# LARGE MULTIDIMENSIONAL DATA VISUALIZATION FOR MATERIALS SCIENCE

*Materials scientists use scientific visualization to explore very large multidimensional data sets. The Atomsviewer visualization system enables telepresence and provides multimodal views of simulation data.*

How does a crack propagate in a composite material? How does a high-speed projectile interact with its target? And how can we use this knowledge to make fracture- and impact-resistant materials? Materials science poses such questions, and we can answer them by simulating materials on supercomputers and analyzing their properties. Scientific visualization is an excellent tool in this analysis, and researchers have developed systems to explore very large, multidimensional data sets from materials simulations in immersive and interactive environments.

Material processes such as fracture and hypervelocity impact involve multiple length scales and phenomena—for example, the collective motions of individual atoms give rise to higher-level structures such as a network of topological defects, which in turn determine macroscopic properties such as toughness. Furthermore, these material properties result from complex causality relationships involving multiple phenomena such as structural transformation, melting, and flow. Materials scientists abstract these complex phenomena in large multidimensional space (that is, billions of atoms, each with multiple attributes such as species, 3D coordinate and velocity vectors, and stress tensor) into simplified scenarios. A fracture in ceramic-fiber composite materials, for example, involves the sliding of fibers and ceramic matrix; the associated frictional energy explains these materials' high toughness value.[1] In hypervelocity impact, high pressure and temperature in front of a projectile cause structural transformation, which changes materials properties such as stiffness, and melting, which causes materials flow. These effects in turn determine how the projectile's kinetic energy will dissipate.

Visualization enhances this type of materials research. Imagine a materials scientist shrunk to the size of an atom walking through a material to investigate a fracture. The scientist has video cameras with filters to view information such as atomic species, temperature, and pressure. Whenever the scientist finds an interesting viewpoint (some things are visible only from a certain distance or at a certain angle), he or she leaves a video camera with a filter to record the entire fracture process. By playing these multiple videos simultaneously, the scientist attains telepresence and multiple modalities of vision.

1521-9615/03/$17.00 © 2003 IEEE
Copublished by the IEEE CS and the AIP

ASHISH SHARMA, RAJIV K. KALIA, AIICHIRO NAKANO, AND PRIYA VASHISHTA
*University of Southern California*

The Collaboratory for Advanced Computing and Simulations at the University of Southern California has developed the Atomsviewer system, which visualizes large data sets from materials simulations involving billions of atoms, allowing materials scientists to view fractures from many angles and distances. We are currently developing a multicamera, multimodal extension to Atomsviewer that will let scientists simultaneously view different parts of a system.

## Atomsviewer Overview

The large data sets required in large-scale molecular dynamics (MD) simulations of materials make visualization difficult. An MD data set per time step for a billion atoms is a table of a billion entries, each with columns of data for physical attributes such as atomic species, positions, velocities, and stresses, and occupies 100 Gbytes of disk space.[2] Few workstations can process this volume of data for interactive rendering.

Atomsviewer solves this problem by visualizing billion-atom data sets at interactive speeds in an immersive environment. In visualization, polygon rendering on a graphics pipeline is the primary bottleneck; thus, we minimize the pipeline workload by processing only the data the viewer will see.[3] To do this, we use data-management techniques based on the *octree* data structure.[4,5] Novel algorithms and techniques, such as our probabilistic approach, to remove hidden atoms can further reduce the rendering pipeline's workload. Furthermore, we offload all processing that precedes rendering to a PC cluster and dedicate the graphics server to rendering through a parallel and distributed design. The resulting architecture provides multiple viewpoints, thus enhancing the user experience. Figure 1 shows a block diagram of Atomsviewer.

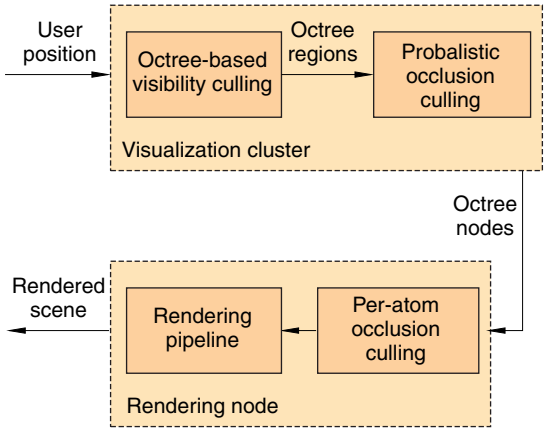



**Figure 1. Overview of the Atomsviewer visualization system. To prevent bottlenecking caused by polygon rendering, the system processes only those data that the viewer will see.**

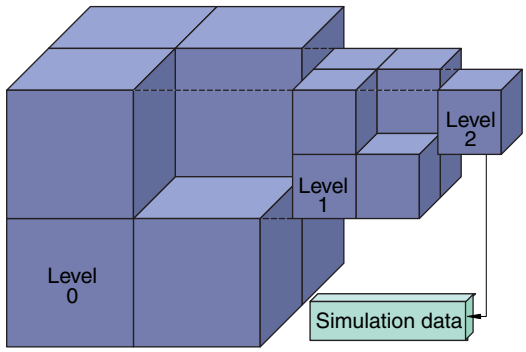



**Figure 2. A three-level octree implementation. Each octree node contains the coordinate bounds of the corresponding subspace. A terminal node at level 2 points to a structure that stores data associated with atoms in the region defined by the terminal node.**

## Data Management

Visualizing 100-Gbyte data sets requires a fast and scalable data management system. Although the data set is large, users only see a relatively small subset at any given instant. Thus, our data management scheme efficiently extracts atoms in the users' field of view. Computer graphics systems define field of view with a truncated pyramid shape called a *frustum*. *View frustum culling* is the process of removing atoms that lie outside the frustum.

To achieve interactive speed and scalability for view frustum culling, we cluster spatially close atoms hierarchically using an octree data structure. An octree is a 3D extension of a binary search tree generated by recursively subdividing the 3D space into smaller subregions, as Figure 2 shows. Each octree node is an abstraction of the atoms in its region. We thus transform the process of removing atoms outside the frustum to one of extracting regions inside the frustum, which significantly reduces computation time. Total computation time includes culling and rendering time for the reduced set of atoms. The number of rendered atoms is largely independent of the viewpoint most of the time, whereas the culling time scales linearly with the total number of atoms. Using octree abstraction, we cull many atoms at a time, reducing the computational complexity from $O(N)$ to $O(\log N)$.

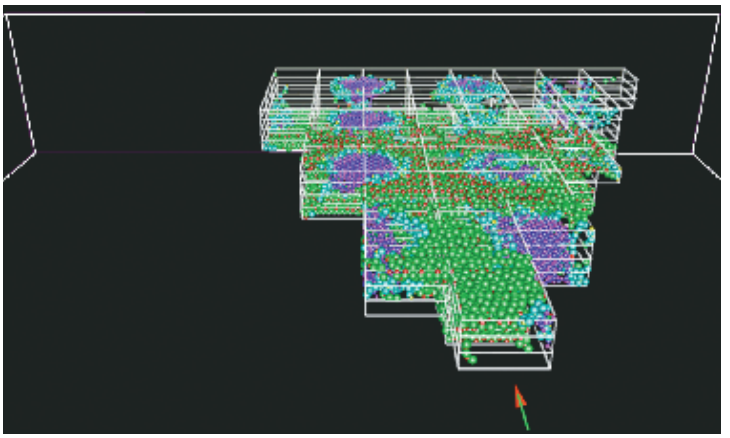


**Figure 3. The octree data structure overlain on the atomistic data. The figure shows only the atoms that are selected for subsequent rendering.**

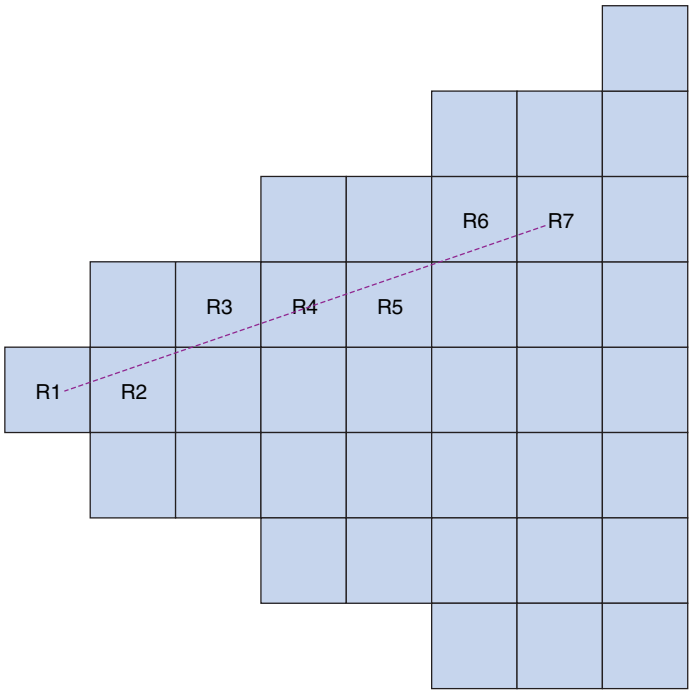



**Figure 4. Probabilistic occlusion culling. The viewer is in region R1. To calculate the visibility of region R7, for example, we trace a line from R7 to R1 and calculate the visibility using the visibility of the regions along this line. The visibility of R7 is therefore a function of the visibility of R6 and the distance of R7 from R1.**

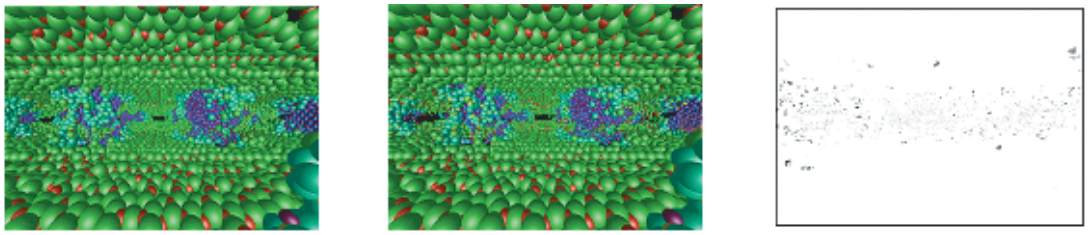
By trading frustum approximation quality for computation time, we can empirically determine the number of subdivisions (the octree *depth*) such that each region represented by a terminal node contains approximately 500 atoms. A larger depth implies smaller regions and therefore a more accurate representation of the frustum, but requires more computation. Our tests involving million- to billion-atom data sets show no significant visual gains with a finer granularity than 500 atoms per region.

We implement view frustum culling through a series of *bounding volumes*: shapes that let us select all octree nodes that intersect the shape or are contained in them. The user's position and orientation determines the shape's size, position, and orientation in 3D space. The first of these shapes is a sphere $S$ that fully encloses the frustum.[6] We perform a coarse extraction by approximating all octree nodes as spheres and selecting those that intersect $S$. To do this, we traverse only those children nodes whose parent-node sphere intersects $S$. The end result is a set of terminal nodes, or regions, all of which intersect $S$. We then prune these regions by testing them against a cylinder and subsequently a cone, each of which improves the frustum approximation. Figure 3 shows the extracted view frustum.

## Probabilistic Occlusion Culling and Per-Atom Occlusion Culling

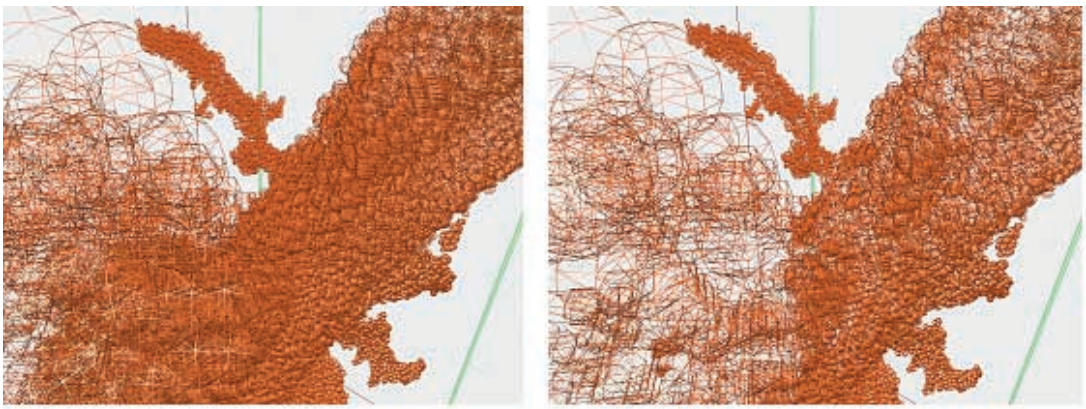
Using the data management techniques described earlier, we identify all nodes in the view frustum. Not all the atoms in the extracted regions are visible, however. Our tests show that 60 percent of the extracted atoms typically are invisible because atoms lying between them and the viewer *occlude* them—that is, they hide them. To prevent the unnecessary rendering of so many atoms, we perform *occlusion culling*—we remove objects that are in the frustum but are hidden by other objects closer to the viewer.

Occlusion culling on large data sets requires enormous computing, but we can exploit the octree abstraction to minimize the complexity. To do this, we introduce a probabilistic approach in occlusion culling, illustrated in Figure 4. The basic idea comes from the observation that the octree regions closest to the user will likely occlude regions that are further away. We can thus define a *visibility value* (the fraction of visible atoms) for each region. The visibility value decreases as the object's distance from the viewer increases. A visibility function for a region $R$ is a recursive function dependent on the visibility of the nearest neighboring region that lies along the shortest path between $R$ and the viewer. Once Atomsviewer has calculated the visibility values for all octree regions, it forwards them to the renderer,



(a) (b) (c)

**Figure 5. Occlusion culling in Atomsviewer: (a) an image produced by the system without probabilistic occlusion; (b) an image produced by the system with probabilistic occlusion; and (c) the difference between the two images.**



(a) (b)

**Figure 6. A wireframe rendering of a scene (a) without occlusion for rendering and (b) with occlusion for rendering. Figure 6a is rendered at 0.94 frames per second and includes 90,000 atoms. Figure 6b is rendered at 3.22fps and includes only 4,500 atoms.**

where a random number generator decides which atoms in a region will be drawn.[6,7]

Data loss due to our probabilistic occlusion culling is small. Figure 5 shows the algorithm's effect and the resulting data loss. Our tests have shown that although the algorithm produces images that typically have 5 percent pixel loss, it achieves a threefold speed up.

The tests performed so far use the octree abstraction, working on a per-region level rather than the actual data set. At this stage, however, we can perform a more traditional occlusion culling that works on a per-atom level.[8] Atomsviewer uses per-atom occlusion culling, which simulates a buffer with pixel data for the rendered image (a *depth buffer*). Next, the system generates a rectangle that approximates an atom's projection on screen. Using the rectangle and the atom's distance from the viewer, we test the approximated shape against our depth buffer across several test points to determine whether the viewer can see any part of the shape. If, for any test point, the depth is lower than the depth buffer value at that point, Atomsviewer marks the atom as visible, draws it, and updates the depth buffer. The system can use the octree abstraction again to speed up the occlusion process by approximating a node as a rectangle and testing this shape against the depth buffer. If the test is negative, it skips the testing process for all atoms in the node.

Figure 6 shows a pair of images, rendered without and with per-atom occlusion. Per-atom occlusion reduces the number of atoms processed from 90,000 to 4,500, thereby increasing the frame rate from 0.94 frames per second to 3.22 fps.

We are now ready to render the atoms. In rendering, distant objects appear to have fewer de-

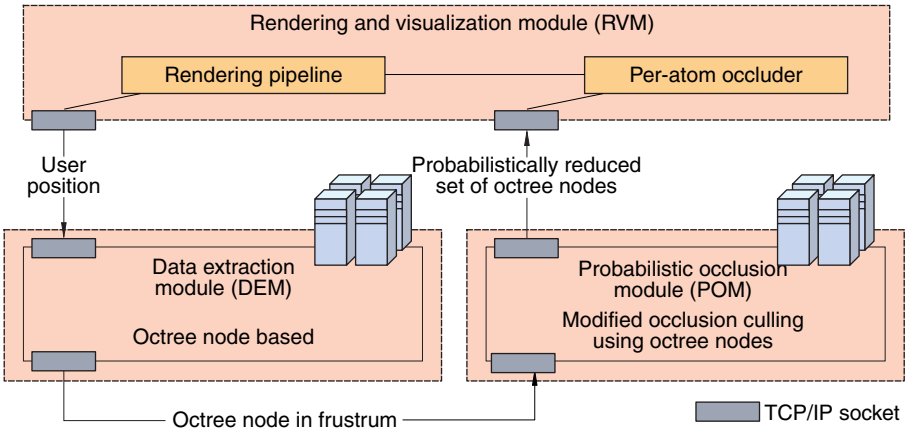



**Figure 7. Parallel and distributed implementation of Atomsviewer. By delegating data-reduction functions to the two nongraphic modules (DEM and POM) and dedicating the graphics server (RVM) to rendering, we significantly increase the frame rate.**

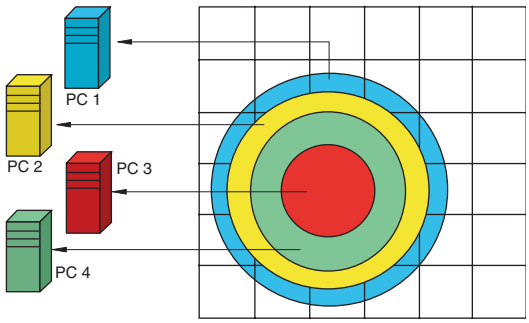



**Figure 8. The parallel implementation of the octree-based view frustum culling. The concentric shells, each of equal volume, are distributed over a PC cluster to perform a coarse extraction of octree regions.**

tails. Therefore, we use *multiresolution rendering*—we render atoms that are farther away as spheres with fewer polygons or even as points. To determine the *atom resolution* (the number of polygons used to draw an atom), we define resolution as an exponentially decreasing function of the atom's distance from the viewer. Now that our set contains only visible atoms and their respective resolutions, we can pass this information to the rendering pipeline.

## Parallel and Distributed Architecture

As mentioned previously, polygon rendering is the primary bottleneck in a visualization system. Offloading the data-reduction techniques to separate computing platforms and dedicating the graphics server to the rendering operation significantly increases frame rate. To achieve this, we divide Atomsviewer into three independent modules, as Figure 7 shows:

- Data extraction module (DEM), which uses the octree-based view frustum culling algorithm
- Probabilistic occlusion module (POM)
- Rendering and visualization module (RVM), which performs per-atom occlusion culling and multiresolution rendering

Because the nongraphic modules (DEM and POM) use the octree-based data abstraction and hence can be isolated from the actual atomistic data during runtime, Atomsviewer executes them on a PC cluster. The application keeps only the atomistic data on the graphics server to reduce the amount of network transfer.

### Latency Hiding

Although DEM, POM, and RVM are largely independent modules that collaborate to deliver graphics, a certain level of interdependency exists among them. For instance, RVM must send DEM the viewer's position before DEM can begin its function. Subsequently, DEM must send POM a set of regions in the view frustum before POM can perform probabilistic occlusion culling. Finally, POM must send RVM a set of atom IDs for rendering. To ensure that network delays don't affect module operations, we overlap intermodule communication with module computation.[9]

This overlap occurs in RVM, where it triggers the event sequence driving DEM and POM.

In the traditional dataflow scheme, 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 deliver the data to be rendered, and returns to the rendering step for time $t$. This approach involves a significant wait time between sending a data request and receiving the data. By introducing a one-time-step lag, the RVM can render the scene at time $t - 1$ while waiting for the data for time $t$ generated by the computations in DEM and POM. We use multiple threads in each of the modules to implement this communication–computation overlapping scheme. Each module creates three threads, two of which are responsible for sending and receiving data while a third thread performs the actual computation. Such a multithreaded design makes the communication nonblocking because the application can queue incoming data while it processes current data.

We are investigating a scheme to significantly boost the frame rate by combining parallel and distributed rendering with predictive user position prefetching. Furthermore, to ensure that a user's position can act as a corrective variable, we could add a priority queue to ensure that the actual viewer position has precedence over a predicted position.

### Parallelized Data Management

When processing a billion atoms, DEM must handle a few million octree nodes, so its serial implementation cannot achieve interactive speeds. Parallelizing DEM can resolve this bottleneck. Specifically, decomposing the bounding sphere into a set of concentric shells and an inner sphere, all of which have equal volume, parallelizes the coarse extraction process, as Figure 8 shows. The number of shells is one fewer than the number of available processors (one processor is allocated to the innermost sphere, or *core*). The sequence of tests used to extract the regions is similar to that used in the serial implementation; however, the tests in the DEM visit an octree child-node shell only if the parent-node shell is intersected.

Because we perform all tests in the culling process on a per-region basis and the parallelization spatially decomposes the sphere into equal volumes, the computational load involved in extracting regions is nearly balanced across all processors. Each processor keeps a copy of the complete octree to reduce communication overhead. This also allows for fault tolerance: should a disturbance occur, such as the loss of a processor, the module need only recalculate the radii of its bounding shells.

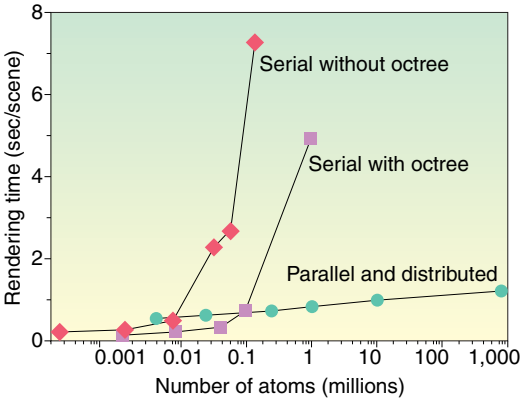



**Figure 9. Rendering time per scene as a function of the number of atoms for the parallel and distributed Atomsviewer versus rendering time for the serial Atomsviewer with and without the octree enhancement.**

We have tested the scalability of our parallel and distributed architecture and associated techniques. Figure 9 compares the serial Atomsviewer's timing results (both with and without the octree-based view frustum culling) with the parallel and distributed application's results. 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.

## Multimodal Multidisplay System

Although our visualization system provides significant information about the simulation data, the amount of information scientists can extract depends on the view they adopt. In complex data sets, however, a single view rarely provides the necessary insight; thus, a series of interconnected viewpoints is preferable. To provide such a user experience, we have extended Atomsviewer to let scientists use many viewpoints displaying a variety of data attributes (or modalities). This provides simultaneous control and viewpoints from all cameras across all desired modalities.

The multimodal visualization system works on top of the Atomsviewer graphics framework. For a system offering $n$ viewpoints, we initialize $n$ rendering pipelines, each of which abstracts a

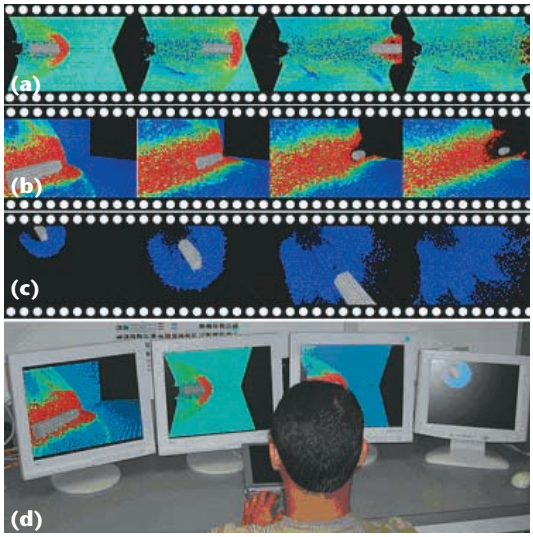



**Figure 10. A scientist investigates the hypervelocity impact damage caused by a projectile using the multimodal, mulitdisplay visualization system. The top three images show (a) the pressure distribution, (b) temperature, and (c) coordination number in the MD system across time. The fourth image (d) shows a scientist simultaneously viewing the different modalities from different viewpoints.**

user-controlled camera. The user initializes the camera positions at desired viewpoints and uses one of these cameras to perform a walkthrough at runtime. The other cameras, on their respective displays, simultaneously show the simulation data time sequence. This extension has had little impact on the system's overall performance because each display runs its own Atomsviewer and is synchronized by a server that transmits the user's position and motion.

Figure 10 shows an application of the multimodal, multidisplay visualization system to an MD simulation of hypervelocity impact of a projectile on aluminum nitride (AlN) ceramic, which materials scientists have considered for armor applications—armor used in US Army tanks and other vehicles. In this simulation, understanding ballistic damage initiation and evolution during the hypervelocity impact of a projectile is essential to improving ceramics armor efficiency.

Figure 10a shows a time sequence of the atomic-level pressure distribution during the projectile's penetration of the AlN plate (we've cut out a quarter of the system to show the damage inside the AlN plate). The projectile's penetration of the AlN plate creates a high-pressure region ahead of the projectile. Figure 10a also shows regions with residual pressure around the cavity formed by the projectile penetration. In these regions, the system undergoes a high-pressure structural transformation. A structural analysis in Figure 10c, which shows the distribution of six coordinated atoms indicating the high-pressure phase, verifies this result.

Figure 10b shows the temperature distribution. The high-impact velocity melts the atoms in front of the projectile, accelerating the structural transformation. The multicamera multimodal animation, which reveals complex microscopic processes of impact damage, identifies spatial–temporal correlations and cause–effect relationships among various physical quantities.

The multimodal display system is amenable to parallel and distributed processing on a Grid of distributed computing and visualization resources. Grid computing is a relatively new computing paradigm in which an application can harness a collection of computing resources as one large computing resource. In such a system, a Grid would discover and allocate computing resources on demand as the user adds cameras; thus, the user would experience no delay as the number of cameras increased. Such a Grid-enabled multimodal display system would be useful in collaborative sessions in which researchers from around the world could simultaneously explore the same simulation data from different viewpoints and modalities.

Scalable implementation of multidisplay Atomsviewer on the Grid would be useful in many areas that require visualization of large multidimensional data sets such as those generated in material science, biology, medicine, and other physical and engineering sciences. The idea of using multiple viewpoints to simultaneously view and track a multitude of data attributes could be implemented in different scientific visualization applications.

## Acknowledgments

*Parts of the research presented in this article were performed in collaboration with Paulo Brancio, Xinlian Liu, Paul Miller, and Wei Zhao at Louisiana State University, and Timothy Campbell and Andy Hass at the Naval Oceanographic Office Major Shared Resource Center. The US Army Research Lab, NASA, the US National Science Foundation, the US Department of Energy, USC-Berkeley-Princeton Defense University*

*Research Initiative on Nanotech, and the Louisiana Board of Regents supported this work. Simulation data were generated using parallel computers at the Major Shared Resource Centers at the Department of Defense under Challenge CHSSI projects.*

## References

1. A. Nakano et al., "Scalable Atomistic Simulation Algorithms for Materials Research," *Proc. Supercomputing 2001*, ACM Press, 2001.
2. A. Omeltchenko et al., "Scalable I/O of Large-Scale Molecular Dynamics Simulations," *Computer Physics Comm.*, vol. 131, pp. 78–85.
3. D. Aliaga et al., "MMR: An Integrated Massive Model Rendering System Using Geometric and Image-Based Acceleration," *Proc. Symp. Interactive 3D Graphics (I3D)*, http://citeseer.nj.nec.com/aliaga99mmr.html, 1999, pp. 199–206.
4. J.H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Comm. ACM*, vol. 19, no. 10, pp. 547–554.
5. D.V. Pinsky et al., "An Error-Controlled Octree Data Structure for Large-Scale Visualization," *Crossroads—The ACM Student Magazine*, Spring 2000, pp. 26–31.
6. A. Sharma et al., "Immersive and Interactive Exploration of Billion-Atom Systems," *Proc. IEEE Virtual Reality 2002*, IEEE CS Press, 2002, pp. 217–223.
7. J.E. Bresenham, "An Algorithm for Computer Control of a Digital Plotter," *IBM Systems J.*, vol. 4, no. 1, pp. 25–30.
8. H. Zhang, *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*, PhD diss., Univ. of North Carolina at Chapel Hill, 1998.
9. S. Barnard et al., "Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid Using Globus," *Proc. 7th Symp. Frontiers of Massively Parallel Computation*, IEEE CS Press, 1999, pp. 60–67.

**Ashish Sharma** is a doctoral student in the Computer Science Department at the University of Southern California (USC), Los Angeles. His research interests are scientific visualization, computer graphics, and the use of parallel, distributed, and Grid computing in the same. He received a BS in electrical engineering from Punjab Engineering College, Chandigarh, India. Contact him at the Collaboratory for Advanced Computing & Simulations, 3651 Watt Way, VHE 608, Los Angeles, CA 90089-0242; sharmaa@usc.edu.

**Rajiv K. Kalia** is a professor in the Physics, Material Science, Computer Science and Biomedical Engineering Departments at USC. He received a PhD in physics from Northwestern University. Contact him at the Collaboratory for Advanced Computing & Simulations, 3651 Watt Way, VHE 608, Los Angeles, CA 90089-0242; rkalia@usc.edu.

**Aiichiro Nakano** is an associate professor in the Computer Science, Physics, Material Science, and Biomedical Engineering Departments at USC. He received a PhD in physics from the University of Tokyo. Contact him at the Collaboratory for Advanced Computing & Simulations, 3651 Watt Way, VHE 608, Los Angeles, CA 90089-0242; anakano@usc.edu.

**Priya Vashishta** is a professor in the Material Science, Physics, Computer Science, and Biomedical Engineering Departments at USC. He received a PhD in physics from Indian Institute of Technology. Contact him at the Collaboratory for Advanced Computing & Simulations, 3651 Watt Way, VHE 608, Los Angeles, CA 90089-0242; priyav@usc.edu.