International Journal of CAD/CAM Vol. 9, No. 1, pp. 111~120(2009)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

Volume Rendering using Grid Computing for Large-Scale Volume Data

Kunihiko Nishihashi*, Toru Higaki, Kenji Okabe, Bisser Raytchev, Toru Tamaki, and Kazufumi Kaneda
Graduate School of Engineering, Hiroshima University, Hiroshima, Japan

Abstract – In this paper, we propose a volume rendering method using grid computing for large-scale volume data. Grid computing is attractive because medical institutions and research facilities often have a large number of idle computers. A large-scale volume data is divided into sub-volumes and the sub-volumes are rendered using grid computing. When using grid computing, different computers rarely have the same processor speeds. Thus the return order of results rarely matches the sending order. However order is vital when combining results to create a final image. Job-Scheduling is important in grid computing for volume rendering, so we use an obstacle-flag which changes priorities dynamically to manage sub-volume results. Obstacle-Flags manage visibility of each sub-volume when line of sight from the view point is obscured by other sub-volumes. The proposed Dynamic Job-Scheduling based on visibility substantially increases efficiency. Our Dynamic Job-Scheduling method was implemented on our university's campus grid and we conducted comparative experiments, which showed that the proposed method provides significant improvements in efficiency for large-scale volume rendering.

Keywords: **Distributed volume rendering, grid computing, obstacle-flag, dynamic job-scheduling, visibility**

1. Introduction

This paper proposes the use of grid computing for direct volume rendering. There are two types of direct volume rendering: ray casting [Levoy 88] and splatting [Lee 89]. The computational cost for both methods depends on the size of volume data, computational time increasing proportional to three times the cube of the size. Several methods have been proposed to reduce the computational time: visibility sorting [Callahan 05], [Hofsetz 08], slicing a volume into planes [Keles 08], use of GPU [Callahan 05], [Hofsetz 08], [Keles 08], cluster computing [Frank 05], [Lacroute 96], [Matsui 04], [Matsui 03], [Sasaki 02], [Stompel 03], [Peggy 97], and grid computing [Cordoba 05], [Norton 03], [Bethel 03], [Iwabuchi 07]. Here, we focus our attention on grid computing because of the availability of idle resources in many medical institutions and hospitals.

In distributed computing for volume rendering, the volume data is divided and each piece is rendered individually, then the results are returned to a central manager where they are combined to create the final image. When results are combined, each result has a level of opacity which ultimately affects the final results of any given section. Thus sub-volumes can only be rendered when the line of sight from the screen is not occluded by other sub-volumes.

Furthermore, the order in which data is send to the agent computers must be considered carefully. Previously, the order of sending data was based on a z-value, and was determined by the sub-volumes' distance from the screen. In grid computing, computing resources often change significantly in short periods of time, thus making sequential job-scheduling unsuitable. In our method each sub-volume receives an obstacle-flag, which is dynamically updated, and is used to determine a sub-volume's current visibility. The processing order can be determined based on current visibility values rather than initial values, thus improving efficiency.

In Section 2, we cover related work, and Section 3 covers the general method of volume rendering using grid computing and disadvantages of sequential job-scheduling. In Section 4, we describe the Dynamic Job-Scheduling and the details of the proposed method. In Section 5, we review and evaluate our results, and we conclude in Section 6.

2.Related Work

Volume rendering has been around for a notable amount of time as a method for rendering and thus has received many various improvements over time, many of which recently are performed on GPUs, as in the following. In [Callahan 05] a hardware assisted visibility sorting algorithm is implemented and operates both in object-space and image-space, primarily making use of GPU based calculations. [Hofsetz 08] makes use of pre-sorting primitives in object-space using a list for each axis, then combining the lists using graphics hardware through converting each list into a texture. [Keles 08] uses the GPU to create texture slices which are grouped to form a texture slab. The method relies on hardware z-occlusion culling and hardware occlusion queries to accelerate

*Corresponding author:
Tel: +81-82-424-7663
Fax: +81-82-420-0048
E-mail: nissin@eml.hiroshima-u.ac.jp

ray traversals.

Other non-GPU based improvements have also been made including [Sasaki 02] which achieves real time volume rendering of a 1024^3 volume data at 1.5 frames/sec through lowering communications at image compositing and balancing the load among processors in a computing cluster. Later in [Matsui 03] early ray termination was enabled where agents were allowed to avoid rendering invisible objects, which led to 5 frames per second volume rendering. In [Matsui 04] the method of early ray termination was improved upon and offered better performance for not only dense objects but transparent objects as well. In [Lacroute 96] a parallel shear-warp algorithm was proposed and several good results were demonstrated. In [Frank 05] was proposed a distribution manager for volume data to reduce the required memory. They applied the method to rendering high-resolution volume data, and showed the reduction of data storage as well. In [Stompel 03] a parallel image compositing algorithm called Schedules Linear Image Compositing (SLIC) was presented and its performance on a PC cluster was shown. In the SLIC method, each processor's compositing load becomes less view dependent because image space partitioning for compositing tasks is crucial to load balancing and only the pixels in the overlapping areas need to be sent to the processors responsible for compositing the corresponding areas. This method can achieve interactive rendering for images at resolution up to 1024×1024 pixels. The ParVox system [Peggy 97] is a parallel volume rendering system that is capable of visualizing large volume datasets. In the system, a splatting-based rendering algorithm with both object space and image space decomposition was designed and implemented.

Volume visualization on the grid has been discussed in several articles. Grid computing is a promising technique when projecting a large-scale volume data, which exceeds the resources of most common PCs [Cordoba 05]. A method for enabling progressive volume visualization of data on the computing grid was proposed in [Norton 03]. They developed visibility-driven compression scheme based on wavelet encoding to alleviate the network bandwidth problem. A network protocol to accelerate network throughput of grid-based distributed visualizations was also developed in [Bethel 03]. These approaches mainly focus on network communication in grid computing environment.

In this paper, we focus on job-scheduling of volume rendering in grid computing. In [Iwabuchi 07] a distributed rendering system was developed, but this system is simply a process of decentralization. The method is not suitable for volume rendering, since it cannot handle opacities and efficient job-scheduling is not taken into account.

3. Volume Rendering using Grid Computing

In grid computing volume rendering, large-scale volume data is divided into smaller sub-volumes and sent to various computers (agents). Each sub-volume is processed and the result is returned to the central manager where the final image is combined. Figure 1 shows a simple diagram of this process.

The order in which the results are combined is extremely important and care must be taken. Two methods exist for combining the results, back-to-front and front-to-back. Both methods determine their order based on the distance of the sub-volume from the screen. We use front-to-back processing so that if a section of the image has an overall opacity of one, then sections further away can be ignored since their results will not be visible.

Using grid computing, the result of each agent is used in the same manner as splatting in volume rendering, where highly visible sub-volumes are processed first. A common problem which affects grid computing is that agents almost always have different computing capabilities. In addition, people are free to use the agents which drastically reduces their resources to spend on calculating rendering data.

3.1 Dividing a Volume

Various methods for dividing sub-volumes exist. In [Keles 08] planes at specified distances from the camera are sliced to define the sub-volumes. In [Sasaki 02] volume data is divided along the axes to create sub-cubes and in [Sung 91] an octree structure is used to divide sub-volumes. These dividing methods have two good aspects: one is that renderings can be done efficiently and another is that the required amounts of memory can be quite small. However in the case of the octree the central manager has to make many calculations and the order of combining the results is often complex. In the case of plane slicing the order of combining sub-volumes is very rigid, which doesn't lend these methods to work well with grid computing.

In this research we divide the volume data into equally sized cubic sub-volumes, so that we may place the screen at any location and calculate the processing order, as opposed to using the plane slicing method

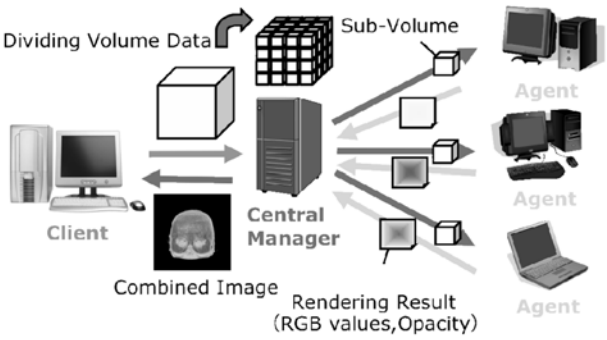


Fig. 1. System Configuration.

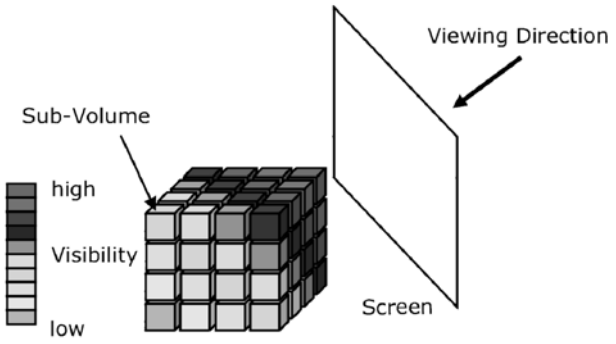


Fig. 2. Visibility of Sub-Volume.

which doesn't allow for such flexibility. Dividing the data in this way allows easy implementation of obstacle-flags for efficiently determining job-scheduling.

In Figure 2, a sub-volumes' level of visibility is shown by its color, so when results are combined, sub-volumes with the same color can be processed in any order. Being able to process the results in a more flexible manner is especially beneficial in grid computing. In the next section we address the details of job-scheduling.

3.2 Disadvantages of Sequential Job-Scheduling

In volume rendering the object to render is transparent so much care must be taken when creating the order of combining the results. It is necessary to combine the results based on the level of visibility (from the screen) for each sub-volume. Each sub-volume has a relationship with neighboring sub-volumes based on their line of sight with the viewpoint, i.e. whether a sub-volume's line of sight to the camera is obscured by a neighboring sub-volume or not. The order in which sub-volumes are sent to agent computers is determined by the sub-volume's initial level of visibility. A sub-volume's visibility is determined from both the screen's location and its relation with neighboring sub-volumes.

If the order of processing is determined by the initial values of visibility for each sub-volume (we call this ordering, sequential job-scheduling), there is a problem for volume rendering based on grid computing. To simplify the explanation in this section we will only consider the two dimensional case. In Figure 3(a) visibility is determined by view direction which sets high visibility to sub-volumes which are close to the screen. In Figure 3(b) sub-volumes 3 and 6 have been combined in the results, so sub-volume 9 becomes ready to be combined. However sub-volume 9 is not send-able as a result of sequential job-scheduling. The reason sub-volume 9 cannot be sent is because sequential job-scheduling only uses the initial visibility values for assigning order. Thus in sequential job-scheduling, sub-volume 9 is not sent to an agent until sub-volume 2 has been combined into the result. Of course this limits the efficiency of grid computing. By implementing a dynamic method for defining the sending order, this problem can be alleviated.

To solve the problems with sequential job-scheduling, we propose to dynamically update the visibility parameter of the sub-volumes as they are rendered. To handle dynamic management of sub-volumes we propose using an obstacle-flag, which we cover in the next section.

4. Dynamic Job-Scheduling

When rendering results are combined the visibility parameter of sub-volumes change which affects sub-volumes' combinability. When a sub-volume is not obscured by any neighboring sub-volumes, then that sub-volume becomes combinable. By using obstacle-flags we maintain visibility relationships which enable us to dynamically manage job-scheduling.

4.1 The Obstacle-Flag

Obstacle-flags manage relationships between sub-volumes, specifically whether a sub-volume is obscured from the other sub-volumes. Naturally, some sub-volumes are obscured by other sub-volumes and do not have line of sight to the screen, however as the view direction changes so do the relationships of which sub-volumes obscure which. In the two dimensional case, we have a 4 bit obstacle-flag, which defines in which directions, a neighboring sub-volume has obscuring sub-volumes. Each bit corresponds to a single direction, and we are only interested in the bits which represent directions that could contain obscuring sub-volumes. We ignore the case where sub-volumes lie further away, we only care about the neighboring sub-volumes.

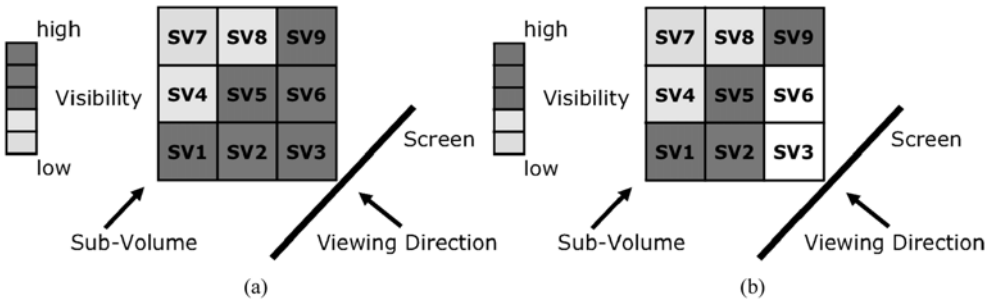


Fig. 3. Visibility of Sub-Volume.

The maximum number of obscuring sub-volumes is two. If a sub-volume's obstacle-flag is all zeroes, this means that there are no obscuring sub-volumes and that sub-volume is combinable. All sub-volumes have an obstacle-flag.

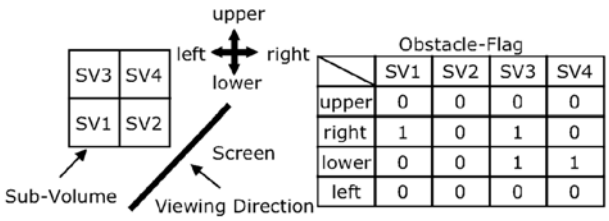
Figure 4 shows the related obstacle-flags for the example sub-volume and view-directions. In the case of sub-volume 3, the right and lower directions contain obscuring sub-volumes so those values are set to 1.

An obstacle-flag has three meaningful states, which is determined by the number of 1's in the 4 bit obstacle-flag. An obstacle-flag can have zero, one or two 1's. The number of 1's in the obstacle-flag is the obscure count. In Figure 4 the obscure count for sub-volume 2 is zero, for sub-volume 1 is one, and for sub-volume 3 is two. In the three dimensional case the obstacle-flag has 6 bits and the obscure count has a maximum value three.

4.2 Visibility of Sub-Volume Based on Obstacle-Flag

To determine visibility of sub-volumes dynamically, we propose a method that determines the visibility of sub-volumes based on obstacle-flags. In the case of sequential job-scheduling the sending process will wait for prior results to be combined thus making the process inefficient. However by using the obstacle-flags we are able to continuously observe the obscuring volumes, so we can dynamically monitor sub-volumes' visibility.

Figure 5 shows the visibility of sub-volumes based on obstacle-flags. In the proposed method we use the obscure count to determine the level of visibility of a sub-volume. In Figure 5 we can see the new visibility levels of the sub-volumes from the case in Figure 3 used in the previous method. As you can see sub-volumes 5 and 9 have different values in the proposed method, when compared to those in Figure 3.



Obstacle-Flag

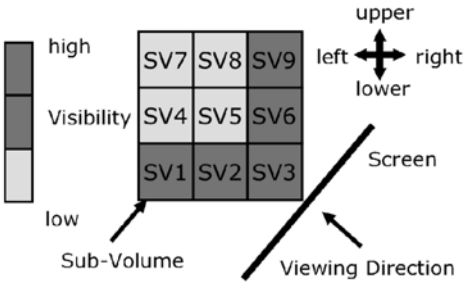
	SV1	SV2	SV3	SV4
upper	0	0	0	0
right	1	0	1	0
lower	0	0	1	1
left	0	0	0	0

Fig. 4. Obstacle-Flag.

4.3 Dynamic Job-Scheduling using Obstacle-Flag

In this section, we propose the Dynamic Job-Scheduling method using obstacle-flags. In Figure 6, sub-volumes 3 and 6 have already been rendered and combined. After a result is returned from an agent, sub-volumes' obstacle-flags are updated and new combinability values are assigned. From their new visibility levels are determined which then allows new sub-volumes to be sent to the agents for rendering. Thus, sub-volumes 2, 5 and 9 have their obstacle-flags updated.

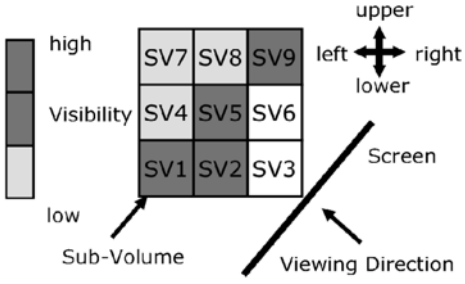
In the case of sequential job-scheduling sub-volume 9 would be combinable but would not be sendable because the visibility of sub-volume 9 would not be updated and thus still be considered obscured. Thus it would be necessary to wait for sub-volume 2 to be combined before being sent to an agent.



Obstacle-Flag

	SV2	SV3	SV5	SV6	SV9
upper	0	0	0	0	0
right	1	0	1	0	0
lower	0	0	1	1	1
left	0	0	0	0	0

Fig. 5. Visibility of Sub-Volumes Based on Obstacle-Flags.



Obstacle-Flag

	SV2	SV3	SV5	SV6	SV9
upper	0	0	0	0	0
right	0	0	0	0	0
lower	0	0	1	0	0
left	0	0	0	0	0

Fig. 6. Dynamic Job-Scheduling using Obstacle-Flag.

However when using obstacle-flags, we can see that sub-volume 9 has a obscure count of zero, meaning it is not obscured and it can be sent to an agent for rendering. When combining results and using obstacle-flags, we can dynamically manage obscuring relationships between sub-volumes, thus avoiding unnecessary waiting periods.

After a sub-volume is combined with the result, that sub-volume's remaining neighbors have their obstacle-flags updated. To determine the remaining neighboring sub-volumes, we use a simple method based on the grid structure and view direction.

Here we give some details of our implementation of the Dynamic Job-Scheduling using obstacle-flags. First, we generate obstacle-flags for each sub-volume. Next, we determine which sub-volumes have zero bits in the obstacle-flag. If a sub-volume has zero bits in the obstacle-flag we add it to the processing list and the central manager. If a sub-volume is in the processing list, it is currently being rendered. Once a sub-volume's result is returned and combined into the final result, it is removed from the processing list and the neighboring sub-volumes' obstacle-flags are updated (see Figure 7).

Only combinable sub-volumes are sent to agents. If non-combinable sub-volumes were sent, it would be necessary to store their results in memory on the central manager until they became combinable. With wider ranges of computing powers and more sub-volumes the probability of needlessly using memory increases. Next we discuss how to best utilize our agents.

4.4 Improving Agent Utilization by Exception Handling

To improve efficiency, partially-occluded sub-volumes can also be sent to an agent if available. The sub-volumes have three states: not occluded (the obscure count is 0), partially-occluded (the obscure count is 1), and fully-occluded (the obscure count is 2). If there are no sub-volumes whose obscure count is 0, partially-occluded volumes are made to be ready to send (See Figure 8). The system waits for results to be combined, if there are neither partially-occluded nor non-occluded volumes. Results are not combined unless all bits in the obstacle-flag are zero. This procedure we call Exception Handling (EH). It can minimize waiting time and memory use, while maximizing agent utilization.

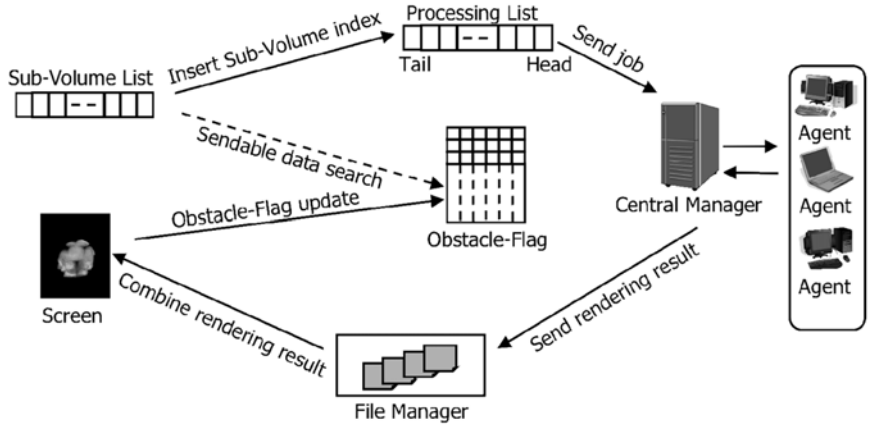


Fig. 7. Distributed Volume Rendering Procedure.

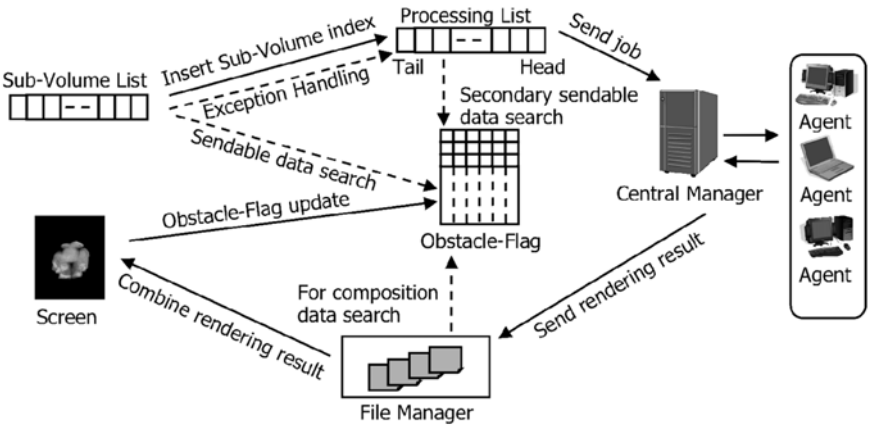


Fig. 8. EH Implementation Procedure.

5. Experiments

5.1 Simulation Verification

Through simulation we verified the proposed method's effectiveness. As shown in Table 1, we used 12 agents with three types of computing powers. Computing powers were low, medium and high where rendering results required 1000sec., 500sec. and 250sec., respectively. Next, we set four agents to each computing power. Result composition, job ordering and data transfer were all completed instantly. Our volume data consisted of 100 sub-volumes.

In addition, agents are regularly used by third parties during which rendering is interrupted. If a job has been sent and someone uses that agent, then rendering is temporary stopped. When that agent becomes free again, rendering continues. Figure 9 shows the agent usage schedule. High means an outside source used the agent, while low means the agent is free to render sub-volumes. We set low spec machines to have short interval interruptions, while high spec machine had long interruptions. At any given time, always six agents were available and six were being interrupted.

When comparing sequential job-scheduling to the Dynamic Job-Scheduling, the later sends jobs when agents are free and have all zero bits in the obstacle-flag, while sequential job-scheduling also requires the previous sub-volumes to be combined with the result.

When using the Dynamic Job-Scheduling a 65 percent reduction in computational costs was achieved for large-scale volume rendering. According to our simulation results in Table 2, the Dynamic Job-Scheduling offers a significant improvement in elapsed times and agent utilization. In Figure 10, we can also see a steeper rendering progress compared to sequential job-scheduling.

Table 1. Simulation Environment

Agents	4 (Low-Spec) 4 (Middle-Spec PC) 4 (High-Spec)
Server	Job order decision and Result composition take 0 sec.
Network	All transfer times are 0 sec.
Sub-Volume	100 (regular grid)

Table 2. Simulation Result

	Dynamic Job-Scheduling	Sequential Job-Scheduling
Elapsed Time [sec.]	10000	15500
Agent Utilization [%]	54.2	38.4

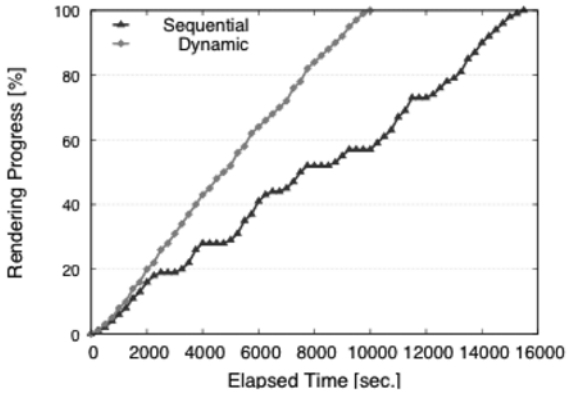


Fig. 10. Rendering Progress.

5.2 Verification using Test Data

We performed experiments, using our university's campus grid. The computer grid's managing software is Condor and other specifications are given in Table 3. After job-scheduling is performed by the proposed method, each job is submitted to the Central Manager of the Condor system, and the Central Manager distributes jobs to agents.

Figure 11 shows the campus grid network diagram. Between agents, the central manager, and a NFS, the

Table 3. Experimentation Environment

Number of Agents	OS	CPU	Memory
34	Linux	Xeon 3.06 GHz	2 GB
469		Pentium4 3.06 GHz	990 MB

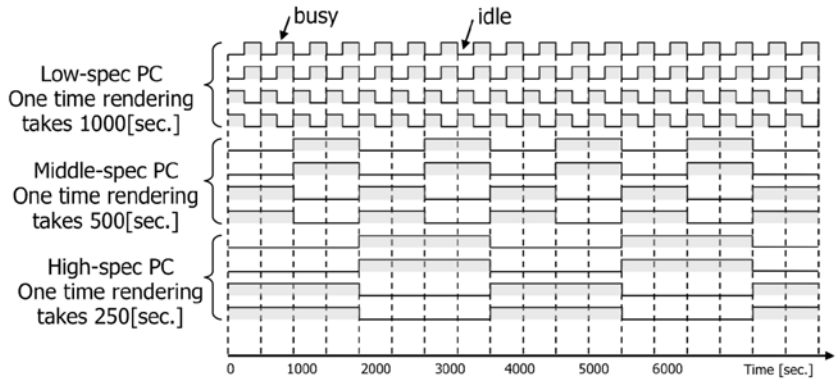


Fig. 9. Agent Usage Schedule.

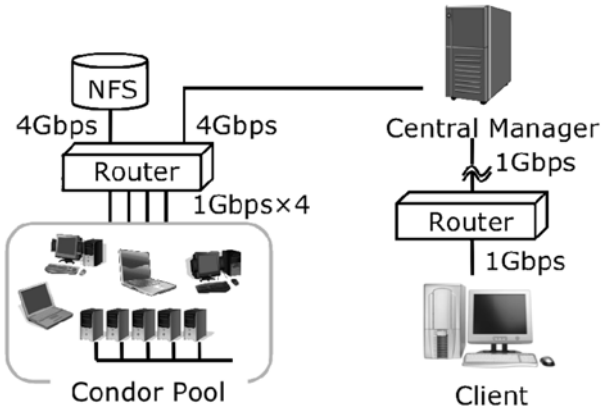


Fig. 11. Network Configuration.

network speed is 4Gbps, while manager to client is 1 Gbps. Table 4 shows three sets of parameters of the test data used.

Figure 12 (a) shows the elapsed times and (b) shows the average number of agents utilized. Each experiment was conducted five times and the median value of the results was used. In Figure 12 (a), the vertical axis is elapsed time. The shorter the elapsed time, the better the job-scheduling. In Figure 12 (b), the vertical axis is the average number of agents utilized. The larger the number of agents utilized, the better the job-scheduling.

We can see that the Dynamic Job-Scheduling uses more agents, showing the benefit of using the obstacle-flags. However, elapsed time was improved in the two cases: the 64 sub-volumes case and 512 sub-volumes with 4096^3 voxels case.

Generally, agents utilization was increased and elapsed time was also improved. However, in the case of $2048^3/512$ elapsed time was not improved. The reason for this is in the setting of our grid computing system, in which Condor allows only 1 job per second per person.

To summarize the results, the Dynamic Job-Scheduling and the exception handling reduce elapsed time in the case of $4096^3/512$ and $2048^3/64$. In both cases, the sub-volume size is relatively large (see Table 4).

5.3 Verification under Various Agent Processing Times

We investigate the performance of the Dynamic Job-Scheduling and the exception handling, when changing the agent processing times to 60, 120 and 180 seconds. The number of sub-volumes is fixed to 512.

The results are shown in Figures 13 (a) and (b). Each experiment was conducted five times and the median value of the results was used. In Figure 13 (a), the longer the agent processing time becomes, the better the performance is for both the Dynamic Job-Scheduling and the exception handling. In Figure 13 (b), the average number of agents used depends on the number of sub-volumes, whatever the agent processing time is.

5.4 Verification in the Case of Interruptions

In grid computing, computing resources often change significantly over time. Rendering processes executed on agents are interrupted when the agents are used by third parties. That is, the job that has been sent to an agent is temporary stopped, if someone uses the agent. When the agent becomes free, the job continues. We investigate the effectiveness of the Dynamic Job-Scheduling

Table 4. Test Data Parameters (VD: Volume Data, SV: Sub-Volume)

Resolution[voxel]	VD size[GB]	Number of Divisions	SV size[MB]	Screen size[pixel]
2048^3	16	64	256	3000 × 3600
		512	32	
4096^3	128	512	256	5800 × 7200

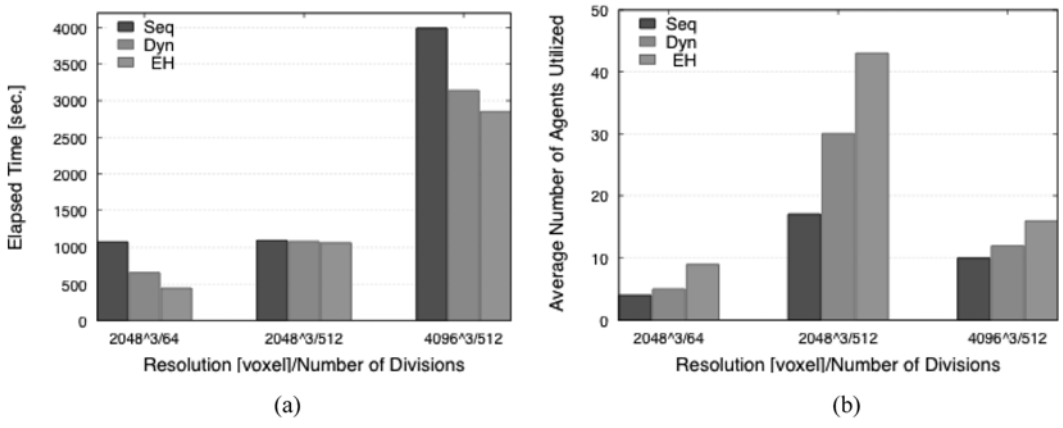


Fig. 12. Experimental results: (a) Elapsed time and (b) Average number of agents utilized (Seq: Sequential Job-Scheduling, Dyn: Dynamic Job-Scheduling, EH: Exception Handling).

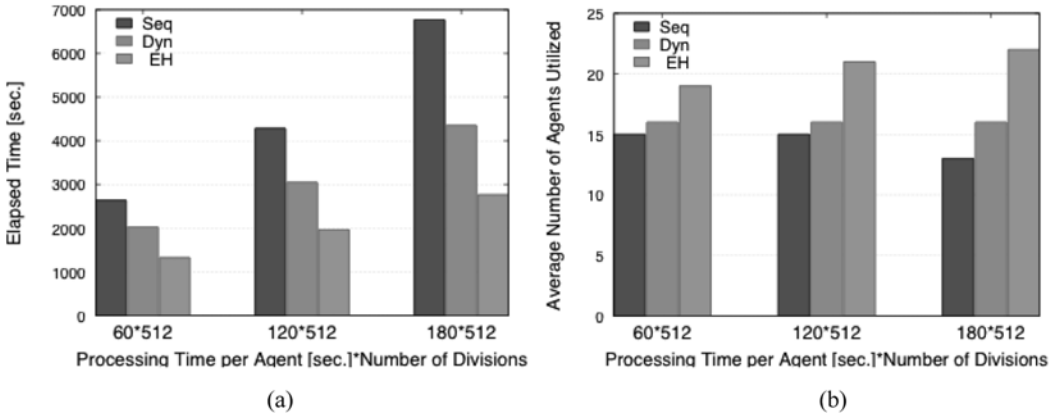


Fig. 13. Experimental results: (a) Elapsed time and (b) Average number of agents utilized.

in that kind of situation.

In our experiment, the number of sub-volumes is 512, and the agent processing time is 60 seconds. The agents that receive job numbers 100, 200, 300, 400, and 500 are interrupted for a certain period. We change the interruption time in two types: short and long periods, 120 sec. and 480 sec., respectively.

The results are shown in Figures 14 (a) and (b) where the interruption times are 120 sec. and 480 sec., respectively. Each experiment was conducted five times and the median value of the results was used. The rendering progress of the Dynamic Job-Scheduling is better than that of the sequential job-scheduling. The sequential job-scheduling is greatly affected by the interruptions, while the Dynamic Job-Scheduling is less affected.

Table 5 shows speed-up ratios of the Dynamic Job-Scheduling compared to the sequential job-scheduling.

Table 5. Speed-up Ratios of the Dynamic Job-Scheduling

Interruption Time[sec.]	Speed-up Ratio[%]
0	23.14
120	39.42
480	41.42

The speed-up ratio shows how much the elapsed time is shortened, and the larger the ratio, the shorter the elapsed time. As can be seen, the performance of the Dynamic Job-Scheduling increases as larger interruption times are applied.

6. Conclusions

In this paper we proposed a new method for large-scale volume data rendering in a grid computing system, Dynamic Job-Scheduling using obstacle-flags. The proposed method generally performs better than the sequential job-scheduling: simulation results showed the reduction of elapsed time and the improvement of average number of agents utilized. The rendering progress of the proposed method was also smoother than the sequential job-scheduling. The experimental results also indicated that the longer the agent processing time is the more effectively the Dynamic Job-Scheduling and the exception handling work.

In the future, we intend to conduct a larger-scale experiment: increasing the number of agents, the size of the volume data (such as terabyte and petabyte volume data) and the number of divisions.

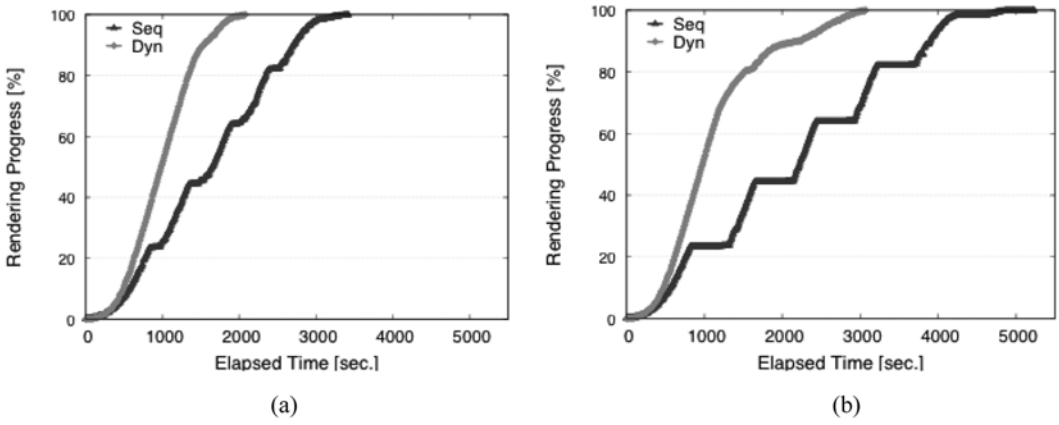


Fig. 14. Experimental results: (a) 120 sec. and (b) 480sec., respectively.

References

[1] Bethel, E. W., and Shalf, J., "Grid-distributed visualizations using connectionless protocols," IEEE Computer Graphics and Applications, vol. **23**, no. 2, pp. 51-59, 2003.

[2] Cordoba de Alfonso, I. Blanquer, V. H., "Large medical datasets on the grid," Methods of Information in Medicine 2005, vol. 44, no. **2**, pp. 172-176, 2005.

[3] Callahan S. P., Ikits, M., Comba, J. L., and Silva, C. T., "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering," IEEE Transactions on Visualization and Computer Graphics, vol. **11**, no. 3, pp. 285-295, 2005.

[4] Frank, S., and Kaufman, A., "Distributed volume rendering on a visualization cluster," in CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics, pp. 371-376, IEEE Computer Society, 2005.

[5] Hofsetz, C., Max, N., and Bastos, R., "Object-Space Visibility Ordering for Point-Based and Volume Rendering," Computer Graphics Forum, vol. **27**, no. 1, pp. 91-101, 2008.

[6] Iwabuchi, E., and Watanabe, I., "Development of decentralize rendering system using grid technology," in Proc. Visual Computing / Graphics and CAD Symposium 2007, pp. 181-184, 2007. (in Japanese).

[7] Keles, H. Y., ES, A., and Isler, V., "Acceleration of direct volume rendering with programmable graphics hardware," The Visual Computer, vol. **23**, no. 1, pp. 15-24, 2006.

[8] Lacroute, P., "Analysis of a parallel volume rendering system based on the shear-warp factorization," IEEE Trans. on Visualization and Computer Graphics, vol. **2**, no. 3, pp. 218-231, 1996.

[9] Lee Westover, "Interactive volume rendering," in Proc. Workshop on Volume Visualization, pp. 9–18, 1989.

[10] Levoy, M., "Display of surface from volume data," IEEE Computer Graphics & Applications, vol. **8**, no. 3, pp. 29-37, 1988.

[11] Matsui, M., Ino, F., and Hagihara, K., "Efcient parallel volume rendering for visualizing large-scale datasets," IPSJ Transactions on Advanced Computing Systems, vol. 45, no. 11, pp. 346–355, 2004. (in Japanese).

[12] Matsui, M., Takeuchi, A., Ino, F., and Hagihara, K., "Reducing the complexity of parallel volume rendering by propagating accumulated opacity," IEICE technical report. Computer systems, vol. **103**, no. 249, pp. 13-18, 2003. (in Japanese).

[13] Norton, A., and Rockwood, A., "Enabling view-dependent progressive volume visualization on the grid," IEEE Computer Graphics and Applications, vol. **23**, no. 2, pp. 22-31, 2003.

[14] P. Peggy. Li and Scott Whitman and Roberto Mendoza and James Tsiao., "ParVox - a parallel splatting volume rendering system for distributed visualization," Parallel Rendering Symposium, vol. **0**, p. 7, 1997.

[15] Sasaki, T., Ino, F., Fujimoto, N., and Hagihara, K., "High-resolution volume rendering on distributed-memory parallel machines," ITE Technical Report, vol. **26**, no. 22, pp. 7-12, 2002. (in Japanese).

[16] Stompel, A., Ma, K.-L., Lum, E. B., Ahrens, J., and Patchett, J., "SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering," in PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics, p. 6, IEEE Computer Society, 2003.

[17] Sung, K., "A DDA octree traversal algorithm for ray tracing," in Eurographics '91, pp. 73-85, North-Holland, 1991.

Kunihiko Nishihashi is master course student at the Department of Information Engineering, Graduate School of Engineering, Hiroshima University in Japan. He received bachelor degree in Electric Engineering from Hiroshima University, Japan in 2008. He is currently a student member of IPSJ.

Toru Higaki is Ph.D student at the Department of Information Engineering, Graduate School of Engineering, Hiroshima University in Japan. He received his B.E., M.E. degrees from Hiroshima University in 2006 and 2008, respectively. He was a exchange student in Harvard Medical School from 2008 to 2009. He is research fellow of Japan Society for the Promotion of Science since April 2009.

Kenji Okabe is currently working as Software Engineer at GE Healthcare Japan Corp, Japan. He received bachelor degree in Electric Engineering from Hiroshima University, Japan in 2005 and masters in the Department of Information Engineering, Graduate School of Engineering, Hiroshima University in Japan in 2007. He is currently designing software for medical devices.

Bisser Raytchev received his Ph.D. in Informatics from Tsukuba University, Japan in 2000. After being a research associate at NTT Communication Science Labs and AIST, he is presently an assistant professor in the Department of Information Engineering, Hiroshima University, Japan. His current research interests include computer vision, pattern recognition, high-dimensional data visualization and image processing.

Toru Tamaki received his B.E., M.S. and Ph.D degrees in information engineering from Nagoya University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is an associate professor at the Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Japan. His research interests include computer vision and image recognition.

Kazufumi Kaneda is a professor in Graduate School of Engineering at Hiroshima University. He joined Hiroshima University in 1986. He was a visiting researcher in Engineering Computer Graphics Laboratory at Brigham Young University in 1991. Kaneda received the BE, ME, and DE in 1982, 1984, and 1991, respectively, from Hiroshima University. His research interests include computer graphics, scientific visualization, and image processing.

Kunihiko Nishihashi

Toru Higaki i

Kenji Okabe

Bisser Raytchev

Toru Tamaki

Kazufumi Kaneda