Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

Interactive Parallel Visualization of Large Particle Datasets

Kevin Liang Patricia Monger Huge Couchman

Research & HPC Support
McMaster University
Hamilton, ON L8S 4M1
Email: [xkliang|monger]@mcmaster.ca

Department of Physics and Astronomy
McMaster University
Hamilton, ON L8S 4M1
Email: couchman@physics.mcmaster.ca

Abstract
This paper presents a new interactive parallel method for direct visualization of large particle datasets. Based on a parallel rendering cluster, a frame rate of 9 frames-per-second is achieved for 256^3 *particles using 7 render nodes and a display node. This provides real time interaction and interactive exploration of large datasets, which has been a challenge for scientific visualization and other real time data mining applications. The system allows scientists to study and to analyze the simulation results by viewing the particle cube from different perspectives, flying through the simulation field, or diving into the internal structure of the particles. A dynamic data distribution technique is designed for visualizing a highlighted subset of the particle volume. It maintains the load balance of the system and minimize the network traffic by updating the rendering pipeline through reconfiguration of the rendering chain. The method can be easily extended to other large datasets such as hydrodynamic turbulence, fluid dynamics, and so on.*

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Distributed / network graphics, I.3.2:C3Computer GraphicsReal-time and embedded systems, I.3.8Comp[uter GraphicsApplications

1. Introduction

One of the scientific visualization problems is to interactively visualize simulated large particle datasets, usually containing more than a hundred million particles. Three methods have been used to visualize large particle datasets. Researchers view the particles directly, convert the particles to volumetric data representing particle density [MHS99a], or explore the particles using a combined direct- and volume-based rendering method [WMQR02]. Direct particle rendering takes longer time on desktop graphics workstations; therefore, it is usually difficult to obtain interactive frame rate. Volume rendering and combined methods can provide interactive frame rates but have the limitation of missing fine structures. In this paper, we investigate a direct rendering method using a parallel rendering cluster, the Sepia [MHS99b] from Hewlett-Packard. The aim is to develop a system which can interactively explore 256^3 and 512^3 particle datasets without missing fine structures.

Interactive particle rendering is very attractive to scientists who are interested in exploring particle datasets in full detail and from many different perspectives. It helps study the development of cosmic structure, such as star formation,

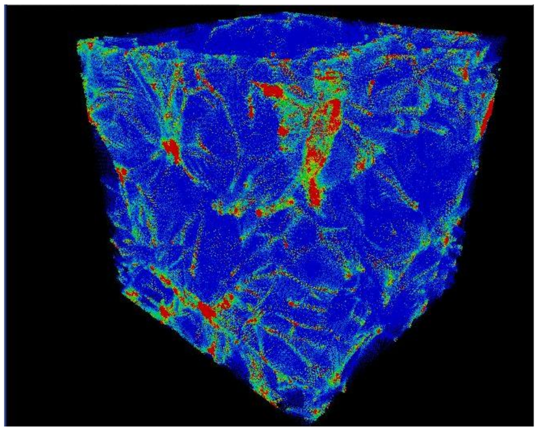
Figure 1: *Direct rendering of a* 256^3 *particle dataset.*

planet formation, and galaxy evolution [WSQ04, JFP*98]. This, however, is so computationally and graphically intensive that a single desktop computer is unable to handle the problem. Our research is based on a rendering cluster with a Sepia-2a compositing board [MHS99b] at each node. The cluster has 8 nodes, configured as 7 render nodes and a display node. For a 256^3 particle dataset, as shown in Figure 1, a frame rate of 9 frames-per-second (*fps*) is obtained when running on 7 render nodes and a display node. The system is designed to allow scientists to interactively navigate the particle volume and to study simulation results using various particle visual representations.

For 512^3 particle datasets, the system runs at 1 *fps* using 7 render nodes and a display node. By using more render nodes, integrating more advanced rendering techniques, and applying other optimization techniques, we expect to obtain higher frame rate.

1.1. Related Work

Visualization of simulated particle datasets has focused on direct particle rendering and volumetric rendering. The direct rendering method views each particle using a simple geometric primitive, often a single point. Volume-based particle rendering first converts the particles to volumetric data and then displays the particles as iso-surfaces or as a series of closely-spaced parallel texture mapped planes [Lev88]. Direct rendering has limitations for interactive exploration of large particle datasets using desktop graphics workstations. A workstation either is unable to process such large datasets or takes too long for rendering. Using a hierarchical data structure and principal component analysis technique, Hopf and Ertl [HE03] proposed a splatting method to visualize large scattered data. The hierarchical data structure separates clusters from point data. Clusters are rendered as splats to accelerate the rendering process as a trade-off of image quality on standard workstations. Our method maintains all the fine features of the particle volume and to provide interactive frame rate using parallel rendering technique on a rendering cluster. Other methods, such as those proposed in [RL00] and [ZPvBG01], are more focused on rendering surfaces instead of points as in our work.

In volume rendering, the range covered by the particle dataset is evenly divided into voxels, and each voxel value is assigned a density based on the number of particles inside it. The data are then converted into a texture and rendered on the screen as a series of parallel texture-mapped planes, creating the illusion of volume. This method can provide real-time visualization when using lower resolutions [MHS99a]. A major drawback of volume rendering is its missing fine structures due to limited voxel resolution. A combined method [WMQR02] provides a compromise solution to interactive visualization of large particle datasets, but it suffers the same low resolution and missing fine structure problems as volume rendering does.

Moll et. al. [MHS99b] proposed a scalable 3D compositing architecture, called Sepia, to interactively render partitioned datasets. A Sepia cluster consists of a number of nodes, each with a Sepia-2a compositing card that implements a sort-last image composition engine. It provides scalability for interactive volume rendering [LMS*01] and visualization of large scientific datasets [HM99]. A high bandwidth compositing network makes interactivity possible by its speed, scalability, and low latency. Each node in the cluster can be configured as either a render node or a display node. Render nodes, the workers, are responsible for portions of rendering task. A display node, the master, is the last node in the Sepia rendering pipeline. It can be used to control user interaction, coordinate operations among render nodes, and project the composited images to the target display.

Using a Sepia system avoids the memory problem as experienced in stand-alone graphics workstations and allows for dealing with increasingly large scientific datasets. In addition a scalable application can be easily designed on the Sepia system. The more render nodes in the system, the larger particle datasets can be processed or the higher frame rates can be achieved.

2. Data Distribution and Parallel Visualization

2.1. Static Data Distribution

In this work, we focus on datasets from a series of simulations in a periodic volume. The dataset is organized as randomly distributed 3D points. Each particle is defined by its 3D position, mass, density, and velocity. Based on a static workload distribution method, the dataset is initially distributed to individual render nodes. An equal portion of particles is allocated to each render node. This static equal distribution strategy guarantees that no individual node becomes the performance bottleneck because each has almost same number of particles to process. The current static distribution is solely based on the data organization of the simulation results, in which particles are randomly distributed within the volume. This data organization obviously prevents some better sorting methods to improve the rendering process. A pre-process procedure could be applied to sort the particles into a hierarchical data structure such as octree first, as in [HE03], and then the static distribution method is used to distribute the particles based on the sorted hierarchy of the datasets. Because, in this case, the master program has to load all the particles into its memory and sort the dataset, a single node in our current system does not have enough memory to hold all the particles when dealing with 512^3 particle datasets. In addition, for a series of datasets, this will greatly increase the loading time and distribution time. Therefore, currently we simply distribute particles based on the original particle organization of the dataset. The dataset is partitioned into equal portion of particles and each render node concurrently loads only its portion of the particles.

2.2. Dynamic Data Distribution

When exploring the particle internal structure, scientists often want to exam a chosen subset in detail while being able to see the entire structure of the particle dataset. One way is to apply different color maps to the subset and the rest of the particles. This, however, can't provide a clear view of the subset particles if there are many clusters of particles around the subset. Another approach is to use OpenGL's blending operation. Particles inside the subset are rendered as opaque points while particles outside the subset as transparent points. In this case, particles need to be redistributed among the render nodes in order to apply alpha blending operation along the rendering pipeline. Subset particles need to be rendered before the rest of the particles. This can be done by designating a few render nodes for processing subset particles and remaining render nodes for the rest of the particles. The number of render nodes designated for subset particles can be determined based on the number of particles within the subset. The basic strategy is to maintain the particle load balance among all render nodes. This is possible because the last render node for the subset can have particles outside the subset. One way to do this is to choose the first few render nodes in the initial rendering pipeline as the designated subset render nodes. All particles within the subset at each render node are moved to the designated subset render nodes. Others are moved to the remaining render nodes. This is obviously not an efficient method because it involves massive particle exchange among render nodes.

A better way is to choose render nodes with higher number of subset particles for subset render nodes and to change the rendering pipeline according to the number of subset particles contained in each render node. This will minimize the data exchange among all render nodes as only a small portion of the particles at most nodes needs to be exchanged. In the new pipeline, the subset render nodes will be moved to the front of other render nodes. This is done using Sepia's dynamic routing functions. For each node, a new upstream node and a downstream node are set based on the number of subset particles at each node.

The redistribution process uses a similar strategy to that of the initial distribution in which each node gets approximately equal share of particles. The particle redistribution process is detailed in Figure 2.

Figure 3 shows an example of dynamic routing on a 6 render node case. At first, the system creates a rendering pipeline according to the number of render nodes allocated to the current running session. This initial pipeline is usually based on the physical node configuration. Each render node, except the first and the last, is directly connected to its upstream render node and its downstream render node. The first render node has no upstream render node. The last render node has no downstream render node and is connected to the display node. The top part of Figure 3 shows an initial rendering pipeline when using 6 render nodes. Each render node renders its share of particles and combines its rendering result with the result from the previous node in the pipeline (details are described in Section 2.4). The composited result is then transfered to the next render node in the pipeline.

- choose a subset.
- find the bounding box of the subset.
- calculate the number of particles within the subset at each node.
- calculate the total number of subset particles.
- determine the number of render nodes required for the subset, i.e.,

$$n_{subset} = n_{total} \times \frac{\textit{number of subset particles}}{\textit{total number of particles}}$$

- select the top n_{subset} nodes that have more particles in the subset.
- exchange particles between nodes so that the first $n_{subset} - 1$ nodes contains only subset particles, the n_{subset}'th node contains subset particles and possibly particles outside the subset, and other nodes contains only particles outside the subset.
- change the rendering chain so that the n_{subset} nodes are in front of other nodes in the rendering pipeline.
- reset the upstream and downstream nodes for each node.

Figure 2: *The dynamic particle distribution algorithm.*

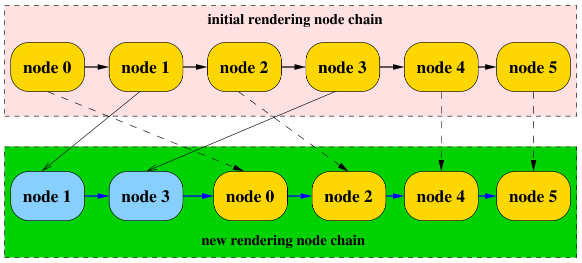

Figure 3: *Dynamic routing rendering pipeline.*

During dynamic data redistribution, the system first determines that two nodes are required for rendering subset particles. It then chooses those two nodes with the most subset particles, i.e. nodes 1 and 3 in this case. All particles outside the subset on node 1 will be redistributed to node 0, 2, 4, or 5. Depending on the total number of particles in the subset, node 3 may only need to redistribute part of its particles outside the subset to node 0, 2, 4, or 5. Particles inside the subset on nodes 0, 2, 4, and 5 are transfered to either node 1 or node 3 in exchange of particles outside the subset. The redistribution algorithm maintains the particle load balance among all render nodes. In general, the number of particles on each render node remains the same as before the redistribution. The 6 node rendering pipeline is finally changed so that the two render nodes containing subset particles are set in front of all other render nodes, as illustrated in Figure 3.

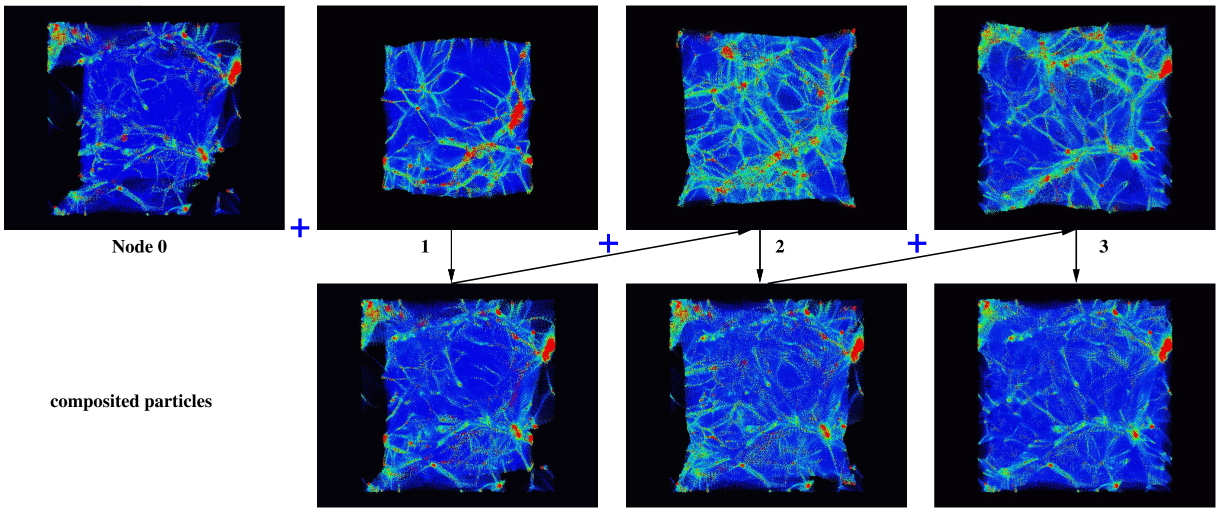

Figure 4: *Particle image compositing on four render nodes.*

2.3. Rendering Particle Primitives

In direct rendering each particle is represented using a single point primitive. All point primitives use the same radius, which is 1 in our examples. Using a uniform point representation avoids the OpenGL rendering overhead that may be incurred by changing the point size for each particle and performing complex transformations required to position different primitives. In this case, particles at each node can be rendered to a display list in a single loop or using a single OpenGL array rendering call.

Particle densities are usually used to reveal the structure of the particle volume. To visualize the particles, particle densities are transformed to RGB or RGBA colors based on a given color map. This is usually done using a logarithm mapping which maps the density of a particle to a specific color in the color map. Linear mapping and other non-linear mappings can be applied as well, depending on the application's characteristics. Non-linear color mapping method can be conveniently used to highlight particles in a particular range of densities. Images in this work are produced using a variety of color maps.

2.4. Image Composition Scheme

The Sepia cluster offers a variety of compositing operations, such as Z-comparison using OpenGL comparison operators, alpha blending, full scene anti-aliasing, and user downloadable re-programmable computational logic. For direct particle visualization we use Z-comparison to composite the rendered image at each node with the image from its upstream node in the pipeline. Figure 4 illustrates rendered particles at each node and composited particles after each node in a case of 4 render nodes and a display node.

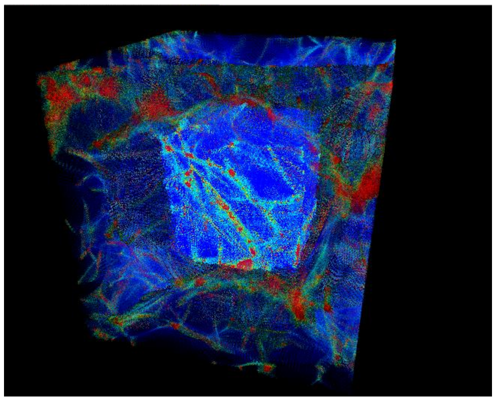

Figure 5: *Subset particles rendered using alpha blending.*

Using alpha blending, a chosen subset of the particle dataset can be highlighted using the algorithm in Figure 2. As shown in Figure 5, particles outside the selected subset are rendered as transparent particles while particles within the subset are rendered as opaque particles. This allows for detailed study of particular regions of particles while users are exploring the whole particle dataset.

2.5. Rendering Synchronization

Users interact with the master using tools in a graphical user interface. For each operation, the master distributes a command to all workers or render nodes using a TCP/IP based protocol. The workers then interpret the received command and perform the corresponding operation. In a parallel rendering system, operations on different nodes need to be synchronized so that particles are rendered at the same time on

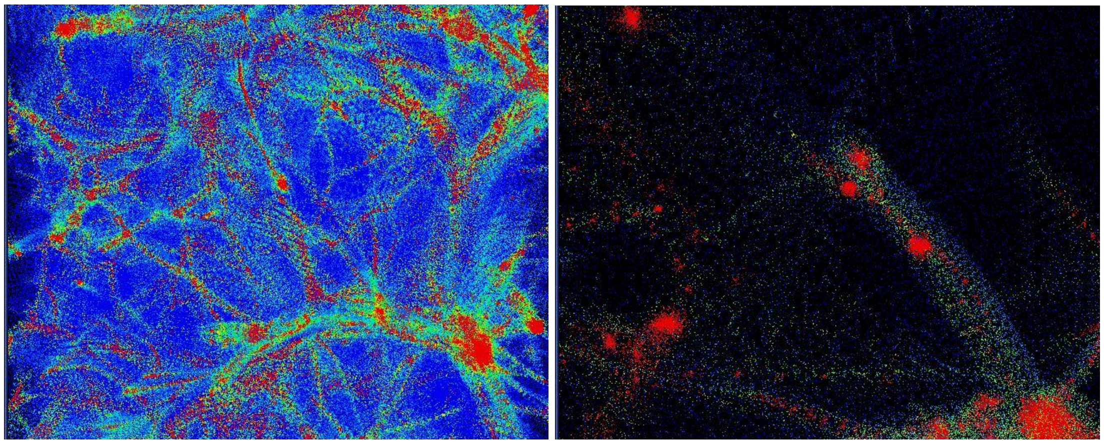

Figure 6: *Particles represented using inverse rainbow color mapping: a) front view; b) a close look.*

all render nodes in response to the same command. Otherwise it may be possible that some render nodes have different view points from others. This obviously will result in a compositing image that is not a correct representation of the given particle dataset.

The system uses a reactor communication pattern for dispatching handles for synchronous events. The master listens to and accepts user requests and concurrently dispatches the requests to all render nodes. A frame controller is used at both master node and render nodes. The master first sends a command to all render nodes to initialize the controller. It then sends each command to the render nodes, along with the frame controller. At each step, the render nodes respond to the master's commands and run only commands associated with the same frame controller. This ensures that all render nodes are working on the same frame at the same time.

3. Results and Discussion

3.1. Implementation

At each render node, a set of OpenGL display lists are used for fast rendering uniform particle point primitives. Particles at each render node are built into the display lists when particle properties are changed. Using OpenGL display lists greatly enhances the system performance when users navigate the particle dataset. Another method for fast rendering particles as uniform point primitives is to use OpenGL vertex arrays, but with OpenGL display lists, little extra performance improvement is obtained from using vertex arrays when rendering 256^3 particle datasets in our current system. As OpenGL display lists have to be stored in the graphics memory, this may become the constraints of the system performance for increasing number of particles. In this case, vertex arrays may be a better choice than display lists. We are still investigating this on the 512^3 particle datasets.

3.2. Particles Visual Representations

Figures 6 and 7 show different visual representations of the 256^3 particle cube. In Figure 6, a front view of the directly rendered particles is modeled using an inverse rainbow color map. The right image shows a close look at a specific part of the particle cube. Clusters of red particles represent high density areas. Figure 7(left) shows the same 256^3 particles cube using another color map, in which high density particles are mapped to brown while low density particles are mapped to light black. A linear color mapping from black to yellow is applied to the same particle cube to create a different particle visual representation in Figure 7 (right).

3.3. Results on 256^3 Particle Dataset

A comprehensive test has been done for the 256^3 particle dataset using 1 to 7 render node(s) and a display node. In general, the performance is constrained by particle rendering, data acquisition, and image composition. Particle rendering depends on the number of particles at the render nodes. Data acquisition and image composition are dependent on the Sepia hardware and its associated visualization toolkits. Table 1 shows time used for individual operations on 4, 5, 6, and 7 render nodes. The time shown in the table is taken when running a series of *navigation* operations. Therefore, it includes only one data acquisition and image compositing time that is independent on the number of render nodes.

There are two types of operations, *navigation* and *particle update*.

- *Navigation* operations include zooming, rotation, and translation. These operations allow users to look around and to go inside the particle volume. Because the particles visual representations are not changed, the system simply calls the OpenGL display lists created while doing *parti-*

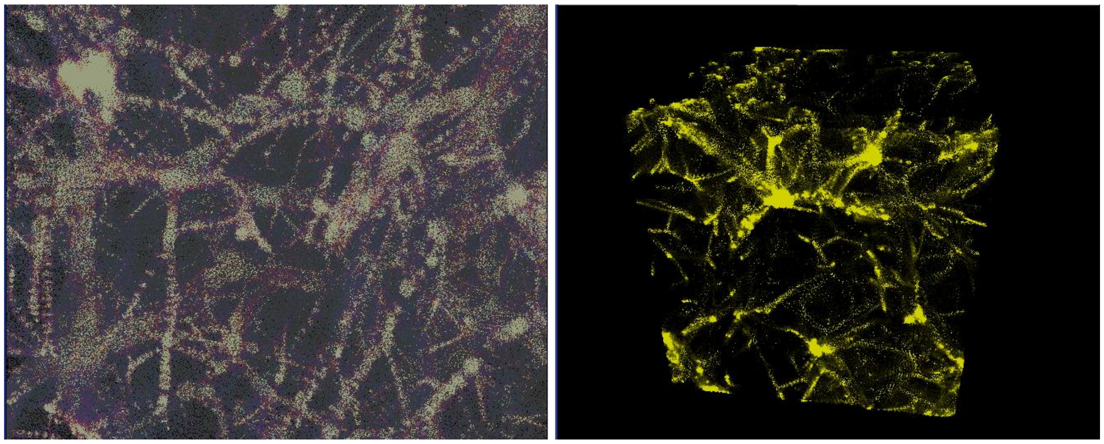

Figure 7: *Particles represented using light hue (left) color mapping and from black to yellow linear color mapping (right).*

Operations/Nodes	4	5	6	7
navigation	0.112	0.112	0.112	0.112
particle update	3.05	2.513	2.06	1.56
change color map	3.06	2.46	2.15	1.60
sort by color	3.21	2.47	2.15	1.58
change color mapping	3.20	2.50	2.06	1.60

Table 1: *Operation time (seconds) on* 256^3 *particle dataset.*

cle update operations, providing a high interactive frame rate.

- *Particle update* operations change the particles attributes and build the OpenGL display lists. The particle attributes can be changed by modifying particles' positions, colors and alpha channels using different color maps and/or different color mapping methods. *Particle update* operations allow users to explore the overall structure and fine features of the particle volume, and to study a specific region of interest in the particle volume.

As shown in Table 1, the time used for *navigation* operations remains the same when using more than 4 render nodes. This is because each render node takes very little time to update a frame by calling the already built OpenGL display lists. For 256^3 dataset, each render node processes about 4 million particles. Our experience showed that the number of display lists is crucial to obtain this performance on a given graphics card. The performance would drop dramatically if we use less display lists or the number of particles on each node is over a certain limit. This can be seen from Figure 8 as the *navigation* operation time drops rapidly from about 0.78 seconds on 1 render node to about 0.112 seconds on 3 render nodes.

On the other hand, *particle update* operations take much longer time than *navigation* operations because they need to build the display lists for rendering all the particles. Their performance is totally dependent on the rendering capabilities of each render node. The less the number of particles at each render node, the faster the operation.

In practice scientists are more interested in *navigation* operations because these operations provide sufficient power for them to view the particle dataset from different view points, to fly through the particle cube, and to dive into specific region for detailed study. Based on the data in Table 1, a frame rate of $1/0.112 = 8.928$ *fps* is obtained for *navigation* operations on the 256^3 particle dataset using 4 or more render nodes. This result is very promising and satisfactory for interactive visualization of the 256^3 particle dataset. Operations such as rotation and zooming in and out run very smoothly.

Figure 8 illustrates *navigation* operation time against the number of nodes. As seen in the figure, the rendering time drops as the number of nodes increases from 1 to 3 nodes. From 3 to 7 nodes, the rendering time is close to 0 and the overall *navigation* operation time is dominated by the acquisition and compositing time. For *particle update* operations, as shown in Figure 9, the rendering time drops as the number of render nodes is increased. This justifies the scalability of the system. The *particle update* operation time is dependent on the graphics processing capability at each render node. The performance gain is mainly from the speedup of individual nodes. For instance, for system running on 4 and 6 nodes, each node takes $(64-42)*256^2$ less particles in 6 nodes rendering than in 4 nodes rendering, and, therefore, the overall performance is improved by about 30%.

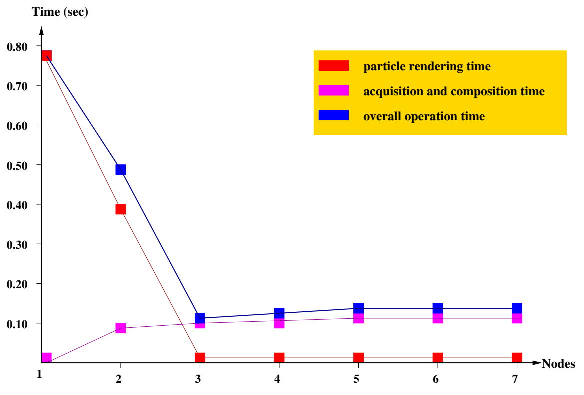

Figure 8: *Navigation operation time (seconds) vs. the number of render nodes on* 256^3 *particles.*

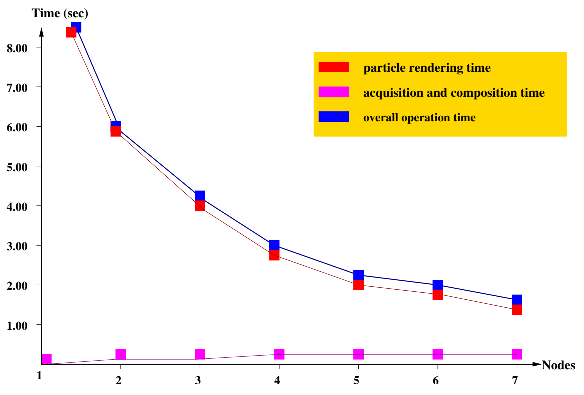

Figure 9: *Particle update operation time (seconds) vs. the number of render nodes on* 256^3 *particles.*

In Table 2, the *navigation* operation time is split into particle rendering time and image data acquisition and composition time for system running on 4, 5, 6, and 7 render nodes. As mentioned earlier, the rendering time is the time used for calling a set of display lists built in a *particle update* operation. The data show that the *navigation* operation time is completely dependent on the acquisition and compositing time, which is constrained by the Sepia hardware.

Operation/Node	4	5	6	7
rendering particles	0.0007	0.0006	0.0006	0.0006
acquisition & composition	0.1082	0.1121	0.1118	0.1118

Table 2: *Time split for navigation operations.*

In conclusion, the performance of the current system is basically determined by the number of render nodes, especially for *particle update* operations. The image data acquisition and composition time at each node is dependent on the Sepia data transfer network and data compositing engine. When using more than 3 render nodes, the Sepia data acquisition and compositing engine has been identified as the bottleneck for *navigation* operations of the system.

3.4. Results on 512^3 Particle Dataset

A preliminary test on 512^3 particle dataset has been done using 6 and 7 render nodes and a display node. As shown in Table 3, the system takes much longer time to perform the *particle update* operations than on 256^3 particles. For a 512^3 particles dataset, each render node processes 8 times more particles. The large number of particles pushes the system to the limit of both memory and graphics processing capabilities.

Similarly, time used for *navigation* operations is greatly increased as well. The number of particles on each render node is much more over the limit at which the performance of calling OpenGL display lists drops dramatically. This is the same as in the 256^3 case when running on 1 or 2 render nodes.

Operation/Nodes	6	7
navigation	1.157	1.007
particle update	189.5	154.58

Table 3: *Operation time (seconds) on* 512^3 *particle datasets.*

The frame rate is about 1 *fps* for *navigation* operations when the system is running on 7 render nodes. Although much more time required for both types of operations, the result shows that the system is well scalable. The more render nodes, the faster the rendering operations.

To be able to navigate the particle volume smoothly, a much higher frame rate for 512^3 particle datasets is expected. Several techniques are currently under investigation. First, current system uses a serial mode for data acquisition and image composition, i.e. image composition process starts only after the data acquisition process has finished. Using a parallel mode will reduce about 50% combined time for data acquisition and image composition. Second, a vertex program is to be applied to render particle points directly using the graphics processor. This may considerably increase the rendering speed and relieve the graphics memory stress. Third, advanced data structure, such as the hierarchical data structure in [HE03], for representing the particle volume is being investigated. In this case, software based rendering pre-process can be done to reduce the workload on the graphics processor.

3.5. Limitations

The method presented in this paper renders all particles using uniform rendering primitives. Either all particles are rendered as opaque points or some of them are rendered as transparent points. This may make it difficult to identify exact particle structures when particles in a given dataset are highly clustered. Although different color mapping methods and subset visualization technique help to highlight particular part of the particles, the overall structure of the particle dataset, especially internal structure, is hard to be recognized. In this case, we may want to render all particles as transparent points. This will allow us to visualize the overall particles structure and internal features in the same way as in texture mapping based volume rendering method.

4. Conclusions and Future Work

We have described an interactive parallel visualization system for viewing large particle datasets using the Sepia rendering cluster. The test on 256^3 particle dataset shows that the system provides an interactive rate at about 9 *fps* for common visualization tasks, such as zooming, rotation, and translation. The system allows for interactively changing particles' visual properties and highlighting particular regions of interests for detailed analysis. An 1 *fps* is obtained at an initial test on 512^3 particle dataset using 6 or 7 render nodes and a display node.

The most challenging task is to improve the frame rate for the 512^3 particle datasets. We are currently investigating a number of techniques. These include parallelization of the data acquisition and image compositing, integrating better data structures for representing particle volumes, and applying other rendering optimization techniques.

Acknowledgment

The authors would like to thank Hewlett Packard for providing the Sepia system and technical support to the system. We would also like to thank the department of Research and High Performance Computing Support and SharcNet for the use of the Sepia cluster. We are very grateful for James Wadsley for his particle dataset and helpful discussions on many practical particle visualization issues from the point of view of physics scientists. We appreciate very much to the anonymous reviewers for their constructive suggestions.

References

[HE03] HOPF M., ERTL T.: Hierarchical splatting of scattered data. In *Visualization '03* (2003), IEEE, pp. 433 – 440. 2, 7

[HM99] HEIRICH A., MOLL L.: Scalable distributed visualization using off-the-shelf components. In *1999 IEEE symposium on parallel and large-data visualization and graphics* (1999), ACM Press, New York, NY, USA, pp. 55–59. San Francisco, California, United States. 2

[JFP*98] JENKINS A., FRENK C., PEARCE F., P.A.THOMAS, COLBERG J., WHITE S., COUCHMAN H., PEACOCK J., EFSTATHIOU G., NELSON A.: Evolution of structure in cold dark matter universes. *Astrophysical Journal 499* (1998), 20–40. The Virgo Consortium. 2

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37. 2

[LMS*01] LOMBEYDA S., MOLL L., SHAND M., BREEN D., HEIRICH A.: Scalable interactive volume rendering using off-the-shelf components. In *IEEE 2001 symposium on parallel and large-data visualization and graphics* (2001), IEEE Press, Piscataway, NJ, USA, pp. 115–121. San Diego, California, United States. 2

[MHS99a] MEISSNER M., HOFFMANN U., STRAβER W.: Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *IEEE Visualization '99* (1999), pp. 207–214. 1, 2

[MHS99b] MOLL L., HEIRICH A., SHAND M.: Sepia: scalable 3D compositing using PCI Pamette. In *IEEE Symposium on FPGAs for Custom Computing Machines* (Los Alamitos, CA, 1999), Pocek K. L., Arnold J., (Eds.), IEEE Computer Society Press, pp. 146–155. 1, 2

[RL00] RUSINKIEWWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00* (2000), ACM SIGGRAPH, pp. 343 – 352. 2

[WMQR02] WILSON B., MA K.-L., QIANG J., RYNE R.: Interactive visualization of particle beams for accelerator design. In *ICCS 2002 Workshop on High Performance Computing in Particle Accelerator Science and Technology* (April 21-24 2002), 2002 International Conference on Computational Science, pp. 352–361. Amsterdam, The Netherlands. 1, 2

[WSQ04] WADSLEY J. W., STADEL J., QUINN T.: Gasoline: a flexible, parallel implementation of TreeSPH. *New Astronomy 9* (Feb. 2004), 137–158. 2

[ZPvBG01] ZWICKER M., PHISTER H., VAN BAAR J., GROSS M.: EWA volume splatting. In *Visualization '01* (2001), IEEE, pp. 29 – 36. 2