Once the POM calculates the visibility value for the octree regions, these values are forwarded to the RVM, where a pseudo-random number generator is used to randomly decide which atoms in a region will be drawn.

4.1 Probabilistic Visibility Function

The probabilistic visibility function, $\nu(\alpha)$—the fraction of atoms in that region which are probably seen by the user—is given by $\nu(\alpha) = [1 - D(\alpha')]\nu(\alpha')$, where α' is some region occluding α and $D(\alpha')$ is the density of the region α'. The density of region α is defined as $D(\alpha) = \Sigma_{i\in\alpha} 4\pi r_i^3/3V(\alpha)$, where r_i is the radius of the i^{th} atom and $V(\alpha)$ is the volume of region α. (By choosing r_i to be less than half the interatomic distance, $D(\alpha)$ can be normalized so that $0 \le D(\alpha) \le 1$.)

Given a region, $\alpha^{(0)}$, that the viewer currently occupies, $\nu(\alpha)$ is computed by defining a recursive relation on the viewing functions of the sequence of regions $\alpha^{(0)}, \alpha^{(1)}, \ldots, \alpha^{(n-1)}, \alpha^{(n)}(=\alpha)$. To determine this sequence, we employ a line-drawing algorithm (Bresenham, 1965; Heckbert, 1990), modified for three dimensions (Pendleton, 1992). For a region $\alpha^{(i)}$ $(1 \le i \le n)$, we determine the nearest neighbor, $\alpha^{(i-1)}$, using the equation, $\alpha^{(i-1)} = (\alpha_x^{(i)} - \Delta x, \alpha_y^{(i)} - \Delta y, \alpha_z^{(i)} - \Delta z)$, where

$$\Delta x = \begin{cases} \text{sgn(x)} & \text{if } |2\,dx| \ge \max(|dy|, |dz|) \\ 0 & \text{else} \end{cases}$$

$dx = \alpha_x^{(i)} - \alpha_x^{(0)}$ and α_x is an integer to specify the position of region α along the x axis. Likewise, we can derive equations for dy, dz and Δy, Δz. The recursive relation to compute the visibility value, ν_i, for region $\alpha^{(i)}$ is

$$\nu_i = \begin{cases} 1 & i = 0 \\ [1 - D(\alpha^{(i-1)})]\nu_{i-1} & \text{else.} \end{cases}$$

In figure 4, we compare scenes rendered with and without the POM. In this example, the POM reduced the number of atoms by 68% and thereby tripled the framerate with only a small loss in visual detail. (A quantitative measure of this loss is given in table 3.)

5 Rendering and Visualization Module

The RVM performs all the rendering tasks and is divided into two main units: per-atom occlusion culling, and rendering the atomic representations at varying

LOD using the OpenGL graphics API (Woo, Neider, Davis, & Shreiner, 1999).

5.1 Per-Atom Occlusion Culling

Per-atom occlusion culling is implemented by using a simulated depth buffer and a sequence of tests against this buffer (Zhang, 1998). Initially for every object rendered, a shape that approximates its screen space is generated in the simulated depth buffer. For simplicity, we use a circumscribed rectangle to represent an atom in the depth buffer, and this shape also has a constant depth value representative of the atom's distance from the viewer. Using the shape and its depth value, we test the approximated shape against our depth buffer across a number of test points to determine whether any part of that area is visible to the viewer. If, for any of the test points, the *Z*-value of our shape is found to be lower than the value in the depth buffer at that point, the atom is marked as visible at that point and is drawn. Additionally, the depth buffer is updated for that shape. It is important here to mention that, for the occlusion culling mechanism to perform optimally, the objects must be sorted in ascending order of distance from the viewer. A latency hiding technique, explained in section 6, ensures that the sorting time is negligible.

Figure 5 shows a sequence of images, without and with the per-atom occlusion. The per-atom occlusion reduced the number of atoms processed from 90,000 to 4,500, thereby increasing the framerate from 0.94 fps (frames per second) to 3.22 fps.

5.2 Multiresolution Rendering

The use of multiple resolutions is the last optimization technique that we employ. Atoms that are closer to the viewer are drawn at higher resolutions whereas those that are farther away from a user use fewer polygons or are represented as points. The resolution is calculated from an exponential function of distance of the atom from the viewer.

6 Parallel and Distributed Architecture

As mentioned earlier, the primary bottleneck in a visualization application is the polygon rendering on the graphics server. Although data reduction techniques such as visibility culling and optimization techniques such as multiresolution rendering reduce the workload of the graphics server, dedicating the graphics server to the rendering operation increases the framerate. We thus divide our application into three independent modules and offload the nongraphic modules, DEM and POM, to a PC cluster. The decision to offload the DEM and the POM to a cluster is also influenced by the fact that these modules use a data abstraction for their operations and hence can be isolated from the actual data during runtime. The data is kept on the graphics server, which reduces the amount of network transfer. Additionally, to account for network and computational delay, we employ a latency hiding technique in the RVM to create a larger window in which the DEM and the POM can function and transmit their result to the RVM.

6.1 Latency Hiding

Although the design of our system consists primarily of three independent modules that collaborate to deliver the graphics, there is a certain level of interdependency among the modules. For instance, the DEM needs to receive the viewer's position from the RVM to start its function. Subsequently, the POM needs to receive from the DEM a set of regions in the view frustum before it performs the probabilistic occlusion culling. Finally, the RVM needs to receive from the POM a set of atom IDs for rendering. To ensure that module operations are not affected by network delays, we overlap the intermodule communication with the module computation (Barnard et al., 1999) as shown in figure 6. This overlap is done in the RVM because it triggers the sequence of events driving the DEM and the POM.

In the traditional data flow scheme, the RVM at time t renders the scene of time $t - 1$, obtains the viewer's new position at time t, waits for the other modules to

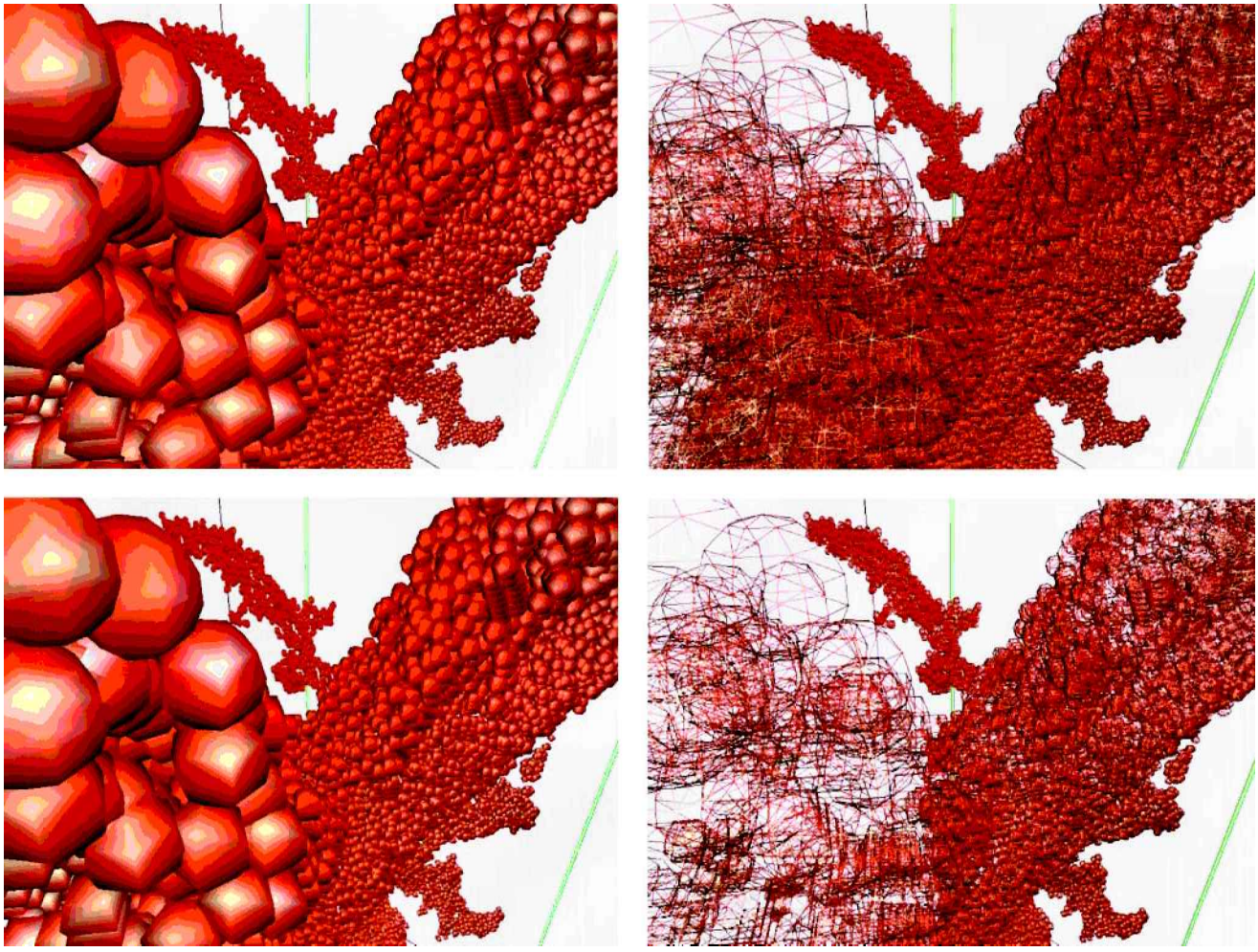

Figure 5. *Rendering of a scene without (top) and with (bottom) occlusion using polygons (left) and wireframe (right) for rendering. The figures on the top are rendered at 0.94 fps and include 90,000 atoms. The figures on the bottom are rendered at 3.22 fps and include only 4,500 atoms.*

deliver the data to be rendered, and goes back to the rendering step for time t. This approach involves a significant wait between the time a request for data is sent and the time the data is received. However, by introducing a lag of one time step, the RVM can render the scene at time $t - 1$ while waiting for the data for time t generated by the computations in the DEM and the POM. This communication/computation overlapping scheme is implemented by employing multiple threads in each of the modules. Each module creates three threads, two of which are responsible for sending and receiving data and a third that performs the actual computation. Such a multithreaded design makes the communication nonblocking because it allows incoming data to be queued while current data is being processed. This scheme will be greatly enhanced when it is combined with a predictive pre-fetching technique. A priority queue can be easily added to ensure that actual viewer position will have higher preference over a predicted viewer position.

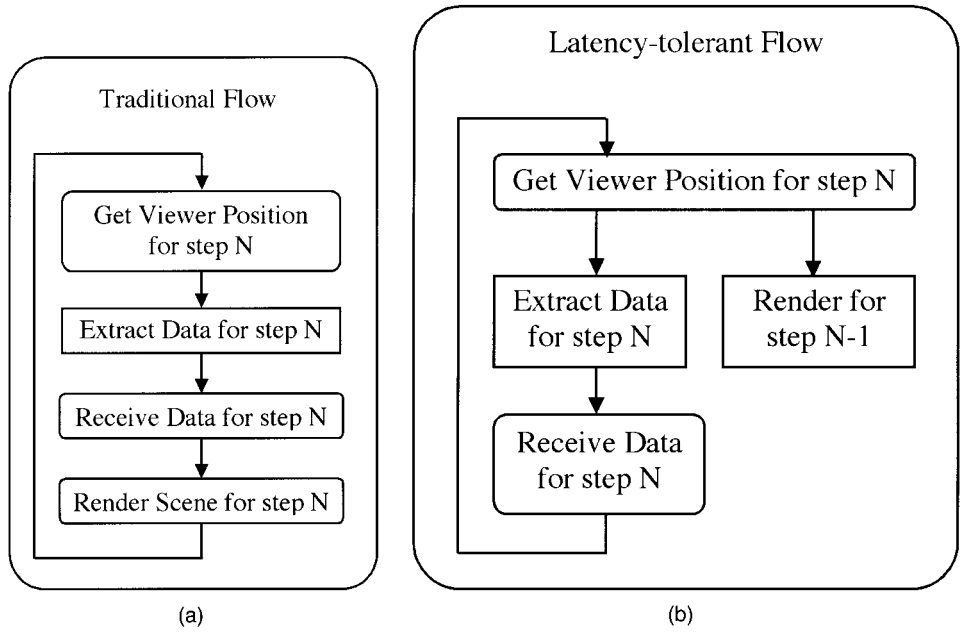

Figure 6. *Dataflow overlapping communication and computation (right) is contrasted with traditional dataflow (left).*

7 Results

Atomsviewer has been implemented on an ImmersaDesk virtual environment (Fakespace Systems, Inc.). The ImmersaDesk consists of a pivotal screen, an Electrohome Marquee stereoscopic projector, a head-tracking system, an eyewear kit, IR emitters, and a wand with a tracking sensor and a tracking I/O subsystem. A programmable wand with three buttons and a joystick allows interactions between the viewer and the virtual objects. The rendering system is an SGI Onyx2 with two R12000 processors (300 MHz), 4 GB RAM, and an InfinityReality2 graphics pipeline. The PC cluster used for the computations of the DEM and the POM comprises four PCs running Linux 7.2, each with an 800 MHz Pentium III processor and 512MB RAM.

We have performed a scalability test of Atomsviewer involving up to a billion-atom data set. Figure 7 compares the timing results of the serial Atomsviewer with and without the octree-based view frustum culling with that of the parallel and distributed Atomsviewer. We see

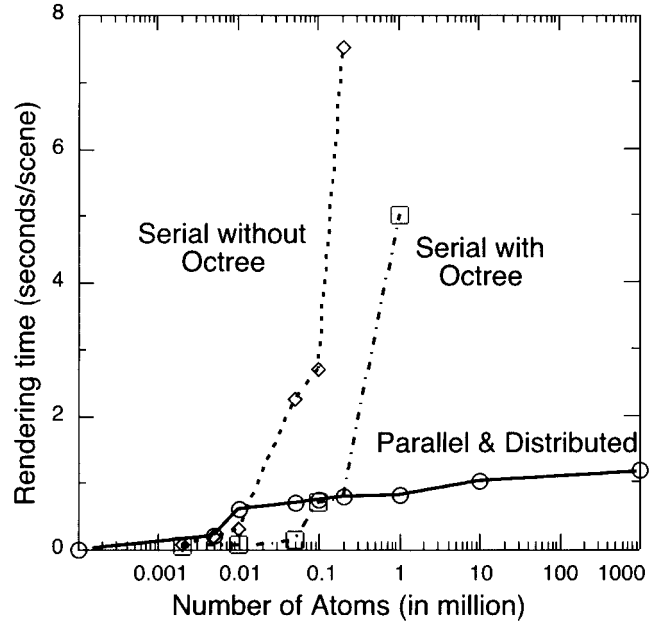

Figure 7. *Rendering time per scene as a function of the number of atoms for the parallel and distributed* Atomsviewer *is compared with those for the serial* Atomsviewer *with and without the octree enhancement.*

Table 1. *Average Reduction in Time Taken Per Rendering with the Individual Use of Each of the Three Modules.*

Number of Particles	DEM	POM	RVM
300,000	47.00%	58.73%	68.27%
500,000	50.81%	59.21%	72.36%
1,300,000	39.17%	80.42%	73.91%

Table 2. *Average Reduction in Atoms Processed Per Rendering with Each of the Three Modules.*

Number of Particles	DEM	POM	RVM
300,000	20.00%	63.84%	88.33%
500,000	72.80%	63.04%	95.44%
1,300,000	68.75%	80.15%	83.82%

that the time to extract and render the atoms within the field of view is nearly a constant function of the number of atoms. The communication overhead is successfully overcome by the communication/computation overlapping technique.

To quantify the effect of the octree-based culling in the DEM, the probabilistic occlusion in the POM, and the per-atom occlusion and the multiresolution rendering in the RVM, we ran a series of tests on various atomic simulations from our research group and collaborators. The systems that were tested had 0.3, 0.5 and 1.3 million atoms. To quantify the effectiveness of each of the three modules, a set of three runs were made on each data set. These tests were run with only one of the three modules active, and the number of atoms processed and the rendering time were recorded. Additionally, images were taken at regular intervals. These images were then subtracted from those images taken with none of the modules enabled revealing the pixel loss incurred from each of the three modules.

Table 1 and 2 show the reduction in rendering time and number of atoms processed by the graphics system with the use of each of the three modules, respectively. It is important to mention that in table 2 the reduction in the number of processed atoms in the POM is relative to the number of atoms culled in the DEM and not the absolute reduction. Table 3 shows the average pixel loss that is encountered by the use of each of the three modules. Thus, we can see that each of the modules results in a very small loss of data but provides significant data reduction and the subsequent increase in speed.

Table 3. *Average Pixel Loss Per Rendering with the Individual Use of Each of the Three Modules.*

Number of Particles	DEM	POM	RVM
300,000	0.0%	3.50%	0.22%
500,000	0.0%	4.50%	0.59%
1,300,000	0.0%	4.80%	0.95%

8 Summary

We have demonstrated a walkthrough of a billion atoms in a virtual environment using parallel and distributed computing employing an octree-based view frustum culling, probabilistic occlusion culling, and multiresolution rendering. Our implementation is scalable and has been ported to a range of platforms including Windows and Mac OS X. We are also developing a GUI with greater functionality and a Perl-based data analysis toolkit, allowing a user to specify certain rules and visualizing data that conforms to those rules. Additionally, we are also developing a neural network-based pre-fetching scheme that predicts the user's next movement and caches the data from the cluster. These would significantly reduce the effect of module latency. Atomsviewer has also been used by Drs. Burgoynne and Buerman at the LSU Eye Center in their research of glaucoma, in which it was used to visualize the path taken by axon bundles in the optic nerve head in a bovine eye.

Acknowledgments

This work was supported by ARL, AFOSR, NASA, NSF, DOE, USC-Berkeley-Princeton DURINT and the Health Excellence Fund from the Louisiana Board of Regents. The visualization was performed at the Concurrent Computing Laboratory for Materials Simulations (CCLMS) at Louisiana State University. Simulation data were generated using parallel computers at the Major Shared Resource Centers at the Department of Defense under CHSSI and Challenge projects. Finally, we would like to thank Dr. Brent Neal and Dr. Hideaki Kikuchi for the various discussions that were of immense help.

Reference

Airey, J. M., Rohlf, J. H., & Brooks, F. P. (1999). Towards image realism with interactive update rates in complex virtual building environments. *Proc. of ACM Symposium on Interactive 3D Graphics, 24*(2), 41–50.

Aliaga, D., Cohen, J., Wilson, A., Zhang, H., Erikson, C., Hoff, K., Hudson, T., Stuerzlinger, W., Baker, E., Bastos, R., Whitton, M., Brooks, F. P., & Manocha, D. (1999). MMR: An integrated massive model rendering system using geometric and image-based acceleration. *Proc of Symposium on Interactive 3D Graphics (I3D),* 199–206.

Bajaj, C., & Cutchin, S. (1999). Web based collaborative visualization of distributed and parallel simulation. *Proc. of IEEE Parallel Visualization and Graphics Symposium,* 47–54.

Bajaj, C., Ihm, I., & Park, S. (2000). Visualization-specific compression of large volume data. TICAM Tech. Rep. 00-17, University of Texas at Austin.

Barnard, S., Biswas, R., Saini, S., Van der Wijngaart, R. F., Yarrow, M., Zechter, L., Foster, I., & Larsson, O. (1999). Large scale distributed computational fluid dynamics on the information power grid using globus. *Proc. of the Seventh Symposium on the Frontiers of Massively Parallel Computation,* 60–67.

Bresenham, J. E. (1965). An algorithm for computer control of a digital plotter. *IBM Systems Journal, 4*(1), 25–30.

Clark, J. H. (1976). Hierarchical geometric models for visible surface algorithms. *Comm. of the ACM, 19*(10), 547–554.

Funkhouser, T. A., Sequin, C. H., & Teller, S. J. (1992). Management of large amounts of data in interactive building walkthroughs. *Proc. of SIGGRAPH Symposium on Interactive 3D Graphics,* 11–20.

Funkhouser, T. A., Teller, S. J., Sequin, C. H., & Khorramabadi, D. (1996). The UC Berkeley system for interactive visualization of large architectural models. *Presence: Teleoperator and Virtual Environments,* 45–60.

Heckbert, S. P. (1990). Digital line drawing. in A. Glassner (Ed.), *Graphics gems* (pp. 99–100), Boston: Academic Press.

Johnson, C., Parker, S., Hansen, C., Kindlmann, G., & Livnat, Y. (1999). Interactive simulation and visualization. *IEEE Computer, 32*(12), 59–65.

Ma, K. L., & Camp, D. (2000). High performance visualization of time-varying volume data over a wide-area network. *Proc. of Supercomputing 2000.* Available online: http://www.sc2000.org/proceedings/techpaper/papers/pap254.pdf. Retrieved Nov. 17, 2002.

Nakano, A., Bachlechner, M. E., Kalia, R. K., Lidorikis, E., Vashishta, P., Voyiadjis, G. Z., Campbell, T. J., Ogata, S., & Shimojo, F. (2001a). Multiscale simulation of nanosystems. *IEEE/AIP Computing in Science and Engineering, 3*(4), 56–66.

Nakano, A., Kalia, R. K., Vashishta, P., Campbell, T. J., Ogata, S., Shimojo, S., & Saini, S. (2001b). Scalable atomistic simulation algorithms for materials research. *Proc. of Supercomputing 2001.* Available online: http://www.sc2001.org/papers/pap.pap263.pdf. Retrieved Nov. 17, 2002.

Omeltchenko, A., Campbell, T. J., Kalia, R. K., Liu, X., Nakano A., & Vashishta, P. (2000). Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm. *Comp. Phys. Comm., 131,* 78–85.

Parker, S., Shirley, P., Livnat, Y., Hansen, C., Sloan, P., & Parker, M. (1999). Interacting with gigabyte volume datasets on the Origin 2000. *The 41st Annual Cray User's Group Conference.* Available online: http://www.sci.utah.edu/publications/cug99/cug99.pdf. Retrieved Nov. 17, 2002.

Pendleton, R. (1992). line3d (version 1) [code]. Retrieved from http://sources.isc.org/dirlist.perl?dir=devel/func/graphics/&tarball=line3d/line3d/dl.c

Pinsky, D. V., Meyer, J., Hamann, B., Joy, K. I., Brugger,

E. S., & Duchaineau, M. A. (2000). An error-controlled octree data structure for large-scale visualization. *Crossroads—The ACM Student Magazine* (spring 2000), 26–31.

Sharma, A., Miller, P., Liu, X., Nakano, A., Kalia, R. K., Vashishta, P., Campbell, T. J., & Haas, A. (2001). Million atom walkthrough: Octree-based fast visibility culling and multiresolution rendering for scalable atomistic visualization. Dept. of Computer Science Tech. Rep., Louisiana State University.

Teller, S., & Sequin, C. H. (1992). Visibility preprocessing for interactive walkthroughs. *Proc. of ACM SIGGRAPH 1992,* 55–64.

Woo, M., Neider, J., Davis, T., & Shreiner, D. (1999). *The OpenGL programming guide,* 3rd ed, Reading, MA: Addison-Wesley.

Zhang, H. (1998). *Effective occlusion culling for the interactive display of arbitrary models.* Unpublished doctoral dissertation, University of North Carolina at Chapel Hill.