A Divided-Screenwise Hierarchical Compositing for Sort-Last Parallel Volume Rendering

Fumihiko Ino[1] Tomomitsu Sasaki[2*] Akira Takeuchi[2] Kenichi Hagihara[1]

[1] Graduate School of Information Science and Technology, Osaka University
[2] Graduate School of Engineering Science, Osaka University
1–3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: ino@ist.osaka-u.ac.jp

Abstract

In this work, to render at least 512^3 voxel volumes in real-time, we have developed a sort-last parallel volume rendering method for distributed memory multiprocessors. Our sort-last method consists of two methods, Hsu's segmented ray casting and our divided-screenwise hierarchical (DSH) compositing, in which each processor produces a subimage and merges all the produced subimages into the final image. This paper describes the DSH method, which aims at achieving high performance compositing on a large number of processors. Our implementation on a 64-node PC cluster can composite a 512^2 pixel image about twice as fast as an existing method, the binary-swap method, so that can render a 512x512x224 voxel volume at approximately eight frames per second (fps).

1. Introduction

Direct volume rendering [6, 21] is a useful technique to visualize the internal parts of the patient's body, which are usually invisible. For example, visualizing the invisible parts helps medical doctors in diagnosing patients and planning surgical strategies [3, 17]. To observe such invisible objects, X-ray computed tomography (CT) scans create three-dimensional (3D) volumes by scanning the objects, and volume rendering methods produce two-dimensional (2D) images by mapping the volumes onto the screen, or the image plane.

In general, medical diagnoses require both interactive and high-resolution rendering: rendering of 512^3 voxel volumes at ten frames per second (fps). To achieve high-speed rendering, we require sufficient computing resources such as fast processors to perform the compute-intensive rendering and large memories to store the entire 3D volume.

One way to meet these requirements is to parallelize serial volume rendering methods on distributed memory multiprocessors. Many researchers have proposed parallel methods [2, 4, 7, 8, 12, 13, 16, 18, 20, 22, 23]. In particular, some recent works [12, 22] have challenged to real-time rendering of at least 512^3 voxel volumes.

The above previous methods are classified as *sort-last* methods [11], which partition the volume into subvolumes and distribute the subvolumes to each processor. The sort-last method consists of two phases: (1) the rendering phase and (2) the compositing phase. In the rendering phase, each processor independently produces a subimage by rendering its own assigned subvolume. Thus n-subimages have been produced after the rendering phase, where n is the number of processors. In the subsequent compositing phase, the processors produce the final image by merging n-subimages in a back-to-front [21] or front-to-back [6] order.

The performance of the rendering phase can scale with n, because every processor performs $1/n$ of the entire rendering task and no communication occurs in this phase. For example, *segmented ray casting* (SRC) [4], which exploits data parallelism in *ray casting* (RC) [6], renders efficiently on distributed memory multiprocessors.

On the other hand, the compositing task grows with n. Furthermore, processors have to communicate each other to produce the final image. Therefore, to develop an efficient sort-last method that gives a linear speedup with n, we also have to develop an efficient compositing method that provides high performance compositing on a large n.

In [8], Ma et al. have introduced four existing compositing methods: *binary tree* (BT), *binary swap* (BS) [8], *direct send* (DS) [4, 13], and *projection* (PJ) [2] methods.

The BT method, which completes the image compositing with $\log n$ stages, is a simple method and suites to hardware implementations [12]. At the first stage, all processors are paired up, and one of a pair sends its own subimage to the other. Half the processors that received a subimage composite the subimage and proceed to the succeeding stage. Repeating the above processes on all $\log n$ stages produces the final image. Since the number of idle processors increases at every stage, this method is inefficient for software implementations.

Next, the BS method is an improvement on the BT method. Each pair of processors splits its own subimage in two pieces and exchanges one of the two pieces. At the k-

*Presently with Toyota Motor Corporation.

th stage, every processor composites $1/2^k$ of the subimage, so that no processor becomes idle during the compositing phase. The BS method completes the image compositing with $\log n$ stages and transmits at most $2.43n^{1/3}p$ pixels, where p is the number of pixels in the final image. Many sort-last methods [10, 16, 22, 23] use this method.

The DS method divides the screen into non-overlapping regions and statically assigns each compositing region to a processor. This method transmits approximately $n^{1/3}p\,(1-1/n)$ pixels, the least among the four listed above. However, it takes $n-1$ stages at the worst case.

Last, the PJ method composites by propagating the subimages through processors in a front-to-back order. On average, this method transmits $O(n^{1/3}p)$ pixels and takes $O(n^{1/3})$ stages. The PJ method is behind the above methods in these performance metrics but completes the image compositing sequentially from the processor located in the front. This allows us to process different viewing angles at the same time like the parallel pipeline method [5, 18].

In this work, to render high-resolution volumes in real-time, we have developed a sort-last parallel volume rendering method that provides high-speed rendering on distributed memory multiprocessors. Our method renders by using Hsu's SRC method and composites by using our *Divided-Screenwise Hierarchical* (DSH) method.

Our DSH method focuses on keeping up high performance with the increase of n. To achieve this, we adopt three compositing policies. To reduce the number of transmitted pixels, our method (1) divides the screen into non-overlapping regions and composites each region as the DS method does. Furthermore, to reduce the number of compositing stages, our method (2) dynamically assigns each compositing region to the appropriate processors and (3) composites each region in a hierarchical order. The order is dynamically determined for every viewing angle according to the expected number of transmitted pixels.

The rest of this paper is organized as follows. Section 2 describes an existing sort-last parallel volume rendering method. Section 3 gives the details of our DSH method and presents its theoretical performance analysis. Section 4 presents some experimental results on a 64-node PC cluster. At last, Section 5 concludes this paper.

2. Sort-Last Parallel Volume Rendering

This section describes an existing sort-last parallel volume rendering method using the SRC and BS methods. We also describe the RC method, the base of the SRC method.

2.1. Ray casting (RC)

The RC method [6] produces an image by casting rays from the viewpoint through the screen into the viewing volume as illustrated in Figure 1(Ph0). Let S_s and S_t be the horizontal and the vertical resolution of the screen, respectively. Let $\mathbf{r}_{s,t}$ also be the ray that penetrates point (s,t) on the screen. A pixel value $P(s,t)$, where both $1 \le s \le S_s$

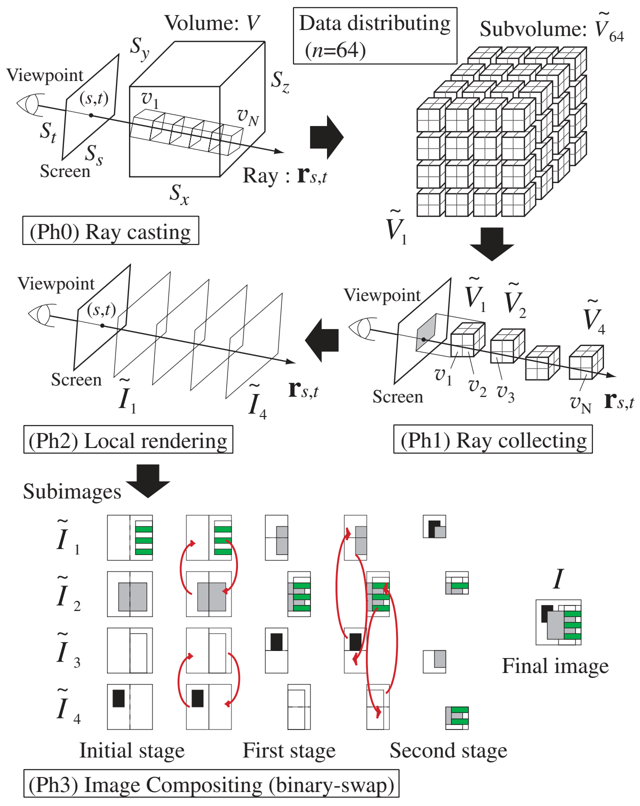


Figure 1. Sort-last parallel volume rendering using segmented ray casting and binary-swap methods.

and $1 \le t \le S_t$, is computed by accumulating the colors and opacities of the voxels that ray $\mathbf{r}_{s,t}$ penetrates:

$$P(s,t) = \sum_{i=1}^{N} \alpha(v_i)\, c(v_i) \prod_{j=0}^{i-1} (1-\alpha(v_j)), \qquad (1)$$

where $v_1, v_2, \ldots, v_N$ are the voxels that ray $\mathbf{r}_{s,t}$ penetrates in a front-to-back order; $c(v_i)$ and $\alpha(v_i)$ are the color and the opacity of voxel v_i, respectively; and $\alpha(v_0) = 0$.

Computing the values of all pixels on the screen produces the final image, I.

2.2. Segmented ray casting (SRC)

The color and opacity accumulating represented as Equation (1) is based on the associative **over** operator, which allows us to segment a ray and accumulate the ray segments independently. Thus the SRC method [4] exploits data parallelism in the RC method via associativity of the Porter and Duff's **over** operator [15].

In the following we assume that the entire volume, V, is partitioned into equal-sized 3D blocks, or subvolumes, and subvolume $\tilde{V}_q$ is distributed to processor q for all $1 \le q \le n$. Figure 1 illustrates this block distribution scheme.

In the SRC method, for all processors $1 \leq q \leq n$, q produces a subimage, $\tilde{I}_q$, by casting rays through its own subvolume $\tilde{V}_q$, then merges n-subimages into the final image. The SRC method consists of the following three phases:

Phase 1: Ray collecting (Figure 1(Ph1)). The penetrated voxels, $v_1, v_2, \ldots, v_N$, specified in Equation (1) are selected for every rays that penetrate the screen. This is performed in two steps. First, for all processors $1 \leq q \leq n$, to know which rays penetrate subvolume $\tilde{V}_q$, q projects its own subvolume $\tilde{V}_q$ onto the screen and determines ray set R_q that consists of any rays that traverse the subvolume $\tilde{V}_q$. Second, for all rays $\mathbf{r}_{s,t} \in R_q$, penetrated voxels $v_1, v_2, \ldots, v_N$ are determined by q. This phase requires only one communication, the broadcast of the viewpoint's coordinate.

Phase 2: Local rendering (Figure 1(Ph2)). For all processors $1 \leq q \leq n$, q produces subimage $\tilde{I}_q$ by rendering its own subvolume $\tilde{V}_q$. That is, q computes a part of Equation (1), the only terms that contain voxel v_i such that $v_i \in \tilde{V}_q$. For example, as shown in Figure 1(Ph1), if processor 2 owns v_3 and v_4, then it computes two terms, $\alpha(v_3)c(v_3) + \alpha(v_4)c(v_4)(1 - \alpha(v_3))$ and $(1 - \alpha(v_3))(1 - \alpha(v_4))$. In this phase, all processors refer only their local voxels, so that communication among processors is unnecessary to produce all subimages.

Phase 3: Image compositing (Figure 1(Ph3)). After the rendering phase, n-subimages have been produced. The processors then merge the produced subimages, $\tilde{I}_1, \tilde{I}_2, \cdots, \tilde{I}_n$, into the final image, I, by applying all computed terms into Equation (1) for every point on the screen. To do this, communication is necessary in this phase.

2.3. Binary-swap (BS) compositing

The BS method [8] is an efficient image compositing method, which produces the final image in a hierarchical order as shown in Figure 1(Ph3).

At every compositing stage, for all processors $1 \leq q \leq n$, q splits its own subimage $\tilde{I}_q$ into two pieces and takes responsibility for one of the two pieces. By assigning equalized compositing tasks to all processors, the BS method balances the compositing workloads that the BT method fails.

At the k-th stage, where $1 \leq k \leq \log n$, the BS method transmits $1/2 \cdot 2^{-(k-1)/3} n^{1/3} p$ pixels among n-processors [8]. Therefore, the average compositing time, t_{BS}, can be bounded as follows:

$$\begin{aligned} t_{BS} &= \sum_{k=1}^{\log n} \left\{ (g + t_c) \, \frac{1}{2n} \, 2^{-(k-1)/3} n^{1/3} p + o \right\} \\ &\leq p\,(g + t_c) \cdot 2.43 n^{-2/3} + o \cdot \log n, \end{aligned} \qquad (2)$$

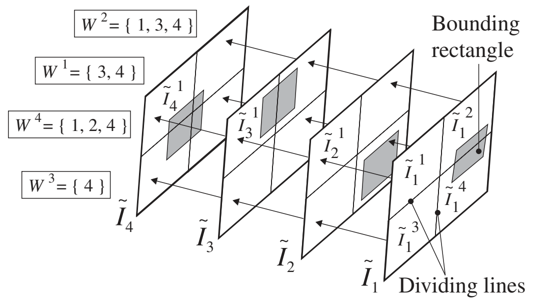


Figure 2. Divided-screenwise image compositing ($n = 4$ and $D = 4$).

where o is the communication overhead that a processor takes for a transmission, and g and t_c is the transmitting time and the computing time per pixel, respectively.

Notice that only non-blank pixels affect the composited results. Thus the BS method exploits the sparsity of subimage by using a bounding rectangle, which encloses the entire non-blank region of the subimage. Every processor composites and transmits within this bounding rectangle.

3. Divided-Screenwise Hierarchical (DSH) Compositing

This section describes our DSH method in detail.

3.1. Compositing policies

The DSH method has three compositing policies:

Policy 1: To reduce the number of transmitted pixels (see Figure 2). For all processors $1 \leq q \leq n$, we divide subimage $\tilde{I}_q$ into D-regions and obtain divided-subimages $\tilde{I}_q^1, \tilde{I}_q^2, \ldots, \tilde{I}_q^D$, where D is the number of screen division. Then, for all regions $1 \leq d \leq D$, we composites d by merging $\tilde{I}_1^d, \tilde{I}_2^d, \ldots, \tilde{I}_n^d$ into one.

Policy 2: To reduce the number of communicating processor pairs (see Figure 2). We shape each region into a tile. In addition, for all regions $1 \leq d \leq D$, we dynamically determine a processor set, W^d, that consists of any processors that have the bounding rectangle of divided-subimage $\tilde{I}_q^d$ within d. Any processors $q \in W^d$ merges region d for all $1 \leq d \leq D$.

Policy 3: To reduce the number of compositing stages (see Figure 4). For all regions $1 \leq d \leq D$, we dynamically determine a hierarchical compositing order, O^d, and concurrently composite d according to O^d.

In the following we introduce why our DSH method adopts the above three policies.

First, the BS method transmits more pixels than the DS method, because it can composite two subimages that fail

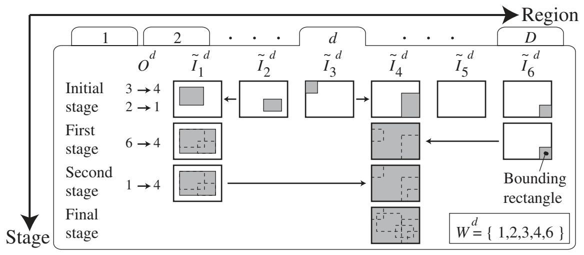


Figure 4. Concurrent processing of hierarchical image compositing.

in overlapping non-blank pixels. For example, in Figure 1(Ph3), processors 3 and 4 composite two subimages $\tilde{I}_3$ and $\tilde{I}_4$ at the first stage. However, both the subimages lack the common non-blank region, so that the processors fail in merging non-blank pixels and the communication for this exchange results in redundant. This redundant communication, which brings no fruitful compositing, happens at two of three stages, because each subvolume locates on the 3D grid [8]. Therefore, we adopt the policy 1. That is, to eliminate such redundant communications, we divide the screen into regions and composites each region like the DS method, which transmits the fewest pixels among the four methods introduced in Section 1.

Next, both the policies 2 and 3 are the ideas to solve the performance problem of the DS method. In the DS method, to ensure load-balancing, the screen is divided into small regions and each region is statically assigned to a processor in a random or interleaved distribution. However, this distribution scheme makes the communication pattern complicated. At the worst case, each processor communicates to every other processor, so that the number of compositing stages increases up to $n - 1$. Therefore, we adopt the policy 2 for our method, which aims at achieving high performance compositing with the increase of n. That is, to make the communication pattern simple, (1) we divide the screen into tile-shaped regions and (2) for all regions $1 \leq d \leq D$, we dynamically assign region d to processor set W^d. Notice that processor set W^d excludes any processors that lack a non-blank pixels within d.

Furthermore, to reduce the number of compositing stages, we composites each region in a hierarchical order like the BT method. For all regions $1 \leq d \leq D$, d is composited in concurrent according to a hierarchical order, O^d. The algorithm to determine O^d is described in Section 3.2.

To determine processor set W^d, we have to examine the coordinates of all distributed bounding rectangles. Since these coordinates are fixed after the rendering phase, our method requires a broadcasting at the beginning of the compositing phase. However, we require only the coordinates, so that the communication cost for this preprocessing is relatively low to the entire compositing cost, which contains the communication cost for transmitting all pixels within the bounding rectangles.

While only one processor merges the final image in the BT method, at most D-processors merge the final image in our method. Therefore, given an appropriate value for D, our method avoids extremely low efficiency.

3.2. Algorithm to determine compositing order

To determine the set of compositing order, $O = \{O^d \mid 1 \leq d \leq D\}$, our algorithm requires four inputs: (1) n, the number of processors; (2) D, the number of screen division; (3) $B = \{\tilde{B}_q^d \mid 1 \leq q \leq n, 1 \leq d \leq D\}$ and (4) $Z = \{\tilde{B}_q \mid 1 \leq q \leq n\}$, the sets of bounding rectangles, where $\tilde{B}_q^d$ and $\tilde{B}_q$ are the bounding rectangles of divided-subimage $\tilde{I}_q^d$ and projected subvolume $\tilde{V}_q$, respectively. Here, the coordinates of $\tilde{B}_q$ are obtained at the ray collecting phase and broadcasted with those of $\tilde{B}_q^d$ at the beginning of the compositing phase.

In our method, a processor, q_s, that has a small bounding rectangle $\tilde{B}_{q_s}^d$ sends $\tilde{B}_{q_s}^d$ to a processor, q_l, that has a large bounding rectangle $\tilde{B}_{q_l}^d$. The receiver processor q_l takes the responsibility for the received $\tilde{B}_{q_s}^d$ (see Figure 4). When pairing up processors q_s and q_l from processor set W^d, we take the receiver first (RF) scheme. That is, we select the receiver q_l prior to the sender q_s. Indeed we can take the sender first (SF) scheme but we take the RF scheme from some experimental results shown later in Section 4.3.

Figure 3 presents the algorithm. First, for all regions $1 \leq d \leq D$, processor set W^d is determined by examining if each processor has a bounding rectangle within d (line 12). Here, we reduce the search space by referring the coordinates of $\tilde{B}_q$ (line 10–11).

We then select processor q_l such that q_l has the largest bounding rectangle in processor set W_{stg}^d (line 28). Here W_{stg}^d is a processor set initialized with W^d and consists of any processors that can be paired up with their front/back neighbors at the current compositing stage. We also select processor q_s from W_{stg}^d such that q_s has a smaller bounding rectangle between q_l's front/back neighbors (line 30).

If such processor $q_s \in W_{stg}^d$ exists, we then add a compositing task that sends bounding rectangle $\tilde{B}_{q_s}^d$ to q_l (line 33) and update the coordinates of bounding rectangle $\tilde{B}_{q_l}^d$ (line 34). Otherwise, since both front and back neighbors of q_l are engaged in other compositing task at this stage, we leave the compositing of bounding rectangle $\tilde{B}_{q_l}^d$ at later stages (line 38).

Repeating the above processes until W_{stg}^d becomes an empty set determines a compositing order for one stage (line 27–39). Moreover, repeating them until every processor $q \in W^d$ engages at least one compositing task determines O^d, or the compositing order for region d (line 25–40).

3.3. Theoretical performance analysis

This section presents a theoretical performance analysis of our DSH method. We give the average composit-

```
Inputs: (1) $n$: Number of processors.
        (2) $D$: Number of screen regions.
        (3) $B$: Set of bounding rectangles of rendered images divided by regions, $B \equiv \{\tilde{B}_q^d \mid 1 \le q \le n, 1 \le d \le D\}$.
        (4) $Z$: Set of bounding rectangles of projected subvolume determined at phase (Ph1), $Z \equiv \{\tilde{B}_q \mid 1 \le q \le n\}$.
Output: (1) $O$: Set of compositing order lists, $O \equiv \{O^d, 1 \le d \le D\}$.

 1: Algorithm DetermineHierComposOrder($n$,$D$,$B$,$Z$,$O$);
 2: begin
 3:   $O := \emptyset$;   /* Initialize $O$ as an empty set. */
 4:   foreach $d \in \{1, 2, \ldots, D\}$ do begin
 5:     $W^d := \emptyset$; $O^d := \emptyset$;
 6:   end
 7:   /* 1. Identify the processors that engage in
 8:          compositing of region $d$. */
 9:   foreach $q \in \{1, 2, \ldots, n\}$ do begin
10:     $D_q := \{d \in \{1, 2, \cdots, D\} \mid d \cap \tilde{B}_q \ne \emptyset\}$;
11:     foreach $d \in D_q$ do begin
12:       if ($A(\tilde{B}_q^d) \ne 0$) then  /* $A(\tilde{B}_q^d)$: Area of $\tilde{B}_q^d$ */
13:         $W^d := W^d \cup \{q\}$;
14:     end
15:   end
16:   /* 2. Determine each of compositing order. */
17:   foreach $d \in \{1, 2, \ldots, D\}$ do begin
18:     DetermineRegion($W^d$, $d$, $B$, $O^d$);   /* line24 */
19:     $O := O \cup \{O^d\}$;
20:   end
21: end
22: /* Procedure to generate $O^d$. */
23: Procedure DetermineRegion($W^d$, $d$, $B$, $O^d$);
24: begin
25:   while ($W^d \ne \emptyset$) do begin   /* All stages */
26:     $W_{stg}^d := W^d$;
27:     while ($W_{stg}^d \ne \emptyset$) do begin   /* One stage */
28:       Select $q_l \in W_{stg}^d$ such that
29:         $\forall q \in W_{stg}^d \; (A(\tilde{B}_q^d) \le A(\tilde{B}_{q_l}^d))$;
30:       Select $q_s \in W_{stg}^d$ such that $q_s$ has lower
31:         $A(\tilde{B}_{q_s}^d)$ between $q_l$'s front/back neighbors;
32:       if $\exists q_s \in W_{stg}^d$ then begin
33:         Add a compositing task '$q_s \rightarrow q_l$' to $O^d$;
34:         $\tilde{B}_{q_l}^d := \tilde{B}_{q_l}^d \cup \tilde{B}_{q_s}^d$;  /* Update coordinates */
35:         Delete $q_s$ and $q_l$ from $W_{stg}^d$;
36:         Delete $q_s$ from $W^d$;   /* $q_s$ completes */
37:       end else
38:         Delete $q_l$ from $W_{stg}^d$;
39:     end
40:   end
41: end
```

Figure 3. Algorithm to determine a hierarchical compositing order.

Table 1. Summary of theoretical analysis. s_{DSH}, c_{DSH}, and t_{DSH} represent number of compositing stages, number of transmitted pixels per processor, and average compositing time, respectively.

s_{DSH}	$\le D\, n^{-2/3} \lceil \log_{3/2}(c\, n^{1/3}) \rceil$	(3)
c_{DSH}	$= s_{DSH} \cdot (e\, p\, n^{-2/3}/D)$	(4)
t_{DSH}	$= \sum_{k=1}^{s_{DSH}} \left\{ (g + t_c) \cdot (e\, p\, n^{-2/3}/D) + o \right\}$	
	$\le p\,(g + t_c) \cdot \left\{ e\, n^{-4/3} \lceil \log_{3/2}(c\, n^{1/3}) \rceil \right\} \; + \; o \cdot \left\{ D\, n^{-2/3} \lceil \log_{3/2}(c\, n^{1/3}) \rceil \right\}$	(5)

n: Number of processors. p: Number of pixels in the final image. D: Number of screen division.
c: Proportionality constant for the number of nonblank pixels transmitted by a processor at a stage.
e: Proportionality constant for the number of processors engaged in a region.

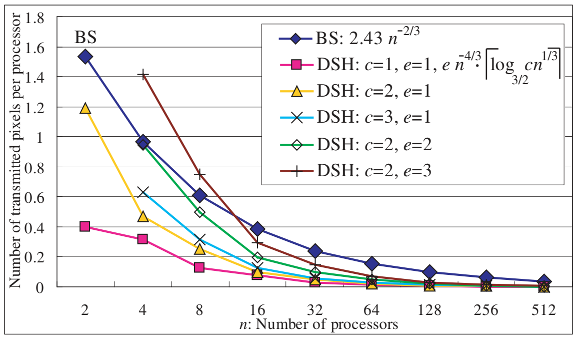
ing time, t_{DSH}, by analyzing the number of compositing stages, s_{DSH}, and the number of transmitted pixels per processor, c_{DSH} (see Table 1). To make this analysis simple, we assume that (A1) the final image, I, is filled with non-blank pixels. We also assume that (A2) each axis of the screen and volume has the same resolution and that (A3) $D \ge n^{2/3}$ (mentioned in later).

First, we analyze s_{DSH}. Consider that any processors in processor set W^d are sorted in a front-to-back order. Since only adjoining processors can be paired up, some processors can be isolated at each compositing stage. Therefore, s_{DSH} becomes worst when isolated processors exist between every pair of processors at every stage. In this worst case, $|W^d|$ reduces by a factor of 2/3 at every stage, so that region d takes $\lceil \log_{3/2} |W^d| \rceil$ stages. On the other hand, it takes $\lceil \log |W^d| \rceil$ stages in the best case, where $|W^d|$ reduces in half at every stage like the BT method.

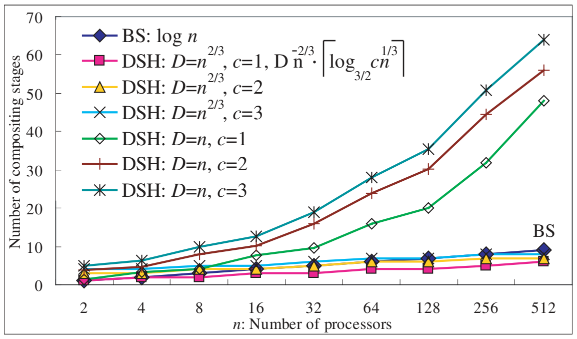
Furthermore, in a block data distribution shown in Figure 1, each processor has approximately $p\, n^{-2/3}$ non-blank pixels at the initial compositing stage [13]. Since we assume that (A1), this means that for all processors $1 \le q \le n$, $n^{-2/3}$ of the entire pixels in subimage $\tilde{I}_q$ are non-blank at the initial compositing stage. Therefore, each processor has non-blank pixels within $D\, n^{-2/3}$ regions.

Thus our DSH method takes $\lceil \log_{3/2} |W^d| \rceil$ stages for each of $D\ n^{-2/3}$ regions. If we have to perform the concurrent compositing in serial, s_{DSH} becomes $D\ n^{-2/3} \lceil \log_{3/2} |W^d| \rceil$. On the other hand, if we can perform the compositing independently, s_{DSH} becomes $max(D\ n^{-2/3}, \lceil \log_{3/2} |W^d| \rceil)$.

Next, we analyze c_{DSH}. At each compositing stage, the size of bounding rectangle $\tilde{B}_q^d$ is proportional to $p\ n^{-2/3}$, or the number of non-blank pixels at the initial compositing stage. It also inversely proportional to D, or the number of screen division. Therefore, each processor transmits $e\, p\, n^{-2/3}/D$ non-blank pixels at a compositing stage, where e is the proportionality constant, so that we obtain



(a) Number of transmitted pixels per processor



(b) Number of compositing stages

Figure 5. Theoretical comparison between DSH and BS methods ($D \geq n^{2/3}$).

Equation (4) (see Table 1).

At last, using the above analysis, we give a bound for t_{DSH}. When we divide the screen into at least $n^{2/3}$ regions, we obtain $|W^d| \leq c\,n^{1/3}$, where c is a proportionality constant. To explain this in brief, we assume that the viewpoint is selected so that we are looking straight down the Z-axis. Then, since we assume that (A2), we are viewing $n^{2/3}$ divided-subimages on the screen. Each of the viewing divided-subimage is overlapping $n^{1/3}$ divided-subimages along the Z-axis. Therefore, dividing the screen into at least $n^{2/3}$ regions, $D \geq n^{2/3}$, gives $|W^d| \leq c\,n^{1/3}$.

Summarizing the above discussions, t_{DSH} for $D \geq n^{2/3}$ is bounded by Equation (5).

3.4. Theoretical comparison between DSH and BS methods

Using Equations (2) and (5), we give a theoretical comparison between the DSH method and the BS method. The coefficients of the two terms, $p\ (g + t_c)$ and o, represent the number of transmitted pixels per processor and that of compositing stages, respectively. Figure 5 plots the values of each term, for $D = n^{2/3}$ and for $D = n$, with varying the values of c and e.

Figure 5(a) indicates that both methods transmit fewer pixels as n increases. But the DSH method decreases more quickly than the BS method, and the performance gap between the two methods extends as n increases. Thus, with the increase of n, the DSH method transmits fewer pixels compared to the BS method.

Figure 5(b) indicates that both methods take more compositing stages as n increases. When $D = n^{2/3}$, the DSH method completes the image compositing with the almost same number of compositing stages as the BS method. On the other hand, when $D = n$, the efficiency of concurrent compositing, or the dependence among compositing tasks, strongly affects the number of compositing stages in the DSH method. If processors have to perform the concurrent compositing in serial, it takes more stages than the BS method as shown in Figure 5(b). However, if processors perform the concurrent compositing in fully parallel, the number of compositing stages become $max(n^{1/3}, \lceil \log_{3/2}(c\ n^{1/3})\rceil)$ as presented in Section 3.3. In this best case, the DSH method takes the almost same number of compositing stages as the BS method.

4. Experimental Results

In this section we present some experimental results on a PC cluster using clinical volumes. We measured following performance metrics:

- *Compositing time* (see Section 4.2). We measured the compositing times of the DSH and BS methods with varying the numbers of processors and screen division.
- *Numbers of transmitted pixels per processor and compositing stages* (see Section 4.3). We measured these numbers of both methods. We also compared the RF and SF schemes mentioned in Section 3.2.

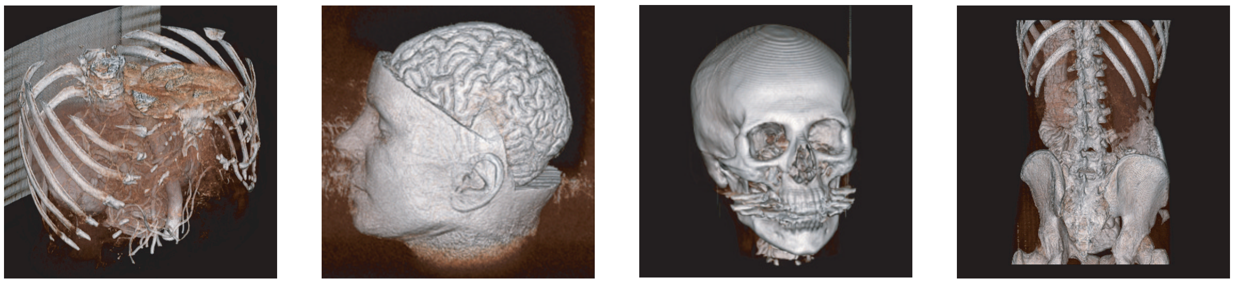
4.1. Experimental environments

Figure 6 shows the rendered images from clinical volumes used in our experiments. These volumes are created by a X-ray CT and magnetic resonance (MR) scan.

We used a PC cluster with 64 symmetric multiprocessor (SMP) nodes for our experiments. Each node in the cluster has two Pentium III 1GHz processors and connects to a Myrinet-2000 [1] switch, which provides sustained bandwidth of 2 Gbps per link.

We have implemented our sort-last parallel volume rendering method and also the BS method using C++ language and MPICH-SCore library [14], which is a fast implementation of Message Passing Interface (MPI) [9].

To achieve high performance rendering on SMP nodes, our implementation uses the POSIX thread in the rendering phase. However, to make the implementation portable, we currently avoid using the POSIX thread in the compositing phase. Few MPI implementations [19] are thread safe at



(a) *VD*1: Breast (512x512x159, CT) (b) *VD*2: Brain (512x512x218, MR) (c) *VD*3: Skull (512x512x224, CT) (d) *VD*4: Belly (512x512x730, CT)

Figure 6. Rendered images from clinical volumes used in experiments.

present, because thread safe mechanism can reduce communication performance. So the number of compositing processors in our current implementation is equal to the number of nodes. In the following let n be the number of nodes, or the number of compositing processors.

4.2. Compositing time

Using the four volumes shown in Figure 6, we measured the compositing time of the DSH method and that of the BS method, T_{DSH} and T_{BS}, respectively.

We rotated the viewpoint 360 degrees around the body axis, or Z-axis, by 15 degrees. We then measured 24 compositing times and averaged them for each T_{DSH} and T_{BS}. The screen was 512^2 pixel large and divided into D-regions: $\sqrt{D}$ regions for horizontal and vertical direction, respectively. Moreover, when we failed to assign the same number of nodes for each three volume axis, we averaged the measured times of all cases such that every axis has at most double nodes compared to the other two axis.

Figure 7 shows the measured results. In many cases we obtain $T_{DSH} < T_{BS}$ except for $n = 2$. Therefore, given an appropriate value for D, the DSH method outperforms the BS method on our PC cluster. For example, when both $n = 64$ and $D = 64$, the maximum speedup of DSH to BS ranges from 1.4 for VD_3 to 2.2 for VD_1.

In the following we describe how we can determine such an appropriate value, D_m, for any given n. In Figure 7, when increasing the value of D at any fixed value of n, T_{DSH} reaches the maximum at $D = 1$, decreases until $D = D_m$, and then turns to increase. Furthermore, we also see that $D_m \geq n$ for all n. Therefore, by increasing the value of D from given n, we can easily identify when T_{DSH} turns to increase and thereby can determine D_m by selecting the value of D that gives the last decrease of T_{DSH}.

Table 2 lists three performance results when $D = D_m$: (1) the ratio of compositing time to the total rendering time, $r_{DSH} = 100 \times T_{DSH}/T$ and r_{BS}, where T is the total rendering time; (2) frames per second, $f_{DSH} = 1/T$ and f_{BS}; (3) speedup ratio, $s_f = 100 \times (f_{BS} - f_{DSH})/f_{BS}$.

In Table 2, r_{DSH} and r_{BS} increase with n and reach the maximum of 22.2% and 16.8%, respectively. Thus, as the number of processors increases, the compositing performance dominates the total rendering performance. As mentioned earlier, this result motivates us to develop an efficient compositing method that provides high performance compositing with the increase of n.

Moreover, in all cases except for $n = 2$, we see that $f_{DSH} > f_{BS}$. We also see that s_f increases with n. That is, with the increase of n, the DSH method becomes faster than the BS method and contributes to the improvement of the total rendering performance. On 64 nodes, the DSH method renders VD_1 and VD_3 at 10.4 and 8.3 fps, yielding parallel speedups of 25.1 and 28.2, respectively.

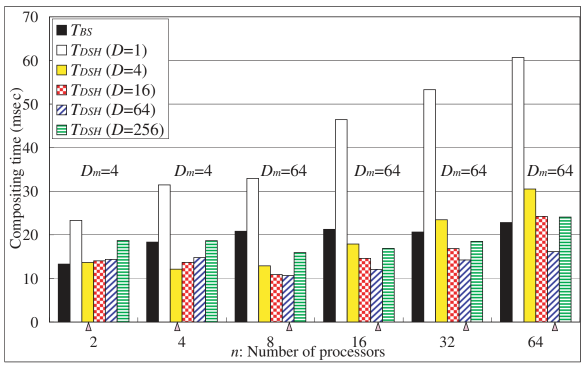
4.3. Numbers of transmitted pixels per processor and compositing stages

By rendering the four volumes in the same way as mentioned in Section 4.2, we measured the numbers of transmitted pixels per processor in the DSH method and in the BS method, C_{DSH} and C_{BS}, respectively. For each viewpoint, we measured the maximum number among all nodes, and then averaged the measured numbers.

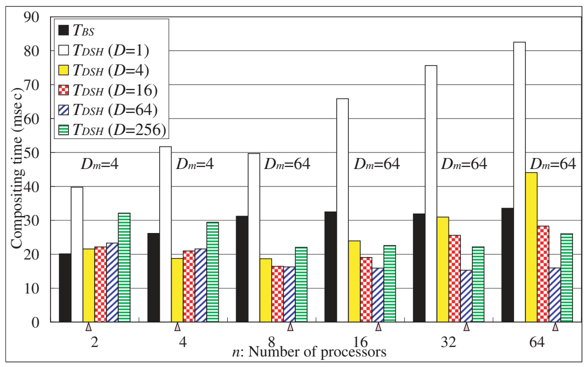
Figure 8 shows both C_{DSH} and C_{BS}. We see that the gap between C_{DSH} and C_{BS} extends as n increases. For example, the minimum value of C_{DSH}/C_{BS} in Figure 8(d) is 0.87 for $n = 2$ and decreases to 0.22 for $n = 64$. This decrease roughly agrees with the theoretical results shown in Figure 5(a).

Furthermore, where $n \geq 8$, C_{DSH} monotonously decreases as D increases. The reason for this is explained as follows. Dividing the screen into small regions prevents the bounding rectangles from extremely growing at each compositing stage. For example, in Figure 4, two subimages $\tilde{I}_3^d$ and $\tilde{I}_4^d$ are composited, and each of the two has a bounding rectangle in the left-top/right-bottom part of region d. Therefore, after this compositing, the bounding rectangle of $\tilde{I}_4^d$ grows to the same size as the entire region d. Since the same bounding rectangle may repeatedly transmitted in hierarchical compositing methods, this extremely grown bounding rectangle can increase the number of transmitted pixels at the following stage. Thus, as D increases, the grown bounding rectangles are left smaller, and thereby C_{DSH} monotonously decreases with the increase of D.

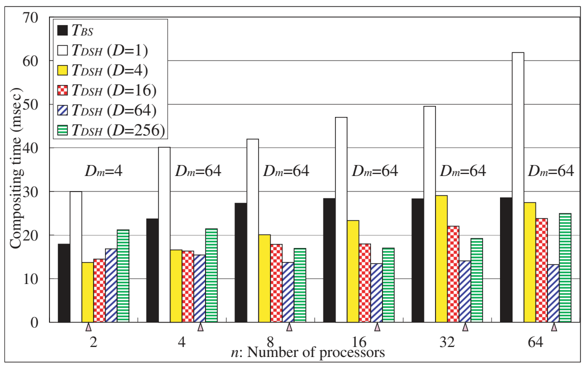
Table 3 shows S_{DSH} and S_{BS}, the numbers of compositing stages in the DSH method and in the BS method, respectively. For the DSH method, we show both the numbers for the RF and the SF schemes.



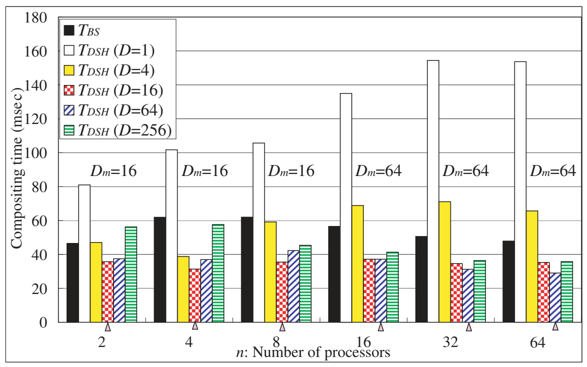
(a) *VD*1: Breast (512x512x159, CT)



(b) *VD*2: Brain (512x512x218, MR)



(c) *VD*3: Skull (512x512x224, CT)



(d) *VD*4: Belly (512x512x730, CT)

Figure 7. Compositing times of DSH and BS methods. Fastest results are obtained where $D = D_m$.

In Table 3, S_{BS} is exactly $\log n$ for any given volumes. On the other hand, S_{DSH} depends on both the viewing volume and the value of D. That is, as mentioned in Section 3.4, the efficiency of concurrent compositing determines the value of S_{DSH}. When increasing the value of D at the fixed value of n, while S_{DSH} slightly increases where $D \le n$, it increases at least twice where $D > n$. Therefore, if we divide the screen into at least n-regions, the number of compositing stages rapidly increases. When the value of D is extremely large for n, the performance of image compositing decreases as we see in Figure 7 where $D = 256$.

Although the DSH method shows better performance than the BS method, we obtain $S_{DSH} \ge S_{BS}$ for all n. Notice here that the processing time per compositing stage differs between the DSH and BS methods. In the DSH method, S_{DSH} increases with D, but each divided region becomes smaller. Thus each compositing task shrinks as D increases, and this brings better performance to the DSH method.

At last, we compare the RF and the SF schemes. From Table 3 we think that both methods are equal in the number of compositing stage. We then compared these two methods on the number of transmitted pixels, and the RF method transmits pixels fewer an average of 4% compared to the SF scheme. Thus we use the RF scheme in our DSH method.

The reduction of 4% is achieved as follows. In the SF scheme, since receiver processor q_l are determined from the neighbors of sender processor q_s, receiver processor q_l can fail to have a large bounding rectangle at each compositing stage. On the other hand, the RF scheme selects receiver processor q_l sequentially from a processor with a larger bounding rectangle. Therefore, compared to the SF scheme, the RF scheme allows fewer bounding rectangles to grow. Thus the RF scheme dropped the number of transmitted pixels by 4%.

5. Conclusions

We have presented a novel compositing method, the DSH method, for sort-last parallel volume rendering on distributed memory multiprocessors. Our method aims at achieving high performance compositing on a large number of processors. To achieve this, the DSH method divides the screen to tile-shaped regions and composites them in concurrent according to a dynamically determined hierarchical order. As the number of screen division increases, we obtain fewer communications with a longer compositing stage.

Our implementation on a 64-node PC cluster can composite a 512^2 pixel image at most 2.2 times faster than the BS method, so that can render a 512x512x224 voxel volume at approximately eight fps.

Acknowledgements

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (C)(2)14580374, JSPS Research for

Table 2. Compositing time ratio to total rendering time and frames per second.

VD_1: Breast (512x512x159)					
n	Comp. time ratio (%)		Frames per sec. (fps)		Speedup ratio (%)
	r_{BS}	r_{DSH}	f_{BS}	f_{DSH}	s_f
2	1.2	1.2	0.89	0.89	–0.0
4	2.6	1.7	1.43	1.44	0.9
8	5.0	2.3	2.40	2.47	2.8
16	8.9	5.3	4.21	4.38	4.0
32	13.3	9.6	6.46	6.74	4.3
64	22.2	16.8	9.73	10.41	7.0

VD_2: Brain (512x512x218)					
n	Comp. time ratio (%)		Frames per sec. (fps)		Speedup ratio (%)
	r_{BS}	r_{DSH}	f_{BS}	f_{DSH}	s_f
2	0.7	0.8	0.35	0.35	–0.1
4	1.7	1.2	0.64	0.65	0.5
8	3.8	1.9	1.23	1.25	2.0
16	6.1	3.1	1.89	1.95	3.2
32	9.9	5.0	3.10	3.27	5.4
64	15.9	8.3	4.73	5.15	9.1

VD_3: Skull (512x512x224)					
n	Comp. time ratio (%)		Frames per sec. (fps)		Speedup ratio (%)
	r_{BS}	r_{DSH}	f_{BS}	f_{DSH}	s_f
2	3.0	2.0	1.65	1.67	1.0
4	5.4	3.3	2.28	2.34	2.3
8	11.2	5.2	4.11	4.38	6.7
16	14.9	7.6	5.24	5.68	8.5
32	18.3	10.0	6.48	7.13	10.1
64	21.1	11.0	7.39	8.34	12.8

VD_4: Belly (512x512x730)					
n	Comp. time ratio (%)		Frames per sec. (fps)		Speedup ratio (%)
	r_{BS}	r_{DSH}	f_{BS}	f_{DSH}	s_f
2	1.4	0.9	0.29	0.29	0.5
4	2.9	1.4	0.47	0.48	1.5
8	4.5	2.6	0.72	0.74	1.9
16	6.6	4.1	1.16	1.19	2.7
32	9.4	6.0	1.86	1.93	3.8
64	14.4	9.1	3.00	3.19	6.1

Table 3. Number of compositing stages in DSH and BS methods.

VD_1: Breast (512x512x159)											
n	S_{BS}	S_{DSH}									
		$D=1$		$D=4$		$D=16$		$D=64$		$D=256$	
		RF	SF	RF	SF	RF	SF	RF	SF	RF	SF
2	1	1	1	$\underline{3}$	$\underline{4}$	7	8	29	29	79	81
4	2	2	2	$\underline{4}$	$\underline{4}$	8	8	23	23	58	58
8	3	3	3	4	4	7	8	$\underline{20}$	$\underline{18}$	48	42
16	4	4	4	5	5	7	7	$\underline{15}$	$\underline{14}$	37	35
32	5	5	5	6	6	6	6	$\underline{12}$	$\underline{11}$	29	28
64	6	6	6	7	6	6	7	$\underline{8}$	$\underline{8}$	21	20

VD_2: Brain (512x512x218)											
n	S_{BS}	S_{DSH}									
		$D=1$		$D=4$		$D=16$		$D=64$		$D=256$	
		RF	SF	RF	SF	RF	SF	RF	SF	RF	SF
2	1	1	1	$\underline{4}$	$\underline{4}$	8	8	32	32	128	128
4	2	2	2	$\underline{4}$	$\underline{4}$	8	8	24	24	83	84
8	3	3	3	4	4	8	8	$\underline{21}$	$\underline{21}$	70	60
16	4	4	4	5	5	8	8	$\underline{19}$	$\underline{18}$	56	50
32	5	5	5	6	6	7	8	$\underline{13}$	$\underline{13}$	39	36
64	6	6	6	6	6	6	6	$\underline{10}$	$\underline{10}$	27	27

VD_3: Skull (512x512x224)											
n	S_{BS}	S_{DSH}									
		$D=1$		$D=4$		$D=16$		$D=64$		$D=256$	
		RF	SF	RF	SF	RF	SF	RF	SF	RF	SF
2	1	1	1	$\underline{3}$	$\underline{3}$	6	6	19	22	74	81
4	2	2	2	4	4	7	7	$\underline{18}$	$\underline{18}$	58	55
8	3	3	3	4	4	8	8	$\underline{17}$	$\underline{17}$	41	44
16	4	4	4	6	5	7	7	$\underline{14}$	$\underline{14}$	28	28
32	5	5	5	6	6	6	6	$\underline{11}$	$\underline{11}$	28	28
64	6	6	6	7	7	7	6	$\underline{8}$	$\underline{8}$	18	18

VD_4: Belly (512x512x730)											
n	S_{BS}	S_{DSH}									
		$D=1$		$D=4$		$D=16$		$D=64$		$D=256$	
		RF	SF	RF	SF	RF	SF	RF	SF	RF	SF
2	1	1	1	4	4	$\underline{9}$	$\underline{13}$	41	55	219	233
4	2	2	2	4	4	$\underline{8}$	$\underline{11}$	28	40	148	141
8	3	3	3	4	4	$\underline{12}$	$\underline{11}$	35	30	120	105
16	4	4	4	5	5	11	10	$\underline{27}$	$\underline{25}$	91	84
32	5	5	5	6	6	9	9	$\underline{21}$	$\underline{20}$	67	66
64	6	6	6	7	6	9	9	$\underline{16}$	$\underline{16}$	49	49

Underlined numbers are the numbers of compositing stages when $D = D_m$.

the Future Program JSPS-RFTF99I00903, and the Foundation for C&C Promotion. We would like to thank Masaki Miyamoto at Medical Imaging Laboratory Co., Ltd., Yoshinobu Sato and Shinichi Tamura at Osaka University for useful discussions regarding this work. We are also grateful to the anonymous reviewers for their valuable comments.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–36, Feb. 1995. http://www.myri.com/.

[2] E. Camahort and I. Chakravarty. Integrating Volume Data Analysis and Rendering on Distributed Memory Architectures. In *Proc. 1993 Parallel Rendering Symp. (PRS'93)*, pages 89–96, Oct. 1993.

[3] P. Hatreiter, C. Rezk-Salama, K. Eberhardt, B. Tomandl, and T. Ertl. Functional Analysis of the Vertebral Column Based on MR and Direct Volume Rendering. In *Proc. 3rd Int'l Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI'00)*, pages 412–421, Oct. 2000.

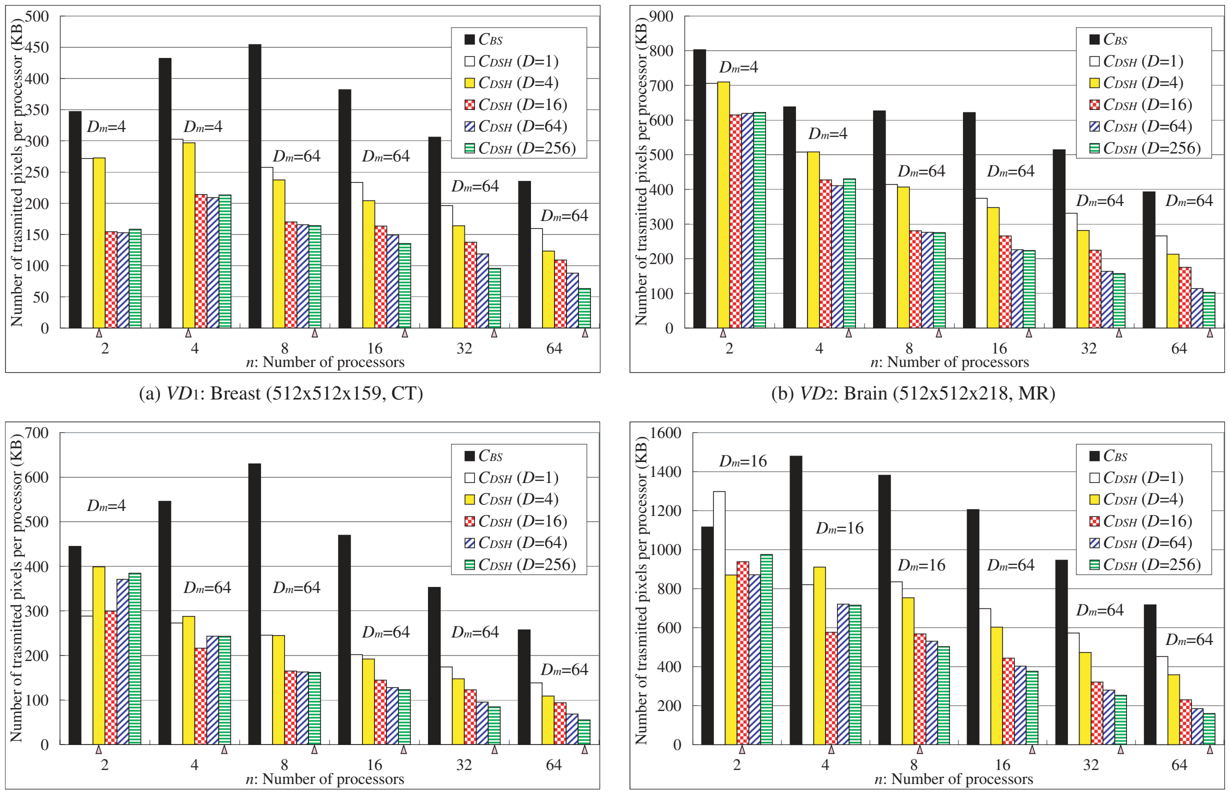
[4] W. M. Hsu. Segmented Ray Casting for Data Parallel Volume Rendering. In *Proc. 1993 Parallel Rendering Symp. (PRS'93)*, pages 7–14, Oct. 1993.

[5] T.-Y. Lee, C. Raghavendra, and J. B. Nicholas. Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers. *IEEE Trans. on Visualization and Computer Graphics*, 2(3):202–217, Sept. 1996.

[6] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[7] C.-F. Lin, D.-L. Yang, and Y.-C. Chung. A Rotate-Tiling Image Composition Method for Parallel Volume Rendering on Distributed Memory Multicomputers. In *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS2002)*, Apr. 2002.

[8] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. Parallel Volume Rendering Using Binary-Swap Composit-



(a) VD_1: Breast (512x512x159, CT)

(b) VD_2: Brain (512x512x218, MR)

(c) VD_3: Skull (512x512x224, CT)

(d) VD_4: Belly (512x512x730, CT)

Figure 8. Number of transmitted pixels per processor in DSH and BS methods.

ing. *IEEE Computer Graphics and Applications*, 14(4):59–68, July 1994.

[9] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *Int'l J. of Supercomputing Applications*, 8(3/4), 1994.

[10] T. Mitra and T. Chiueh. Implementation and Evaluation of the Parallel Mesa Library. In *Proc. 1998 Int'l Conf. on Parallel and Distributed Systems (ICPADS'98)*, pages 84–91, Dec. 1998.

[11] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, July 1994.

[12] S. Muraki, M. Ogata, K.-L. Ma, K. Koshizuka, K. Kajihara, X. Liu, Y. Nagano, and K. Shimokawa. Next-Generation Visual Supercomputing using PC Clusters with Volume Graphics Hardware Devices. In *Proc. High Performance Networking and Computing Conf. (SC2001)*, Nov. 2001.

[13] U. Neumann. Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers. In *Proc. 1993 Parallel Rendering Symp. (PRS'93)*, pages 97–104, Oct. 1993.

[14] F. O'Carroll, H. Tezuka, A. Hori, and Y. Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *Proc. ACM Int'l Conf. on Supercomputing (ICS'98)*, pages 243–250, July 1998. http://www.pccluster.org/.

[15] T. Porter and T. Duff. Compositing Digital Images. *Computer Graphics (Proc. SIGGRAPH'84)*, 18(3):253–259, July 1984.

[16] K. Sano, H. Kitajima, H. Kobayashi, and T. Nakamura. Parallel Processing of the Shear-Warp Factorization with the Binary-Swap Method on a Distributed-Memory Multiprocessor System. In *Proc. 1997 Parallel Rendering Symp. (PRS'97)*, pages 87–94, Oct. 1997.

[17] R. Shahidi, B. Wang, M. Epitaux, R. Grzeszczuk, and J. Adler. Volumetric Image Guidance via a Stereotactic Endoscope. In *Proc. 1st Int'l Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, pages 241–252, Oct. 1998.

[18] C. T. Silva, A. E. Kaufman, and C. Pavlakos. PVR: High-Performance Volume Rendering. *IEEE Computational Science and Engineering Winter 1996*, 3(4):18–28, 1996.

[19] H. Tang and T. Yang. Optimizing Threaded MPI Execution on SMP Clusters. In *Proc. 15th ACM Int'l Conf. on Supercomputing (ICS'01)*, pages 381–392, June 2001.

[20] R. Westermann. Parallel Volume Rendering. In *Proc. 9th Int'l Parallel Processing Symp. (IPPS'95)*, pages 693–699, Apr. 1995.

[21] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics (Proc. SIGGRAPH'90)*, 24(4):367–376, Aug. 1990.

[22] B. Wylie, C. Pavlakos, V. Lewis, and K. Moreland. Scalable Rendering on PC Clusters. *IEEE Computer Graphics and Applications*, 21(4):62–70, July 2001.

[23] D.-L. Yang, J.-C. Yu, and Y.-C. Chung. Efficient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers. In *Proc. 1999 Int'l Conf. on Parallel Processing (ICPP'99)*, pages 200–207, Sept. 1999.