The Journal of Supercomputing, 22, 277–302, 2002

Parallel Shear-Warp Factorization Volume Rendering Using Efficient 1-D and 2-D Partitioning Schemes for Distributed Memory Multicomputers*

CHING-FENG LIN, DON-LIN YANG,† AND YEH-CHING CHUNG {cflin, dlyang, ychung}@fcu.edu.tw

Department of Information Engineering, Feng Chia University, Taichung 407, Taiwan

Abstract. 3-D data visualization is very useful for medical imaging and computational fluid dynamics. Volume rendering can be used to exhibit the shape and volumetric properties of 3-D objects. However, volume rendering requires a considerable amount of time to process the large volume of data. To deliver the necessary rendering rates, parallel hardware architectures such as distributed memory multicomputers offer viable solutions. The challenge is to design efficient parallel algorithms that utilize the hardware parallelism effectively. In this paper, we present two efficient parallel volume rendering algorithms, the 1D-partition and 2D-partition methods, based on the shear-warp factorization for distributed memory multicomputers. The 1D-partition method has a performance bound on the size of the volume data. If the number of processors is less than a threshold, the 1D-partition method can deliver a good rendering rate. If the number of processors is over a threshold, the 2D-partition method can be used. To evaluate the performance of these two algorithms, we implemented the proposed methods along with the slice data partitioning, volume data partitioning, and sheared volume data partitioning methods on an IBM SP2 parallel machine. Six volume data sets were used as the test samples. The experimental results show that the proposed methods outperform other compatible algorithms for all test samples. When the number of processors is over a threshold, the experimental results also demonstrate that the 2D-partition method is better than the 1D-partition method.

Keywords: volume rendering, data partitioning, image compositing, shear-warp factorization, distributed memory multicomputer

1. Introduction

Volume rendering [7] can be used to analyze the shape and volumetric property of three-dimensional objects for medical imaging and computational fluid dynamics. Volume rendering can display semi-opaque objects and provide better visualization of the surface of an object. Three-dimensional scanner devices such as CT and MRI can acquire the three-dimensional image data in machine-readable form. Volume rendering is a popular technique for medical imaging used to understand objects by analyzing the large amount of empirical data obtained from measurements or simulations [24].

* This work was partially supported by the NSC of ROC under contract NSC89-2213-E-035-032.
† Corresponding author.

However, most volume rendering methods that produce effective visualizations are computation intensive [19]. It is very difficult for them to achieve interactive rendering rates for the large amount of volume data. Even with the latest volume rendering acceleration technique advance [8, 9], a few minutes or possibly hours is required to render the images on a single processor machine. In addition, the volume data is too large to be held in the memory of a single processor element. One way to solve the above problems is to parallelize the serial volume rendering techniques onto distributed memory multicomputers [23].

The shear-warp factorization volume rendering method, proposed by Lacroute et al. [11], is the fastest volume rendering algorithm. Paralleling the shear-warp factorization volume rendering algorithm onto a distributed memory multicomputer consists of three stages: the data partitioning stage, the shear-warp rendering stage and the image compositing stage. In the data partitioning stage, an efficient data partitioning method is used to distribute the volume data to processors. In the shear-warp rendering stage, each processor uses the shear-warp factorization volume rendering method to generate a partial final image. In the image compositing stage, the partial final images are composited to form a final image. In this paper, we focus on the data partitioning stage.

Many parallel shear-warp factorization volume rendering methods were proposed in the literature [1, 18]. Sano et al. [18] proposed a slice data partitioning method for distributing volume data to processors in the data partitioning stage. The binary-swap method was used [15] to produce the final image in the image compositing stage. The drawbacks of the slice volume partitioning method are the number of processors is restricted to a power of two and considerable time is required to composite the partitioned sub-volume images during the binary-swap process.

Amin et al. [1] presented a volume data partitioning method for distributing volume data to processors in the data partitioning stage. The partial intermediate images produced by this method have overlapped areas. In the image compositing stage, this method requires extra communication and computation overhead to use the *over* operation for compositing the partial final images. To improve the performance of the volume data partitioning method, they presented a sheared volume data partitioning method for distributing the volume data to the processors in the data partitioning stage. This method produces partial intermediate images without overlapped areas. In the image compositing stage, a simple merge operation is used to produce a final image. The overhead is small. However, in the sheared volume data partitioning method, the volume data is not evenly distributed to the processors. The processor load imbalance may increase the computation time in the shear-warp rendering stage.

To solve the problems stated above, we present efficient parallel one-dimensional and two-dimensional partition shear-warp factorization volume rendering methods for distributed memory multicomputers. For the parallel one-dimensional partition shear-warp factorization volume rendering method (the 1D-partition method for short), in the data partitioning stage, we developed an 1-D partitioning scheme to partition the volume data based on the mathematical formula derived from the number of processors and the viewing angle. The 1-D partitioning method not only reduces the computation time and the communication overhead, but also achieves

load balancing. In the shear-warp rendering stage, we used the shear-warp method proposed in [8] to render the sub-volume and generate the partial final images independently. In the image compositing [17] stage, because the partial final images do not have overlapped areas, a simple merge operation is used for assembling the partial final images into a final image.

According to the performance analysis of the 1D-partition method, we found that the speedup of the 1D-partition method does not increase when the number of processors is over a threshold. Therefore, we present a parallel two-dimensional partition shear-warp factorization volume rendering method (2D-partition method for short) for the case where the number of processors is over a threshold. For the 2D-partition method, in the data partitioning stage, we first decided the number of horizontal and vertical processors. We then developed a 2-D partitioning scheme to partition the volume data based on the mathematical formula derived from the number of horizontal and vertical processors and the viewing angle. In the shear-warp rendering stage, the shear-warp volume rendering method is used to render the sub-volume and generate the partial warped intermediate images independently. In the image compositing stage, the *over* operation is applied first for compositing the partial warped intermediate images from the vertical processors to form the partial final images. A simple merge operation, similar to the image compositing stage of the 1D-partition method, is then used for assembling the partial final images into a final image.

To evaluate the performance of the proposed methods, we implemented them along with the slice data partitioning [18], volume data partitioning [1], and sheared volume data partitioning [1] methods on an IBM SP2 parallel machine. The experimental results show that the 1D-partition and 2D-partition methods outperform the methods proposed in [1] and [18]. The experimental results also show that the 2D-partition method has better performance than the 1D-partition method when the number of processor is over a threshold.

The remainder of this paper is organized as follows. In Section 2, we describe the shear-warp factorization volume rendering method and a number of proposed parallel share-warp factorization volume rendering methods. We then present and analyze the 1D-partition and 2D-partition methods in Sections 3 and 4, respectively. In Section 5, the performance results from the proposed methods are compared with the other parallel shear-warp factorization volume rendering methods on an IBM SP2 parallel machine.

2. Related work

The proposed volume rendering methods can be classified into the following types, ray tracing, splatting, cell projection, multi-pass resampling, and shear-warp factorization methods. The ray tracing method [5, 13, 14] is called the backward projection or the image order method. It traces a ray through the volume data for each image pixel, computes the color and opacity of the volume data along the path, and produces a final image. The splatting method [12] is called the forward projection or the object order method. It computes the contribution of a voxel to the image by

convolving the voxel with a filter that distributes the voxel's value to the neighboring pixels. The cell projection method [21] is similar to the splatting method except that it uses a polygon scan conversion to perform the projection. The multi-pass resampling method [20] operates by resampling the entire volume to the image coordinate system. Catmull and Smith introduced the multi-pass resampling method for warping two-dimensional images. This technique was first applied to render the volume data in Pixar [3].

The shear-warp factorization method proposed by Lacroute and Levey [11] is an object-order volume rendering method. This method has three major stages. In the first stage, a three-dimensional volume data is sheared based on a factorization of the viewing transformation matrix. In the second stage, a projection method is used to generate an intermediate image. In the final stage, a two-dimensional image is warped to form a final image. Figure 1 illustrates the three stages of the shear-warp factorization volume rendering method [10].

Sano et al. [18] proposed a parallel shear-warp factorization volume rendering method on distributed memory multicomputers. In this method, they employ the slice data partitioning method to distribute volume data to each processor. This method first groups volume slices into a set of sub-volumes. Each processor is then assigned several continuous sub-volumes. Each processor uses the shear-warp factorization method to perform run-length encoding and resampling of all allocated sub-volumes. The partial intermediate image from the sub-volumes is then generated in parallel. Finally, these partial intermediate images are composited to form a complete intermediate image and a warping method is used to produce the final image. An example is given in Figure 2. In Figure 2, there are eight volume slices in the volume data. We used horizontal lines to represent the volume data slices. Assume that there are four processors P_0, P_1, P_2, and P_3 available in the system. When the slice volume partitioning method is performed, each processor is assigned two slices denoted by a dotted rectangle. Each processor uses the serial shear-warp factorization to generate a partial intermediate image from the sub-volumes assigned to it. We used thin horizontal lines to denote the partial intermediate

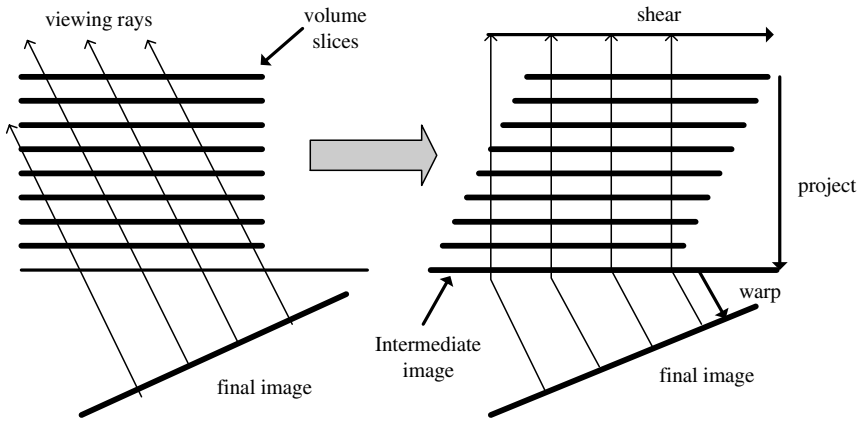


Figure 1. The shear-warp factorization volume rendering method.

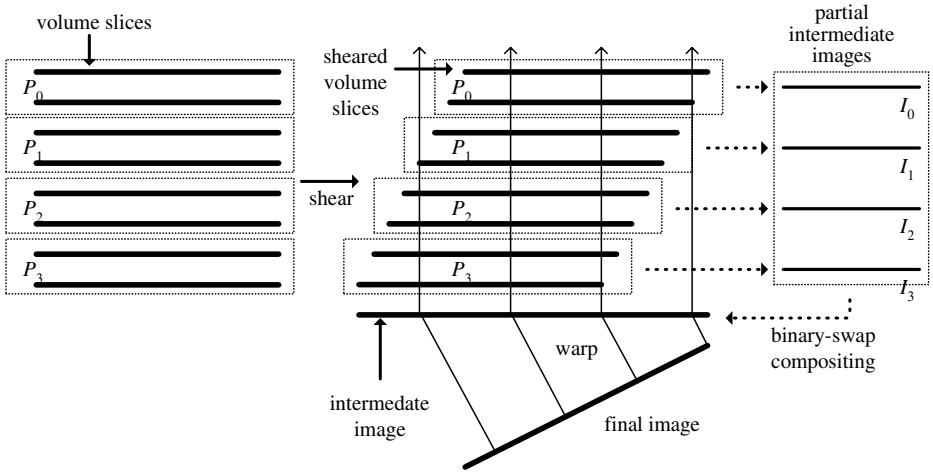


Figure 2. The slice data partitioning method.

images I_0, I_1, I_2, and I_3 on the right. These four partial images are composited to form a complete intermediate image using the binary-swap compositing method. The warp operation is employed in the final step to produce the final image.

The main advantage of the slice volume partitioning method is that it is easily implemented in parallel. The drawbacks of this method are that the number of processors is restricted to a power of two and a lot of time is necessary to composite the partial intermediate images during the binary-swap compositing.

The volume data partitioning method, proposed by Amin et al. [1], is another simple partitioning method that performs better than the slice volume partitioning method. The major improvement of this method is that it avoids the requirement for a large intermediate image in each processor using vertical slicing. In Figure 3, there are four processor elements P_0, P_1, P_2, and P_3. Assume that there are eight volume slices in the volume data, which is sliced into four modules denoted by dash

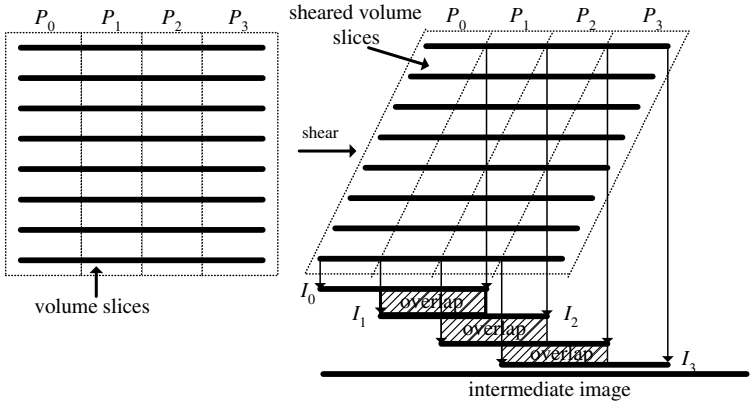


Figure 3. The volume data partitioning method.

line parallelograms. Each slice is divided into four pieces and each module contains one piece from each slice. Each processor has eight pieces that are contained in one parallelogram. After data partitioning, processor P_i uses the shear-warp factorization to generate the partial intermediate image, I_i, from the eight pieces, where $i = 0, \ldots, 3$. An image compositing method is then used to form the complete intermediate image that will be warped to generate the final image.

When shearing the volume data, the partial intermediate images will have overlapped areas and the parts generated from the processors that intersect will be assembled. While the overlapped areas of the partial intermediate images have communication overheads, the compositing intersection parts will have extra computation overhead. Therefore, the disadvantage of the volume data partitioning method is that the communication and computation time will increase when it is sheared at a large angle. As a result, more scan lines are produced and an *over* operation is required to compute the color and opacity for each image pixel.

The sheared volume data partitioning method proposed by Amin et al. [1] is a novel data partitioning method that can avoid the communication overhead of overlapped areas. The volume data is sheared first and then partitioned by slicing orthogonally to the volume data slices according to the viewing angle. Figure 4 shows an example of the sheared volume data partitioning method with eight slices of volume data. The first stage involves shearing in the viewing direction angle, and then deriving rays orthogonally to the slices. Four partitioned modules are therefore generated from the designated volume segments. For example, P_0 contains a triangle formed by the designated slices on the left. P_1 contains a rectangle formed by the middle slices. P_2 and P_3 are similar to P_1 and P_0, respectively. We can see that the partial intermediate images in the processors do not have any overlapped areas. Each processor produces a disjointed partial intermediate image and that partial intermediate image can be warped independently. In this way, no compositing is required across processor boundaries. However, the processor load is not balanced. The processor load imbalance will increase the computation time for the shear-warp process.

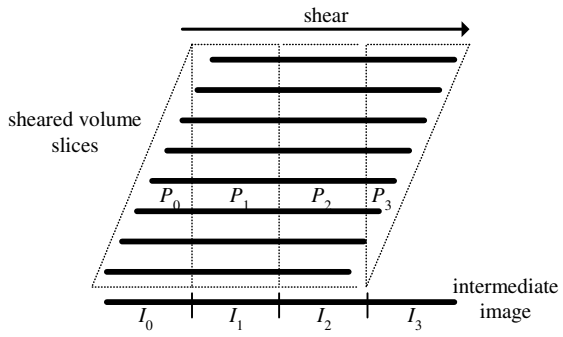


Figure 4. The sheared volume data partitioning method.

3. The 1D-partition method

In this section, we will describe and analyze the 1D-partition method in detail. The 1D-partition method is divided into the following three stages:

Stage 1: *The data partitioning stage.* In this stage, the 1-D partitioning scheme is developed for partitioning volume data into sub-volumes according to the mathematical formulae derived from the viewing angle and the number of processors.

Stage 2: *The shear-warp rendering stage.* In this stage, each processor uses the shear-warp factorization volume rendering method to generate the corresponding partial final image.

Stage 3: *The image compositing stage.* In this stage, a simple merge operation is used for compositing the partial final images to form a final image.

Figure 5 shows the behavior of the 1D-partition method. In the following subsections, we will discuss the data partitioning stage, the shear-warp rendering stage and the image compositing stage of the 1D-partition method.

3.1. The data partitioning stage

The goals of the data partitioning stage are to distribute volume data to the processors evenly and minimize the communication overhead and the image compositing

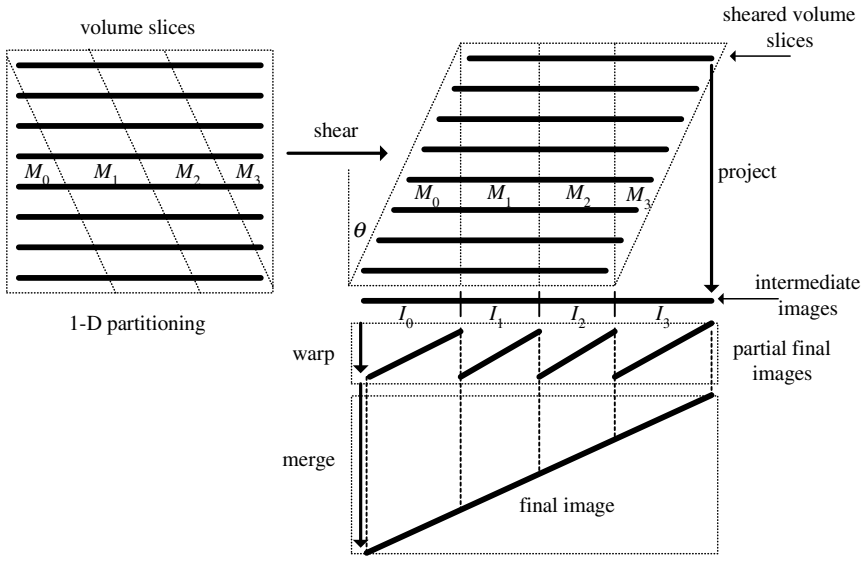


Figure 5. The behavior of the 1D-partition method.

time. The slice volume partitioning method [18] distributes the volume data slices to the processors evenly. However, this method results in high communication overhead and extra compositing time in the image compositing stage. The volume data partitioning method [1] distributes volume data to the processors evenly and minimizes the communication overhead in the image compositing stage. However, extra image compositing time is required in the image compositing stage. The sheared volume data partitioning method [1] minimizes the communication overhead and no extra image compositing time is required in the image compositing stage. The volume data is, however, not evenly distributed to the processors.

The 1-D partitioning scheme concept involves partitioning volume data evenly by partitioning the sheared volume slices orthogonally according to the mathematical formulae derived from the viewing angle and the number of processors. However, in the implementation, the volume slices are not sheared when they are partitioned into modules. According to the mathematical formula derived below, the 1-D partitioning scheme can determine which voxel belongs to which module and distributes the voxels to their corresponding processors. The shear operation is then performed in the shear-warp rendering stage. Figure 5 shows an example of the 1-D partitioning scheme. In Figure 5, we can see that the volume data is partitioned into four equal volume data modules, M_0, M_1, M_2, and M_3, using the derived mathematical formulae that will be described later. The modules are assigned to four processors. Because the partial intermediate images of the four processors, denoted by I_0, I_1, I_2, and I_3, do not have any overlapped area, they can be warped independently and used to form partial final images. In the image compositing stage, the merge operation is used to composite the partial final images to form a final image. The communication overheads can be minimized and no extra image compositing time is required because the partial final images have no overlapped area in the image compositing stage.

In the following, we derive the mathematical formulae used to partition the volume data. To simplify our notations, we assumed that the size of volume data is $n \times n \times n$ and P processors are used, where n is the size of each dimension. P processors form a processor array and are denoted as $P_0, P_1, P_2, \ldots$, and P_{P-1}. Given a viewing angle θ, when partitioning the sheared volume slices orthogonally into P modules such that each module has the same number of voxels, we obtained three cases as shown in Figure 6. Assume that the height of the triangle part (i) shown in Figure 6 is x. The base of the triangle part is $x \tan\theta$. Because the area of the triangle is equal to $\frac{n^2}{P}$, we have

$$\frac{1}{2}x^2 \tan\theta = \frac{n^2}{P} \Rightarrow x = n\sqrt{\frac{2}{P\tan\theta}}.$$

When $x \tan\theta$ is smaller than $n \tan\theta$, we have the case shown in Figure 6(a), that is,

$$x\tan\theta < n\tan\theta \Rightarrow n\sqrt{\frac{2}{P\tan\theta}} \times \tan\theta < n\tan\theta \Rightarrow \tan\theta > \frac{2}{P}.$$

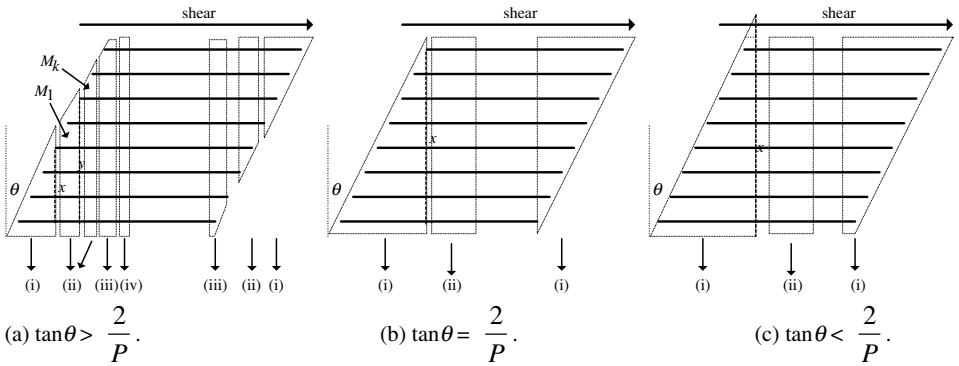


Figure 6. Three cases for the 1-D data partitioning scheme formulae.

When $x \tan\theta$ is equal to $n \tan\theta$, we have the case shown in Figure 6(b), that is,

$$x \tan\theta = n \tan\theta \Rightarrow n\sqrt{\frac{2}{P \tan\theta}} \times \tan\theta = n \tan\theta \Rightarrow \tan\theta = \frac{2}{P}.$$

When $x \tan\theta$ is larger than $n \tan\theta$, we have the case shown in Figure 6(c), that is,

$$x \tan\theta > n \tan\theta \Rightarrow n\sqrt{\frac{2}{P \tan\theta}} \times \tan\theta > n \tan\theta \Rightarrow \tan\theta < \frac{2}{P}.$$

The $\tan\theta$ value can therefore be used to determine the sizes of the partial intermediate images. In the following, we will give the mathematical formulae for determining the sizes of the partial intermediate images for the partitioned modules for the three cases shown in Figure 6. We use I_i to denote the size of the partial intermediate image of partitioned module M_i, where $i = 0, \ldots, P-1$. To avoid a lengthy description, the detailed proofs are omitted.

Case 1: $\tan\theta > \frac{2}{P}$. The formulae for determining the sizes of the partial intermediate images of the partitioned modules for the four different shapes shown in Figure 6(a) are given below.

- The triangle portions (denoted as (i) in Figure 6(a)): The individual size of the partial intermediate images of the triangle portions, M_0 and M_{P-1}, can be determined using Eq. (1),

$$I_i = n\sqrt{\frac{2 \tan\theta}{P}}, \quad \text{for } i = 0 \text{ and } P-1. \tag{1}$$

- The trapezoid portions (denoted as (ii) in Figure 6(a)): According to the values of P, n, and θ, there are $2 \times (\lfloor P \tan\theta/2 \rfloor - 1)$ trapezoid portions, $M_1, \ldots, M_k$ and $M_{P-k-1}, \ldots, M_{P-2}$, where $k = \lfloor P \tan\theta/2 \rfloor - 1$. The individual sizes of the partial

intermediate images of $M_1, \ldots, M_k$ and $M_{P-k-1}, \ldots, M_{P-2}$ can be determined using Eq. (2),

$$I_i = \begin{cases} n\sqrt{\dfrac{2\tan\theta}{P}}(\sqrt{(i+1)}-\sqrt{i}), & \text{for } i=1,\ldots,k \\ n\sqrt{\dfrac{2\tan\theta}{P}}(\sqrt{P-i}-\sqrt{P-1-i}), & \text{for } i=P-k-1,\ldots,P-2 \end{cases}. \quad (2)$$

- The pentagon portions (denoted as (iii) in Figure 6(a)): The individual size of the partial intermediate images of the pentagon portions, M_{k+1} and M_{P-k-2}, can be determined using Eq. (3),

$$I_i = n\left(\frac{k+1}{P} + \frac{1}{2}\tan\theta + 1 - \sqrt{\frac{2(k+1)\tan\theta}{P}}\right), \quad \text{for } i = k+1 \text{ and } P-k-2. \quad (3)$$

- The middle rectangle portions (denoted as (iv) in Figure 6(a)): The individual size of the partial intermediate images of the middle rectangle portions, $M_{k+2}, \ldots, M_{P-k-3}$, can be determined using Eq. (4),

$$I_i = \frac{n}{P-k}\left(\frac{3}{2}\tan\theta - \frac{k+1}{P}\tan\theta - 2\right), \quad \text{for } i=k+2,\ldots,P-k-3. \quad (4)$$

Case 2: $\tan\theta = \frac{2}{P}$. The formulae for determining the sizes of the partial intermediate images of the partitioned modules for the two different shapes shown in Figure 6(b) are given below.

- The triangle portions (denoted as (i) in Figure 6(b)): The individual size of the intermediate images of the triangle portions, M_0 and M_{P-1}, can be determined using Eq. (5),

$$I_i = n\tan\theta, \quad \text{for } i = 0 \text{ and } P-1. \quad (5)$$

- The middle rectangle portions (denoted as (ii) in Figure 6(b)): The individual size of the partial intermediate images of the middle rectangle portions, $M_1, \ldots, M_{p-2}$, can be determined using Eq. (6),

$$I_i = \frac{n}{P-2}(1-\tan\theta), \quad \text{for } i = 1, \ldots, P-2. \quad (6)$$

Case 3: $\tan\theta < \frac{2}{P}$. The formulae for determining the sizes of the intermediate images of the partitioned modules for the two different shapes shown in Figure 6(c) are given below.

- The trapezoid portions (denoted as (i) in Figure 6(c)): The individual size of the partial intermediate images of the trapezoid portions, M_0 and M_{P-1}, can be determined using Eq. (7),

$$I_i = n\left(\frac{1}{P} + \frac{1}{2}\tan\theta\right), \quad \text{for } i = 0 \text{ and } P-1. \quad (7)$$

- The middle rectangle portions (denoted as (ii) in Figure 6(c)): The individual size of the partial intermediate images of the middle rectangle portions, $M_1, \ldots, M_{P-2}$, can be determined using Eq. (8),

$$I_i = \frac{n}{P-2}\left(1 - \frac{2}{P}\right), \quad \text{for } i = 1, \ldots, P-2. \tag{8}$$

3.2. *The shear-warp rendering stage and the image compositing stage*

After the volume data is partitioned into P modules with an equal number of voxels, each processor is assigned one module. Each processor uses the shear-warp factorization volume rendering method for rendering the assigned voxels and then generates the corresponding partial final image independently.

After the shear-warp rendering stage, each processor contains a partial final image. In the image compositing stage, the partial final images generated from each processor are composited to form a final image. Because the partial final image in each processor is generated independently and does not overlap or intersect with any other partial final image, a simple merge operation is used for compositing the partial final images into the final image. By using the gather directives of a message-passing library, such as MPICH, on distributed memory multicomputers, the image compositing time is minimized. Therefore, the advantages of our simple merge operation are twofold:

(1) There is no restriction regarding the number of processors, i.e., the 1D-partition method can be used in cases where the number of processors is not a power of two.
(2) The image compositing time is short and fixed.

The algorithm for the 1D-partition method is given as follows.

Algorithm 1D-partition_Method(V, θ, P) {
/* V is a volume data. */
/* θ is the shearing angle */
/* P is the number of processors. */
/* I is the final image. */

1. Calculate the value of $\tan\theta$;
2. Compute $M = \frac{2}{P}$ and compare M with $\tan\theta$;
3. **if** $\tan\theta > M$ **then** use formulae (1)(2)(3)(4) to partition V
4. **else if** $\tan\theta = M$ **then** use formulae (5)(6) to partition V
5. **else** use formulae (7)(8) to partition V;
6. **for** each processor P_i **do parallel**{
7. Use shear-warp factorization volume rendering method to generate a partial final image A_i;

}

8. $I :=$ merge(A_i);
9. return I; }

end_of_1D-partition_Method

3.3. Performance analysis of the 1D-partition method

The time complexity of the shear-warp rendering and image compositing stages of the 1D-partition method are analyzed in this subsection. The time complexity of the data partitioning stage was not evaluated because this stage is a preprocessing step for distributing the voxels to each processor. A summary of the notations used in the performance analysis is given below.

- T_s is the startup time of a communication channel.
- T_p is the data transmission time per byte.
- $T_{v\text{-}shear}$ is the time for shearing one voxel of a sub-volume.
- $T_{v\text{-}project}$ is the time for projecting one voxel of a sub-volume.
- $T_{p\text{-}warp}$ is the time for warping one pixel in a partial intermediate image.
- P is the number of processors.
- n is the size of each dimension of a volume data.
- A_i is the partial final image size of P_i.
- θ is the viewing angle.

3.3.1. The shear-warp rendering stage. After applying the 1-D partitioning scheme to the volume data in the data partitioning stage, each processor gets an equal number of voxels. Each processor generates a partial intermediate image by shearing and projecting voxels assigned to it; and warping the partial intermediate image independently to form the partial final image. Therefore, the time of the shear-warp rendering stage, denoted by $T_{shear\text{-}warp}$, is the sum of T_{shear}, $T_{project}$, and T_{warp}, where T_{shear}, $T_{project}$ and T_{warp} are the time for a processor to perform shear, project and warp operations for a given sub-volume, respectively. We have

$$\begin{aligned} T_{shear\text{-}warp} &= T_{shear} + T_{project} + T_{warp} = \frac{n^3}{P} T_{v\text{-}shear} + \frac{n^2}{P} T_{v\text{-}project} + \frac{n^2}{P} T_{p\text{-}warp} \\ &= \frac{n^2}{P}(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp}) \end{aligned} \tag{9}$$

3.3.2. The image compositing stage. In the image compositing stage, since there is no overlapping area among the partial final images in the processors, we used a simple merge operation for compositing the partial final images to form a final image. Therefore, the time for the image compositing stage is

$$T_{composite} = \sum_{i=0}^{P-1} (T_p \cdot A_i + T_s) = (n^2 - \varepsilon)T_p + (P-1)T_s, \tag{10}$$

where ε is the partial final image size of the root processor that gathers partial final images from other processors.

3.3.3. Performance upper bound of the 1D-partition method. The total rendering time $T_{1D\text{-}partition}$ of the 1D-partition method is the sum of Eqs. (9) and (10) as follows:

$$\begin{aligned} T_{1D\text{-}partition} &= T_{shear\text{-}warp} + T_{composite} \\ &= \frac{n^2}{P}(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp}) + (n^2 - \varepsilon)T_p \\ &\quad + (P-1)T_s. \end{aligned} \tag{11}$$

To find the performance bound, we compare the total rendering time for P and $P+1$ processors are used. We have

$$T_P = \frac{n^2}{P}(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp}) + (n^2 - \varepsilon_p)T_p + (P-1)T_s$$

and

$$\begin{aligned} T_{P+1} = \frac{n^2}{P+1}&(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp}) + (n^2 - \varepsilon_{P+1})T_p \\ &+ (P+1-1)T_s. \end{aligned}$$

The difference of T_P and T_{P+1} is

$$\begin{aligned} \Delta T = T_P - T_{P+1} = n^2(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp})\frac{1}{P(P+1)} \\ - (\varepsilon_P - \varepsilon_{P+1})T_p - T_s \end{aligned} \tag{12}$$

From Eq. (7), we have

$$\Delta\varepsilon = \varepsilon_P - \varepsilon_{P+1} = n\left(\frac{1}{P} + \frac{1}{2}\tan\theta\right) - n\left(\frac{1}{P+1} + \frac{1}{2}\tan\theta\right) = \frac{n}{P(P+1)}.$$

Therefore, Eq. (12) can be replaced by

$$\begin{aligned} \Delta T &= T_P - T_{P+1} \\ &= n^2(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp})\frac{1}{P(P+1)} - \frac{n}{P(P+1)}T_p - T_s. \end{aligned}$$

By setting $\Delta T = 0$, we have

$$n^2(n \cdot T_{v\text{-}shear} + n \cdot T_{v\text{-}project} + T_{p\text{-}warp})\frac{1}{P(P+1)} = \frac{n}{P(P+1)}T_p + T_s.$$

Since T_p is much smaller than T_s, if $n/P(P+1) < 1$, $(n/P(P+1))T_p + T_s$ will be very close to T_s. Since T_s is a constant, if $\Delta T \to 0$, $n/P(P+1) < 1$. Therefore, we can derive

$$P > \frac{-1 + \sqrt{1+4n}}{2} \approx \sqrt{n}. \tag{13}$$

Given an $n \times n \times n$ volume data set, Eq. (13) indicates that the total rendering time for the 1D-partition method will not be improved when P is greater than $\sqrt{n}$.

4. The 2D-partition method

As we know, a good parallel volume rendering algorithm tries to obtain a linear relationship between the performance speedup and the increase in available processors. From Eq. (13), we know that the number of processors used for the 1D-partition method is bounded by $\sqrt{n}$. To improve the speedup when more than $\sqrt{n}$ processors are used, we developed another method called the 2D-partition method. The 2D-partition method is divided into the following three stages.

Stage 1: *The data partitioning stage.* In this stage, a 2-D partitioning scheme is developed for partitioning volume data into sub-volumes according to the mathematical formula derived from the viewing angle and the number of processors.

Stage 2: *The shear-warp rendering stage.* In this stage, each processor uses the shear-warp factorization volume rendering method to generate a partial final image.

Stage 3: *The image compositing stage.* In this stage, the pixel compositing method is used for compositing the partial final images in the vertical slices to form partial composited final images. The merge operation is then used for assembling the partial composited final images into a final image.

Figure 7 shows the behavior of the 2D-partition method. In the following subsections, we will discuss the data partitioning stage, the shear-warp rendering stage, and the image compositing stage of the 2D-partition method.

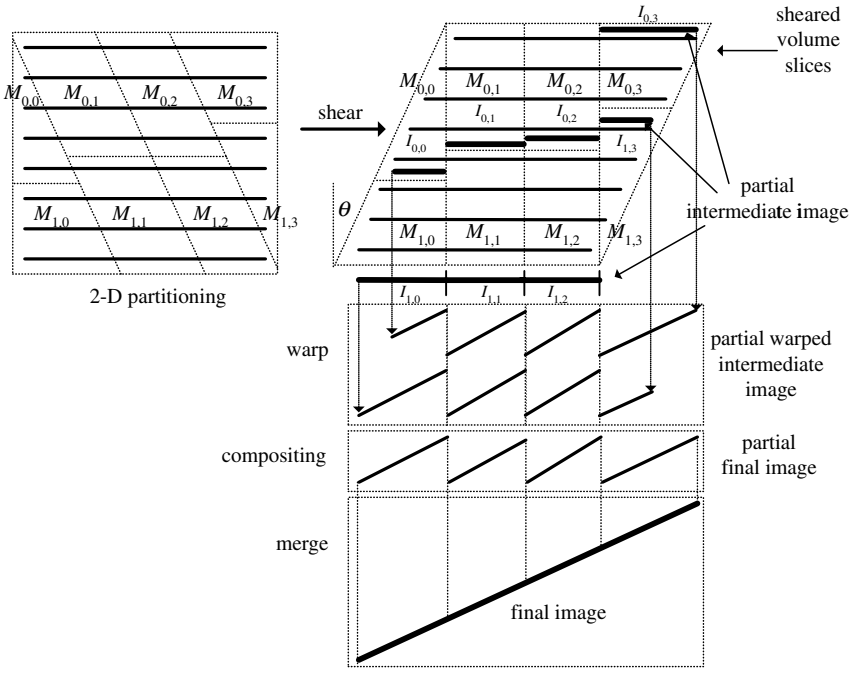


Figure 7. The behavior of the 2D-partition method.

4.1. The data partitioning stage

The 2-D partitioning scheme combines the 1-D partitioning scheme with the slice data partitioning method to partition volume data into modules with approximately the same number of voxels. Given an $n \times n \times n$ volume data set and $P = P_v \times P_h$ processors, the 2-D partitioning scheme first partitions the sheared volume slices into P_h parts using the 1-D partitioning scheme. Each part is then partitioned into P_v modules with approximately the same number of voxels by using the slice data partitioning method. We use $P_{i,j}(M_{i,j})$ to denote the processor (module) in the ith row and the jth column of a processor grid (a partitioned sheared volume slice), where $i = 0, \ldots, P_v - 1$ and $j = 0, \ldots, P_h - 1$. $M_{i,j}$ is assigned to processor $P_{i,j}$. Again, in the implementation, the volume slices are not sheared when they are partitioned into modules. According to the mathematical formula derived below, the 2-D partitioning scheme can determine which voxel belongs to which module and distributes the voxels to their corresponding processors.

According to the values of P_h, P_v, n, and θ values, we can derive a mathematical formula to determine the size of the partial intermediate image of module $M_{i,j}$. In the following, we will give the mathematical formulae for the cases shown in Figure 8. To avoid a lengthy description, the detailed proofs were omitted. We use $M_{*,j}$ to denote the modules in the jth column, that is, $M_{*,j} = \{M_{0,j}, M_{1,j}, \ldots, M_{P_v-1,j}\}$.

Case 1: $\tan\theta > \frac{2}{P_h}$. The formulae for determining the sizes of the partial intermediate images of the partitioned modules for the four different shapes shown in Figure 8(a) are given below.

- The left- and right-side portions (denoted as (i) in Figure 8(a)): The individual sizes of the partial intermediate images of the left- and right-side portions, $M_{*,0}$ and M_{*,P_h-1}, can be determined using Eq. (14),

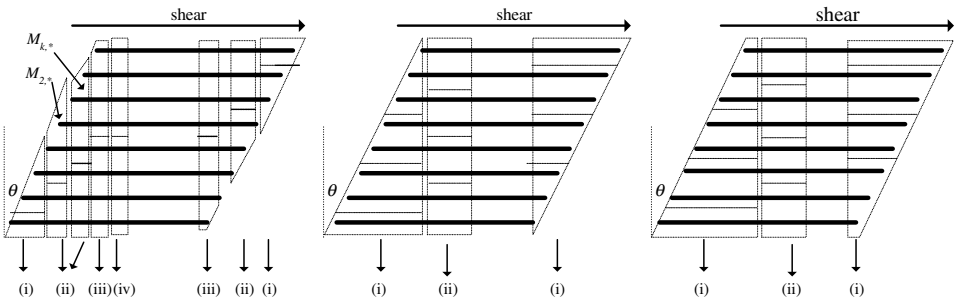


Figure 8. Three cases for the 2-D partitioning scheme.

$$I_{i,j}=\begin{cases} n\sqrt{\dfrac{2\tan\theta}{P_h}}\left(\dfrac{i+1}{\sqrt{P_v}}\right), & \text{for } i=0,\ldots,P_v-1 \text{ and } j=0 \\ n\sqrt{\dfrac{2\tan\theta}{P_h}}\left(\dfrac{P_v-i}{\sqrt{P_v}}\right), & \text{for } i=0,\ldots,P_v-1 \text{ and } j=P_h-1 \end{cases}. \tag{14}$$

- The trapezoid portions (denoted as (ii) in Figure 8(a)): According to the P_h, n, and θ values, there are $2\times(\lfloor P_h\tan\theta/2\rfloor-1)$ trapezoid portions, $M_{*,1},\ldots,M_{*,k}$ and $M_{*,P_h-k-1},\ldots,M_{*,P_h-2}$, where $k=\lfloor P_h\tan\theta/2\rfloor-1$. The individual sizes of the partial intermediate images of $M_{*,1},\ldots,M_{*,k}$ and $M_{*,P_h-k-1},\ldots,M_{*,P_h-2}$ can be determined using Eq. (15),

$$I_{i,j}=\begin{cases} n\sqrt{\dfrac{2\tan\theta}{P_h}}(\sqrt{(j+1)}-\sqrt{j}), \\ \quad \text{for } i=0,\ldots,P_v-1 \text{ and } j=1,\ldots,k \\ n\sqrt{\dfrac{2\tan\theta}{P_h}}(\sqrt{P_h-j}-\sqrt{P_h-1-j}), \\ \quad \text{for } i=0,\ldots,P_v-1 \text{ and } j=P_h-k-1,\ldots,P_h-2 \end{cases}. \tag{15}$$

- The pentagon portions (denoted as (iii) in Figure 8(a)): The individual size of the partial intermediate images of the pentagon portions, $M_{*,k+1}$ and M_{*,P_h-k-2}, can be determined using Eq. (16),

$$I_{i,j}=n\left(\frac{k+1}{P_h}+\frac{1}{2}\tan\theta+1-\sqrt{\frac{2(k+1)\tan\theta}{P_h}}\right), \quad \text{for } i=0,\ldots,P_v-1;\ j=k+1 \text{ and } P_h-k-2. \tag{16}$$

- The middle rectangle portions (denoted as (iv) in Figure 8(a)): The individual size of the partial intermediate images of the middle rectangle portions, $M_{*,k+2},\ldots,M_{*,P_h-k-3}$, can be determined using Eq. (17),

$$I_{i,j}=\frac{n}{P_h-k}\left(\frac{3}{2}\tan\theta-\frac{k+1}{P_h}\tan\theta-2\right), \quad \text{for } i=0,\ldots,P_v-1 \text{ and } j=k+2,\ldots,P_h-k-3. \tag{17}$$

Case 2: $\tan\theta = \frac{2}{P_h}$. The formulae for determining the sizes of the partial intermediate images of the partitioned modules for the two different shapes shown in Figure 8(b) are given below.

- The triangle portions (denoted as (i) in Figure 8(b)): The individual sizes of the partial intermediate images of the triangle portions, $M_{*,0}$ and M_{*,P_h-1}, can be determined using Eq. (18),

$$I_{i,j} = \begin{cases} n\tan\theta\left(\dfrac{i+1}{\sqrt{P_v}}\right), & \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = 0 \\ n\tan\theta\left(\dfrac{P_v - i}{\sqrt{P_v}}\right), & \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = P_h - 1 \end{cases}. \tag{18}$$

- The middle rectangle portions (denoted as (ii) in Figure 8(b)): The individual size of the partial intermediate images of the middle rectangle portions, $M_{*,1}, \ldots, M_{*,P_h-2}$, can be determined using Eq. (19),

$$I_{i,j} = \frac{n}{P_h - 2}(1 - \tan\theta), \quad \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = 1, \ldots, P_h - 2. \tag{19}$$

Case 3: $\tan\theta < \frac{2}{P_h}$. The formulae for determining the sizes of the partial intermediate images of the partitioned modules for the two different shapes shown in Figure 8(c) are given below.

- The trapezoid portions (denoted as (i) in Figure 8(c)): The individual sizes of the partial intermediate images of the trapezoid portions, $M_{*,0}$ and M_{*,P_h-1}, can be determined using Eq. (20),

$$I_{i,j} = \begin{cases} n\left(\dfrac{1}{P_h} + \dfrac{1}{2}\tan\theta\right)\left(\dfrac{i+1}{\sqrt{P_v}}\right), & \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = 0 \\ n\left(\dfrac{1}{P_h} + \dfrac{1}{2}\tan\theta\right)\left(\dfrac{p_v - i}{\sqrt{P_v}}\right), & \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = P_h - 1 \end{cases}. \tag{20}$$

- The middle rectangle portions (denoted as (ii) in Figure 8(c)): The individual size of the partial intermediate images of the middle rectangle portions, $M_{*,1}, \ldots, M_{*,p_h-2}$, can be determined using Eq. (21),

$$I_{i,j} = \frac{n}{P_h - 2}\left(1 - \frac{2}{P_h}\right), \quad \text{for } i = 0, \ldots, P_v - 1 \text{ and } j = 1, \ldots, P_h - 2. \tag{21}$$

4.2. The shear-warp rendering stage and the image compositing stage

After the volume data is partitioned into $P_v \times P_h$ modules with approximately the same number of voxels, each processor is assigned one module. Each processor then uses the shear-warp factorization method for rendering the assigned voxels and generates the corresponding partial warped intermediate image independently.

After the share-warp rendering stage, each processor contains a partial warped intermediate image. In the image compositing stage, the partial warped intermediate images in the same column, $I_{*,j}$, are assembled first, where $j = 0, \ldots, P_h - 1$. Because the partial warped intermediate images in $I_{*,j}$ have overlapped areas, processor $P_{i,j}$ sends its $I_{i,j}$ to processor $P_{P_v-1,j}$ and $P_{P_v-1,j}$ uses the *over* operation to assemble these partial warped intermediate images into a partial final image, where $j = 0, \ldots, P_h - 1$. The merge operation presented in the 1D-partition method is then used for compositing the partial final images $I_{P_v-1,*}$ to form a final image.

The algorithm for the 2D-partition method is given as follows.

Algorithm 2D-partition_Method(V, θ, P) {
/* V is a volume data. */
/* P is the number of processors. */
/* θ is the shearing angle */
/* I is the final image */

1. Calculate the value of $\tan\theta$;
2. Factorize P to form $P = P_h \times P_v$, P_h is the largest value smaller than or equal to $\sqrt{n}$;
3. Compute $M = \frac{2}{p_h}$ and compare M with $\tan\theta$;
4. **if** $\tan\theta > M$ **then** use formulae (14)(15)(16)(17) to partition V
5. **else if** $\tan\theta = M$ **then** use formulae (18)(19) to partition V
6. **else** use formulae (20)(21) to partition V;
7. **for** each processor $P_{i,j}$ **do parallel**{
8. Use the shear-warp factorization volume rendering method to generate a partial warped intermediate image $A_{i,j}$;
9. }
10. $A_{P_v-1,j}$:= pixel_compositing($A_{*,j}$)
11. I := merge($A_{P_v-1,*}$);
12. return I; }

end_of_2D-partition_Method

4.3. Performance analysis of the 2D-partition method

The time complexity of the shear-warp rendering and image compositing stages of the 2D-partition method are analyzed in this sub-section. As in Section 3.3, we did not evaluate the time complexity of the data partitioning stage since this stage is a preprocessing step for distributing the volume data to each processor. A summary of the notations used in the following analysis is given below.

- T_s is the startup time of a communication channel.

- T_p is the data transmission time per byte.
- $T_{v\text{-}shear}$ is the time for shearing one voxel of a sub-volume.
- $T_{v\text{-}project}$ is the time for projecting one voxel of a sub-volume.
- $T_{p\text{-}warp}$ is the time for warping one pixel in a partial intermediate image.
- P is the number of processors.
- n is the size of each dimension of a volume data.
- $A_{i,j}$ is the partial final image size of $P_{i,j}$.
- θ is the viewing angle.

4.3.1. The shear-warp rendering stage. After using the 2-D partitioning scheme to distribute the volume data, each processor renders its own sub-volume by shearing, projecting, and warping operations. The time of the shear-warp stage, denoted by $T_{shear\text{-}warp}$, is:

$$\begin{aligned} T_{shear\text{-}warp} &= T_{shear} + T_{project} + T_{warp} \\ &= \frac{n^3}{P} T_{v\text{-}shear} + \frac{n^3}{P} r(p) \cdot T_{v\text{-}project} + n^2 \left(\frac{1}{P_h} + \frac{1}{2} \tan\theta \right) \cdot T_{p\text{-}warp} \\ &= \frac{n^2}{P} \left(n \cdot T_{v\text{-}shear} + n \cdot r(p) \cdot T_{v\text{-}project} + \left(p_v + \frac{P}{2} \tan\theta \right) \cdot T_{p\text{-}warp} \right), \end{aligned} \tag{22}$$

where $r(p)$ is the data coherence ratio of a partial warped intermediate image [2, 4, 22].

4.3.2. The image compositing stage. In the image compositing stage, the partial warped intermediate images $A_{*,j}$ are first sent to the corresponding processors $P_{P_v-1,j}$, where $j = 0, \ldots, P_h - 1$. The corresponding processors $P_{P_v-1,j}$ then use the *over* operation to composite $A_{*,j}$ for generating the corresponding partial final images $A_{P_v-1,j}$. The time for this step is denoted by $T_{v\text{-}composite}$. Since there is no overhead among the partial final images $A_{P_v-1,j}$ of $P_{P_v-1,j}$, a simple merge operation is used to assemble the partial final images $P_{P_v-1,j}$ to form a final image. The time for this step is denoted by $T_{h\text{-}merge}$. The time for the image compositing stage, denoted by $T_{composite}$, is:

$$\begin{aligned} T_{composite} &= T_{v\text{-}composite} + T_{h\text{-}merge} \\ &= \sum_{i=1}^{P_v - 1} \left\{ \left(T_p + T_{v\text{-}project} \right) \cdot A_{i,j} + T_s \right\} + \sum_{j=1}^{P_h - 1} \left(T_p \cdot A_{P_v-1,j} + T_s \right) \\ &= n^2 \left(T_p + T_{v\text{-}project} \right) \cdot \sqrt{P_v} (P_v + 1) \cdot \left(\frac{1}{2P_h} + \frac{\tan\theta}{4} \right) \\ &\quad + (P_v - 1) T_s + (n^2 - \varepsilon) T_p + (P_h - 1) T_s, \end{aligned} \tag{23}$$

where ε is the partial final image size of the root processor that gathers other partial final images from the other horizontal partitioning processors.

4.3.3. The time complexity of the 2D-partition method. The total rendering time for the 2D-partition method, denoted by $T_{2D\text{-}partition}$, is the sum of Eqs. (22) and (23):

$$\begin{aligned} T_{2D\_rendering} &= T_{shear\text{-}warp} + T_{composite} = T_{shear} + T_{project} + T_{warp} + T_{v\text{-}composite} + T_{h\text{-}merge} \\ &= \frac{n^2}{P}\left(n \cdot T_{v\text{-}shear} + n \cdot r(p) \cdot T_{v\text{-}project} + \left(P_v + \frac{P}{2}\tan\theta\right) \cdot T_{p\text{-}warp}\right) \\ &\quad + n^2(T_{trans} + T_{v\text{-}project}) \cdot \sqrt{P_v}(P_v + 1) \cdot \left(\frac{1}{2P_h} + \frac{\tan\theta}{4}\right) \\ &\quad + (n^2 - \varepsilon)T_p + (P_v + P_h - 2)T_s. \end{aligned} \tag{24}$$

In Eq. (24), given that P is a constant, $\frac{n^2}{P} \cdot n \cdot T_{v\text{-}shear}$ and $(n^2 - \varepsilon)T_p$ are constants. The value of $\frac{n^2}{P} \cdot n \cdot r(p) \cdot T_{v\text{-}project}$ depends on data coherence. It decreases when P_v is close to P_h. When P_v increases, the values of $n^2(\frac{1}{P_h} + \frac{P}{2}\tan\theta) \cdot T_{p\text{-}warp}$ and $n^2(T_p + T_{v\text{-}project}) \cdot \sqrt{P_v}(P_v + 1) \cdot (\frac{1}{2P_h} + \frac{\tan\theta}{4})$ increase but the value of $(P_v + P_h - 2)T_s$ decreases. For a given set of parameters discussed above, the time complexity of the 2D-partition method can be evaluated.

5. Experimental results and performance analysis

To evaluate the performance of the 1D-partition and 2D-partition methods, we implemented them along with the slice data partitioning [12], volume data partitioning [1], and sheared volume data partitioning methods [1] on an IBM SP2 parallel machine [6]. The IBM 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 the system and each node has a 128KB 1st-level data cache, a 32KB 1st-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).

We used C language and the MPICH message-passing library to implement the proposed parallel volume rendering algorithms. The MPICH library is one of the MPI [16] message-passing libraries. MPI is a standard for using message-passing to send and receive data in parallel systems. Our volume rendering algorithm implementations are therefore portable and can be installed in other distributed memory multicomputers.

Six different volume data test samples were used to evaluate the performance of our algorithms. These volume data were selected from the Chapel Hill Volume Rendering Test Dataset [10]. Table 1 lists the dimensions and descriptions of these volume data. The first two test samples are "brain" volume data generated from a MR scan of a human head with two different resolutions (marked as small and large voxel sizes). The next three test samples are CT "head" volume data that have different resolutions (small, medium, and large). The last test sample is an "engine" volume data set, a CT scan of an engine block. Each image is grayscale and contains 256×256 pixels. Figure 9(a–c) shows the three test sample images.

Table 1. Dimensions and descriptions of the six test samples

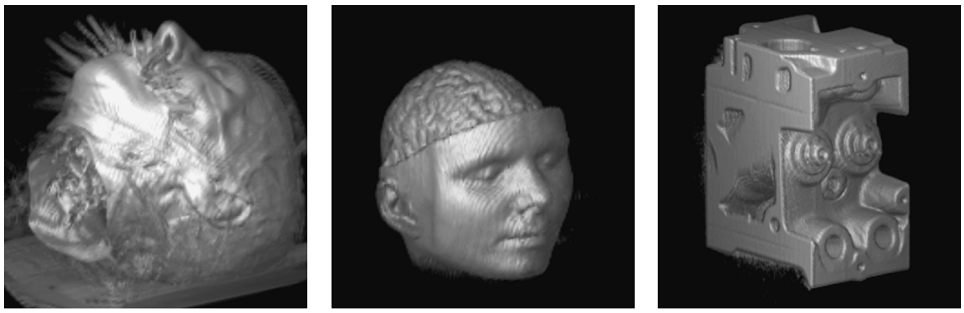
Test samples	Dimensions	Descriptions
Brain (small)	$128 \times 128 \times 84$	Applying a Gaussian filter; no further scaling is necessary
Brain (large)	$256 \times 256 \times 109$	Scaling 1.54× in the Z dimension
Head (small)	$128 \times 128 \times 113$	Applying a box filter; no further scaling is necessary
Head (medium)	$256 \times 256 \times 113$	Scaling 2× in the Z dimension
Head (large)	$256 \times 256 \times 225$	Applying a cubic bspline filter; no further scaling is necessary
Engine	$256 \times 256 \times 110$	No scaling

5.1. Comparison of the shear-warp rendering and image compositing time

Figure 10 shows the experimental results for the shear-warp and image compositing time for the large "head" test sample on 1, 2, 4, 8, 16, and 32 processors. We plotted the results from four different volume rendering methods in these figures for comparison. m_1, m_2, m_3 represent the volume rendering methods using the slice data partitioning, volume data partitioning, and sheared volume data partitioning methods, respectively. m_4 represents the 1D-partition method.

Figure 10(a) shows the shear-warp time results using different numbers of processors. From Figure 10(a), we can see that the shear-warp time for m_4 is less than that for the other methods. The reason is that the 1D-partition method uses the corresponding formulae to compute the partition size for each processor. This method can achieve better partition load balancing than the other methods. Figure 10(b) shows the image compositing time for these four different algorithms. The image compositing time for m_3 and m_4 is much less than that for m_1 and m_2. The reason is that m_3 and m_4 use only a merge method to assemble the partial final images while m_1 and m_2 must use the *over* operation to calculate the color and opacity in the overlapped parts of the partial final images. The shear-warp time and image compositing time results for the other five test samples are similar to this case.

Figure 11(a–f) shows the total rendering time for these four methods for the six test samples listed in Table 1, respectively. The total rendering time contains the shear-warp and image compositing time. In each figure the horizontal axis denotes



(a) (b) (c)

Figure 9. Test samples of parallel volume rendering methods. (a) CT scan "head" test sample ($256 \times 256 \times 225$); (b) MR scan "brain" test sample ($256 \times 256 \times 109$); (c) CT scan "engine" test sample ($256 \times 256 \times 110$).

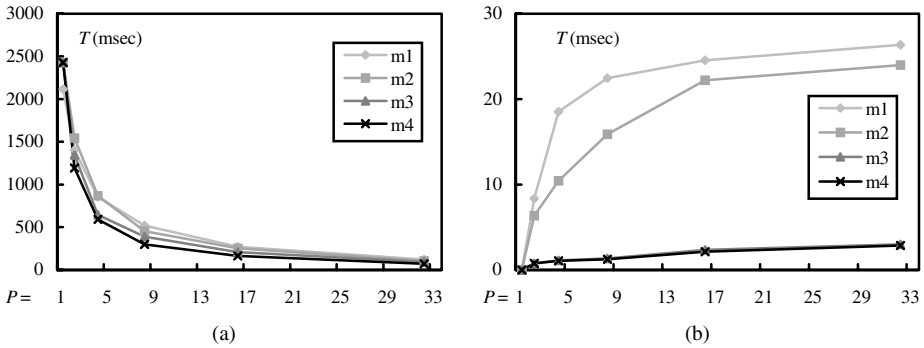


Figure 10. The shear-warp (a) and image compositing (b) time for the "head" test sample.

the number of processors and the vertical axis denotes the total rendering time for these four methods in mini-seconds. In all cases, m_4 has better performance than any of the other methods.

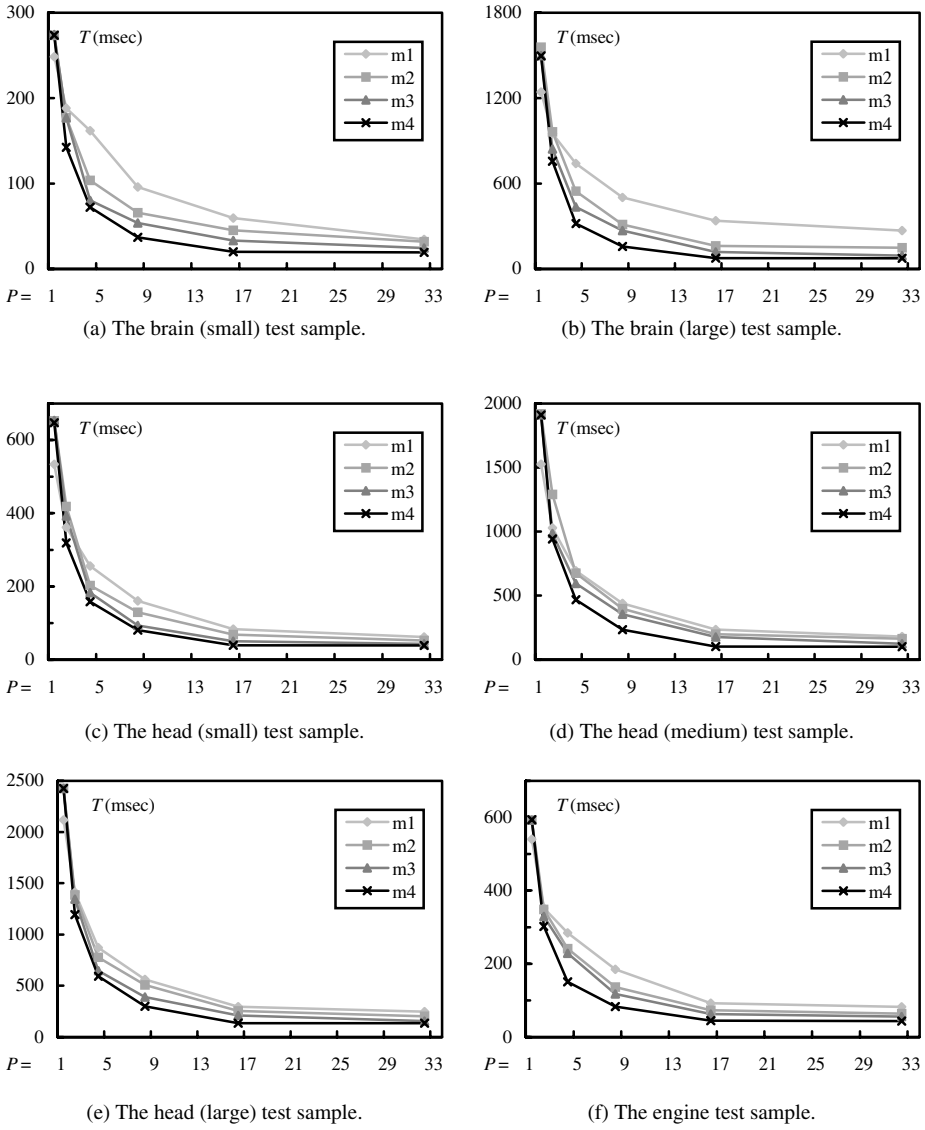
5.2. *The performance bound of the 1D-partition method*

According to Eq. (13), the performance of the 1D-partition method is bounded by $\sqrt{n}$. The experimental results were used to verify the following:

Tables 2 and 3 show the total rendering time for the 1D-partition method using different numbers of processors in the six test samples. According to Eq. (13), when the amount of volume data is small, such as the small "brain" test sample containing $128 \times 128 \times 84$ voxels and the small "head" test sample containing $128 \times 128 \times 113$ voxels, the upper bound appears when P is near $\sqrt{n} = \sqrt{128} \approx 12$. From Table 2, we can see that the total rendering time improvement for a small amount of volume data is very small when the number of processors is greater than 12. For a larger amount of volume data, such as the large "brain" test sample containing $256 \times 256 \times 109$ voxels and the large "head" test sample containing $256 \times 256 \times 225$ voxels, the upper bound appears when P is near $\sqrt{n} = \sqrt{256} = 16$. From Table 3, the observations are similar to that for the small volume data.

5.3. *The 1D-partition and 2D-partition methods performance comparison*

Table 4 shows the total rendering time for the 1D-partition and 2D-partition methods for the brain (small) and head (large) test samples on 32 processors. The total rendering time for the 1D-partition method is indicated in the last column labeled $P = 32$. For the brain (small) test sample, from Table 4, we observe that 4×8 has the shortest time of all. In this case, $\sqrt{n} = \sqrt{128} = 11.2 \approx 12$ is close to $P = 8$. We can see that the 2D-partition method performs better than 1D-partition method when P_h is close to $\sqrt{n}$. For the head (large) test sample, we observe that 2×16 has the shortest time of all. In this case, $\sqrt{n} = \sqrt{256} = 16 < P_h = 32$ and we can also see that the 2D-partition method performs better when P_h is close to $\sqrt{n}$.



(a) The brain (small) test sample.

(b) The brain (large) test sample.

(c) The head (small) test sample.

(d) The head (medium) test sample.

(e) The head (large) test sample.

(f) The engine test sample.

Figure 11. The total rendering time for all test samples.

Table 5 shows the total rendering time for the 1D-partition and 2D-partition methods for the brain (small) and head (large) test samples on 16 processors. The total rendering time for the 1D-partition method is indicated in the last column labeled $P = 16$. For the brain (small) test sample, from Table 5, we observe that 2×8 has the shortest time of all. In this case, $\sqrt{n} = \sqrt{128} = 11.2 \approx 12$ is close to $P = 8$. We can see that the 2D-partition method performs better than 1D-partition method when P_h is close to $\sqrt{n}$. For the head (large) test sample, we observe that 1×16 has the shortest time. In this case, $\sqrt{n} = \sqrt{256} = 16 = P$ and we can see that the 1D-partition method performs better than the 2D-partition method.

Table 2. The total rendering time (ms) for various processors for two small volume data

Test sample	P	10	11	12	13	14	15
Brain (small)	$T_{shear\text{-}warp}$	21.640	20.133	18.944	18.450	17.950	17.778
$128 \times 128 \times 84$	$T_{composite}$	2.190	2.266	2.329	2.394	2.434	2.451
	T_{total}	23.830	22.399	21.273	20.844	20.384	20.229
Head (small)	$T_{shear\text{-}warp}$	56.314	48.649	37.167	36.825	36.328	36.124
$128 \times 128 \times 113$	$T_{composite}$	2.038	2.148	2.279	2.288	2.329	2.378
	T_{total}	58.352	50.797	39.446	39.113	38.657	38.502

Table 3. The total rendering time (ms) for various processors for four large volume data

Test sample	P	14	15	16	17	18	19
Brain (large)	$T_{shear\text{-}warp}$	83.162	78.860	73.434	73.263	73.151	72.942
$256 \times 256 \times 109$	$T_{composite}$	3.457	3.482	3.513	3.527	3.553	3.581
	T_{total}	86.619	82.342	76.947	76.790	76.704	76.523
Head (medium)	$T_{shear\text{-}warp}$	118.557	113.240	119.111	98.745	97.802	97.179
$256 \times 256 \times 113$	$T_{composite}$	3.486	3.599	3.641	3.717	3.791	3.806
	T_{total}	122.043	116.839	102.752	102.462	101.593	100.985
Head (large)	$T_{shear\text{-}warp}$	156.311	143.406	131.033	129.819	128.597	118.332
$256 \times 256 \times 225$	$T_{composite}$	3.816	3.985	4.014	4.198	4.487	4.635
	T_{total}	160.127	147.391	135.047	134.017	133.084	132.967
Engine	$T_{shear\text{-}warp}$	48.219	44.670	40.924	39.972	38.663	38.194
$256 \times 256 \times 110$	$T_{composite}$	3.557	3.652	3.695	3.784	3.812	3.859
	T_{total}	51.776	48.322	44.619	43.756	42.475	42.053

Table 4. The rendering time (ms) for the 1D- and 2D-partition methods on 32 processors

		Partition method						
		2D-partition						1D-partion
Test samples	$P_v \times P_h$	1×32	2×16	4×8	8×4	16×2	32×1	$P = 32$
Brain (small)	T_{shear}	0.25	0.25	0.25	0.25	0.25	0.25	
$128 \times 128 \times 84$	$T_{project}$	15.14	9.55	5.19	7.28	12.65	20.55	
	T_{warp}	2.12	3.89	6.13	11.52	21.42	38.32	
	$T_{v\text{-}composite}$	0.00	3.43	5.43	10.32	13.42	18.32	
	$T_{h\text{-}merge}$	2.57	2.37	2.18	1.92	1.54	0.00	
	T_{total}	20.09	19.49	19.18	31.29	49.28	77.44	19.97
Head (large)	T_{shear}	2.73	2.73	2.73	2.73	2.73	2.73	
$256 \times 256 \times 225$	$T_{project}$	118.08	92.67	87.46	89.26	102.70	127.90	
	T_{warp}	9.27	13.93	26.79	51.75	84.15	107.43	
	$T_{v\text{-}composite}$	0.00	4.34	8.76	16.43	31.53	60.32	
	$T_{h\text{-}merge}$	4.73	4.34	4.12	3.78	3.43	0.00	
	T_{total}	129.81	118.01	129.85	163.94	224.53	298.37	129.17

From Tables 4 and 5, we have the following remarks.

Remark 1 When the number of processors is greater than $\sqrt{n}$ and a $P_v \times P_h$ processor grid is used in the 2D-partition method, better performance can be expected if the value of P_h is close to $\sqrt{n}$.

Table 5. The rendering time (ms) for the 1D- and 2D-partition methods on 16 processors

		Partition method					
		2D-partition					1D-partion
Test samples	$P_v \times P_h$	1×16	2×8	4×4	8×2	16×1	$P = 16$
Brain (small)	T_{shear}	0.52	0.52	0.52	0.52	0.52	
$128 \times 128 \times 84$	$T_{project}$	15.64	8.14	7.39	11.86	19.42	
	T_{warp}	2.46	5.43	8.94	14.43	19.54	
	$T_{v\text{-}composite}$	0.00	3.65	5.84	10.83	19.66	
	$T_{h\text{-}merge}$	2.29	2.15	1.98	1.47	0.00	
	T_{total}	20.91	19.88	24.67	39.10	59.14	20.21
Head (large)	T_{shear}	5.48	5.48	5.48	5.48	5.48	
$256 \times 256 \times 225$	$T_{project}$	114.33	108.32	94.84	110.15	133.06	
	T_{warp}	12.32	17.54	33.85	53.43	87.32	
	$T_{v\text{-}composite}$	0.00	4.43	9.74	16.78	31.43	
	$T_{h\text{-}merge}$	4.31	4.22	3.95	3.34	0.00	
	T_{total}	136.44	139.99	147.86	189.18	257.29	135.05

Remark 2 If the number of processors is greater than $\sqrt{n}$, the 2D-parition method outperforms the 1D-partition method when P_h is close to $\sqrt{n}$.

Remark 3 If the number of processors is less than $\sqrt{n}$, the 1D-parition method outperforms the 2D-partition method.

6. Conclusions

In this paper, we presented the 1D-partition and 2D-partition methods based on shear-warp factorization and demonstrated the performance improvement over three other parallel volume rendering algorithms: the slice data partitioning, volume data partitioning, and sheared volume data partitioning methods. All tests were performed on an IBM SP2 parallel machine. According to the number of processors, we used the either 1-D or 2-D partitioning schemes to partition a given volume data set. This flexible approach can efficiently partition the volume data for each processor with a balanced load distribution. After the data partitioning stage is completed, each processor employs shear-warp factorization to render the sub-volume and generate a partial final image. A simple merge operation is used to composite the final image with very short image compositing time. The experimental results demonstrate that the proposed approaches outperform other compatible algorithms and are viable methods for achieving high-speed volume rendering.

References

1. M. B. Amin, A. Grama, and V. Singh. Fast volume rendering using an efficient scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the 1995 Parallel Rendering Symposium*, pp. 7–14. Atlanta, October 1995.

2. B. Corrie and P. Mackerras. Parallel volume rendering and data coherence. In *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 23–26. San Jose, October 1993.
3. R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of SIGGRAPH'88*, vol. 22, pp. 65–74. Atlanta, 1988.
4. E. Groeller and W. Purgathofer. Coherence in computer graphics. Technical reports TR-186-2-95-04. Institute of Computer Graphics 186-2 Technical University of Vienna, March 1995.
5. W. M. Hsu. Segmented ray casting for data parallel volume rendering. In *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 7–14. San Jose, October 1993.
6. IBM. IBM AIX parallel environment. Parallel Programming Subroutine 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 the 1995 Parallel Rendering Symposium*, pp. 15–22. Atlanta, October 1995.
10. P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2:218–231, 1996.
11. P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. 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. In *Proceedings of SIGGRAPH '91*, vol. 25, pp. 285–288. Las Vegas, July 1991.
13. M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9:245–261, 1990.
14. 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 the 1993 Parallel Rendering Symposium*, pp. 15–22. San Jose, October 1993.
15. K. L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14:59–68, 1994.
16. MPI Forum. MPI: A message-passing interface standard. May 1994.
17. T. Porter and T. Duff. Compositing digital images. In *Proceedings of SIGGRAPH'84*, vol. 18, pp. 253–259, July 1984.
18. K. Sano, H. Kitajima, H. Kobayasi, and T. Nakamura. Parallel processing of the shear-warp factorization with the binary-swap method on a distributed-memory multiprocessor system. In *Proceedings of the 1997 Parallel Rendering Symposium*, October 20–21, 1997.
19. J. P. Singh, A. Gupta, and M. Levoy. Parallel visualization algorithms: Performance and architectural implications. *Computer*, 27:45–55, 1994.
20. C. Upson and M. Keeler. V-BUFFER: Visible volume rendering. In *Proceedings of SIGGRAPH'88*, vol. 22, pp. 59–64. Atlanta, 1988.
21. L. Westover. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH'90*, vol. 24, pp. 367–376. Dallas, 1990.
22. J. Wilhelms and A. Van Gelder. A coherent projection approach for direct volume rendering. In *Proceedings of SIGGRAPH'91*, vol. 25, pp. 275–283, July 1991.
23. C. M. Wittenbrink and A. K. Somani. Permutation warping for data parallel volume rendering. In *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 57–60. San Jose, October 1993.
24. 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*, 12:63–71, 1992.