Parallel Volume Rendering with Early Ray Termination for Visualizing Large-Scale Datasets

Manabu Matsui, Fumihiko Ino, and Kenichi Hagihara

Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
m-matui@ist.osaka-u.ac.jp

Abstract. This paper presents an efficient parallel algorithm for volume rendering of large-scale datasets. Our algorithm focuses on an optimization technique, namely early ray termination (ERT), which aims to reduce the amount of computation by avoiding enumeration of invisible voxels in the visualizing volume. The novelty of the algorithm is that it incorporates this technique into a distributed volume rendering system with global reduction of the computational amount. The algorithm also is capable of statically balancing the processor workloads. The experimental results show that our algorithm with global ERT further achieves the maximum reduction of 33% compared to an earlier algorithm with local ERT. As a result, our load-balanced algorithm reduces the execution time to at least 66%, not only for dense objects but also for transparent objects.

1 Introduction

Direct volume rendering [1] is a technique for displaying three-dimensional (3-D) volumetric scalar data as a two-dimensional (2-D) image. Typically, the data values in the volume are made visible by mapping them to color and opacity values, which are then accumulated to determine the image pixel.

One challenging issue in volume rendering is to realize fast rendering for large-scale datasets. However, finding a solution to this issue is not easy due to the high time and space complexities of volume rendering, both represented as $O(n^3)$ for an $n \times n \times n$ voxel volume. Therefore, fast processors with large memories are necessary to carry out this compute-intensive rendering with in-core processing.

To address this issue, many acceleration techniques have been proposed in the past. Levoy [2] proposes two optimization techniques that reduce the time complexity of volume rendering. The first technique is early ray termination (ERT), which adaptively terminates accumulating color and opacity values in order to avoid useless ray casting. The second technique is a hierarchical octree data structure [3], which encodes spatial coherence in object space in order to skip empty regions of the volume. These techniques reduce the execution time by roughly a factor of between 5 and 11. Nukata et al. [4] present a cuboid-order rendering algorithm that aims to maximize the cache hit ratio by dividing the volume into cuboids, which are then rendered successively. This algorithm enables view-independent fast volume rendering on a single CPU computer.

Another promising approach is parallelization on parallel computers. Hsu [5] proposes the segmented ray casting (SRC) algorithm, which parallelizes volume rendering on a distributed memory parallel computer. This algorithm distributes the volume by using a block-block decomposition and carries out data-parallel processing to generate subimages for each decomposed portion. The subimages are then merged into a final image by using an image compositing algorithm [6, 7].

Though many earlier projects propose a wide variety of acceleration schemes, data distributed parallel schemes [5–7] are essential to render large-scale datasets that cause out-of-core rendering on a single CPU computer. One problem in these schemes is that the increase of computational amount compared to sequential schemes, where the amount can easily be reduced by means of ERT. This increase is due to the low affinity between ERT and data distribution. That is, while data distribution makes processors independently render the volume data, ERT is based on the visibility of the data, determined in a front-to-back order. Therefore, earlier parallel algorithms independently apply ERT to each distributed data in order to perform data-parallel processing. This locally applied ERT, namely local ERT, increases the computational amount compared to global ERT, because the visibility is locally determined, so that processors render locally visible but globally invisible voxels. Thus, earlier data distributed schemes are lacking the capability of global ERT, so that these schemes can suffer in low efficiency especially for large-scale datasets with many transparent objects.

Gao et al. [8] address this issue by statically computing the visibility of the volume data. However, their static visibility culling approach requires pre-processing for every viewing direction, so that its pre-processing stage prevents rapid visualization.

The key contribution of this paper is the development of a parallel algorithm that dynamically realizes global ERT in a distributed volume rendering system. Our algorithm has the following two advantages.

R1: Reduction of the memory usage per processor by data distribution.
R2: Reduction of the computational amount by global ERT without pre-processing.

To realize R1, our algorithm employs a block-cyclic decomposition that is capable of statically balancing the processor workloads. To realize R2, the algorithm employs an efficient mechanism for sharing the visibility information among processors.

The remainder of the paper is organized as follows. Section 2 introduces earlier algorithms and presents the problem we tackled in this work. Section 3 describes the details of our algorithm while Section 4 presents some experimental results on a cluster of 64 PCs. Finally, Section 5 concludes the paper.

2 Volume Rendering

Figure 1(a) shows an overview of the ray casting algorithm [1]. This algorithm produces an image by casting rays from the viewpoint through the screen into the viewing volume. The image pixel $I_{s,t}$ on point (s,t) is determined by accumulating color and opacities values of penetrated voxels $V_1, V_2, \ldots, V_k$: $I_{s,t} = \sum_{i=1}^{k} \alpha(V_i)c(V_i)\prod_{j=0}^{i-1}(1 - \alpha(V_j))$, where $C(V_i)$ and $\alpha(V_i)$ are the color and opacity values of the i-th penetrated voxel V_i, respectively; $0 \le \alpha(V_i) \le 1$; and $\alpha(V_0) = 0$. Computing $I_{s,t}$ for all points (s,t) on the screen, where $1 \le s \le n_s$ and $1 \le t \le n_t$, generates the final image of size $n_s \times n_t$. In the following discussion, let $a_{s,t}(i)$ be the accumulated transparency for voxel V_i, where $a_{s,t}(i) = \prod_{j=0}^{i-1}(1 - \alpha(V_j))$.

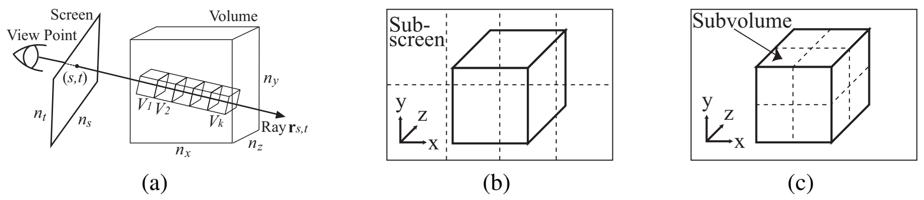

Fig. 1. Ray casting and its parallel schemes. (b) Screen-parallel rendering and (c) object-parallel rendering parallelize (a) ray casting by exploiting the parallelism in screen space and in object space, respectively.

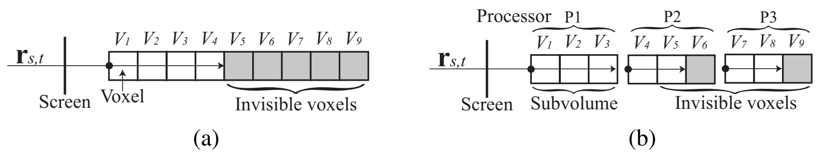

Fig. 2. Early ray termination (ERT). (a) Global ERT for sequential and screen-parallel rendering, and (b) local ERT for object-parallel rendering. While global ERT terminates the ray immediately before invisible voxel V_5, local ERT fails to avoid accumulating locally visible but globally invisible voxels: V_5, V_7, and V_8. Voxels V_6 and V_9 are invisible locally as well as globally.

ERT reduces the computational amount by avoiding accumulation of color and opacity values that do not have influence on the final image. That is, ERT avoids enumerating voxels $V_l, V_{l+1}, \ldots, V_k$ if $a_{s,t}(l) = 0$.

Earlier parallel schemes can be classified into two groups: screen-parallel and object-parallel rendering as illustrated in Figure 1.

Screen-parallel rendering exploits the parallelism in screen space. In this scheme, the screen is divided into p subscreens, where p represents the number of processors, and tasks associated with each subscreen are assigned to processors. Because each processor takes responsibility for the entire of a ray as it does in sequential schemes, ERT can easily be applied to this scheme, as illustrated in Figure 2(a). Furthermore, by assigning the tasks in a cyclic manner, this scheme statically balances the processing workloads. However, it requires large main memory to provide fast rendering for any given viewpoint, because every processor need to load the entire volume into memory. Thus, though screen-parallel rendering is a good scheme for small datasets, which require no data decomposition, it does not suit for large-scale datasets.

In contrast, object-parallel rendering exploits the parallelism in object space. This scheme divides the volume into p subvolumes, and then assigns tasks associated with each subvolume to processors. Parallel rendering of each subvolume generates p distributed subimages, so that image compositing is required to merge subimages into the final image. Thus, this scheme allows us to distribute subvolumes to processors, so that is suitable for large-scale datasets. However, because accumulation tasks of a ray can be assigned to more than one processor, it is not easy to utilize global ERT in this scheme.

Figure 2(b) shows an example of local ERT in object-parallel rendering. In this example, voxels from V_1 to V_4 are visible from the viewpoint while voxels from V_5 to V_9 are invisible. These voxels are assigned to three processors, so that each processor takes responsibility for three of the nine voxels. In object-parallel rendering, the reduction given by ERT is localized in each processor, because processors take account of the local visibility instead of the global visibility. For example, processor P2 fails to identify V_5 as an invisible voxel, because it accumulates opacity values from its responsible V_4 in order to perform data-parallel processing. Furthermore, although P2 terminates the ray after V_5, its back neighborhood P3 is unaware of this termination, so that accumulates V_7 and V_8, according to the local visibility.

3 Data Distributed Algorithm with Early Ray Termination

Our algorithm is based on object-parallel rendering to deal with large-scale datasets. It integrates the following techniques: (1) Data distribution by a block-cyclic decomposition; (2) Concurrent processing of volume rendering and image compositing; (3) Visibility sharing by a master/slave paradigm; (4) Parallel image compositing.

3.1 Data Distribution

Our algorithm distributes the volume data according to a block-cyclic decomposition, aiming to maximize the parallelism that can be decreased due to global ERT. The following discussion describes why we employ this decomposition.

In order to realize global ERT in object-parallel rendering, processors have to share the visibility information, namely accumulated transparency. For example, as illustrated in Figure 2(b), in a case where processors P2 and P3 are responsible for neighborhood voxels and P2 terminates the ray, P3 can avoid accumulating all of its responsible voxels V_7, V_8, and V_9 after it obtains the value of accumulated transparency that P2 has computed for V_6. However, this indicates that casting a ray with global ERT has no parallelism in the viewing direction, because P3 has to wait for P2 to complete rendering of its responsible voxels. Thus, applying global ERT to object-parallel rendering decreases the entire parallelism in object space due to processor synchronization. Note here that the parallelism in screen space is remained.

The key idea to address this decreased parallelism is that exploiting parallelism in vertical planes perpendicular to the viewing direction. That is, we employ (A) a data decomposition that allows every processor to have equal-sized tasks on any cross sections of the volume. Such decomposition minimizes the overhead for the processor synchronization by allowing processors to overlap communication with computation. For example, processor P3 in Figure 2(b) can perform rendering for other rays during waiting for P2, because any processor has its responsible tasks on any vertical plane perpendicular to the viewing direction. Furthermore, this decomposition realizes static load balancing because tasks on any cross sections are assigned equally to processors.

As the results of the above considerations, our algorithm employs a block-cyclic decomposition. Note here that a cyclic decomposition is more appropriate than this decomposition in terms of (A). However, it possibly decreases rendering performance due to frequent communication among processors, because a task in the cyclic decomposition corresponds to a voxel, so that communication occurs for each voxel. Therefore, we use a combination of block and cyclic decompositions in order to have coarse-grained tasks without losing the nature of load balancing.

In addition to this data distribution technique, our algorithm aims to reduce the time complexity by encoding empty regions as Levoy does in [2]. We use an adaptive block decomposition rather than Levoy's hierarchical octree, because it increases traversing overheads with the degree of its hierarchy [9]. The adaptive block decomposition addresses this issue by uniformly space partitioning.

3.2 Concurrent Processing of Volume Rendering and Image Compositing

As mentioned before, ERT aims to efficiently render the volume according to the visibility. Note here that the visibility in object-parallel rendering is determined by compositing of subimages. Therefore, applying global ERT to object-parallel rendering requires concurrent processing of volume rendering and image compositing. Furthermore, to obtain better performance, (B) a rapid read/write access to the visibility information, namely the accumulated transparency for an arbitrary ray, must be provided.

In order to realize (B), we classify processors into two groups as follows: (1) Rendering Processors (RPs), defined as processors that render subvolumes in order to generate subimages; (2) Compositing Processors (CPs), defined as processors that composite subimages and manage accumulated transparency $a_{s,t}$ for all rays, where $a_{s,t}$ denotes the accumulated transparency for ray $\mathbf{r}_{s,t}$. Note here that the relation between CPs and RPs is similar to that between masters and slaves in the master/slave paradigm. The details of CPs and RPs are presented later.

3.3 Visibility Sharing Mechanism

Figure 3 shows the processing flow of our master/slave based algorithm. Let r, c, and v be the number of RPs, CPs, and subvolumes, respectively. The processing flows for RPs and CPs consist of the following phases.

Processing flow for RPs:

1. Data distribution. The volume is divided into at least r subvolumes, which are then distributed to r RPs in a round-robin manner. Data distribution phase occurs only at the beginning of the system.
2. Rendering order determination. Each RP determines the rendering order of assigned subvolumes by constructing a list of subvolumes, L, in which its responsible v/r subvolumes are sorted by the distance to the screen in an ascending order. This ascending order is essential to achieve further reduction by means of ERT. List L is updated every time the viewpoint moves.
3. Accumulated transparency acquisition. Each RP deletes a subvolume from the head of L, then obtains accumulated transparencies from CPs, for all rays that penetrate the subvolume. Let T be a set of accumulated transparencies obtained from CPs.

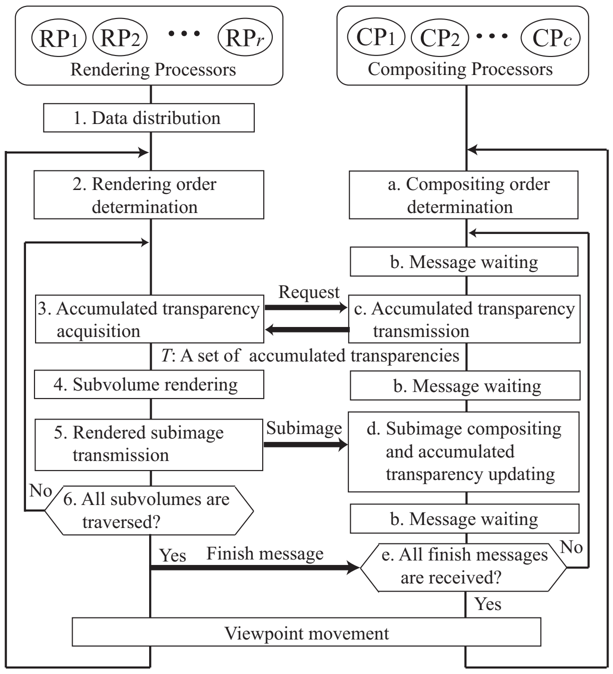
RP1
RP2
RPr
Rendering Processors
CP1
CP2
CPc
Compositing Processors
1. Data distribution
2. Rendering order determination
a. Compositing order determination
b. Message waiting
Request
3. Accumulated transparency acquisition
c. Accumulated transparency transmission
T: A set of accumulated transparencies
4. Subvolume rendering
b. Message waiting
Subimage
5. Rendered subimage transmission
d. Subimage compositing and accumulated transparency updating
No
6. All subvolumes are traversed?
b. Message waiting
Yes
Finish message
e. All finish messages are received?
No
Yes
Viewpoint movement

Fig. 3. Processing flow of proposed algorithm.

4. Subvolume rendering. For all rays $\mathbf{r}_{s,t}$ such that $a_{s,t} \in T$ and $a_{s,t} > 0$, each RP accumulates the voxels penetrated by $\mathbf{r}_{s,t}$ so that generates a subimage. Note here that the algorithm avoids rendering for all rays $\mathbf{r}_{s,t}$ such that $a_{s,t} \in T$ and $a_{s,t} = 0$, according to ERT.
5. Rendered subimage transmission. Each RP transmits the rendered subimages to CPs. Because blank pixels have no influence on the final image, the algorithm transmits only pixels inside the bounding rectangle of the subimages in order to reduce the amount of communication.
6. Completion check. Phases 3., 4., and 5. are repeated until list L becomes empty. Empty L indicates that the RP completes performing all assigned tasks for the current viewpoint, so that it sends a finish message to all CPs.

Processing flow for CPs:

a. Compositing order determination. Each CP determines the compositing order of subimages by constructing a list of subvolumes, M, in which all v subvolumes are sorted by the distance to the screen in an ascending order.
b. Message waiting. Each CP waits for incoming messages from RPs. Such messages contain request messages for accumulated transparency acquisition, data messages including rendered subimages, and finish messages.
c. Accumulated transparency transmission. Each CP transmits T that RPs require.

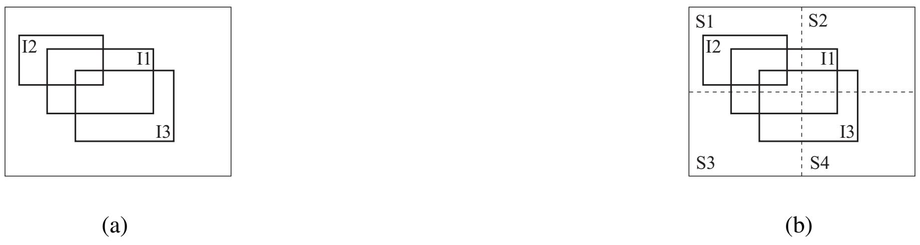

Fig. 4. Screen-parallel compositing. Dividing the screen into subscreens produces more parallelism of image compositing.

d. Subimage compositing and accumulated transparency updating. Each CP updates T by compositing its local subimages with received subimages, according to the order of list M. If keeping this order is impossible due to the lack of the still unrendered subimages, it stores the received subimages into a local buffer for later compositing.
e. Completion check. Phases b., c. and d. are repeated until receiving finish messages from all RPs.

3.4 Parallel Processing of Image Compositing

In the master/slave paradigm, the master becomes a performance bottleneck if it is assigned many slaves beyond its capacity. Therefore, our algorithm parallelizes the master's tasks by exploiting the parallelism in screen space. That is, as screen-parallel rendering does, it divides the screen into at least c subscreens and assigns them to c CPs. Let s be the number of subscreens.

In addition to the benefits of acceleration, this screen-parallel compositing increases the parallelism of image compositing. Figure 4 gives an example of this increased parallelism. In this example, subimages I1, I2, and I3 are rendered from neighborhood subvolumes located in a front-to-back order. As shown in Figure 4(a), compositing I1 and I3 requires rendered I2 if we avoid dividing the screen. In this case, CPs have to wait for RPs to generate I2 before compositing I1 and I3. In contrast, if the screen is divided into subscreens, rendered I2 is unnecessary to perform compositing in subscreens S2, S3, and S4. Therefore, compositing in these subscreens can be carried out without waiting the rendering of I2, so that screen-parallel compositing enables compositing subimages at shorter intervals. This means that accumulated transparencies are updated at shorter intervals, which contribute to achieve further reduction by ERT.

Thus, dividing the screen allows us to exploit more parallelism of image compositing. Furthermore, it also contributes to realize (B) because it enables more frequent updating of accumulated transparencies.

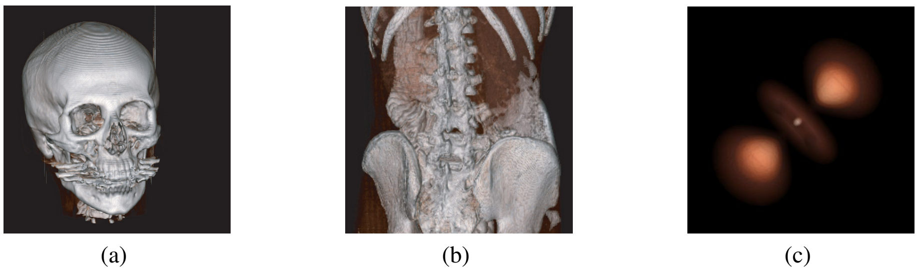

Fig. 5. Rendering results of volume datasets used in experiments. (a) D1: skull volume of size $512 \times 512 \times 448$, (b) D2: abdomen volume of size $512 \times 512 \times 730$, and (c) D3: hydrogen atom volume of size $512 \times 512 \times 512$. Each volume is rendered on a 512×512 pixel screen.

4 Experimental Results

In order to evaluate the performance of our algorithm, we compare it with two earlier algorithms: SRC [5] and SRC with load balancing (SRCLB). We also present a guideline for obtaining the appropriate values for the four parameters: r, c, v, and s.

The SRC algorithm is an object-parallel algorithm that parallelizes the ray casting algorithm with a block-block decomposition. On the other hand, the SRCLB algorithm incorporates a load balancing capability into SRC. To balance processing workloads, it divides the volume into subvolumes with marginal regions. For every viewpoint, it adaptively varies the size of responsible regions inside the subvolumes, according to the execution time measured for the last viewpoint. Therefore, SRCLB requires more physical memory compared to the remaining two algorithms, which use disjoint decompositions. However, it requires no data redistribution during volume rendering. Both the SRC and SRCLB algorithms use an improved binary-swap compositing (BSC) [7] for image compositing. Furthermore, ERT is locally applied to them. All the three algorithms use an adaptive block decomposition [9] to skip empty regions in the volume.

We have implemented the three algorithms by using the C++ language and MPICH-SCore library [10], a fast implementation of the Message Passing Interface (MPI) standard. We used a Linux cluster of 64 PCs for the experiments. Each node in this cluster has two Pentium III 1-GHz processors and 2 GB of main memory, and connects to a Myrinet switch, which provides a link bandwidth of 2 GB/s.

Figure 5 shows images rendered for three employed datasets D1, D2, and D3. In addition, we also used a large-scale dataset D4, skull-big volume of size $1024 \times 1024 \times 896$, generated by trilinear interpolation of D1. The screen sizes are 512×512 pixel for D1, D2, and D3, and 1024×1024 pixel for D4.

4.1 Performance Comparison to Earlier Algorithms

To measure rendering performance, we rotated the viewpoint around the viewing objects, so that obtained an average of 24 measured values. Table 1 shows the averaged

Table 1. Measured execution time and number of rendered voxels for proposed, SRC, and SR-CLB algorithms. On less than eight processors, some execution failed due to the lack of physical memory. N_s represents the number of voxels rendered by a sequential algorithm with ERT.

p	D1: skull volume $N_s = 3.1 \cdot 10^6$ voxels								D2: abdomen volume $N_s = 18.2 \cdot 10^6$ voxels							
	SRC		SRCLB		Proposed		Reduction ratio		SRC		SRCLB		Proposed		Reduction ratio	
	T_1	N_1	T_2	N_2	T_3	N_3	R_T	R_N	T_1	N_1	T_2	N_2	T_3	N_3	R_T	R_N
4	2134	6.7	—	—	2366	4.4	0.90	1.52	6155	23.7	—	—	6497	20.9	0.95	1.13
8	1471	8.3	1185	8.0	1283	6.3	1.15	1.32	3719	25.9	—	—	3316	23.8	1.12	1.09
16	1177	9.9	773	10.2	691	6.3	1.70	1.57	2684	27.9	2350	28.0	1761	23.7	1.52	1.18
32	920	11.4	480	12.3	438	6.8	2.10	1.68	1903	29.6	1432	29.9	934	24.7	2.04	1.20
64	682	12.7	302	14.0	279	8.3	2.44	1.53	1334	31.0	859	31.5	541	25.5	2.47	1.22
128	435	15.0	240	12.6	207	11.0	2.10	1.36	780	32.6	536	27.8	373	27.3	2.09	1.19

p	D3: hydrogen atom volume $N_s = 7.6 \cdot 10^6$ voxels								D4: skull-big volume N_s is unmeasurable							
	SRC		SRCLB		Proposed		Reduction ratio		SRC		SRCLB		Proposed		Reduction ratio	
	T_1	N_1	T_2	N_2	T_3	N_3	R_T	R_N	T_1	N_1	T_2	N_2	T_3	N_3	R_T	R_N
4	2168	8.1	—	—	2759	7.3	0.79	1.11	—	—	—	—	—	—	—	—
8	1201	8.3	—	—	1725	7.5	0.70	1.11	7146	40.8	—	—	5836	26.7	1.22	1.53
16	888	8.3	777	8.3	872	7.6	1.02	1.09	5386	49.2	4736	49.5	3328	31.5	1.62	1.56
32	673	8.3	479	8.4	499	7.8	1.35	1.06	4144	57.2	3172	59.4	2178	36.3	1.90	1.58
64	468	8.4	302	8.6	301	8.1	1.55	1.04	3152	64.8	2365	68.5	1311	41.0	2.40	1.58
128	325	8.5	187	7.5	226	8.2	1.44	1.04	2132	78.8	1640	86.0	927	50.2	2.30	1.57

$R_T = T_1/T_3, R_N = N_1/N_3$

Table 2. Parameter values employed for p processors. Parameters r, c, v, and s represent the number of RPs, that of CPs, that of volume divisions, and that of screen divisions, respectively.

p	D1: skull volume of size $512 \times 512 \times 448$			D2: abdomen volume of size $512 \times 512 \times 730$			D3: hydrogen atom volume of size $512 \times 512 \times 512$			D4: skull-big volume of size $1024 \times 1024 \times 896$		
	r	v	s	r	v	s	r	v	s	r	v	s
4	3	$2 \times 2 \times 2$	23×23	3	$3 \times 3 \times 3$	24×24	3	$6 \times 6 \times 6$	24×24	—	—	—
8	7	$2 \times 2 \times 2$	22×22	7	$8 \times 8 \times 8$	24×24	7	$6 \times 6 \times 6$	18×18	7	$4 \times 4 \times 4$	24×24
16	14	$4 \times 4 \times 4$	24×24	15	$8 \times 8 \times 8$	23×23	14	$6 \times 6 \times 6$	19×19	15	$5 \times 5 \times 5$	23×23
32	27	$7 \times 7 \times 7$	22×22	29	$10 \times 10 \times 10$	20×20	27	$8 \times 8 \times 8$	16×16	27	$7 \times 7 \times 7$	23×23
64	55	$8 \times 8 \times 8$	18×18	59	$9 \times 9 \times 9$	16×16	50	$8 \times 8 \times 8$	10×10	55	$8 \times 8 \times 8$	24×24
128	109	$8 \times 8 \times 8$	14×14	111	$12 \times 12 \times 12$	11×11	106	$8 \times 8 \times 8$	14×14	109	$11 \times 11 \times 11$	20×20

results. In this table, T_1, T_2, and T_3 represent the averaged execution time for SRC, SRCLB, and our algorithms, respectively. N_1, N_2, and N_3 also represent the averaged number of rendered voxels. We measured them by using the best parameter values determined by the guideline presented in the next section (see Table 2). The length for marginal region of SRCLB is given by 256 voxels, which is the maximum length for performing in-core rendering on our cluster.

This table indicates that our algorithm is generally faster than SRC, because it shows $R_T > 1.0$ for all $p > 8$, where R_T is the reduction ratio of the execution time compared to SRC. In particular, on a larger number of processors, our algorithm reduces the execution time in half for datasets D1, D2, and D4.

In contrast, the reduction ratio is relatively small on a smaller number of processors. This small improvement can be explained as follows. The first reason is that RPs in our classification based algorithm is c fewer than that in the remaining two algorithms. This indicates that processor classification is not suited for systems with smaller p, because such systems do not have computing resources enough to deal with compute-intensive rendering. In such small systems, any resource must be dedicated to the performance bottleneck, namely subvolume rendering, in order to achieve faster acceleration. Actually, as presented in Table 2, we obtain $c = 1$ for all $p \leq 8$, so that the number of RPs is insufficient to that of CPs in these situations. The second reason is that tasks associated with the same ray are assigned to a few processors in the SRC algorithm. This indicates that on a smaller number of processors, local ERT is sufficient to terminate rays in an early rendering phase, so that there is no redundant accumulation left for global ERT.

By comparing R_T and R_N, where R_N is the reduction ratio of rendered voxels compared to SRC, we can see that the reduction of the execution time is more than that of rendered voxels. In particular, although ERT achieves few reduction for transparent dataset D3, our algorithm reduces its execution time by 33%. This acceleration is provided by load balancing. As shown in Figure 5(c), the most voxel in dataset D3 is transparent and few opaque voxel is located around the center of the volume. For such datasets, the SRC algorithm constructs smaller blocks located near the center and larger blocks located far from the center, because it uses a combination of an adaptive block decomposition and block-block decomposition. Therefore, processors assigned with center blocks have relatively larger tasks than others, so that the processor workloads become imbalanced. Actually, the average and standard deviation of subvolume rendering time on 128 processors, μ and σ, respectively, are improved from $\mu = 91$ and $\sigma = 88$ ms in the SRC method to $\mu = 105$ and $\sigma = 34$ ms in our algorithm.

Finally, we compare our algorithm to SRCLB by using D1 and its larger version D4. Our algorithm shows better improvement to SRCLB for D4 rather than for D1. On 128 processors, its reduction ratio to SRCLB is $T_2/T_3 = 1.77$ for D4 but 1.16 for D1. This is due to the lack of physical memory required for marginal regions. That is, although we maximized the length for marginal regions, it was not enough to balance the workloads for D4. Thus, compared to our algorithm, SRCLB requires a larger physical memory to balance the workloads for large-scale datasets.

4.2 Parameter Setup

We now present a guideline for obtaining the appropriate values for the four parameters: r, c, v, and s. Figure 6 shows the execution time averaged over RPs for different parameter values. We show the results only for skull dataset D1 because we obtained similar results for others.

We first investigate the influence of r under fixed v and s. Figure 6(a) shows the breakdown of the execution time on $p = 64$ for different r. The time for subvolume rendering decreases as r increases, so that we obtain the shortest time when $r = 55$. This decrease is due to the reduction of computational amount per processor, because each RP is responsible for v/r subvolumes, which decreases with the increase of r. On the other hand, when $r > 55$, the execution time turns to increase because the communication time for accumulated transparency acquisition increases with r. This increase is due to the lack of CPs, which makes RPs wait in the accumulated transparency acquisition phase. Thus, there is a tradeoff between r and c.

The appropriate values for r and c are determined by finding a balancing point as follows. Given a volume of size $n \times n \times n$ and a screen of $n \times n$, the time complexities of volume rendering and image compositing are $O(n^3)$ and $O(n^2)$, respectively.

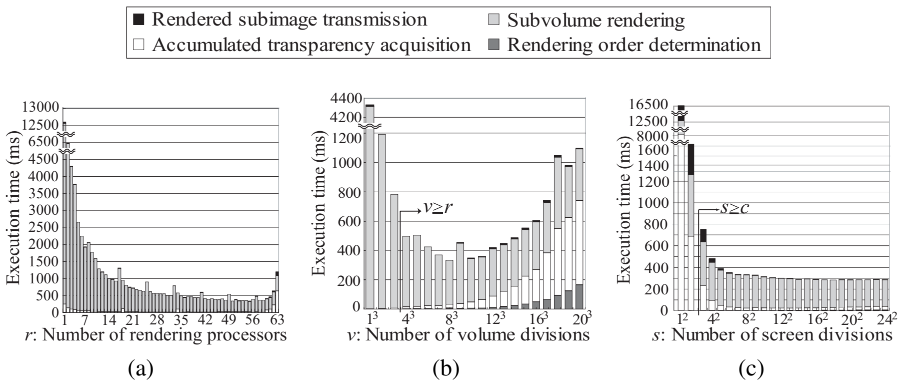

Fig. 6. Execution time measured on RPs for different parameter values using skull dataset D1. Results for (a) $v = 8 \times 8 \times 8$ and $s = 18 \times 18$, (b) $r = 55$ and $s = 18 \times 18$, and (c) $r = 55$ and $v = 8 \times 8 \times 8$.

Therefore, if we assume that the appropriate values for r and c balance the workloads of them, r and c are given by:

$$r/c = (w_1 + w_2)/w_2 \cdot n^3/n^2, \tag{1}$$

$$p = r + c, \tag{2}$$

where $(w_1 + w_2)/w_2$ is a granularity of rendering to compositing, w_1 is the time for trilinear interpolation to determine a scalar value of a voxel, and w_2 is the time for accumulating color and opacity values of a image pixel.

Next, we investigate v (Figure 6(b)). The increase of v means the downsize of task granularity. In addition, the block-cyclic decomposition becomes similar to the cyclic decomposition with the increase of v. Therefore, increasing v leads to better load balancing, which minimizes the execution time for subvolume rendering. However, fine-grained tasks cause frequent communication between RPs and CPs, because they shorten the intervals of accumulated transparency acquisition and updating. Although shorter intervals contribute to achieve further reduction by ERT, but CPs can suffer from network contention when the intervals are too short beyond the network capacity. Therefore, the execution time turns to increase in such a situation. Thus, there is a tradeoff between load balancing and communication frequency.

As same as for r and c, the appropriate value for v also is determined by finding a balancing point. In Table 2, we can see that the best value of v differs among datasets. Therefore, the appropriate value must be determined for each p and dataset by finding the saturated value of at least r. For example, by increasing v from r, we can identify the saturated value when the execution time turns from decrease to increase.

Finally, we investigate s (Figure 6(c)). When $s < c$, the parallelism of image compositing increases with s, so that increasing s reduces the execution time. However, there is no significant difference when $s \geq c$. Furthermore, Table 2 shows similar values of s for datasets D1, D2, and D3. Therefore, the appropriate value for s depends on each p and n. The value can be determined by finding the saturation point with increasing s from c.

In summary, Equations (1) and (2) determine the appropriate values for r and c, which dominate the execution time. The appropriate value for v can be determined by using the tradeoff between load balancing and communication time. The value is given by finding the saturated value of at least r for each p and dataset. The last parameter s has relatively small significance on the execution time. The appropriate value can be determined by finding the saturated value of at least c for each p and n.

5 Conclusions

We have presented an efficient parallel volume rendering algorithm that is capable of rendering large-scale datasets on a distributed rendering system. The novelty of the algorithm is a combination of global ERT and data distribution with static load balancing. To realize this, the algorithm uses the master/slave paradigm where slave processors carry out the rendering tasks to accumulate color and opacity values of voxels while master processors perform compositing tasks and manage the accumulated values to share them among processors. The experimental results show that the reduction given by ERT increases with the size of datasets, and the improvement produced by static load balancing increases with the number of processors.

References

1. Levoy, M.: Display of surfaces from volume data. IEEE Computer Graphics and Applications **8** (1988) 29–37
2. Levoy, M.: Efficient ray tracing of volume data. ACM Trans. Graphics **9** (1990) 245–261
3. Yau, M.M., Srihari, S.N.: A hierarchical data structure for multidimensional digital images. Comm. ACM **26** (1983) 504–515
4. Nukata, M., Konishi, M., Goshima, M., Nakashima, Y., Tomita, S.: A volume rendering algorithm for maximum spatial locality of reference. IPSJ Trans. Advanced Computing Systems **44** (2003) 137–146 (In Japanese).
5. Hsu, W.M.: Segmented ray casting for data parallel volume rendering. In: Proc. 1st Parallel Rendering Symp. (PRS'93). (1993) 7–14
6. Ma, K.L., Painter, J.S., Hansen, C.D., Krogh, M.F.: Parallel volume rendering using binary-swap compositing. IEEE Computer Graphics and Applications **14** (1994) 59–68
7. Takeuchi, A., Ino, F., Hagihara, K.: An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. Parallel Computing **29** (2003) 1745–1762
8. Gao, J., Huang, J., Shen, H.W., Kohl, J.A.: Visibility culling using plenoptic opacity functions for large volume visualization. In: Proc. IEEE VIS'03. (2003) 341–348
9. Lee, C.H., Park, K.H.: Fast volume rendering using adaptive block subdivision. In: Proc. 5th Pacific Conf. Computer Graphics and Applications (PG'97). (1997) 148–158
10. O'Carroll, F., Tezuka, H., Hori, A., Ishikawa, Y.: The design and implementation of zero copy MPI using commodity hardware with a high performance network. In: Proc. ACM ICS'98. (1998) 243–250