Volume Rendering using Grid Computing for Large-Scale Volume Data

Kunihiko Nishihashi, Toru Higaki, Kenji Okabe, Bisser Raytchev, Toru Tamaki, Kazufumi Kaneda
Graduate School of Engineering, Hiroshima University, Hiroshima, Japan
{nissin, higa, okabe, bisser, tamaki, kin}@eml.hiroshima-u.ac.jp

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.

1. Introduction

This paper proposes the use of grid computing for direct volume rendering. There are two types of direct volume rendering: ray casting [1] and splatting [2]. 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 [3], [4], slicing a volume into planes [5], use of GPU [3], [4], [5], cluster computing [6], [7], [8], [9], [10], [11], [12], and grid computing [13], [14], [15], [16]. 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 [3] a hardware assisted visibility sorting algorithm is implemented and operates both in object-space and image-space, primarily making use of GPU based calculations. [4] 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. [5] 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 ray traversals.

• •Other non-GPU based improvements have also been made including [10] 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 [9] early ray termination was enabled where agents were allowed to avoid rendering invisible objects, which led to

978-1-4244-3701-6/09/$25.00 ©2009 IEEE

5 frames per sec volume rendering. In [8] the method of early ray termination was improved upon and offered better performance for not only dense objects but transparent objects as well. In [7] a parallel shear-warp algorithm was proposed and several good results were demonstrated. In [6] 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 [11] 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 [12] is a parallel volume rendering system that is capable of visualizing large volume datasets. In it, 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 [13]. A method for enabling progressive volume visualization of data on the computing grid was proposed in [14]. 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 [15]. 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 [16] 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

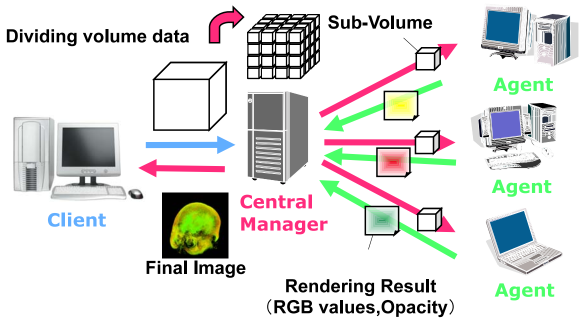


Figure 1. System Configuration

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 [5] planes at specified distances from the camera are sliced to define the sub-volumes. In [10] volume data is divided along the axes to create sub-cubes and in [17] 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 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.

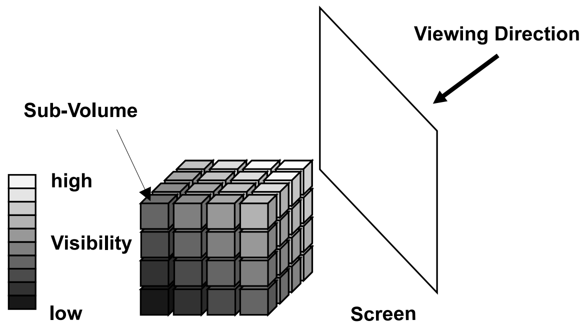


Figure 2. Visibility of Sub-Volumes

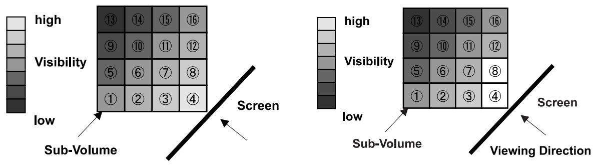


Figure 3. Sequential Job-Scheduling

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 4 and 8 have been combined in the results, so sub-volume 12 becomes ready to be combined. However sub-volume 12 is not send-able as a result of sequential job-scheduling. The reason sub-volume 12 cannot be sent is because sequential job-scheduling only uses the initial visibility values for assigning order. Thus in sequential job-scheduling, sub-volume 12 is not sent to an agent until sub-volume 3 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

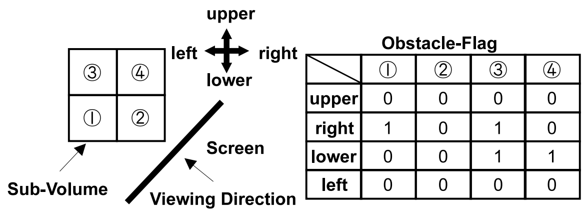
Obstacle-flags manage relationships between sub-volumes, specifically whether a sub-volume is obscured from the other sub-volumes. Naturally, some sub-volumes are obscured by other sub-volumes and do not have line of sight to the screen, however as the view direction changes so do the relationships of 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.

• •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.



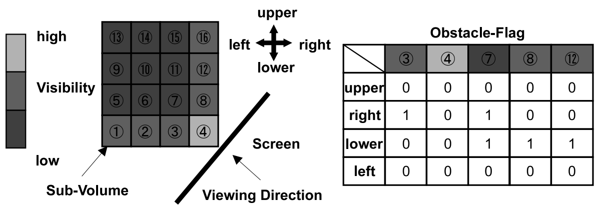
	①	②	③	④
upper	0	0	0	0
right	1	0	1	0
lower	0	0	1	1
left	0	0	0	0

Figure 4. Obstacle-Flag

4.2. Visibility of Sub-Volumes Based on Obstacle-Flags

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 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 7 and 12 have different values in the proposed method, when compared to those in Figure 3.

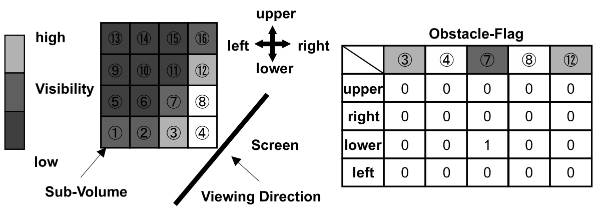


	③	④	⑦	⑧	⑫
upper	0	0	0	0	0
right	1	0	1	0	0
lower	0	0	1	1	1
left	0	0	0	0	0

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

4.3. Dynamic Job-Scheduling using Obstacle-Flags

In this section, we propose the Dynamic Job-Scheduling method using obstacle-flags. In Figure 6, sub-volumes 4 and 8 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 there new visibility levels are determined which then allows new sub-volumes to be sent to the agents for rendering. Thus, sub-volumes 3, 7 and 12 have their obstacle-flags updated. In the case of sequential job-scheduling sub-volume



	③	④	⑦	⑧	⑫
upper	0	0	0	0	0
right	0	0	0	0	0
lower	0	0	1	0	0
left	0	0	0	0	0

Figure 6. Dynamic Job-Scheduling using Obstacle-Flag

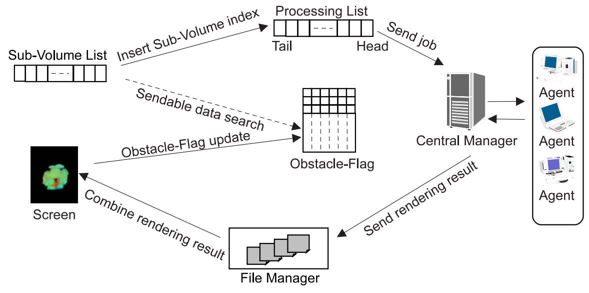


Figure 7. Volume Rendering Procedure

12 would be combinable but would not be sendable because the visibility of sub-volume 12 would not be updated and thus still be considered obscured. Thus it would be necessary to wait for sub-volume 3 to be combined before being sent to an agent.

• •However when using obstacle-flags, we can see that sub-volume 12 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

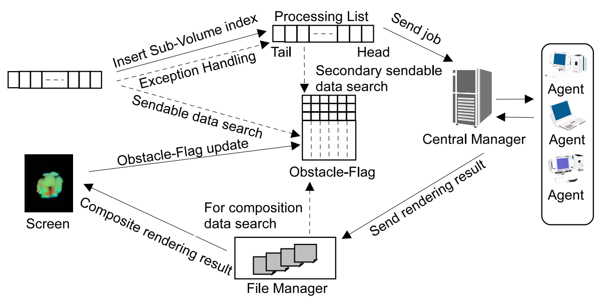


Figure 8. EH Implementation Procedure

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.

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 zero bits in the obstacle-flag, while sequential job-scheduling also requires the previous sub-volumes to be combined with the result.

Results. When using the Dynamic Job-Scheduling a 65 percent reduction in computational costs was achieved for

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-Volumes	100 (regular grid)

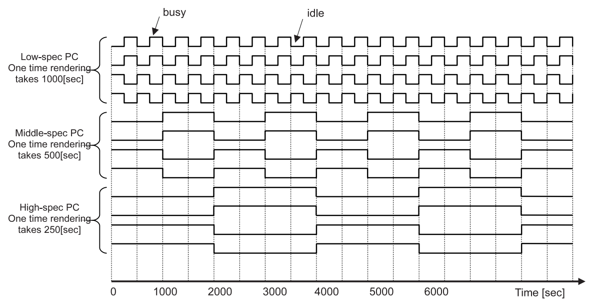


Figure 9. Agent Usage Schedule

Table 2. Simulation Results

	Dynamic Job-Scheduling	Sequential Job-Scheduling
Elapsed Time [sec.]	10,000	15,500
Agent Utilization [%]	54.2	38.4

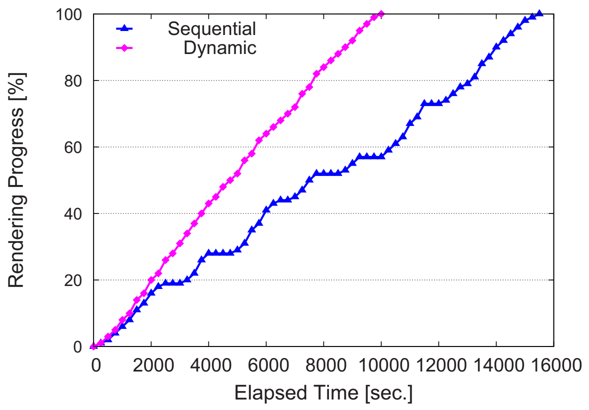


Figure 10. Rendering Progress

Table 3. Experimentation Environment

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

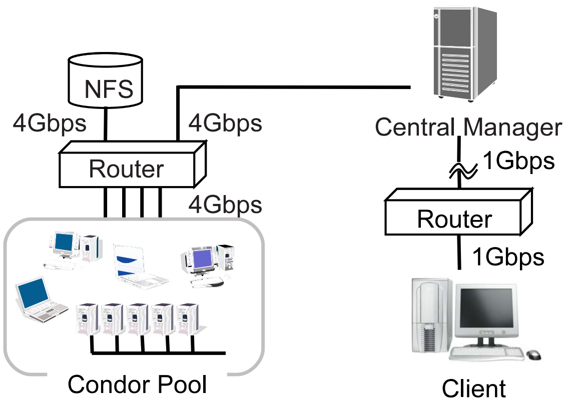


Figure 11. Network Configuration

Table 4. Test Data Parameters

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

VD : Volume Data, SV : Sub-Volume

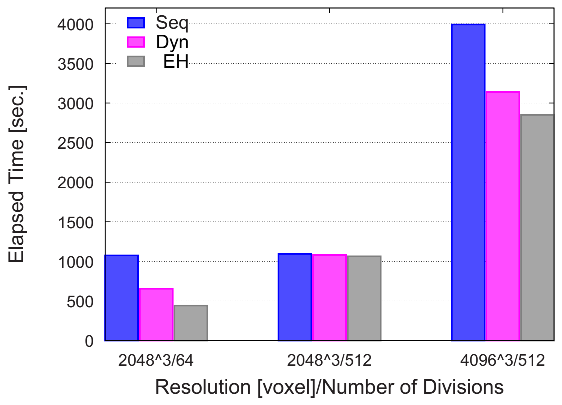


Figure 12. Elapsed Times

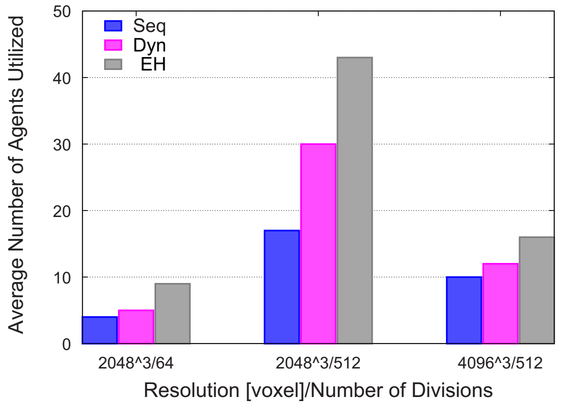


Figure 13. Average Number of Agents Utilized

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 more steeper rendering progress compared to sequential job-scheduling.

5.2. Verification using Test Data

We performed experiments, using our university's campus grid [18]. 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 sequentially. • •Figure 11 shows the campus grid network diagram. Between agents, the central manager, and NFS the network speed is 4Gbps, while manager to client is 1 Gbps.
Table 4 shows three sets of parameters of the test data used.

Results. Figure 12 shows the elapsed times and Figure 13 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, the vertical axis is elapsed time. The shorter the elapsed time, the better the job-scheduling. In Figure 13, 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 only in 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).