A Rotate-Tiling Image Composition Method for Parallel Volume Rendering on Distributed Memory Multicomputers

Chin-Feng Lin, Don-Lin Yang, and Yeh-Ching Chung
Department of Information Engineering,
Feng Chia University, Taichung, Taiwan 407
TEL: 886-4-2451-7250x3700
FAX: 886-4-2451-6101
E-mail: {cflin, dlyang, ychung}@iecs.fcu.edu.tw

Abstract

The binary-swap and the parallel-pipelined methods are two popular image composition methods for volume rendering on distributed memory multicomputers. However, these methods either restrict the number of processors to a power of two or require many steps to transform image data that results in high communication overheads. In this paper, we present an efficient image composition method, the rotate-tiling (RT), for parallel volume rendering on distributed memory multicomputers. The RT method can fully utilize all available processors and minimize the communication overheads. In addition, we provide a data compression method, the template run-length encoding (TRLE), to further reduce the communication data size. To evaluate the performance of the RT method, we compare the proposed method with the binary-swap method and the parallel-pipelined method. Both theoretical analysis and experimental test are conducted. In the theoretical analysis, we analyze the best performance bound of the RT method in terms of the startup time, the data transmission time, the number of processors, and the number of initial block of a sub-image. In the experimental test, we have implemented these three methods on an SP2 parallel machine. Three volume datasets are used as test samples. The experimental results show that our method outperform the binary-swap and the parallel-pipelined methods for all test samples and match the results analyzed in the theoretical analysis. For the TRLE method, the experimental results show that the TRLE method can further reduce the composition time for these three methods.

Index Terms: Image Composition, Volume Rendering, Binary-Swap, Parallel-Pipelined, Run-Length Encoding, Rotate-Tiling.

1. Introduction

Volume rendering [3-5] can be used to analyze the shape and volumetric property of three-dimensional objects in research areas such as medical imaging and computational fluid dynamics. It can not only display the semi-opaque object, but also provide a better visualization of the object surface. However, most volume rendering methods that produce effective visualizations are computation intensive [12,14], and it is very difficult to achieve interactive rendering rates for large datasets. In addition, volume datasets are too large to be stored in the memory of a single processor element. One way to solve the above problems is to parallelize the serial volume rendering techniques on distributed memory multicomputers [21-26].

A parallel volume rendering system on distributed memory multicomputers, in general, consists of three stages, the data partitioning stage, the rendering stage, and the image composition stage. In the data partitioning stage, an efficient partitioning method can be used to distribute the volume dataset to processors. In the rendering stage, each processor uses a serial rendering algorithm to generate a partial image. In the image composition stage, the partial intermediate images of processors are composited to form a final image. When the number of processors is large, the image composition stage becomes a bottleneck of a parallel volume rendering system. Hence, a good image composition method is very important to the performance of a parallel volume rendering system.

In general, there are two ways to improve the performance of the image composition for volume rendering on distributed memory multicomputers. One is to use an efficient communication scheme to send and receive the partial intermediate images of processors such that the communication overheads can be minimized. The other is to use an efficient data compression method to reduce the communication data size. These methods not only reduce the communication time, but also minimize the computation time for the image composition.

For the communication scheme, the binary-swap (BS) method proposed by Ma *et al*. [16-17] is a well-known divide-and-conquer algorithm. The key idea of the binary-swap method is that in each communication step, two processors are paired to exchange half of their sub-images and to composite sub-images they received. After $\log P$ communication steps, each processor contains a portion of the final image, where P is the number of processors. The final image then can be obtained by gathering each potion of the final image from processors. The advantage of this method is that it enables more parallelism in the image composition stage, and keeps all processors busy in all communication steps of the image composition process. However, this method can be only implemented when the number of processors is a power of two.

The parallel-pipelined (PP) method proposed by Lee [13] is another composition method. In the PP method, the sub-image owned by each processor is first divided into P blocks, where P is the number of processors. The topology of processors is treated as a ring. In each communication step, a processor sends one block to its next processor and receives a block from its previous processor. The block received by each processor is then composited using "over" operation. After $P-1$ communication steps, each processor holds one block of the final image. The final image then can be obtained by gathering each block of the final image from processors. The advantage of the PP method is that it can be implemented with arbitrary number of processors. The disadvantage is that each processor needs $P-1$ communication steps to perform the image composition. When P is large, the communication overhead is high.

In the image composition stage, the size of a partitioned sub-image and the number of communication steps affect the composition time. The binary-swap method and the parallel-pipelined method used a fixed way to set these two parameters. They did not optimize these two parameters. In this paper, we present an efficient method, the *rotate-tiling* (RT), to address this issue. Given P processors, in the RT method, a sub-image owned by a processor can be divided into N blocks initially, where $N = 1, \ldots, P$. In each communication step, a processor sends (receives) some blocks to (from) other processors according to a mathematical formula. After the send and receive operations, each processor uses "over" operation to composite blocks it received. Then each block in a processor is divided into two equal halves. Continue the above process, after $\lceil \log P \rceil$ communication steps, each processor contains a block of the final image. The final image then can be obtained by gathering each block of the final image from processors. The only restriction of the RT method is that the value of $P \times N$ is even. According to the values of P and N, the RT method can be divided into two categories, the 2N_RT method for arbitrary number of processors and the N_RT method for even number of processors. In the RT method, different values of N lead to different performance. To analyze the effect of N in the RT method, the composition time is parameterized by the number of processors, the number of initial blocks of a sub-image, the startup time, and the data transmission time. By fixing the number of processors, we can determine the value of N that has the best performance.

Many data compression methods were proposed to reduce the image composition time of a volume rendering system on distributed memory multicomputers [1, 11, 13, 16]. Ma *et al*. [16] suggested using a bounding rectangle of non-blank pixels for each image and only compositing the pixels within the intersection of the bounding rectangles. Lee [13] used the bounding rectangle of a parallel-pipelined composition algorithm. Using the bounding rectangle method, the performance of test data can have 20 to 50 percent improvement. Lacroute and Levoy [1, 11] used the run-length encoding method for a serial volume renderer and promoted the run-length encoding as a general technique applicable to many parallel image composition algorithms. It is suitable for monochrome images. For a gray color image, the compression ratio of the run-length encoding method is low since a gray color image contains many colors. To increase the compression ratio of a gray color image, we propose another data compression method, the template run-length encoding (TRLE). In TRLE method, it uses 16 templates to encode 2×2 pixels. The TRLE codes are used to encode sub-images. A TRLE code is one byte long. The lower four bits of a TRLE code is represented as the template of four pixels. The upper four bits of a TRLE code is represented as the copies of the same template. A bit operation is used to encode and decode the TRLE codes. Hence, the TRLE compression method is easy to be implemented and has a high compression ratio.

To evaluate the performance of the RT method, we compare the proposed method with the binary-swap method and the parallel-pipelined method. Both theoretical analysis and experimental test are conducted. In the theoretical analysis, we analyze the best performance bound of the RT method in terms of the startup time, the data transmission time, the number of processors, and the number of initial block of a sub-image. In the experimental test, we have implemented these threes methods on an SP2 parallel machine. Three volume datasets are used as test samples. The experimental results show that our method outperform the binary-swap and the parallel-pipelined methods for all test samples and match the results analyzed in the theoretical analysis. For the TRLE method, the experimental results show that the TRLE method can further reduce the composition time for these three methods.

The rest of the paper is organized as follows. In Section 2, we present and analyze the RT method for the image composition of volume rendering. We introduce the TRLE data compression method in Section 3. In Section 4, we compare the performance of the proposed composition method with the binary-swap and parallel-pipelined composition method on an SP2 parallel machine. We also evaluate the performance of these three methods with the TRLE method.

2. The RT method for the image composition of volume rendering

According to the number of processors (P) and the number of initial blocks (N) of a sub-image, the RT method can be classified into two cases. One is the $2N$ rotate-titling ($2N$_RT) method for arbitrary number of processors. The other is the N rotate-tiling (N_RT) method for even number of processors. If the number of processors and the number of initial blocks of a sub-image are odd, the RT method cannot be applied. In the following, we describe the proposed method and analyze the communication and computation costs of the proposed method along with the binary-swap and the parallel-pipelined methods.

2.1 The 2*N* rotate-tiling (2*N*_RT) method for any number of processors

In this case, the number of processors (P) can be an arbitrary positive integer and the number of initial blocks of a sub-image ($2N$) is even. To composite sub-images of processors to form a final image, in the $2N$_RT method, each sub-image in a processor is first partitioned into $2N$ blocks with equal size. Processors, then, send and receive blocks to and from other processors according to the *send-receive equations* as given in Equations (1) and (2). Once a processor sent and received blocks, it composites the received blocks. After the composition, it divides each block into two equal halves and repeats the send and receive blocks process until the final image is composited. In the following, we assume that, in the kth communication step, P_r sends block $A_r^k(m)$ to P_i and receives block $A_j^k(n)$ from P_j, where r, i, and j are processor ranks; k is a positive integer; and m, n are block numbers. The send equation is given below,

$$P_r(A_r^k(m)) \rightarrow P_i, \text{ where } \begin{cases} r = 0,1,\ldots,P-1, \\ i = |P\times l - r - k - 1| \bmod P, \text{ for } l = 1,2,\ldots,N/2, \\ m = (P + r + 1 + 2^{k-1}) \bmod 2^l, \text{ for } l = 1,2,\ldots,N/2. \end{cases} \qquad (1)$$

The receive equation is given below,

$$P_r \leftarrow P_j(A_j^k(n)), \text{ where } \begin{cases} r = 0,1,\ldots,P-1, \\ j = |P\times l - r - k + 1| \bmod P, \text{ for } l = 1,2,\ldots,N/2, \\ n = (P + r + 2^{k-1}) \bmod 2^l, \text{ for } l = 1,2,\ldots,N/2. \end{cases} \qquad (2)$$

The algorithm of the $2N$_RT method is given as follows.

```
Algorithm 2N_Rotate-Tiling_Method(P, A, N)
/* P is the number of processors. */
/* A is the initial image of each processor. */
/* N is the even number of partitions of an image. */
1. Each processor partitions its sub-image A into N blocks;
2. for k = 1 to ⌈log P⌉ do {
3.    for each processor P_r do parallel {
4.       P_r sends block A_r^k(m) to Pi using Equation (1);
5.       P_r receives block A_j^k(n) from P_j using Equation (2);
6.       P_r composites the received A_j^k(n) with its local
         block A_r^k(n);
7.    }
8.    Divide each block into two equal halves;
9. }
end_of_2N_Rotate-Tiling_Method
```

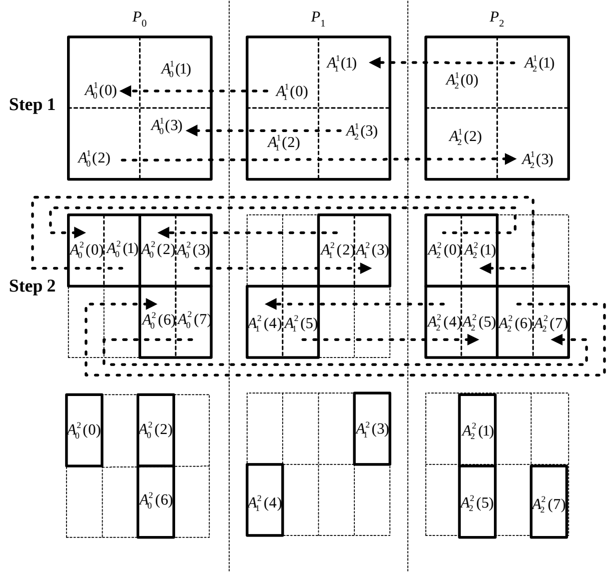


Figure 1: Example of the 2*N*_RT method using three processors and *N* = 4.

An example is given in Figure 1 to illustrate the $2N$_RT method using three processors and four initial blocks. In Figure 1, there are $\lceil \log P \rceil = \lceil \log 3 \rceil = 2$ communication steps. In the first communication step, according to Equations (1) and (2), P_0 sends $A_0^1(2)$ to P_2 and receives blocks $A_1^1(0)$ and $A_1^1(3)$ from P_1, P_1 sends blocks $A_1^1(0)$ and $A_1^1(3)$ to P_0 and receives block $A_2^1(1)$ from P_2, and P_2 sends block $A_2^1(1)$ to P_1 and receives block $A_0^1(2)$ from P_0. After blocks were received, each processor uses the "over" operation to composite the received blocks with its local blocks. For example, P_0 composites received blocks $A_1^1(0)$ and $A_1^1(3)$ with its local blocks $A_0^1(0)$

and $A_0^1(3)$, respectively. Then each block in a processor is divided into two equal halves. In the given example, each processor contains 8 blocks now.

In the second communication step, a similar process performed in the first communication step is continuing. After performing the second communication step, blocks $A_0^2(0)$, $A_0^2(2)$ and $A_0^2(6)$ in P_0, blocks $A_1^2(3)$ and $A_1^2(4)$ in P_1, and blocks $A_2^2(1)$, $A_2^2(5)$ and $A_2^2(7)$ in P_2 form the final image. P_0 uses the "gather" operation to collect these blocks to get the final image.

2.2 The *N* rotate-tiling (*N*_RT) method for the even number of processors

In this case, the number of processors (P) is even and the number of initial blocks of a sub-image (N) can be an arbitrary positive integer. To composite sub-images of processors to form a final image, in the N_RT method, each sub-image in a processor is first partitioned into N blocks with equal size. Processors, then, send and receive blocks to and from other processors according to the send-receive equations shown in Equations (3) and (4). Once a processor sent and received blocks, it composites the received blocks. After the composition, it divides each block into two equal halves and repeats the send and receive blocks process until the final image is composited. In the following, we assume that, in the kth communication step, P_r sends block $A_r^k(m)$ to P_i and receives block $A_j^k(n)$ from P_j, where r, i, and j are processor ranks; k is a positive integer; and m, n are block numbers. The send equation is given below,

$$P_r(A_r^k(m)) \rightarrow P_i, \text{ where } \begin{cases} r = 0, 1, \ldots, P-1, \\ i = (P \times l - r - k) \bmod P, \text{ for } l = 1, 2, \ldots, N, \\ m = 2 \times l - 1, \text{ for } l = 1, 2, \ldots, N. \end{cases} \quad (3)$$

The receive equation is given below,

$$P_r \leftarrow P_j(A_j^k(n)), \text{ where } \begin{cases} r = 0, 1, \ldots, P-1, \\ j = (P \times l + r + k) \bmod P, \text{ for } l = 1, 2, \ldots, N, \\ n = 2 \times (l-1), \text{ for } l = 1, 2, \ldots, N. \end{cases} \quad (4)$$

The algorithm of the N_RT method is given as follows.

```
Algorithm N_Rotate-Tiling_Method(P, A, N)
/* P is the even number of processors. */
/* A is the initial image of each processor. */
/* N is the number of partitions of an image. */
1. Each processor partitions its sub-image A into N blocks;
2. for k = 1 to ⌈log P⌉ do {
3.    for each processor P_r do parallel {
4.          P_r sends block $A_r^k(m)$ to Pi using Equation (3);
5.          P_r receives block $A_j^k(n)$ from P_j using Equation (4);
6.          P_r composites the received $A_j^k(n)$ with its local
             block $A_r^k(n)$;
7.    }
8.    Divide each block into two equal halves;
9. }
end_of_N_Rotate-Tiling_Method
```

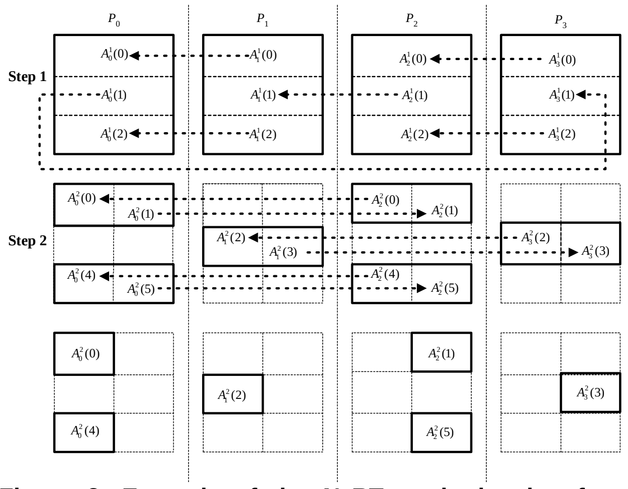


Figure 2: Example of the *N*_RT method using four processors and *N* = 3.

Figure 2 illustrates the N_RT method using four processors and three initial blocks. The description for Figure 2 is similar to that of Figure 1 except that the N_RT method uses Equations (3) and (4) to determine the send and receive processors and blocks.

2.3 Theoretical analysis of the RT and other composition methods

In this subsection, we first derive the theoretical performance of the BS, PP, and RT methods. Then, we analyze the effect of the number of initial blocks for the RT method. A summary of the notations used in this theoretical analysis is given below.

- $T_{comm}(M)$ – The total communication time of method M.
- $T_{comp}(M)$ – The total computation time (over operation) of method M.
- T_s – The startup time of a communication channel.
- T_p – The data transmission time per byte.
- T_o – The computation time of the "over" operation per pixel.
- P – The number of processors.
- A – The image size in pixels.
- $A_k(M)$ –The block size in pixels of the k-th communication steps of method M.
- $S(M)$ – The number of communication steps of method M.
- N – The number of initial blocks of a sub-image.

We summarize the number of communication steps, the block sizes sent or received by a processor in each communication step, the total computation time, and the total communication time in Table 1.

Table 1: The theoretical performance comparisons.

M	$S(M)$	$A_k(M)$	T_{comm}	T_{comp}
BS	$\log P$	$\frac{A}{2^k}$	$\sum_{k=1}^{\log P}(T_s + \frac{A}{2^k} \times T_p)$	$\sum_{i=1}^{\log P}(\frac{A}{2^i} \times T_o)$
PP	$P-1$	$\frac{A}{P}$	$\sum_{k=1}^{P-1}(T_s + \frac{A}{P} \times T_p)$	$\sum_{i=1}^{P-1}(\frac{A}{P} \times T_o)$
$2N\_RT$	$\lceil \log P \rceil$	$\frac{A}{N \cdot 2^{k-1}}$	$\sum_{k=1}^{\lceil \log P \rceil}\left(k \times T_s + (\frac{k \times A}{N \cdot 2^{k-1}}) \times T_p\right)$	$\sum_{i=1}^{\lceil \log P \rceil}\frac{k \cdot A}{N \cdot 2^{k-1}} \times T_o$
N_RT	$\lceil \log P \rceil$	$\frac{A}{N \cdot 2^{k-1}}$	$\sum_{k=1}^{\lceil \log P \rceil}\left(\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right)(T_s + \frac{A \cdot T_p}{N \cdot 2^{k-1}})\right)$	$\sum_{i=1}^{\lceil \log P \rceil}(\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) \times \frac{A}{N \cdot 2^{k-1}} T_o)$

The composition time of the $2N$_RT method T_{2N_RT} is the sum of $T_{comm}(2N\_RT)$ and $T_{comp}(2N\_RT)$, that is, $T_{N\_RT} = T_s \cdot N^{\lceil \log P \rceil} + \frac{A}{N}(T_p + T_o \cdot \lceil \log P \rceil \times (1-(\frac{1}{2})^{\lceil \log P \rceil})) \times (1-(\frac{1}{2})^{\lceil \log P \rceil})$. The composition time is parameterized by the number of processors, the number of initial blocks of a sub-image, the startup time, and the data transmission time. To find the performance bound of N, we compare the composition time when the numbers of initial blocks are $2N$ and $2(N+1)$, respectively. Let $T_{2N\_RT}(2N)$ and $T_{2N\_RT}(2(N+1))$ represent the composition time when the numbers of initial blocks are $2N$ and $2(N+1)$, respectively. We have $T_{2N\_RT}(2N) = T_s \cdot N^{\lceil \log P \rceil} + \frac{A}{N}(T_p + T_o \cdot \lceil \log P \rceil \cdot \times (1-(\frac{1}{2})^{\lceil \log P \rceil})) \times (1-(\frac{1}{2})^{\lceil \log P \rceil})$ and $T_{2N\_RT}(2(N+1)) = T_s \cdot (N+2)^{\lceil \log P \rceil} + \frac{A}{N+2}(T_p + T_o \cdot \lceil \log P \rceil \times (1-(\frac{1}{2})^{\lceil \log P \rceil})) \times (1-(\frac{1}{2})^{\lceil \log P \rceil})$. The difference of $T_{2N\_RT}(2N)$ and $T_{2N\_RT}(2(N+1))$ is $\Delta T = T_s(N^{\lceil \log p \rceil} - (N+2)^{\lceil \log p \rceil}) + \frac{2A}{N(N+2)} \times \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right) \times \left(T_p + T_o \cdot \lceil \log P \rceil \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right)$. By setting $\Delta T < 0$, we have $T_s((N+2)^{\lceil \log p \rceil} - N^{\lceil \log p \rceil}) < \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right) \times \frac{2A}{N(N+2)}\left(T_p + T_o \cdot \lceil \log P \rceil \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right) \times$. By using the following equation, we can determine the performance bound of N,

$$N(N+2)\left((N+2)^{\lceil \log p \rceil} - N^{\lceil \log p \rceil}\right) < \frac{2A}{T_s}\left(T_p + T_o \cdot \lceil \log P \rceil \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right) \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right). \quad (5)$$

For example, assume that the values of P, T_s, T_p, and T_o are 32, 0.005, 0.00004, and 0.0002, respectively. According to Equation (5), the performance bound of N is 4.3. It means that when N is equal to 4.3, the $2N$_RT method can produce the best performance than other values of N. Since N must be an even number, N is equal to 4 for this example.

In the N_RT method, to find the performance bound of N, we compare the composition time when the numbers of initial blocks are N and $N+1$, respectively. Let $T_{N\_RT}(N)$ and $T_{N\_RT}(N+1)$ represent the composition time when the numbers of initial blocks are N and $N+1$, respectively. We have $T_{N\_RT}(N) = T_s \cdot N^{\lceil \log P \rceil} + \frac{A}{N}(T_p + T_o \cdot \lceil \log P \rceil)(1-(\frac{1}{2})^{\lceil \log P \rceil})$ and $T_{N\_RT}(N+1) = T_s \times (N+1)^{\lceil \log P \rceil} + \frac{A}{N+1}(T_p + T_o \cdot \lceil \log P \rceil)(1-(\frac{1}{2})^{\lceil \log P \rceil})$. The difference of $T_{N\_RT}(N)$ and $T_{N\_RT}(N+1)$ is $\Delta T = T_N - T_{N+1} = T_s(N^{\lceil \log p \rceil} - (N+1)^{\lceil \log p \rceil}) + \frac{2A}{N(N+1)} \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right) \times \left(T_p + T_o \cdot \lceil \log P \rceil \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right)$. By setting $\Delta T < 0$, we have $T_s((N+1)^{\lceil \log p \rceil} - N^{\lceil \log p \rceil}) < \frac{2A}{N(N+1)} \times \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right) \times \left(T_p + T_o \cdot \lceil \log P \rceil \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right)$. By using the following equation, we can determine the performance bound of N,

$$N(N+1)\left((N+1)^{\lceil \log p \rceil} - N^{\lceil \log p \rceil}\right) < \frac{2A}{T_s}\left(T_p + T_o \cdot \lceil \log P \rceil \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right)\right) \cdot \left(1-(\frac{1}{2})^{\lceil \log P \rceil}\right) \quad (6)$$

For example, assume that the values of P, T_s, T_p, and T_o are 32, 0.005, 0.00004, and 0.0002, respectively. According to Equation (6), the performance bound of N is 3.4. It means that when N is equal to 3.4, the N_RT method can produce the best performance than other values of N. Since $N \geqq 1$, N is equal to 3 for this example.

3. The TRLE compression Method

Data compression is another way to reduce the communication time for the image composition of volume rendering. To couple a composition method with an efficient data compression method, the performance of a composition method can be further enhanced. However, data compression requires extra computation. The less time a data compression scheme takes, the better the performance. In the following, we introduce the proposed data compression method in detail.

The run-length encoding (RLE) method is a compression technique that reduces file sizes, especially for black-and-white images. In the RLE method, it uses a single character to represent continuous pixels with the same color (black or white). The more runs there are and the longer the run sequence, the greater the compression. However, in gray images, the run-length encoding is not efficient since the pixels' values are more varied.

For this reason, we propose another data compression method, the template run-length encoding (TRLE) method, to efficiently compress gray images. In the TRLE method, we use templates to encode data. A template is a 2×2 pixels. There are total 16 templates used in the TRLE method and their corresponding codes are given in Figure 3. In the TRLE method, a gray image is represented by TRLE codes. The lower four bits of a TRLE code represent the code of a template. The higher four bits of a TRLE code represent the number of times of the same template replication. The maximal number of the replication template can be represented in a TRLE code is 16. Since the TRLE method uses the lower four bits to indicate the templates, the TRLE codes can be easily construct by the bit operation. As a result, the TRLE method is more efficient since it requires less computation time.

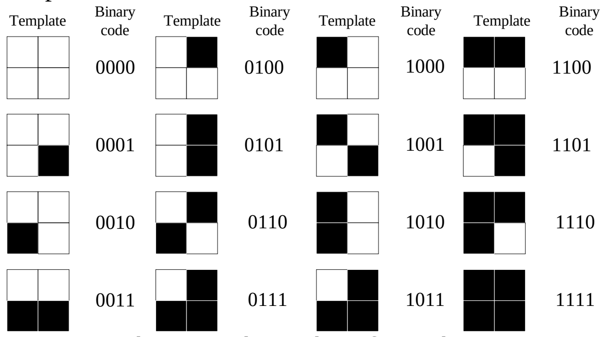


Figure 3: The codes of templates.

We now give an example to compare the compression ratio of the RLE method and the TRLE method. Figure 4 illustrates two scanlines with twenty-four pixels. If the RLE method is used, the RLE code for the first scanline is 121113111, and 121112211 for the second scanline. The total size of the RLE codes is 18 bytes. If we use the TRLE method, the TRLE codes for the scanlines are 5 26 15 8 10. The total size of the TRLE codes is 5 bytes. In this example, the compression ratio of the RLE method and the TRLE method is 18:5. The TRLE method results in a better compression ratio than the RLE method.

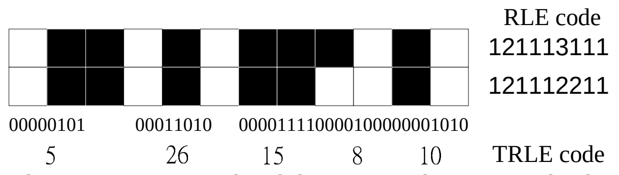


Figure 4: An example of the RLE and TRLE methods.

4. Experimental results

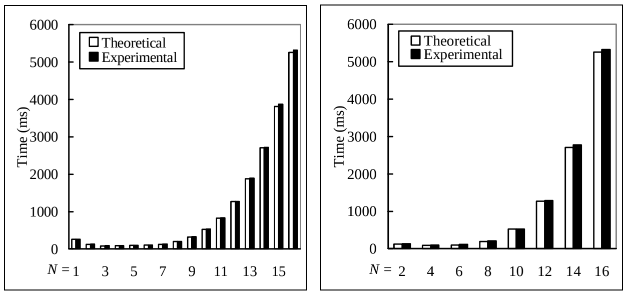
To evaluate the performance of the RT method, we have implemented the RT method along with the BS method and the PP method on an SP2 parallel machine [6]. The SP2 parallel machine is located at the National Center of High Performance Computing (NCHC) in Taiwan. This super-scalar architecture uses an IBM RISC System/6000 POWER2 CPU with a clock rate of 66.7 MHz. There are 40 IBM POWER2 nodes in this system, and each node has a 128KB first-level data cache, a 32KB first-level instruction cache, and 128MB of memory space. Each node is connected to a low-latency, high-bandwidth interconnection network called the High Performance Switch (HPS).

The volume rendering system consists of three main stages: the data partitioning stage, the rendering stage, and the image composition stage. To implement these composition methods, in the data partitioning stage, we use the efficient 1-D and 2-D partitioning schemes [15] to distribute the volume dataset to processors. In the render stage, each processor uses the shear-warp factorization [11] to generate a partial image. In the image composition stage, the partial images are composited using these composition methods to form a final image.

Three volume datasets are used to evaluate the performance of these composition methods. They are selected from the Chapel Hill Volume Rendering Test Dataset [11]. The first test sample is an "engine" dataset, which is the CT scan of an engine block. The second test sample is a "brain" dataset generated from the MR scan of a human head. The third test sample is a CT "head" dataset. Each image is grayscale and contains 512 × 512 pixels.

4.1 Performance comparisons of three composition methods

In this section, we first evaluate the performance of the RT method. Then we compare the performance of the RT method with the BS method and the PP method.



(a) The *N*_RT method (b) The 2*N*_RT method

Figure 5: The theoretical and the experimental composition time of the *N*_RT and the 2*N*_RT methods for test sample "engine" with various numbers of initial blocks of a sub-image on 32 processors.

Figure 5(a) and Figure 5(b) show the theoretical and the experimental composition time of the *N*_RT and the 2*N*_RT methods for test sample "engine" with various numbers of initial blocks of a sub-image on 32 processors, respectively. From Figure 5(a), we can see that the theoretical performance is close to the experimental performance. Therefore, the theoretical analysis matches the experimental analysis for the test sample. In Figure 5(a), when the number of initial blocks is equal to 3, the *N*_RT method can produce the best theoretical and experiment performance for the test sample. For the performance of the 2*N*_RT method illustrated in Figure 5(b), we have similar observations as those of the *N*_RT method and when the number of initial block is equal to 4, the 2*N*_RT method can produce the best theoretical and experiment performance for the test sample. For test samples “head” and “brain”, we have similar results.

According to the results shown in Figure 5, Figure 6 shows the theoretical and the experimental composition time of the BS, PP, 2*N*_RT and *N*_RT methods for test sample “engine” on 32 processors. The numbers of initial blocks of the 2*N*_RT and *N*_RT methods are 4 and 3, respectively. From Figure 6, we observe that the *N*_RT method has much better performance than the BS and PP methods. The reason is that the *N*_RT method can find the best performance point ($N = 3$) for the startup time, the data transmission time, and the computation time. In the BS method, it minimizes the startup time, but do not reduce the data transmission time and the computation time. For the PP method, it decreases the data transmission time and the computation time, but do not reduce the startup time. The performance of the *N*_RT method is slightly better than that of the 2*N*_RT method. The reason is that when the number of processors is even, the best performance point found by the *N*_RT method is more accurate than that of the 2*N*_RT method because the numbers of initial blocks used by the *N*_RT method and the 2*N*_RT method can be an arbitrary number and an even number, respectively. For test samples “head” and “brain”, we have similar observations for these four methods.

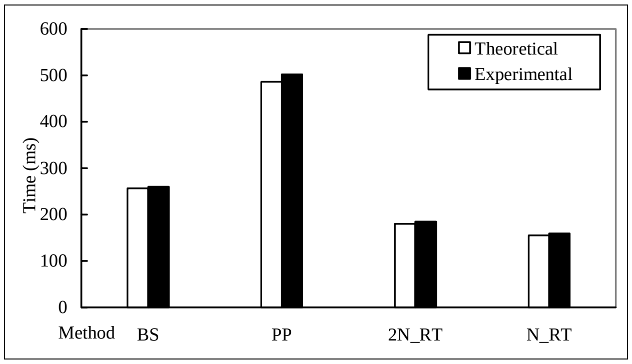
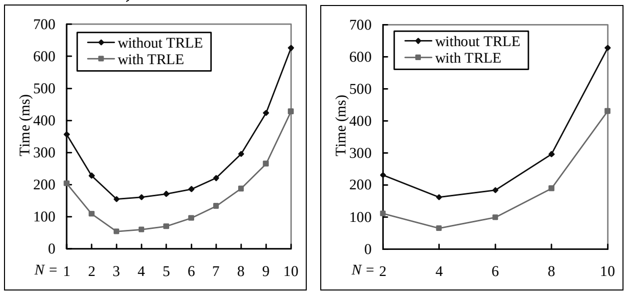


Figure 6: The theoretical and the experimental composition time of the BS, PP, 2*N*_RT, and *N*_RT methods for test sample “engine” on 32 processors. The numbers of initial blocks of the 2*N*_RT and *N*_RT methods are 4 and 3, respectively.

4.2 Performance improvements of the composition methods with TRLE

Figure 7(a) and Figure 7(b) show the theoretical and the experimental composition time of the *N*_RT and the 2*N*_RT methods with and without TRLE method for test sample “engine” for various numbers of initial blocks of a sub-image on 32 processors, respectively. In Figure 7(a), when the number of initial blocks is equal to 3, the *N*_RT method with TRLE method can produce the best performance for the test sample. For the performance of the 2*N*_RT method illustrated in Figure 7(b), we have the best performance for the test sample when the number of initial blocks is equal to 4. From Figure 7, we can see that the TRLE method did improve the performance of the RT method a lot. For test samples “head” and “brain”, we have similar results.

Figure 8 shows the theoretical and the experimental composition time of the BS, PP, 2*N*_RT and *N*_RT methods with and without the RLE method and the TRLE method for test sample “engine” on 32 processors. The numbers of initial blocks of the 2*N*_RT and *N*_RT methods are 4 and 3, respectively. From Figure 8, we can see that both the TRLE method and the RLE method can reduce the composition time for the BS, PP, 2*N*_RT and *N*_RT methods. However, the composition methods with the TRLE method have better performance than the methods with the RLE method. For test samples “head” and “brain”, we have similar observations.



(a) The *N*_RT method (b) The 2*N*_RT method

Figure 7: The composition time of the RT method with and without the TRLE method for test sample “engine” with various numbers of initial blocks of a sub-image on 32 processors.

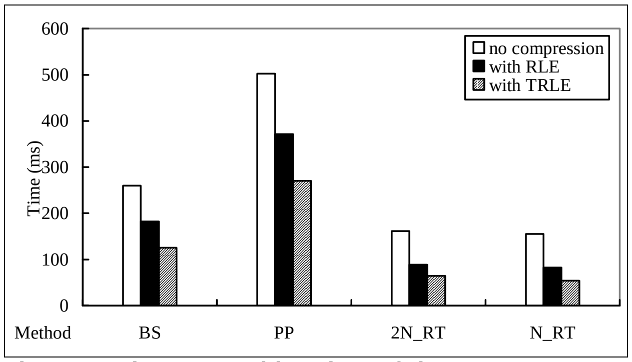


Figure 8: The composition time of the BS, PP, 2*N*_RT, and *N*_RT methods with and without the RLE and the TRLE methods for test sample “engine” on 32 processors. The numbers of initial blocks of the 2*N*_RT and *N*_RT methods are 4 and 3, respectively.

5. Conclusions

In this paper, we have proposed an efficient method, the rotate-tiling (RT) method, for the image composition of volume rendering on distributed memory multicomputers. According to the number of processors and the number of initial blocks of a sub-image, the RT method can be classified into two categories, the 2*N*_RT method for arbitrary number of processors and the *N*_RT method for even number of processors. We also devised a data compression method, the template run length encoding (TRLE) method, to improve the performance further. The proposed methods have been implemented on an IBM SP2 machine along with the binary-swap and the parallel-pipelined methods for performance evaluation. Both theoretical analysis and experimental test were conducted. The performance results show that the RT method has better performance than that of the binary-swap and parallel-pipelined methods. The experimental results also show that a composition method with the TRLE method can produce better performance than that without the TRLE method.

References

[1] J. Ahrens and J. Painter, "Efficient Soft-Last Rendering Using Compression-Based Image Compositing," *Proceedings of the second Eurographics Workshop on Parallel Graphics and Visualization*, 1998.

[2] M. Cox and P. Hanrahan, "Pixel Merging for Object-Parallel Rendering: A Distributed Snooping Algorithm," *Proceedings of 1993 Parallel Rendering Symp.*, pp. 49-56, New York, 1993.

[3] R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering," *Proceedings of SIGGRAPH'88*, vol 22, no. 4, pp.65-74, Atlanta, 1988.

[4] E. Groeller, W. Purgathofer, "Coherence in Computer Graphics," *Technical Reports TR-186-2-95-04*, Institute of Computer Graphics 186-2 Technical University of Vienna, Mar 1995.

[5] W.M. Hsu, "Segmented Ray Casting for Data Parallel Volume Rendering," *In Proceedings of 1993 Parallel Rendering Symposium*, pp. 7-14, San Jose, Oct. 1993.

[6] IBM, IBM AIX Parallel Environment, Parallel Programming Subroutune Reference.

[7] A. Kaufman (Eds.), Volume Visualization, *IEEE Computer Society Press*, 1991.

[8] P. Lacroute, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation," *PhD dissertation*, Stanford University, 1995.

[9] P. Lacroute, "Real-time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization," *In Proceedings of 1995 Parallel Rendering Symposium*, pp. 15-22, Atlanta, Oct. 1995.

[10] P. Lacroute, "Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 218-231, 1996.

[11] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation," *Computer Graphics* (*In Proceedings of SIGGRAPH '94*), pp. 451-458, Orlando, July 1994.

[12] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics* (*In Proceedings of SIGGRAPH '91*), vol. 25, pp. 285-288, Las Vegas, July 1991.

[13] Tong-Yee Lee, "Image Composition Schemes for Soft-Last Polygon Rendering on 2D Mesh Multicomputers," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 3, pp. 202-217, Sep. 1996.

[14] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, vol. 9, no. 3, pp. 245-261, July 1990.

[15] Chin-Feng Lin, Don-Lin Yang, and Yeh-Ching Chung, "Parallel Shear-Warp Factorization Volume Rendering Using Efficient 1-D and 2-D Partitioning Schemes on Distributed Memory Multicomputers," *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 555-560, Nov. 1999, USA.

[16] K.L. Ma, J.S. Painter, C.D. Hansen, and M.F. Krogh, "A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering," *In Proceedings of 1993 Parallel Rendering Symposium*, pp. 15-22, San Jose, Oct. 1993.

[17] K.L. Ma, J.S. Painter, C. D. Hansen, and M.F. Krogh, "Parallel Volume Rendering Using Binary-Swap Composition," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 59-68, July 1994.

[18] MPI Forum, MPI: A Message-Passing Interface Standard, May 1994.

[19] T. Porter and T. Duff, "Composition Digital Images," *Computer Graphics* (*In Proceedings of SIGGRAPH'84*), volume 18, pp. 253-259, July 1984.

[20] Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayasi, and Tadao Nakamura. "Parallel Processing of the Shear-Warp Factorization with the Binary-Swap Method on a Distributed-Memory Multiprocessor System", *In Proceedings of 1997 Parallel Rendering Symposium* (*PRS'97*), October 20-21, 1997.

[21] J.P. Singh, A. Gupta, and M. Levoy, "Parallel Visualization Algorithms: Performance and Architectural Implications," *Computer*, vol. 27, no. 7, pp. 45-55, July 1994.

[22] C. Upson, and M. Keeler, "V-BUFFER: Visible Volume Rendering," *Computer Graphics* (*In Proceedings of SIGGRAPH'88*), vol. 22, no. 4, pp. 59-64, Atlanta, 1988.

[23] L. Westover, "Footprint Evaluation for Volume Rendering", *Computer Graphics* (*In Proceedings of SIGGRAPH'90*), vol.24, pp. 367-376, Dallas, 1990.

[24] J. Wilhelms and A. Van Gelder, "A Coherent Projection Approach for Direct Volume Rendering," *Computer Graphics* (*In Proceedings of SIGGRAPH'91*), vol. 25, no. 4, pp. 275-283, July 1991.

[25] C.M. Wittenbrink and A.K. Somani, "Permutation Warping for Data Parallel Volume Rendering," *In Proceedings of 1993 Parallel Rendering Symposium*, pp. 57-60, San Jose, Oct. 1993.

[26] T.S. Yoo, U. Neumann, H. Fuchs, S.M. Pizer, T. Cullip, J. Rhoades, and R. Whitaker, "Direct Visualization of Volume Data," *IEEE Computer Graphics & Applications*, vol. 12, no. 4, pp. 63-71, July 1992.