

2002

## Techniques and algorithms for immersive and interactive visualization of large datasets

Ashish Sharma

*Louisiana State University and Agricultural and Mechanical College*

Follow this and additional works at: [https://digitalcommons.lsu.edu/gradschool\\_theses](https://digitalcommons.lsu.edu/gradschool_theses)



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Sharma, Ashish, "Techniques and algorithms for immersive and interactive visualization of large datasets" (2002). *LSU Master's Theses*. 1165.

[https://digitalcommons.lsu.edu/gradschool\\_theses/1165](https://digitalcommons.lsu.edu/gradschool_theses/1165)

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact [gradetd@lsu.edu](mailto:gradetd@lsu.edu).

**TECHNIQUES AND ALGORITHMS FOR IMMERSIVE AND INTERACTIVE  
VISUALIZATION OF LARGE DATASETS**

A Thesis

Submitted to the Graduate Faculty of the  
Louisiana State University and  
Agricultural and Mechanical College  
in partial fulfillment of the  
requirements for the degree of  
Master of Science in Systems Science

in

The Department of Computer Science

by  
Ashish Sharma  
B.E., Punjab University, Chandigarh, India 1999  
May 2002

## **ACKNOWLEDGEMENTS**

I would take this opportunity to thank my advisor, Dr. Aiichiro Nakano, for his support and encouragement in my research and graduate studies at Louisiana State University. The time spent at Concurrent Computing Laboratory for Materials Simulations has been a great experience, and I would like to thank Drs. Priya Vashishta, Rajiv K. Kaila and Aiichiro Nakano for letting me a part of it.

I am indebted to my colleagues and friends Paul A. Miller, Brent Neal and Wei Zhao for the numerous discussions, suggestions and programming tips critical to my research. I am grateful to Hideaki Kikuchi for his help on all hardware issues, without which none of this would have been possible. I am also grateful to all the members of the CCLMS group with who I have had the opportunity to interact: Gurcan Aral, Sanjay Kodiyalam, Jabari Lee, Elefterios Lidirokis, Cindy Rountree, Laurent Van Brutzel, Satyavani Vemparala, Liu Xinlian and Cheng Zhang. A special thanks to Jade Etheridge, the CCLMS Coordinator, for the innumerable times she has helped me.

I am lucky to have had some great friends without whom graduate school would have been murder. Last but not the least I thank my parents, my sister and my grandparents for all their love, support and encouragement.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b> . . . . .	ii
<b>ABSTRACT</b> . . . . .	iv
<b>CHAPTER 1. INTRODUCTION</b> . . . . .	1
1.1 Document Organization . . . . .	2
<b>CHAPTER 2. PROBLEM DESCRIPTION</b> . . . . .	3
2.1 Introduction to Scientific Visualization . . . . .	4
2.2 Limitations of Current Tools and Techniques . . . . .	5
2.3 System Overview . . . . .	7
2.3.1 Hardware Description . . . . .	8
2.3.2 Programming Languages, Methodologies and APIs used . . . . .	9
<b>CHAPTER 3. OCTREE DATA STRUCTURE</b> . . . . .	11
3.1 Introduction . . . . .	11
3.2 Design and Implementation of the Octree . . . . .	12
3.3 Octree Based Visibility Culling . . . . .	16
3.3.1 Parallelized and Distributed Implementation . . . . .	18
<b>CHAPTER 4. GRAPHICAL SYSTEM</b> . . . . .	20
4.1 Occlusion Culling Algorithm . . . . .	20
4.1.1 Probabilistic Occlusion Culling . . . . .	21
4.1.2 Depth Based Occlusion Culling . . . . .	22
4.2 Multiresolution Algorithm . . . . .	23
<b>CHAPTER 5. PARALLEL AND DISTRIBUTED OPERATION</b> . . . . .	25
5.1 Distributed Design and Implementation . . . . .	25
5.2 Latency Hiding – Communication / Computation Overlap . . . . .	27
<b>CHAPTER 6. RESULTS AND CONCLUSIONS</b> . . . . .	29
6.1 Serial Implementation . . . . .	29
6.2 Parallel Implementation . . . . .	31
<b>REFERENCES</b> . . . . .	33
<b>VITA</b> . . . . .	36

## **ABSTRACT**

Advances in computing power have made it possible for scientists to perform atomistic simulations of material systems that range in size, from a few hundred thousand atoms to one billion atoms. An immersive and interactive walkthrough of such datasets is an ideal method for exploring and understanding the complex material processes in these simulations. However rendering such large datasets at interactive frame rates is a major challenge. A scalable visualization platform is developed that is scalable and allows interactive exploration in an immersive, virtual environment. The system uses an octree based data management system that forms the core of the application. This reduces the amount of data sent to the pipeline without a per-atom analysis. Secondary algorithms and techniques such as modified occlusion culling, multiresolution rendering and distributed computing are employed to further speed up the rendering process. The resulting system is highly scalable and is capable of visualizing large molecular systems at interactive frame rates on dual processor SGI Onyx2 with an InfinteReality2 graphics pipeline.

# CHAPTER 1

## INTRODUCTION

This thesis presents algorithms and techniques that were developed to allow an immersive and interactive visualization of large scientific datasets in a virtual environment. These algorithms are designed to be scalable and have been successfully implemented in a visualization tool called *Atomsviewer* that was developed at Concurrent Computing Laboratory for Materials Simulations (CCLMS), Louisiana State University. The datasets primarily visualized are from simulations of material properties, fracture etc. that were performed by the researchers at CCLMS.

Visualization is a relatively new tool that allows a researcher to better understand his SYSTEM before, during and after a simulation. The *Atomsviewer* project was started in CCLMS because of a growing need for a visualization tool for the material simulations that were being performed. Besides giving the researcher a tool for the final analysis, it would allow visualization of the initial system and correction of any errors or anomalies, thus preventing any wastage of time or computational resources. The early versions were capable of visualizing systems no larger than a few hundred thousand atoms. However with increased computational power, larger simulations of a few million atoms were being performed and this version was incapable of handling such large systems. Furthermore with the addition of an immersive environment, the need to visualize physical systems in an immersive virtual environment arose. Consequently *Atomsviewer* was brought back to the drawing boards and the techniques and algorithms that are proposed in this thesis were

developed to facilitate an immersive and interactive virtual visualization system that was scalable, portable and easy to use.

## **1.1 Document Organization:**

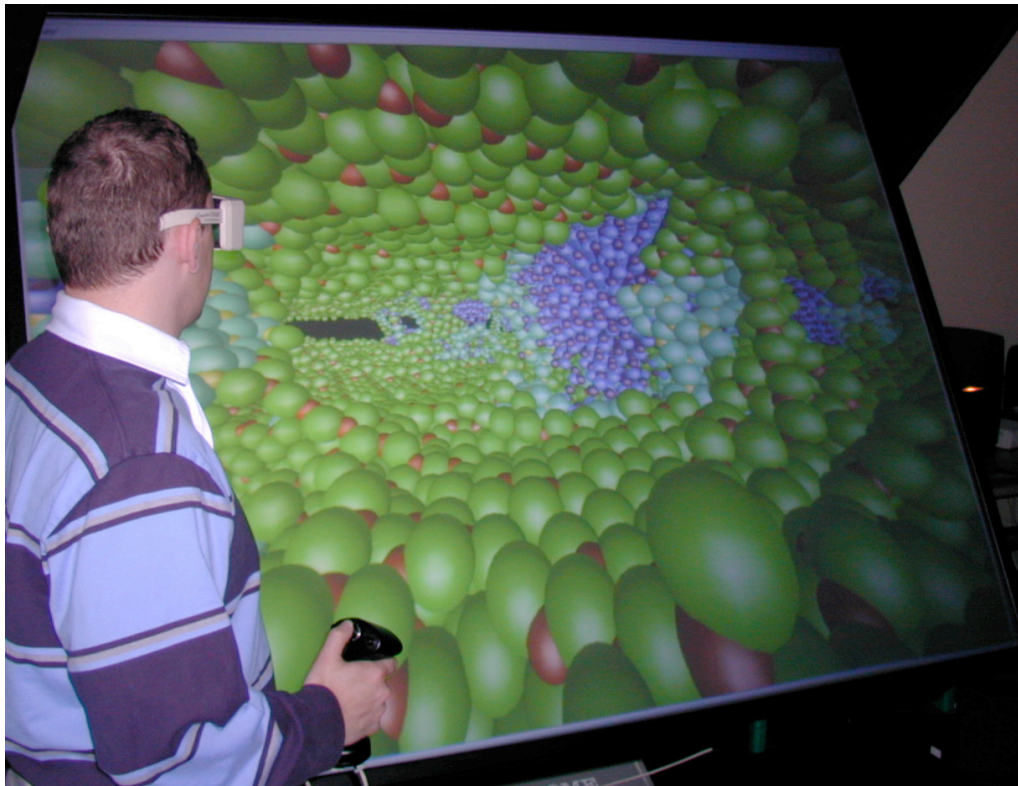
This thesis is organized as follows. In Chapter 2 we present some of the common scientific visualization techniques, discuss the various state-of-art visualization algorithms and tools and techniques and discuss why these tools cannot be used for large atomistic visualization. Finally we present an overview of the system, the different hardware components and the programming languages and APIs (Application Programming Interfaces) that are used in *Atomsviewer*. In Chapter 3 the octree-based data management technique is presented. This is the primary reason behind the speed and scalability of *Atomsviewer*. In Chapter 4 the graphical system is presented and the various optimizations that have been built into it are discussed.

In order for *Atomsviewer* to be truly scalable it needs to be able to visualize systems of any size. However if we attempt to visualize multi-million atoms the program loses interactivity due to hardware limitations. To remove this bottleneck, the system is parallelized and distributed over a network. Chapter 5 discusses the various techniques that go into the parallel and distributed operation. Chapter 6 presents the results of the various tests that were conducted in the serial and parallel versions of *Atomsviewer* to quantitatively demonstrate its interactivity and scalability.

## CHAPTER 2

### PROBLEM DESCRIPTION

Large-scale atomistic simulations generate an enormous amount of data. For example, a billion-atom molecular dynamics (MD) simulation produces 100 gigabytes of data per frame including atomic species, positions, velocities, and stresses [1 - 4]. Interactive exploration of these simulations is important for identifying and tracking atomic features that are responsible for macroscopic phenomena, and an immersive and interactive virtual environment is an ideal platform for such explorative visualization, see Fig. 1.



**Figure 1.** A researcher investigating a ceramic fiber-composite material, rendered in an immersive virtual environment.



This chapter provides an introduction to scientific visualization and discusses the need for a tool for large-scale particle visualization. It discusses the state-of-the-art in this field, the work that has been done in large-scale visualization and the limitations of these techniques when used for atomistic visualization. Last of all the various hardware and the software components of the visualization application are discussed.

## **2.1 Introduction to Scientific Visualization:**

Scientific Visualization is one of the most actively researched areas in computer sciences and has greatly altered the way many industries function. As an example, consider the recent development of the Joint Strike Aircraft undertaken by Boeing and Lockheed Martin. Through an extensive use of modeling and visualization, the companies were able to conduct collaborative fly-through sessions and virtual build, assembly and maintenance tasks to ensure concurrent engineering, first-time quality and low development times.

The primary reason for the success of virtual reality in visualization comes from the basic idea of using computer-generated models to gain information and understanding from data (geometry) and relationships (topology) [5]. Scientific visualization can be broadly classified into 3 categories namely:

1. Volume based visualization: This is the most actively researched area and involves visualization of systems where measurements have been made at regular intervals. Examples of volume-based visualization include visualization of CFD experiments and simulations, medical systems and geological studies such as oil exploration.

2. Texture based visualization: This is one of the fastest visualization techniques and involves the use of textures (predetermined pictures that overlay a mesh of the object that is being represented). Typical applications include areas where the system configuration is known beforehand such as architectural walkthroughs and virtual worlds.
3. Particle based Visualization: It involves an actual rendering of a three dimensional object using graphic primitives. This approach makes this computationally expensive and hence is not used very often. Typical applications include biomolecular visualization and atomistic visualization.

In our systems, discrete atoms whose individual positions cannot be precomputed are being visualized. This restricts us to particle-based visualization, which is a relatively newer area where such large-scale visualizations have not been attempted.

## **2.2 Limitations of Current Tools and Techniques:**

Large-scale visualization and achieving an interactive frame rate, which can be defined as 10 or more frames per second, is a major challenge. In a virtual environment it is harder to maintain this interactivity since every frame consists of two drawings, one for the left eye and the other for the right eye. Visualization of large datasets is not a very new field and considerable progress has been made in this area. One of the earliest works by Clark [6] has introduced a hierarchical representation of models, which provides a fast mechanism to clip insignificant data and compute levels-of-detail (LOD).

Major efforts to visualize large datasets have focused on volumetric data. The Center for Computational Visualization at the University of Texas, Austin, led by Bajaj [7, 8] has visualized large volumetric datasets on stereoscopic display systems using data compression and parallel processing. Hamann and Ma [9 - 11] at the University of California, Davis have developed multiresolution algorithms and parallel and distributed systems to accomplish large data visualization. The Scientific Computing and Imaging Institute at the University of Utah led by Johnson [12, 13] have developed tools to visualize large computational fields for areas such as biomedicine and oil exploration. To alleviate the burden of the rendering stage Crawfis *et al.* at Ohio University have suggested ‘Image Based Rendering Assisted Volume Rendering (IBRAVR)’ [14] that partially pre-renders large datasets on a powerful computer and performs the final rendering on a graphics workstation. Visapult project [14 - 16] at the University of California, Berkeley has exploited the IBRAVR for remote visualization of large scientific datasets. Visapult consists of two components: A viewer and a back-end pre-renderer. The back-end is a parallel application that loads large datasets using domain decomposition and performs software rendering on each sub-domain, thereby producing layered images of sub-domains. The viewer, another parallel application, implements IBRAVR to re-combine images produced by the back-end application.

Another application area for large-dataset walkthrough is architectural modeling. Several systems to accomplish this [17 - 19] utilize the properties of architectural models by subdividing the system into cells that correctly map onto the

rooms and other physical domains of the system. The UC Berkley walkthrough of a building (SODA Hall) used a hierarchical representation of the model and incorporated visibility culling and LOD. Funkhouser *et al.* [20] have proposed numerous techniques for the management of large datasets using adaptive display algorithms. More recently the work done at the University of North Carolina, Chapel Hill [21] has used culling, data management and rendering enhancements to achieve an interactive walkthrough in a tanker comprised of 82 million triangles.

The visualization systems described above have been developed for volume rendering and 3D modeling. For volume rendering, achieving speedup by parallelization is relatively simple because occlusion culling is not involved and different parts of the scene can be blended. On the other hand, 3D modeling can utilize the structure of the model in graphic optimizations such as culling and LOD control. In contrast, datasets in atomistic simulations consist of positions and other attributes (such as species, velocities and stresses) of discrete atoms, which are distributed in an irregular manner, and application of the techniques developed for volume rendering and 3D modeling is not straightforward.

### **2.3 System Overview:**

The techniques and algorithms that have been developed have been implemented in a visualization system named *Atomsviewer* that allows a viewer to walk through a very large atomic system. It is highly scalable and can visualize billion-atom systems. Atoms are rendered as discrete spheres at multiple resolutions in

an immersive environment. To achieve scalability without losing the immersive experience, we reduce the data subset that is sent to the graphics pipeline. Since the data size processed by the graphics pipeline is limited by the user's field of view and is independent of the data size, the *Atomsviewer* can render very large datasets. Data reduction is achieved by culling data that is not visible, at the initial data extraction stage. When the data subset reaches the graphics pipeline, optimizations such as a multiresolution algorithm are employed to speed up the rendering process.

### **2.3.1 Hardware Description:**

The hardware can be broken up into two distinct components namely the processing unit and the display unit. In an immersive, virtual environment, these are physically distinct units. In these cases the display system is the *Immersadesk* which 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 simulated objects. The *Immersadesk* is currently driven by an SGI Onyx2 system powered by two R12000 MIPS processors and has 4GB of system RAM and an SGI proprietary graphics pipeline known as the "Infinite Reality 2". In future versions of *Immersadesk*, the driver will be a PC cluster. The SGI machine also doubles as the processing unit in the serial version of *Atomsviewer*. In cases where immersive visualization is not a requirement, *Atomsviewer* is portable and can successfully run on a Windows driven PCs, Linux workstations and Apple desktops.

For systems that necessitate a parallel implementation of *Atomviewer*, the only hardware restriction is that the operating system on all nodes be POSIX<sup>1</sup> compliant. Operating systems, which fall in this category, include most flavors of UNIX and the new Mac OS X. Work is underway to port the parallel implementation to the Microsoft Windows architecture. As before, the *Immersadesk* acts as the virtual platform and is driven by the SGI. It is strongly recommended that, in a parallel implementation, the machine driving the virtual environment also act as the rendering node, thus minimizing network latency.

### **2.3.2 Programming Languages, Methodologies and APIs used**

The first versions of *Atomviewer* were developed using C and the structured programming methodology. The basic flow involved reading in the atom list from a file, drawing this onto the screen and finally allowing the viewer to navigate the scene. All graphic routines were written using Open GL [22] – an industry standard graphic API. During the design process of the updated versions of *Atomviewer*, it was felt that the system ought to be modular enough so that independent development can be undertaken and subsequent additions do not necessitate a complete rewrite of the source code. Consequently major parts of the application were rewritten using the Object Oriented methodology in C++ [23]. This allowed code reuse and permitted the use of different optimized features of C++ such as referencing and the various template libraries. The graphical modules are still in Open GL.

---

<sup>1</sup> POSIX: Portable Operating System Interface

When visualizing in a virtual environment, a set of libraries known as the CAVELib[24] are required. These libraries simplify the generation of a virtual environment and greatly simplify the tracking mechanism. The CAVE Libraries have the added advantage of being portable across most of the virtual immersive environments that are commercially available.

In a distributed environment the program is broken up into distinct modules each of which communicates with other modules using TCP/IP sockets conforming to the POSIX standards. Furthermore each module is multithreaded and the inter-process communication takes place over POSIX compliant mutex locks.

In a desktop version *Atomsviewer* has a user-friendly interface. This has been developed using the Java JFC/Swing 1.1 API. This allows the front end to be portable across all platforms and since the underlying application uses the OO methodology the GUI-Application interface is trivial and can easily be upgraded if ever there is a need.

## CHAPTER 3

# OCTREE DATA STRUCTURE

The central issue that is faced in the visualization and analysis of large size data sets is data management. A large system is defined as one with approximately one million or more atoms and has data over few tens of time steps. The data that is usually required to completely define a single atom consists of a specie identifier, a three dimensional coordinate and a set of physical attributes such as charge, temperature, stress etc. In most systems, therefore, an atom would take up about 30bytes. Thus we see that a large system with multiple frames is typically in the range of a few gigabytes. Such systems cannot be tackled with a brute force approach of loading up the entire data in memory and rendering it. Instead a smart data structure needs to be devised that is fast and does not require large amounts of memory. In *Atomsviewer*, this data structure is the octree. In this section the octree is explained in greater detail and a distributed implementation is introduced that allows data management of very large size systems ( $\sim 1.5\text{GB}$  per time step).

### 3.1 Introduction

An octree is an extension of the binary search tree (BST) where each node spawns eight children. While a BST works very well for one dimensional data, a multi-dimensional environment would involve the use of nested trees which is a rather cumbersome and expensive approach on account of the ensuing computational complexity and memory overheads involved in creating and managing such a complex structure. Therefore for higher dimensions we use multi-way trees. An octree is one



such multi-way tree that can be used to handle three-dimensional data. Each of its children represents one of the eight octants in subdivided space. It acts as a logical representation of three-dimensional data and permits rapid searching of data since each search reduces the amount of data being searched to an eighth. The idea of using such a hierarchical data structure was first proposed by Clark [6]. He proposed its use in determining varying levels-of-detail of an object. The details in an object (granularity used in its rendering) can be stored at various levels of an octree and the desired detail can be picked by associating the depth with the distance of the object from the viewer. He also proposed its use in clipping where we exploit the spatial ordering of data to extract the subset of interest by reducing the problem to one of extracting a set of octree nodes that belong to the subset of interest. These are all easily implemented and have a logarithmic time complexity.

### **3.2 Design and Implementation of the Octree**

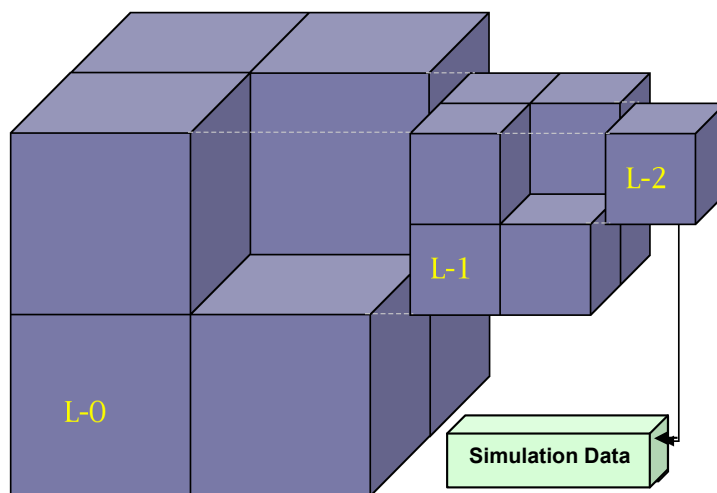
An octree is obtained by hierarchically subdividing a three-dimensional space into eight parts (octants) [25, 26]. This subdivision is continued recursively until subsystems of a desired size have been created. While this approach is simplistic and incurs memory wastage (in the nodes that do not contain any data and are needlessly subdivided) it is preferred because the cost of memory wastage is recovered from the computational speed that is achieved in searching a very regular data structure. This is particularly suitable to our problem, where we have a large geometric dataset and the primary objective is to extract region of interest as fast as possible. However it is

possible to avoid this memory wastage and these techniques are subsequently discussed.

Before the implementation and usage is discussed certain terms need to be understood that greatly improve the overall understanding. The *depth* of an octree is defined as the number of levels in the octree and equals the number of times the system is partitioned. Since the partition is uniform, there are  $8^l$  subsystems at level  $l$ . The terminal nodes (nodes at the lowest level) are known as *regions*. The *depth* is calculated at the preprocess stage and is determined by keeping approximately 500 atoms in each region. Thus if there are  $N$  atoms in a system then the depth  $d$  of the octree is defined as  $d = \log_8(N / 500)$ . This limit is empirically determined and is a measure of the desired visual granularity. Since the field-of-view is reconstructed from octree regions, a higher granularity results in a better approximation of the field-of-view. However for systems of ten million to a billion atoms, our benchmark tests showed us that increased granularity does not provide noticeable visual gains.

At this point it should be mentioned that the octree by itself does not store the atomic data but only acts as an abstraction of the atomic data. The atomic coordinates and related atomic attributes for each *region* are stored in arrays and it is these array indices that are contained in the individual regions of the octree see Fig. 2. This design permits greater flexibility since it only requires a single octree for a large multi-frame system. Furthermore the data can be stored in a physically separate location from the octree and the data extraction process. This second point is particularly significant in the design of the distributed implementation of the octree since it allows a set of nodes

to extract the regions that can be viewed and convey this information to the visualization module without actually sending the viewable data resulting in reduced network load and consequently smaller network latency.

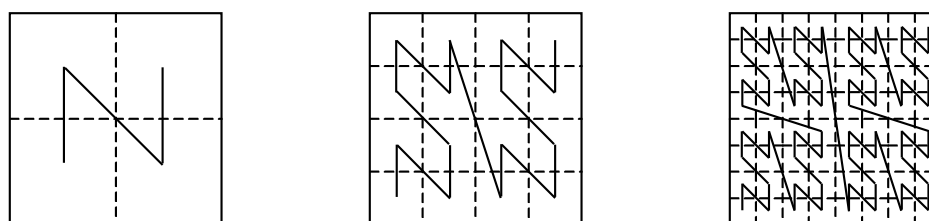


**Figure 2.** A three-level octree implementation. Each node contains the coordinate bounds of the corresponding subspace. A terminal node at level 2 contains a pointer to a structure that stores data associated with atoms in the region defined by the terminal node.

Since the octree is acting as an abstraction it must be created before atoms can be inserted. This is done by calculating the maximum extents of the system across all frames and creating a cubical system of these dimensions such that the cubical system fully encloses the physical system over all the time steps. The reason behind using a cubical system comes from the fact that in the extraction process a spherical approximation of a region is used and a cube accurately approximates a sphere. The process of spherical extraction is explained in the next section.

Once the octree has been created, individual atoms are read and based on their coordinates, their respective regions are determined. The process of determining the respective region of an atom does not require the traversal of the octree and as long as

the depth and the extents of the octree is known, the atoms can be inserted. The insertion mechanism draws from the observation that if an in-order traversal of an octree is conducted, and the order in which the terminal nodes are visited is mapped, a space filling curve, known as the Z curve emerges. A Z curve is obtained by joining nodes that are in a certain sorted order. It is recursively defined and successive refinement of one such curve is illustrated in Fig. 3.



**Figure 3.** An in-order quadtree traversal producing a Z curve

Now it is trivial to determine the relative position of an atom with respect to the minimum octree extents. This information is used to determine the center coordinates of the region that contains the atom. Let us assume that  $\langle x, y, z \rangle$  are the center coordinates of a certain region. The octree index R can be easily determined by interleaving the  $x, y$  and  $z$  bits. Now, for example if  $x = 12, y = 10$  and  $z = 20$  then R is computed as follows:

$$\begin{array}{rccccccccc}
 x = & 0 & & 1 & & 1 & & 0 & & 0 \\
 y = & & 0 & & 1 & & 0 & & 1 & & 0 \\
 z = & & & 1 & & & 0 & & 1 & & 0 & & 1 \\
 \hline
 R = & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & = 7505
 \end{array}$$

Therefore an atom which is contained in the region of center  $\langle 12, 10, 20 \rangle$  is stored in region number 7505 of an octree of depth 5. This method is significant not only

because it clearly distinguishes the abstraction (the octree) from the data, but also because it gives a quick and easy method to determine the neighbors of a particular octree node.

### **3.3 Octree Based Visibility Culling:**

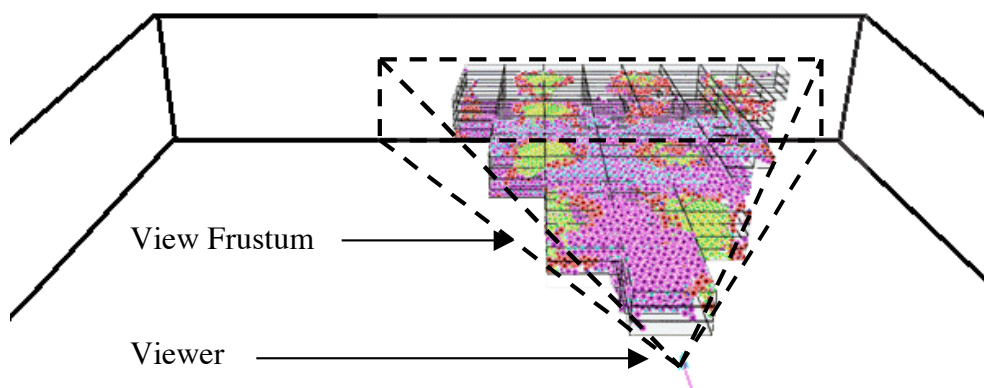
The key idea behind the use of the octree was to have a scalable data structure that was capable of storing large systems and would permit fast extraction of a subset that satisfied certain properties. The octree achieved this and in the pervious section we explained the techniques that are used in the setup of the data structure. The key idea in the extraction of viewable regions is a two step process involving a coarse extraction of all regions contained in the field-of-view and a subsequent refinement to ensure that a minimal set of viewable regions is passed to the graphic modules without any compromise to the immersive feel of the visualization.

The first step in the visibility-culling scheme involves traversing the octree and extracting a superset of regions that are contained in viewers field-of-view. To do so, we approximate every region as a sphere, which is centered at the center of the region and has a radius that would allow the cube to be fully contained within the sphere. Since all regions at a given level will be of the same size therefore the only information that needs to be stored by each node of the octree is its center. Next we approximate the viewers field-of-view as a sphere which is centered at a point in front of the viewer at a distance which is half the depth-of-field of the system and in the direction of the viewers line-of-sight. Thus the viewer lies on the surface of the viewing sphere. Now the octree is traversed and all spheres (region abstractions) that are contained within or

intersect with the viewing sphere. Since the data structure is a tree structure, the algorithm is implemented recursively.

Once a coarse viewable subset has been extracted, it is refined by removing all those regions that are in the viewing sphere but not in the field-of-view. To do this we employ a set of bounding volumes like a cylinder and a cone. The radius of the cylinder and the angle of the cone are a function of the viewing extent.

In the final step the selected regions are sorted based on their distance from the viewer. This step is essential since it allows features like multi-resolution rendering of a region based upon its distance from the viewer. Fig. 4 illustrates the subset of regions that have been extracted after a traversal.



**Figure 4.** Illustration of Octree based visibility culling. About two hundred octree leaves remain in the final frustum that goes through graphics rendering pipeline. A bounding-box depicting the view frustum and the position of the viewer are included in the figure to serve as guides.

The total time complexity of the octree-based data management includes the time  $T_1$  for initial setup of the octree and the time  $T_2$  for processing the octree at each frame of visualization. The time complexity for the setup is  $T_1 = O(N)$  [27] where  $N$  is the total number of the atoms in the system. For each visualization frame during

walkthrough, a view-frustum culling involves testing for whether a terminal node in the octree is within the field of view. Such tests are conducted by traversing the octree from the root to the terminal nodes that are within the field of view. The time complexity for extraction is  $T_2 = O(m \log_8(N/c))$ , where  $m$  is the total number of such nodes and  $c$  is the number of atoms per node. The total number of atoms in the field of view determines  $m$  and therefore it is independent of  $N$ . The parameter  $c$  is determined by the average atomic number density and is also independent of  $N$ .

### 3.3.1 Parallel and Distributed Implementation

The system described so far is capable of handling systems that are under a few million atoms. However a serial implementation is incapable of handling multi-million atom systems because of the sheer size of the data, which is typically in the range of a few GB per frame. For such systems a parallel and a distributed approach ensures that the visualization tool remains interactive and scalable. The data structure is still the octree, however the octree based culling is moved to a set of nodes. This is possible on account of the octree based atomic abstraction described earlier, which ensures that the data and its abstraction can be physically separate.

In the parallel implementation each processor keeps a copy of the complete octree. The only difference between the serial implementation and the parallel implementation of the octree is that in the serial version the process of extracting a coarse viewable subset uses one sphere centered at a point one-half of the depth-of-field, in front of, and in the direction of, the viewer. In the parallel version the single sphere is replaced by a set of shells and an inner sphere, all of equal volume. The

number of shells is one less than the number of available processors<sup>1</sup>. These shells then perform the same form of traversal that was used in the single processor implementation, with the obvious modification of a child being visited if its parent lies in the bounding shell. Breaking the bounding volumes in this fashion allows load balancing among the processors. While it may seem wasteful to keep multiple copies of the octree across all nodes, this is done in-order to reduce the communication and make fault tolerance and process migration easier. In the event of a network disturbance such as the loss of a node, the module will only have to recalculate the radii of its bounding shells and repeat the data extraction process for the instance at which the disturbance occurred, to recover from the network disturbance. The design and implementation of a parallel and distributed implementation of an octree in a visualization system is described in greater detail in Chap. 5.

---

<sup>1</sup> One processor is allocated to the innermost sphere, *i.e.*, the core.



## **CHAPTER 4**

### **GRAPHICAL SYSTEM**

The graphical system is responsible for the rendering process and since this process is by far the slowest, various optimizations have to be introduced so as to keep the visualization interactive. One such technique is visibility culling, which involves removing all the data that is not in the viewer's field-of-view. This is done by the octree, which culls all such data. However not all data that is in the viewers field-of-view can be seen by the viewer since a significant part of it is occluded by data in front of the viewer. This is achieved by a process known as occlusion culling, which removes all such potentially visible but practically invisible (on account of occlusion) data. Finally observation tells us that not all components need to be drawn with the same resolution. Thus a sphere which is very far from the viewer can be depicted as a four or five-sided polygon rather than a smooth circle with realistic illumination. This section describes various such techniques that are used to optimize the graphic routines.

#### **4.1 Occlusion Culling Algorithm:**

Occlusion culling refers to the process of culling those polygons that are in the viewer's field-of-view but are not visible since they have been occluded by other polygons that are nearer to the viewer. This is a very cumbersome process since it involves processing the graphical entities on a per-primitive basis. Since a typical scene would contain about 200,000 atoms with an average resolution (all atoms are not drawn at the same resolution) of 30 polygons, there are approximately 6 million

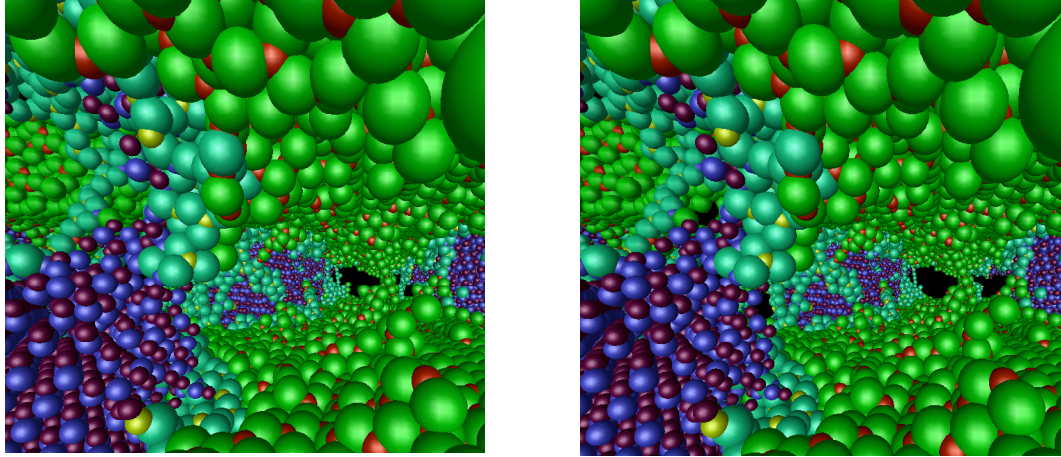
polygons. Thus we can see that a traditional occlusion-culling algorithm would incur a significant computational penalty, which destroys the savings in rendering time, gained by occlusion culling. Therefore two modified approaches to the problem are adopted, each of which can be used independently or in succession. These are based on a probabilistic model [28] and a depth-based model [28].

#### **4.1.1 Probabilistic Occlusion Culling:**

This methodology is based upon the assumption that in most systems the atomic distribution in every octree region is uniform. To understand this, let us create 4 sets of visible regions whose radial distance from the viewer is within a certain range and mark these sets as A, B, C and D. Now if we assume that all of regions contained in A are completely visible then it can be said that there is a high probability that A will occlude a certain percentage of B. Similarly B will occlude some percentage of C and so on. Now by calculating the density of each region and using the set of regions that occlude the region of interest we can set up a recursive relationship to determine the visibility function. Once the visibility function for all regions has been determined, it is used to randomly select atoms from each of the regions that will be drawn.

Implementation of this algorithm shows that there is a visually insignificant loss of information and whatever loss does occur is in atoms at a distance. The computational costs are also very small since most of the computations are performed on a per-region basis. In Fig. 5, the picture on the left is drawn without probabilistic occlusion while the one on the right has this occlusion enabled. As we can see there is very little loss in visual detail but the computational gains are significant. In this

instance, the occlusion culling had reduced the number of atoms drawn by 68% and increasing the frame rate three times.



**Figure 5.** An illustration of the effect of probabilistic occlusion. The image on the left is rendered without the algorithm, while the one on the right uses it, resulting in 68% fewer atoms and a three times increase in the frame rate.

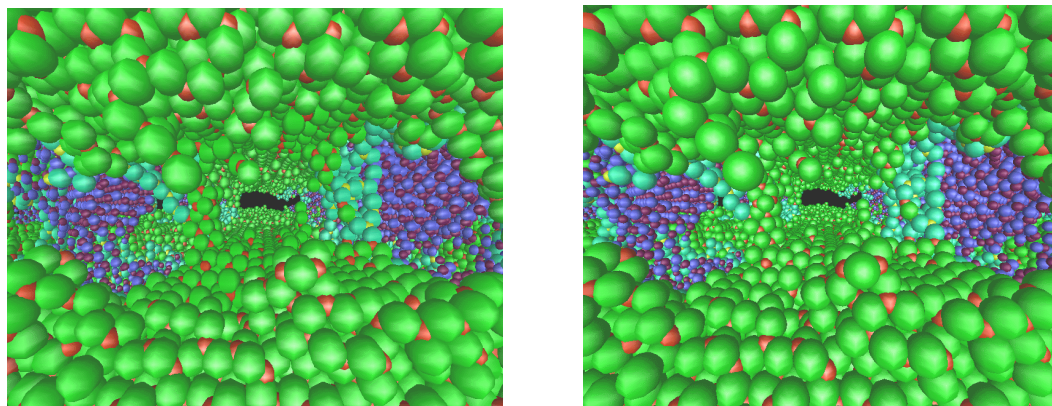
#### 4.1.2 Depth Based Occlusion Culling:

This is a slightly more expensive process than the probabilistic technique and is performed on a per-polygon level by the use of a simulated depth-buffer and a configurable sequence of tests against that buffer. While being on a per-polygon level it is faster than a traditional occlusion culling technique on account of its use of a simulated depth buffer. Here for every atom that needs to be drawn, a shape is generated that approximates the screen space that will be occupied by the atom. For simplicity, we use a rectangular shape, with which we associate a constant depth value representative of the atom's distance from the viewer. Using that shape and its depth value, we test it against our depth buffer across a number of test points to determine whether any part of that area is visible to the viewer and should consequently be drawn. If for any of the test points, the value of our shape is lower than the value in the

depth buffer at that point, then the atom is determined to be visible at that point. Consequently, the depth tests at subsequent points are bypassed, the depth buffer is updated for that shape, and the atom is passed on to the rendering module to be rendered. If for all of the test points, the shape is not determined to be visible, then we do not update the depth buffer and the atom is “discarded”.

## 4.2 Multiresolution Culling Algorithm:

The various optimizations described so far act upon the atomic data that needs to be rendered. However if we analyze Fig. 6, we see two pictures that are identical. However what is not visible is the fact that the picture on the left uses the same number of polygons for each of the three spheres while the picture on the right uses a fewer polygons per sphere as the distance from the viewer increases. Thus we see that a significant saving in rendering time can be obtained if we draw spheres with varying number of polygons.



**Figure 6.** An illustration of the multiresolution algorithm. The image on the left uses the same resolution to draw all constituent atoms while the image on the right uses five different resolutions. Both images have the same maximum resolution.

To implement this scheme we predetermine the various resolutions at which spheres will be drawn and the range of distance-from-viewer associated with each resolution. Finally we create display lists (an Open GL macro that allows a graphical object to be drawn in the least amount of time) for each resolution. Our tests reveal that good visual results can be obtained by using 8 to 300 polygons per sphere.

A modification of this algorithm involves associating the speed of the viewer with the resolution. By studying the characteristics of a user it was observed that if the viewer is traveling fast then the objective is moving to a new location. However if the viewer is traveling slowly then in all likelihood the viewer is observing the system. Therefore if we draw atoms with a lesser resolution when the user is traveling fast and at higher resolutions when the user is traveling slowly, then rendering time and the refresh rate decreases noticeably. Similarly speed can be associated with the extent of the viewers field-of-view wherein the user has a wider field-of-view when moving slowly and a narrow field-of-view when moving fast.

## **CHAPTER 5**

### **PARALLEL AND DISTRIBUTED OPERATION**

As mentioned earlier, visualization of multi-million particle systems is a computationally intensive task. While data reduction techniques such as octree based visibility culling and graphic optimization techniques such as occlusion culling and multi-resolution culling do reduce the load on system; assigning all graphics operations to a single node can significantly decrease the frame rate. This section describes the design and methodology that are used to achieve a distributed operation.

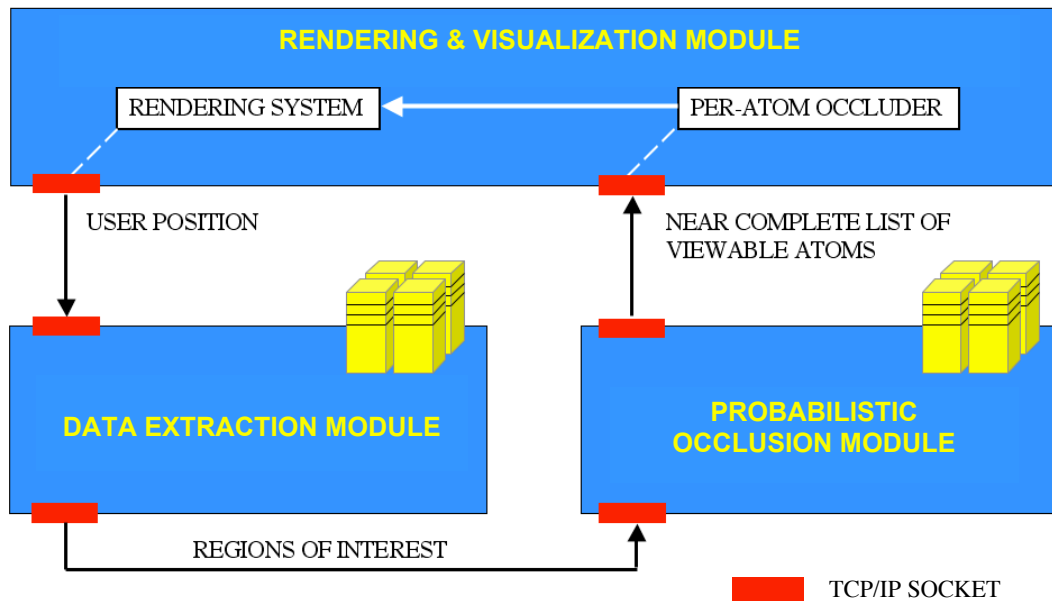
#### **5.1 Distributed Design and Implementation:**

A distributed system capable of interactive visualization of very large systems is designed by breaking up the serial implementation into three logically distinct components. Each of these components is further parallelized if necessary. The three components are:

1. Data Extraction Module (DEM);
2. Probabilistic Occlusion Module (POM);
3. Rendering and Visualization Module (RVM);

Fig. 7 shows a schematic of the system in which the decomposition of the system into the three components and their mutual interaction are represented by dotted rectangles and arrows, respectively. At runtime the user position and orientation are tracked and their change triggers a message to be sent to the Data Extraction Module (DEM). The DEM employs an octree-based data abstraction (described in Section 3) to obtain a subset of the total system, which is contained in the viewer's field-of-view. This

subset is forwarded to the Probabilistic Occlusion Module (POM), which employs a probabilistic occlusion culling technique (see Section 4) to refine the data subset by removing atoms in the viewer’s field-of-view but occluded by other atoms. The result of the POM operation is forwarded to a Rendering and Visualization Module (RVM), in which the atoms are rendered after pruning the data subset to a minimal set of atoms visible to the viewer.



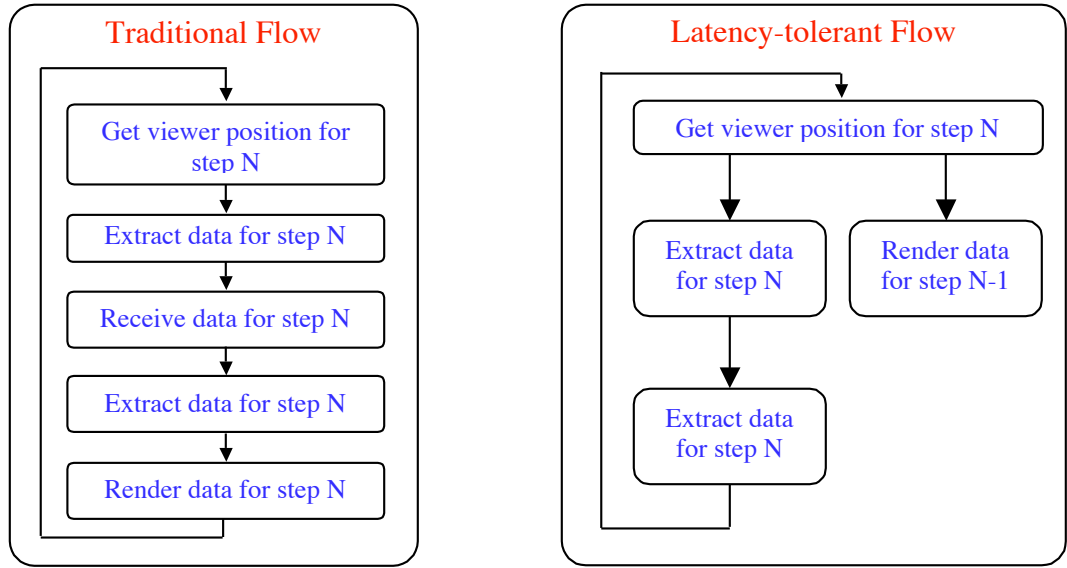
**Figure 7.** Distributed architecture of *Atomviewer*.

Inter- and Intra- (in a parallelized module) modular network communication is conducted over TCP/IP sockets. The main reason behind using sockets for communication was speed. Since the sockets are the lowest form of network communication available they are more flexible, easily manageable and have very little overhead. Each of the modules is a multithreaded application with one pair of threads managing the network communication and one or more application threads. A network deadlock between the modules, on account of a slow application thread, is avoided by

the use of data queues on the networking threads. A schematic showing the various threads and the network operation is provided in the appendix.

## 5.2 Latency Hiding – Communication / Computation Overlap:

While 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 viewer’s field-of-view, before it performs occlusion-driven data reduction. Finally the RVM needs to receive, from the POM, a set of atom IDs. To ensure that module operations are not affected by network delays, we overlap the inter-module communication with the module computation [29]. This overlap is achieved in the RVM, since it triggers the sequence of events driving the DEM and the POM.



**Figure 8.** An illustration of the data-flow in a communication / computation overlap (right) and a traditional system (left).



In the traditional data-flow scheme, the RVM at time  $t$  renders the scene at 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 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 overlap scheme is schematically shown in Fig. 8. It is implemented by employing multiple application threads in each of the modules and a queuing mechanism on the data-receiver thread. This approach makes the communication non-blocking and is especially useful in the DEM where it can accept multiple viewer positions.

## CHAPTER 6

### RESULTS AND CONCLUSIONS

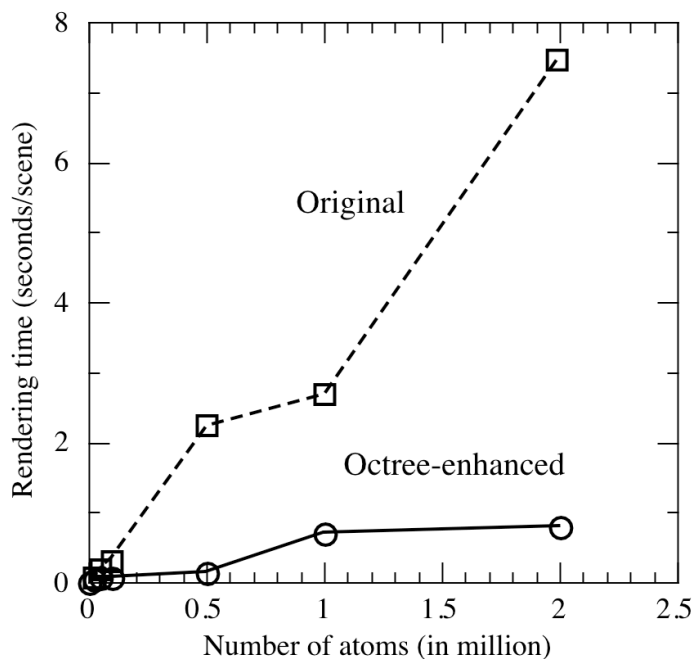
As mentioned earlier, the various techniques described here have been implemented in a visualization application called *Atomviewer*. For the purpose of testing *Atomviewer*, the serial version is tested on an SGI Onyx2 with two R12000 processors (300 MHz), 4 GB RAM, and an InfinityReality2 graphics pipeline. The virtual environment is *Immersadesk* [30] and 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 simulated objects.

For the parallel version *Atomviewer* was tested on a cluster made up of four PCs running Linux 6.2 each with an 800 MHz Pentium III processor and 512MB RAM and the SGI machine described earlier. The PC cluster run the DEM and the POM while the RVM runs on the SGI machine.

#### 6.1 Serial Operation:

We have performed a scalability test of *Atomviewer* on data comprising of up to two million atoms. Figure 9 compares the timing results of *Atomviewer* with and without the fast visibility culling based on the octree data structure. With the octree enhancement, the time to extract and render the atoms within the field of view is an almost constant function of the number of atoms. This is in contrast to the conventional rendering for which rendering time increases linearly with the number of

atoms. The scalability demonstrated in Fig. 9 is due to the reduced number of polygons to be rendered by a sequence of optimization techniques.



**Figure 9.** Rendering time per scene as a function of the number of atoms for *Atomsvviewer* with and without the octree based fast visibility culling. Lines are guide to the eyes.

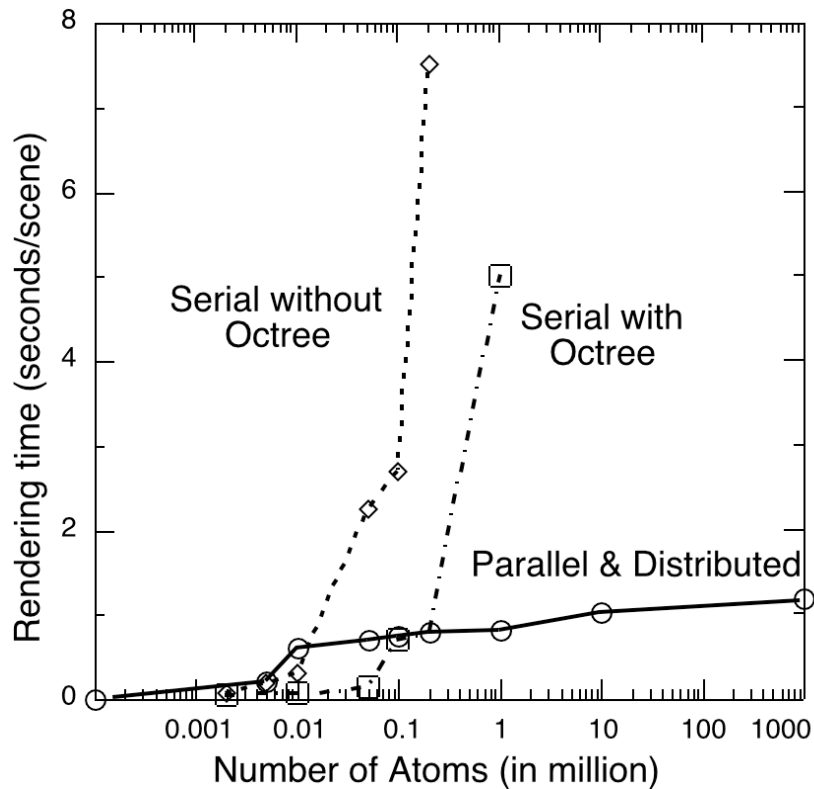
**Table 1.** Performance results of various techniques used to reduce the numbers of atoms and polygons for an 800,000-atom dataset.

Optimization method	Atoms remaining	% Atoms remaining	Polygons remaining	% Polygons remaining
None	800,000	100	26,072,352	100
Octree extraction	408,548	51	11,073,536	42
Visibility culling	108,000	13	3,661,280	14
Multiresolution	108,000	13	782,170	3

In Table 1 we can see the reduction achieved by various routines for a 800,000 atom system in terms of the total number of atoms and the number of polygons that have been eliminated from the graphics pipeline.

## 6.2 Parallel Operation:

We have performed a scalability test of *Atomsviewer* involving up to a billion-atom system. Fig. 10 compares the timing results of the serial *Atomsviewer* with and without the octree-based fast visibility culling with that of the parallel and distributed *Atomsviewer*. We see 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.



**Figure 10.** 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.

Fig 10 also demonstrates the scalability of the system due to the suppression of the communication overhead and the reduced number of polygons to be rendered by a sequence of optimization techniques. In Table 2 we can see the reduction achieved by various routines for a billion-atom system in terms of the total number of atoms and the module latency for the same system. It also shows the average network latency for the complete system.

**Table 2.** Performance results and module latency for the three modules of *Atomsviewer* visualizing a one billion-atom system.

Data Reduction Method	Atoms Remaining	Avg. Time (in msec)
DEM	500,000	10
POM	100,000	35
RVM without depth test	100,000	140
RVM with depth test	50,000	103
Network Latency		240

## REFERENCES

1. Kalia, R.K., Campbell, T.J., Chatterjee, A., Nakano, A., Vashishta, P., and Ogata, S.: Multiresolution algorithms for massively parallel molecular dynamics simulations of nanostructured materials. *Comp. Phys. Comm.* 128, 245-259 (2000)
2. Nakano, A., Bachlechner, M.E., Branicio, P., Campbell, T.J., Ebbsjö, I., Kalia, R.K., Madhukar, A., Ogata, S., Omeltchenko, A., Rino, J.P., Shimojo, F., Walsh, P., and Vashishta, P.: Large-scale atomistic modeling of nanoelectronic structures. *IEEE Trans. Electron Devices* 47, 1804-1810 (2000)
3. Nakano, A., Kalia, R.K., and Vashishta, P.: Scalable molecular-dynamics, visualization, and data-management algorithms for materials simulations. *Comp. Sci. Eng.* 1(5), 39-47 (1999)
4. Shimojo, F., Campbell, T.J., Kalia, R.K., Nakano, A., Vashishta, P., Ogata, S., and Tsuruta, K.: A scalable molecular-dynamics-algorithm suite for materials simulations: design-space diagram on 1,024 Cray T3E processors. *Future Generation Comp. Sys.* 17, 279-291 (2000)
5. Nielson, G.M., Mueller, H. and Hagen, H.: *Scientific Visualization Overviews, Methodologies and techniques*, IEEE Computer Society Press 1997
6. Clark, J.H.: Hierarchical geometric models for visible surface algorithms. *Comm. ACM* 19(10), 547-554 (1976)
7. Bajaj, C. and Cutchin, S.: Web based collaborative visualization of distributed and parallel simulation. *Proc. of IEEE Parallel Visualization and Graphics Symposium*, pp. 47-54. San Francisco, CA: IEEE 1999
8. Bajaj, C., Ihm, I., and Park, S.: Visualization-specific compression of large volume data. *TICAM Technical Report 00-17*. University of Texas at Austin 2000
9. Ma, K.L. and Camp, D.: High performance visualization of time-varying volume data over a wide-area network. *Proc. of Supercomputing 2000*. Dallas, TX: IEEE 2000
10. Olson, A.J., Pailthorpe, B.A., Toga, A.W., Wunsch, C.I., Genetti, J.D., Nadeau, D.R., Sanner, M.F., Bajaj, C., Bordes, Hamann, B., Takanashi, I., Thompson, and Meyer, J.: Scalable visualization toolkits for brains to bays. *Envision*, Vol. 16, No. 4, pp. 8-9. San Diego, CA: San Diego Supercomputer Center 2000

11. Pinskiy, D.V., Meyer, J., Hamann, B., Joy, K.I., Brugger, E.S., and Duchaineau, M.A.: An error-controlled octree data structure for large-scale visualization. *Crossroads—The ACM Student Magazine*, Spring 2000, pp. 26-31. ACM 2000
12. Johnson, C., Parker, S., Hansen, C., Kindlmann, G., and Livnat, Y.: Interactive simulation and visualization. *IEEE Computer* 32(12), 59-65 (1999)
13. Parker, S., Shirley, P., Livnat, Y., Hansen, C., Sloan, P.-P., and Parker, M.: Interacting with gigabyte volume datasets on the Origin 2000. *The 41st Annual Cray User's Group Conference*. 1999
14. 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. *Proc. of Supercomputing 2000*. Dallas, TX: IEEE 2000
15. Bethel, W.: Visualization dot com. *IEEE Computer Graphics and Application* 20(3), 17-20 2000
16. Bethel, W.: A prototype remote and distributed visualization application and framework. *Proc. of SIGGRAPH 2000*. New Orleans, LA: ACM 2000
17. Airey, J.M., Rohlf, J.H., and Brooks F.P.: Towards image realism with interactive update rates in complex virtual building environments. *Proc. of ACM Symposium on Interactive 3D Graphics* 24(2), 41-50 (1990)
18. Funkhouser, T.A., Teller, S.J., Sequin, C.H., and Khorramabadi, D.: The UC Berkeley system for interactive visualization of large architectural models. *Presence* 5, 45-60 (1996)
19. Teller, S. and Sequin, C.H.: Visibility preprocessing for interactive walkthroughs. *Proc. of ACM SIGGRAPH 1992*, pp. 55-64. ACM 1992
20. Funkhouser, T.A., Sequin, C.H., and Teller, S.J.: Management of large amounts of data in interactive building walkthroughs. *Proc. of SIGGRAPH Symposium on Interactive 3D Graphics*. Boston, MA: ACM 1992
21. Aliaga, D., Cohen, J., Wilson, A., Zhang, H., Erikson, C., Hoff, K., Hudson, T., Stuerzlinger, W., Baker, E., Bastos, R., Whitton, M., Brooks, F., and Manocha, D.: A framework for the real-time walkthrough of massive models. *Computer Science Technical Report TR98-013*. Univ. of North Carolina 1998.
22. Woo, M., Neider, J., Davis, T., and Shreiner D.: *The OpenGL Programming Guide*, 3<sup>rd</sup> Ed. Reading, MA: Addison-Wesley 1999

23. Stroustrup, B.: The C++ Programming Language, Special Ed. Addison-Wesley Longman, Inc. 2000
24. VRCO Inc. <http://www.vrco.com>
25. Jackins, C.L. and Tanimoto, S.L.: Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing* 14, 249-270 (1980)
26. Wilhems, J. and Gelder, A.V.: Octrees for faster isosurface generation. *ACM Trans. Graphics* 11(3), 201-227 (1992)
27. Omeltchenko, A., Campbell, T.J., Kalia, R.K., Liu, X. Nakano, A., and Vashishta, P.: Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm. *Comp. Phys. Comm.* 131, 78-85 (2000)
28. Sharma, A *et al.* Immersive and Interactive Visualizations for billion atom systems.: *Proc. of IEEE Virtual Reality 2002*, 217-223 (2002)
29. Barnard, S., Biswas, S., Saini, R.F., Van der Wijngaart, Yarrow, M., Zechter, L., Foster, I., and Larsson, O.: Large Scale Distributed Computational Fluid Dynamics on the Information Power Grid Using Globus”, *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, Annapolis, Maryland, Feb. 1999
30. Fakespace Systems Inc. <http://www.fakespacesystems.com>



## **VITA**

Ashish Sharma was born on June 24, 1977, in New Delhi, India, to Gp. Capt. Rajiv and Suman Sharma. He completed high school from Bhartiya Vidya Bhavan in Chandigarh, India.

He studied at Punjab Engineering College, Chandigarh, India from 1995 to 1999 and graduated with a Bachelor of Engineering degree in Electrical Engineering.

Ashish came to the United States in July of 1999 to study Computer Science at Louisiana State University in Baton Rouge, Louisiana. He enrolled in the Department of Computer Science to study towards obtaining the degree of Master of Science in Systems Science. His future plans include getting a doctoral degree from the Department of Computer Science under his advisor Dr. Aiichiro Nakano.