A PC Cluster System for Simultaneous Interactive Volumetric Modeling and Visualization

Shigeru Muraki* Eric. B. Lum† Kwan-Liu Ma† Masato Ogata‡ Xuezhen Liu‡

*AIST, Japan †University of California, Davis, U. S. A. ‡Mitsubishi Precision, Co., Ltd., Japan

Abstract

A number of problems are well suited for volumetric representation for both simulation and storage, however, the large amount of data that needs to be processed and rendered with these volumes makes interactive manipulation extremely challenging.

In this paper, we present a scalable PC cluster system (VG cluster) designed specifically to enable simultaneous volumetric computation and visualization, using compositing hardware devices and the latest PC graphics accelerators. We demonstrate the flexibility and performance of this system with several different applications that include reaction-diffusion simulation, volumetric image processing, and vector field visualization. We also discuss how to improve the visual computing performance of this system with some load balancing techniques.

CR Categories: I.3.2 [COMPUTER GRAPHICS] Graphics Systems - Distributed/network graphics; I.3.6 [COMPUTER GRAPHICS] Methodology and Techniques - Interaction techniques; I.6.8 [SIMULATION AND MODELING] Types of Simulation - Visual;

Keywords: Graphics hardware, PC clusters, parallel volume rendering, reaction-diffusion patterns, visual computing, volume graphics, volume modeling

1. Introduction

A wide variety of phenomena lend themselves naturally to volumetric representations such as tomographic imaging or the results of the simulation of fluids and gasses. These volumes are typically stored as a regular 3D grid of data that can contain hundreds of millions of voxels making the interactive processing and rendering of these volumes a tremendous challenge. Interactivity, however, is often required when dealing with such problems, where it is essential to give the user the ability to run a simulation and receive immediate visual feedback for the tuning of simulation parameters.

e-mail: s-muraki@aist.go.jp
e-mail: {ma, lume}@cs.ucdavis.edu
e-mail: {ogata, liu}@mpcnet.co.jp

In this paper, we present a PC cluster system (*VG cluster*. Fig. 1) that is capable of performing interactive, simultaneous modeling and visualization of large-scale volumetric problems. All calculations and communication for volume rendering are mapped to hardware subsystems so that the main CPU and network resources can be devoted exclusively to volumetric processing or simulation. We will explain how this system efficiently solves large-scale problems, and will also discuss what new problems particular to this system occur and how they can be avoided.

2. Previous Work

Parallel supercomputers have been widely used for large-scale simulations, while high-end graphics workstations are well suited for small-scale visualization [1]. In recent years there has also been increasing use of PC clusters for graphics and visualization calculations, where each node is equipped with graphics hardware [2]. Most of these systems can be classified as having either image-space (sort-first) or object-space (sort-last) parallelism [3].

Sort-last volume rendering is suitable in regular 3D grid applications. For volume rendering, there is a commercially available accelerator for PCs, i.e. Volume Pro 1000 [4]. More recently there has been a trend toward the use of the less expensive texture hardware found in commodity PC graphics cards for volume rendering [5,6,7]. Sort-last volume rendering requires image compositing, which blends the output image from each PC in depth order using opacity values. Although a few efficient image compositing algorithms have been proposed [8,9], these techniques are not suited for simultaneous simulation and visualization of volume data since network and/or CPU resources are used for image compositing.

Because of the remarkable advances in PC graphics hardware, image compositing is increasingly becoming the bottleneck of parallel renderings, and there have been efforts in building hardware image compositing devices [10,11,12,13,14]. The systems of [10,11] send image data from the DVI digital output of a graphics accelerator and do not require either frame buffer read backs or specialized PCI interface cards. However, the compositing order of [10,11] is fixed for *Z* value comparison of polygons, and is not suitable for volume rendering in which the compositing order changes according to the view angle. In addition, DVI output does not include depth and alpha values, thus requiring auxiliary RGB images to encode this information. The systems of [12,13,14] are for volume rendering and have functionality to handle compositing order change, though they need frame buffer read backs and custom PCI interface cards. The system of [13], which is the successor of [12], combined VIA-based network interfaces and a single-stage, eight-port, crossbar switch (*ServerNet-2*) to change the compositing order at will. The latency, however, increases linearly with the number of

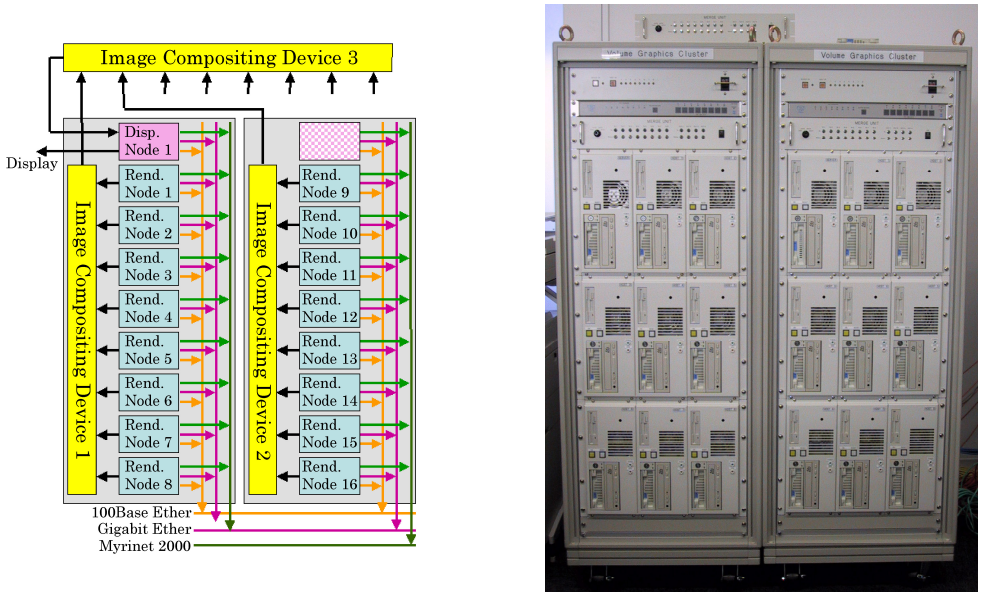


Figure 1 VG cluster system (17 nodes)

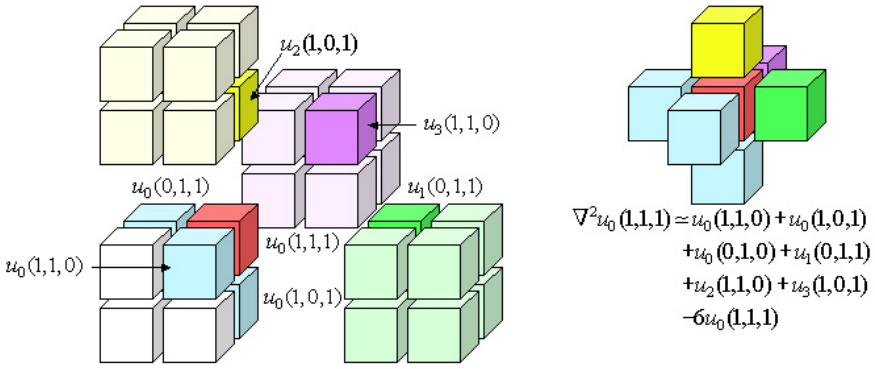


Figure 2 Boundary-data exchange between four 2^3 sub-volumes (u_0, u_1, u_2, u_3) for Laplacian approximation.

PCs (n), as well as requiring more complex switching as the size of a cluster increases. The device of [14] employed a binary compositing tree to simplify the compositing order change and reduced the latency to the order of log (n).

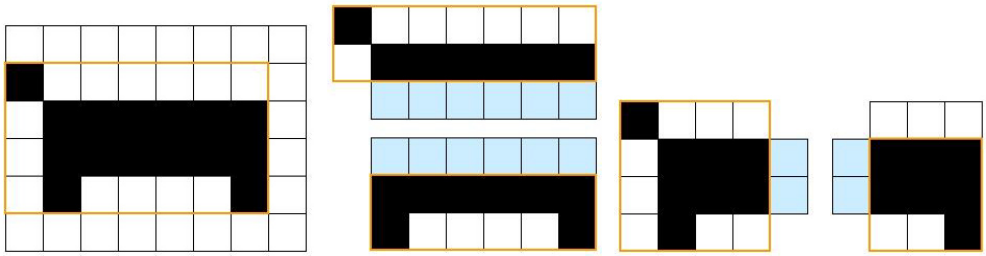
3. System Design

In our system, all rendering and compositing operations have been mapped to hardware subsystems so that traditional CPU and network resources can be devoted exclusively to volumetric processing or simulation. Fig. 1 illustrates our VG cluster system, using PCs for volume processing (Render-node) and a single PC for the display (Display-node). This system consists of two 9-PC clusters using image-compositing hardware manufactured by Mitsubishi Precision, Co. Ltd. [14], and another image-compositing device connects them together. Each PC node is a dual processor Intel Xeon 1.7 GHz system with one gigabyte of memory and has a graphics accelerator supporting hardware 3D texture mapping (nVIDIA GeForce 4 Ti 4600). Three different network systems (100Base-TX, Gigabit Ethernet and Myrinet 2000) are employed for the inter PC communications, with RedHat Linux 7.3 based SCore 5.2 [15] as the operating system. With this configuration, we can evaluate the performance of the system by changing the number of Render-nodes from 1 to 16.

We implemented a volume renderer for each Render-node by using the multi-texture functionality of the GeForce 4 Ti4600. Two 3D textures, a single 32-bit texture for three normal vector components and an attribute value and a single 8-bit texture for indexed RGBA are employed [20]. Since the memory capacity of the graphics card is 128 megabytes and our volume renderer uses 5 bytes for each voxel, the maximum subvolume size for each PC is 256^3 voxels.

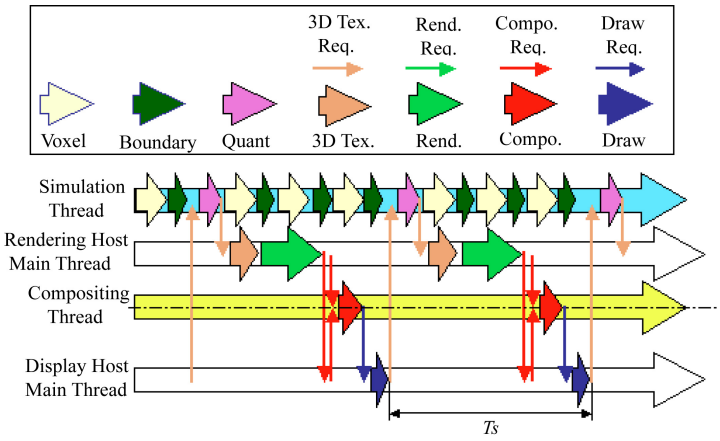
4. Load Balancing

Our system was designed to meet the challenge of interactive volume computation and visualization. For simplicity, we deal with only structured grid data in this paper. The types of volume computation we deal with in this paper are local operation, e.g. differential operations like the *Laplacian* or mathematical

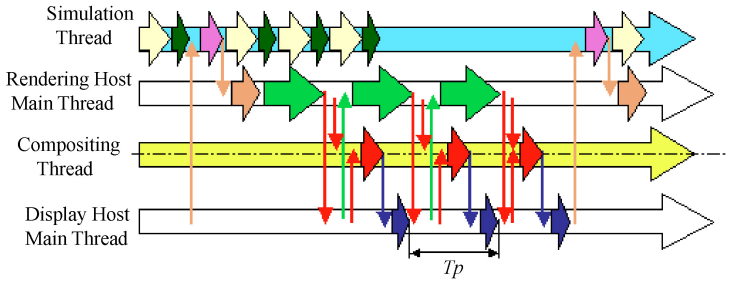


(a) Original volume (b) Vertical division (c) Horisontal division

Figure 3 Adaptive subdivision method concerning on the number of foreground (black) voxels.



(a) Monitored simulation mode



(b) Pipelined visualization mode

Figure 4 Interactive visual simulation modes

morphology operations, which are performed over a finite 3D extent around each voxel.

For these operations we use the divide and conquer strategy. A regular grid computation space is divided into subvolumes and dispatched to the nodes of a PC cluster for both computation and sort-last volume rendering. Since the volume computations have a finite 3D extent, this strategy causes several problems. For example, in computational fluid dynamics (CFD), the Laplacian is often approximated by the six neighboring central differences, as depicted in Fig. 2. In this case, the voxels along the sub-volume boundaries need to obtain the necessary voxel values by communicating with neighboring PCs. This procedure is referred to as *boundary-data exchange* (or *ghost-point exchange*) and can consume a considerable amount of time depending on the performance of the available network.

There are several space subdivision methods for balancing computational loads, but many are not suitable for visualization. For simplicity we demonstrate these methods using the 2D diagrams shown in Fig 3. Fig. 3(a) represents an object in 8×6 volume with foreground (black) voxels. Suppose that the volume computations are performed only on the foreground voxels, we can then fit the bounding box (orange square) by removing unnecessary background (white) voxels. To perform the volume computations with two CPUs, the subdivision that equalizes the numbers of foreground voxels efficiently balances the computational loads. If we allow only subdivisions along the coordinate axes, there are two possible methods as shown in Figs. 3(b) and 3(c). If we use texture-base volume rendering method, rendering performance is often fill-rate limited leading to a strong correlation between the projected size of a volume and frame rate. Therefore, the method shown in Fig. 3(b) yields better load balancing for visualization applications, since the left subvolume

of Fig. 3(c) is larger than the others. However, because of the costs of exchanging boundary data points (blue voxels), the method shown in Fig. 3(c) gives better overall superior.

As observed in these examples, load balancing is problematic when the volume computation and visualization are performed simultaneously. Fig. 4(a) presents a flow diagram for simultaneous processing and rendering, with the simulation results shown every three iterations. The procedure is divided into seven subprocesses: i) simulation computation at each voxel (*Voxel*), ii) boundary-data exchange (*Boundary*), iii) quantization of voxel values for the graphics accelerator (*Quant.*), iv) 3D texture generation (*3D Tex.*), v) rendering (*Rend.*), vi) image compositing (*Compo.*), and vii) drawing (*Draw*). Since independent threads are used for simulation, rendering, and image compositing, the visualization proceeds without stopping the simulation process. Our system utilizes image compositing devices, to improve the rendering performance by overlapping image compositing and rendering processes as shown in Fig. 4(b). This pipelined visualization mode reduces the display interval T_p to less than T_s as shown Fig. 4(a). This type of pipelining is not possible with compositing methods that make use of the graphics accelerator [7].

5. Applications

A number of applications containing large amount of data that require simultaneous computation and visualization were mapped to our system in order to evaluate its effectiveness.

5.1. Reaction-Diffusion simulation

Fig. 5 illustrates two 3D *Turing patterns* [16] generated on the body surface of a dog model based on *reaction-diffusion* equations:

$$\begin{cases} \partial u/\partial t = \nabla^2 u + u^2 v + a - u, \\ \partial v/\partial t = d\nabla^2 v + b - u^2 v, \end{cases} \quad (1)$$

The reaction-diffusion equations can generate a variety of patterns with slight parameter changes and have been used by computer graphics researchers as a 2D texture generation technique [17]. However, when this technique is extended to 3D, the amount of computation increases significantly.The dog model is represented as a binary volume of resolution 512×348×143, and is generated as the body surface voxels of some thickness based on Euclidian distance transformation. Solving Equation (1) on the body surface, whose thickness is five voxels (1,105,026 voxels), generates the patterns of Fig. 5. To solve Equation (1), we compute the right hand side of the equation for every body surface voxel and iteratively update *u,*v as

$$\begin{cases} u_{T+1} = u_T + \left(\partial u/\partial t\right)_T \Delta T, \\ v_{T+1} = v_T + \left(\partial v/\partial t\right)_T \Delta T, \end{cases} \quad (2)$$

with an appropriate ΔT until their convergence. As we mentioned before, the system needs to perform the boundary-data exchanges for each time-step because of the Laplacian in Eqs. (1). The patterns of Figs. 5(a) and 5(b) appear after 10,000 and 150,000 iterations respectively at ΔT=0.005.

5.2. Volumetric Image Processing

The effective visualization of the 3D data produced by tomographic imaging systems often requires post-processing to remove noise and enhance features of interest. Since the type of post-processing that must be performed is entirely dependent on the nature of data being viewed and the visualization goals of the user, interactivity in this type of processing is essential, since it allows the tuning of this processing to properly bring out the features of interest to that user.

We therefore mapped a suite of image-processing operations to our system in a tool that allows for their interactive application. The user can apply different operations to a volume and immediately see how it affects the resulting visualization.

Fig. 6 illustrates an example of the result of recursively applying grayscale morphology for the "CT-Head" data set (256×256×113) to extract brain area. Using 16 Render-nodes, this process was completely interactive. Since a 3×3×3 kernel is used for this operation, the same boundary-data exchanges as in the previous example were required after each operation. Filtering using a kernel larger than 3×3×3 would require considerably more time for boundary-data exchange.

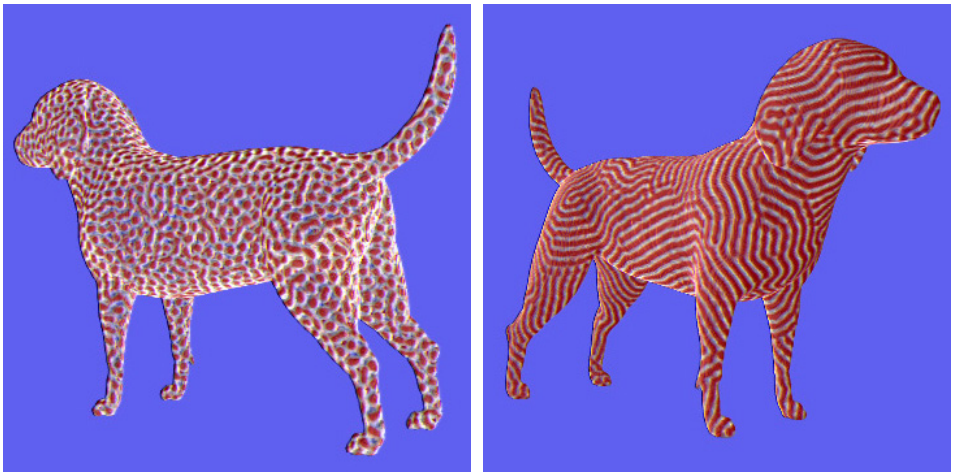
5.3. Vector Field Visualization

Vector field visualization is a particularly challenging problem since the volumes dealt with are not only large, but it is necessary to illustrate fine changes in vector direction across these voxels. Line integral convolution (LIC) [18] is a common technique used in vector-field visualization and consists of blurring a noise function along a vector field to illustrate direction. By applying phase-shifted filter kernels, we can generate time-varying 3D LIC volumes of 3D vector field data as an animation. Although the 3D LIC computation itself is not fully interactive, once a time series of 3D LIC volumes is generated, our system can visualize the animated volumes very efficiently.

Since our system uses object space subdivision, each graphics accelerator needs to store a small subset of this 4D data. Fig 7(a) shows a single time step of a "4D LIC" volume (120×120×120 for 16 different time steps). In this case it is possible to store all time steps of the 120×120×120 animated LIC volume in the aggregated texture memory of the system for the display of interactive animations that allow for changes in viewpoint and transfer function.

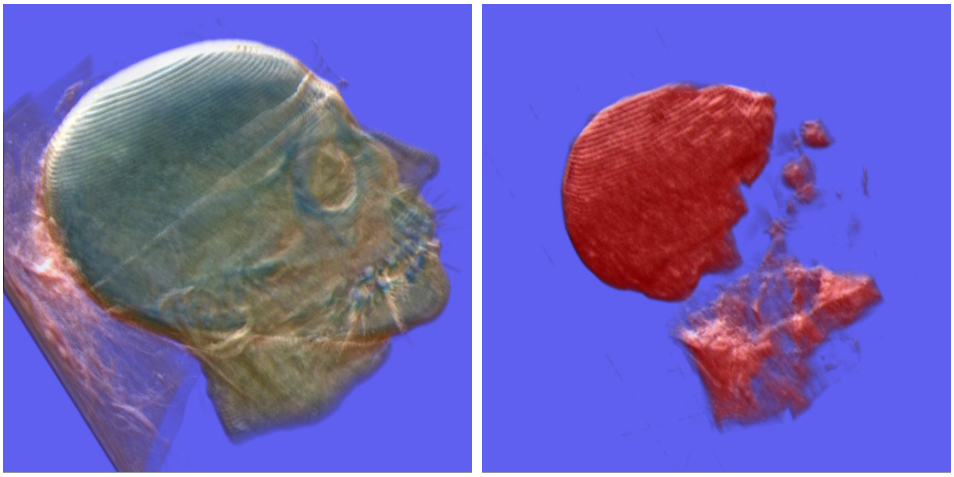
6. Performance Analysis

To evaluate our volume graphics PC cluster system, four different volume data sets were used ("Dog," "CT-Head," "4D-LIC," and the Visible Human Male (VHM) dataset (Fig. 7(b)). Frame rates for rendering these volumes to various sized windows with differing number of rendering nodes is shown in Fig. 8. Because of the limited amount of graphics memory capacity, we only evaluated the performances of "Dog" and "VHM" using 8 and 16 nodes. We used the standard *Cartesian* volume subdivision method because of its simplicity. Changing the number of Render-nodes from 2 to 16 linearly increased frame-rates, however, all data set exhibited a sudden frame-rate change when the number of nodes was small. This shows that the volume rendering performance of the graphics accelerator severely decreases when the size of the subvolume is large. Our divide and conquer approach avoids this problem and efficiently brings out the maximum performances of latest graphics accelerators. By increasing the number of Render-nodes, our system can achieved over 45 frames per second for a 512×512 image size by hiding the compositing time behind the volume rendering time as shown in Fig. 8. Although this performance is more than three times faster than a software implementation[7] of the *binary-swap compositing* [8] using Myrinet, Fig. 8 also reveals that our compositing hardware does not always take full advantage of the full performance of the graphics accelerators. Notice that the frame rates of "4D LIC" with more than four nodes are almost identical



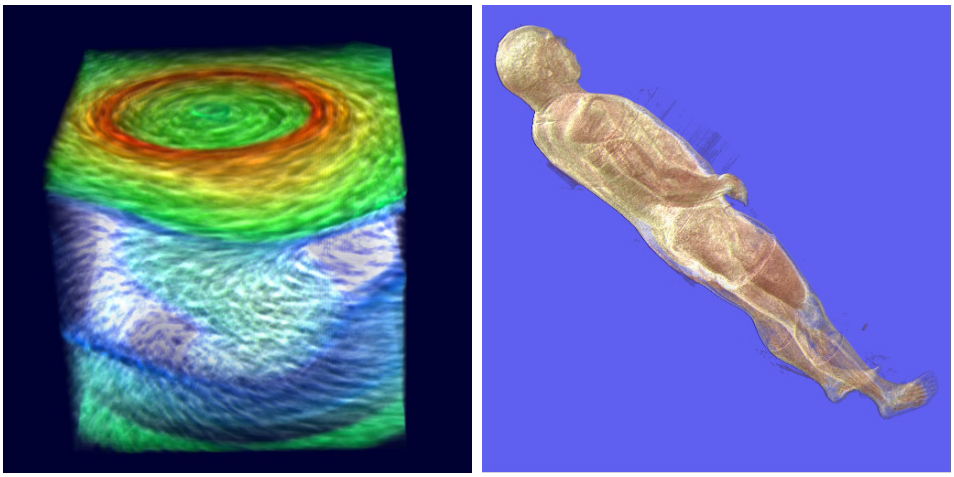
(a) a=0.1, b=1.1, d=20 (b) a=0.1, b=1.45, d=20

Figure 5 Examples of the 3D Reaction-Diffusion simulation.



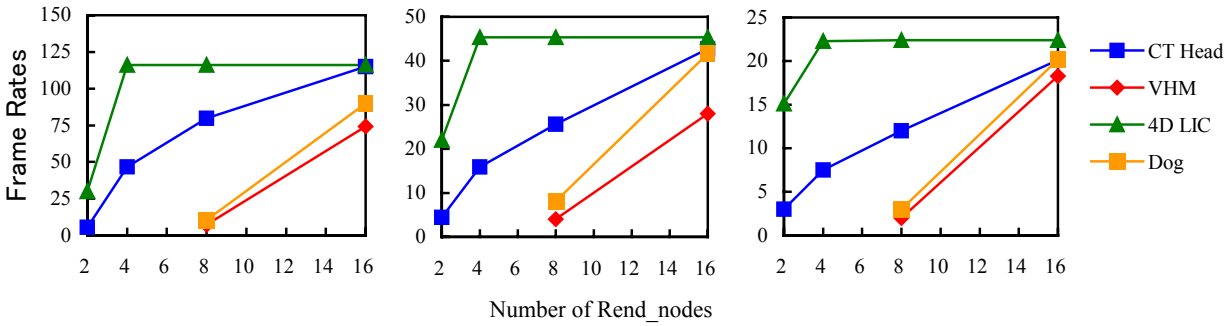
(a) Before filtering (b) After filtering

Figure 6 Volumetric image processing reveals the brain region



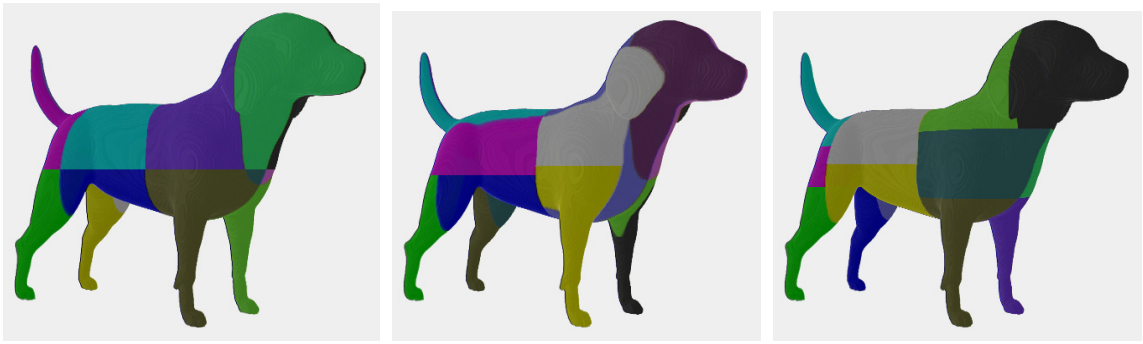
(a) 4D_ LIC (120×120×120×16) (b) VHM (430×240×939)

Figure 7 Test Volume Data



(a) 256×256 (b) 512×512 (c) 768×768

Figure 8 Frame rates (Hz) at pipelined visualization mode (Cartesian subdivision)



(a) Cartesian (b) Adaptive subdivision 1 (c) Adaptive subdivision 2

Figure 9 Space subdivision methods

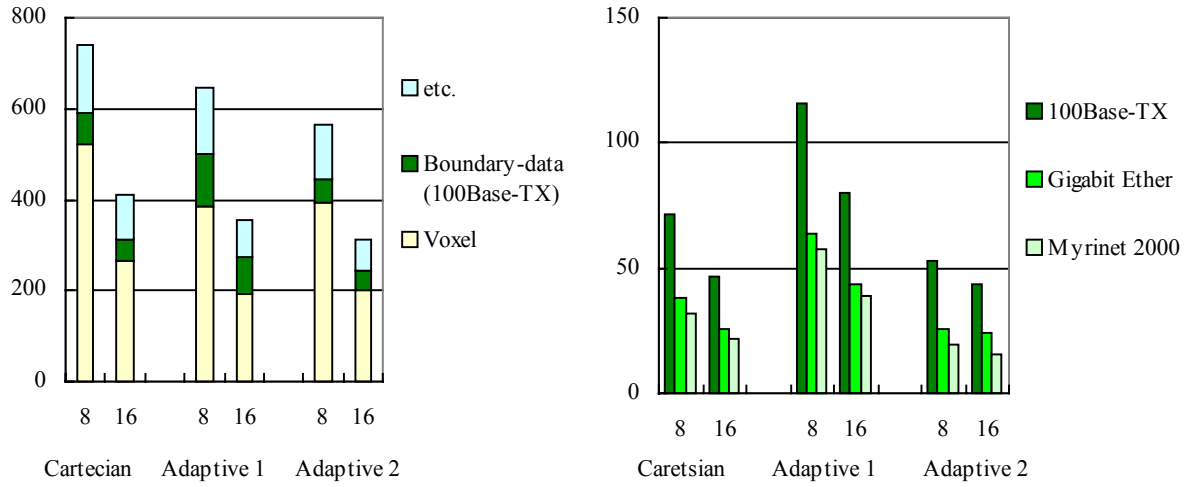
because the small volume size at each time step makes rendering time extremely short. There are two reasons for this loss of performance. One is the amount of time required to read each rendered image from the frame buffer, while the other is the time needed to send each image across the PCI bus to the compositing hardware. Frame buffer reads could be accelerated by using the faster 8x AGP graphics bus, while in the future we plan to develop a new compositing system that employs a faster bus like *PCI Express*.

Next, we evaluated the visual simulation performances of our system by using the reaction-diffusion example. Dividing the body surface voxels of the dog model into subvolumes as in Fig. 9 and dispatching them into the Render-nodes of our PC cluster system, simultaneous simulation and visualization are possible. Fig. 9(a) presents the Cartesian subdivision., and Figs. 9(b) and 9(c) demonstrate adaptive subdivisions. There are three possible axes for division to occur in adaptive subdivision. Fig. 9(b) (Adaptive 1) uses subdivision along the axis that minimizes the size of the largest sub-volume to balance the size of the rendering task for each node. The subdivision scheme also affects the performance of boundary-data exchange. Fig. 9(c) (Adaptive 2) illustrates the use of the axis that minimizes the number of boundary points. Asymmetric voxel distributions, illustrated in Figs. 9(b) and 9(c), improve computational load balancing between nodes. However, these irregular subdivisions can simultaneously degrade volume-rendering performance since the rendering task can become less balanced and in some cases a node might no longer have sufficient texture memory to fit its subvolume. For example, we were not able to subdivide the VHM data set into 16 nodes by using these adaptive methods.

Fig. 10(a) illustrates the computation time (seconds) of 10,000 iterations to obtain the polka-dot pattern of Fig. 5(a). When the number of Render-nodes is less than 8, the simulation cannot be displayed because the aggregated texture memory storage across the system is insufficient to store the volume. During the simulation, the value u in Eqs. (1) is visualized every 50 iterations using a schedule similar to that shown in Fig. 4(a), permitting the user to see the simulation as it progresses at interactive rates. Although the visualization rate of every 50 iterations corresponds to only 0.4 Hz (using 16 nodes), the user still can interact with the latest volume date. Most of the simulation time was spent for "Voxel" operation, and the subdivision method of Fig. 9(c) yielded the best results. Changing the number of Render-nodes from 8 to 16 reduces the computation time nearly in half.

Fig. 10(b) compares the "Boundary" operation times for different network systems. It is clear that the Adaptive 2 efficiently reduced the cost of boundary exchange. Since this operation includes processes other than communication (e.g. data reorganization), Gigabit Ethernet exhibited less than twice the performance of 100Base-TX, and the difference between Myrinet 2000 and Gigabit Ethernet was even less.

Fig. 11 displays the average frame rates for the pipeline mode in Fig. 4(b) with the volume displayed to a 512×512 window as shown in Fig. 5(a). We compared the frame rates for differing number of nodes (Render-nodes) using the three subdivision methods of Fig. 9. It was our expectation that Adaptive 1 would yield the best results since it was designed to improve the visualization performance. Contrary to our expectations, Adaptive 2 yielded better results when used with eight Render-nodes. Since our subdivision method is greedily adaptive, and not guaranteed optimal, the simple decision rule may have resulted in inadequate



(a) Consumption times of the visual simulations (seconds) (b) Boundary-data exchange times (seconds)

Figure 10 Reaction-diffusion simulation performances for 10000 iterations.

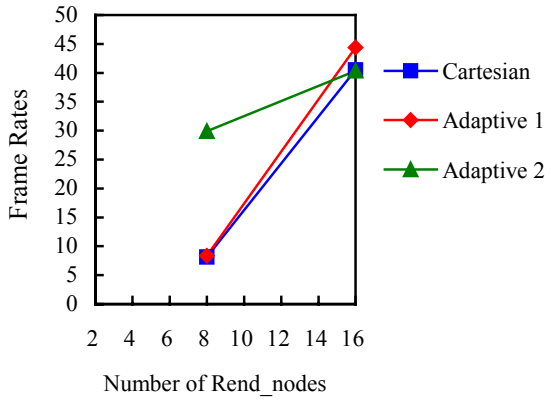


Figure 11 Frame rates (Hz) at pipelined visualization mode

subdivision. Finding the optimal solution, however, would require checking all possible subdivisions which would be too computationally expensive, since there are 3^{15} (14,348,907) combinations for 16 Render-nodes.

As shown in Fig. 11, frame rate differences between subdivision methods were very small when we used 16 Render-nodes. Since the visualization occurs less frequently than the simulation update, the subdivision technique to reduce the amount of boundary-data exchange is crucial to achieve the high performance visual simulation. The latest graphics accelerators support floating-point computation and can perform most of the simulation computations inside them [21]. In this situation it would not be necessary to load the computed result from main memory to texture memory for visualization, but it would still be necessary to copy texture data into main memory for boundary-data exchange. Thus, efficient boundary-data exchange will continue to be important in future visual simulations.

7. Conclusions

The increasing performance and decreasing price of commodity PC systems as well as graphics and communication subsystems enable the construction of low-cost, high-performance systems to study problems in science and engineering with computational requirements that were previously prohibitively expensive. The cluster system we have built takes full advantage of commodity PC hardware to deliver high-performance computing and graphics, and is the first one that demonstrates tightly coupled modeling and visualization for a suite of application problems.

Most real-world simulation problems demand a large cluster system to achieve the required accuracy or turnaround time. Our system permits scalable, real-time volume-rendering performance (45 frames per second for a 512×512 image size), which makes possible interactive visualization-based modeling of large-scale problems. For example, in addition to using the reaction-diffusion equations for texture generation, our system can be employed to simulate brain nerve excitement using the *Hodgkin-Huxley* equations [19], and observe a visual presentation during the whole course of the simulation. We believe that the simulation of regions of the brain of lower animals will be possible by scaling our system to thousands of nodes. For such a large system, one relevant problem that remains to be investigated is optimal object-space decomposition to facilitate both simulation and visualization calculations.

References

[1] Kniss, J., McCormick, P., McPherson, A., Ahrens, J., Painter, J., Keahey, A., Hansen, C., T-Rex: Interactive Texture-Based Volume Rendering for Large Data Sets, *IEEE Computer Graphics & Applications*, Vol. 21, No. 4, 2001.

[2] Wylie, B., Pavlakos, C., Lewis, V., Moreland, K., Scalable Rendering on PC Clusters, IEEE CG&A, Vol. 21, No. 4, pp.62-70, 2001.

[3] Molnar, S., Cox, M., Ellsworth, D., Fuchs, H., A Sorting Classification of Parallel Rendering, *IEEE CG&A*, Vol.14, No.4, pp.23-32, 1994.

[4] http://www.terarecon.com/products/volumepro_prod.html

[5] Cabral, B., Cam, N., Foran, J., Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware, *Proc. ACM Symp. on Volume Visualization*, 1994.

[6] Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., Ertl, T., Interactive Volume Rendering on Standard PC Graphics Hardware

Using Multi-Textures and Multi-Stage Rasterization, *Proc. ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware 2000*, pp. 109-118, 2000.

[7] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, *ACM Trans. Graphics (Proc. SIGGRAPH 2002)*, Vol. 21, No. 3, pp.693-702, 2002.

[8] Ma, K.-L., Painter, J.S., Hansen, C.D., Krog, M.F., Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE CG&A*, Vol.14, No.4, pp.59-68, 1994.

[9] Lee, T-Y, Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers, *IEEE TVCG*, Vol. 2, No. 3, pp.202-217, 1996.

[10] Stoll, G., Eldrige, M., Buck, I., Patterson, D., Webb, Art., Berman, S., Levy, R., Caywood, C., Taveira, M., Hunt, S., Hanrahan, P., Lightning-2: A High-Performance Display Subsystem for PC Cluster, *Proc. SIGGRAPH 2001*, pp.141-148, 2001.

[11] Zhang, X., Bajaj, C., Blanke, W., Scalable Isosurface Visualization of Massive Datasets on COTS Clusters, *Proc. IEEE 2001 Symp. Parallel and Large-Data Visualization and Graphics*, pp.51-58, 2001.

[12] Moll, L., Heirich, A., Shand, M., Sepia: scalable 3D compositing using PCI Pamette, Proc. IEEE Symposium on FPGAs for Custom Computing Machines, pp. 246-155, 1999.

[13] Lombeyda, S.,Moll, L., Shand, M., Breen, D., Heirich, A., Scalable Interactive Volume Rendering Using Off-the-Shelf Component, *Proc. IEEE Sympo. Parallel and Large-Data Visualization and Graphics*, October 2001.

[14] Muraki, S., Ogata, M., Kajihara, K., Ma, K.-L., Koshizuka, K., Liu, X., Nagano, Y., Shimokawa, K., Next-Generation Visual Supercomputing using PC Clusters with Volume Graphics Hardware Devices, *Proc. IEEE SC2001*, 2001.

[15] http://www.pccluster.org/index.html.en

[16] Turing, A. M., The chemical basis of morphogenesis, *Phil. Trans. Roy. Soc.*, B237, pp. 37-72, 1952.

[17] Turk, G., Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion, *Computer Graphics (Proc. SIGGRAPH 91)*, Vol. 25, No. 4, pp. 289-298, July 1991.

[18] Cabral, B., Leedom, L.: Imaging vector field using line integral convolution, *Computer Graphics (Proc. SIGGRAPH 93)*, pp.263-270, August 1993.

[19] Beeman, B., *The Book of GENESIS, 2nd Ed*., Splinger- Verlag, 1997.

[20] M. Tokunaga, S. Ando, Y. Suzuki, S. Muraki, Y. Takeshima, N. Takahashi, I. Fujishiro, Realtime Rendering of 3D Attributed LIC Textures Using Programmable GPU, *Proc. Visual Computing 2003*, pp. 213-218, June 2003. (in Japanese)

[21] W. R. Mark, R. S. Glanville, K. Akeley, M. Kilgard, Cg: A system for programming graphics hardware in C-like language, *acm Trans. on Graphics (Proc. SIGGRAPH 2003)*, vol. 22, no. 3, pp. 896-907, July 2003.