1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

AN EFFICIENT PARALLEL VOLUME RENDERING ALGORITHM BASED ON SHEAR-WARP FACTORIZATION ON DISTRIBUTED MEMORY MULTIPROCESSOR SYSTEMS

Don-Lin Yang, Ching-Feng Lin and Yeh-Ching Chung

Department of Information Engineering and Computer Science,
Feng Chia University, Taichung, Taiwan 407, R.O.C.
E-mail:{dlyang, cflin, ychung}@pine.iecs.fcu.edu.tw

ABSTRACT

Volume Rendering is a technology to illustrate the shape and volumetric property of the objects in the areas like medical imaging and computational fluid dynamics. Because volume rendering takes a lot of time to process a large volume of data, it is very difficult to generate three-dimensional image in real-time. In this paper, we present an efficient parallel volume rendering algorithm based on shear-warp factorization for medical imaging on distributed memory multiprocessor systems. We first use a novel data partition method to distribute voxels into each processor element and to render partial subvolume images independently. In the reassembling stage, it only requires *merge time* to produce the final image. Our parallel volume rendering algorithm takes less memory space and rendering time on distributed memory multiprocessor systems. In our implementation, it can render a medical dataset of $256 \times 256 \times 225$ voxels at about 75 mini-second per frame on a 32-processor IBM SP2 distributed memory multiprocessor system. Therefore, our algorithm can be used to make real-time volume rendering.

1. INTRODUCTION

Volume Rendering is a technique to visualize three-dimensional volume data and to display it in two-dimensional image. The traditional rendering techniques are mostly used to display the surface of objects. Volume rendering [1] is now used to understand and analyze the shape of the objects in medical imaging and computational fluid dynamics. It can display the object with semi-opaque and also has a quality better than that of the surface. In medical imaging, the three-dimensional scanning devices (CT, PET, MRI and Ultrasounds) can directly capture the properties of three-dimensional objects in machine-readable format. Volume rendering can be used to study natural objects by analyzing large empirical data sets obtained from measurements or simulations.

Unfortunately, most of the existing volume rendering methods are very computationally intensive and therefore fail to achieve interactive rendering rates for large datasets. The volume data are too large to hold in the memory of a single processor element. Even with the latest volume rendering acceleration techniques, it still takes a few minutes to a few hours to render volume images in a single processor machine. It is not a real-time interactive volume rendering at all. Although today's computer technology continues to advance, the power of computer processing never seems to catch up with the increases in data volume. Even volume rendering acceleration techniques are available, one still faces the problem of handling a large amount of data. Besides, large datasets are very expensive in the storage and communication costs. In order to achieve interactive volume rendering, users often reduce the original data size and get inferior visualization results. From the above problems, we know that the visualization of volume data should be performed in a parallel fashion.

One can parallelize a volume rendering algorithm that uses the optimization technique to reduce computation costs. In distributed computing environments such as a campus-network, there are many workstations and personal computers that might be idle for many hours a day. One can integrate these workstations and personal computers to become a parallel computing environment. This allows us to distribute the large amount of data as well as the time-consuming rendering process to the available distributed computing resources.

Parallel volume rendering parallelizes the traditional serial rendering techniques. It distributes volume data into a lot of processor elements and reduces the computation time. Different parallel system architectures have different approaches and encounter various problems. For example, in a distributed memory multiprocessor system, the distributed volume rendering algorithm uses message passing to exchange data. The communication overhead is the main factor affecting the rendering speedup. However, in a shared memory multiprocessor system, Philippe Lacroute [2] found that a shared memory implementation of the shear-warp algorithm was not dominated by the costs of communication. Therefore, shared memory implementations of volume rendering do not have the communication problem.

A good volume rendering algorithm must ensure that all large data structures are distributed among the processors without wasteful duplication. In this paper, we present an efficient volume rendering method based on shear-warp factorization on distributed memory multiprocessor systems. The shear-warp factorization volume rendering algorithm is a fast algorithm used in many volume rendering algorithms. It was presented by Philippe Lacroute et al. [3] and had been implemented in the shared memory multiprocessor system such as DASH to perform high speedup rendering. However, the preprocessing time is too long and it requires a lot of memory as well. To solve these problems, we use an efficient and adaptive partition method to distribute volume datasets. Then each processor renders subvolume independently. By using a merge method, one can assemble rendered images to form the final image.

The rest of this paper is organized as follows. In Section 2, we describe a number of volume rendering algorithms which had been proposed recently. Then we explain the serial shear-warp factorization volume rendering algorithm in Section 3. Section 4 presents our efficient parallel volume rendering algorithm on distributed memory multiprocessor systems. Section 5 shows the implementation on IBM SP2 for our parallel volume rendering algorithm and compares the results with that of other volume rendering algorithms on IBM SP2. Section 6 concludes our paper.

2. RELATED WORK

A number of algorithms have been proposed for volume rendering. They can be broadly classified as ray tracing algorithms, splatting algorithms, cell projection algorithms, multi-pass resampling algorithms, or shear-warp factorizations.

Ray tracing algorithms [4][5] are called backward projection or image order algorithms. They first trace a ray through the volume data for each image pixel, compute their color and opacity, and then produce final image. Splatting algorithms [6] are called forward projection or object order algorithms. They compute the contribution of a voxel to the image by convolving the voxel with a filter that distributes the voxel's value to a neighborhood of pixels. The cell projection algorithms [7] are similar to splatting except that the former use polygon scan conversion to perform the projection. The multi-pass resampling algorithms [8] operate by resampling the entire volume to the image coordinate system. Catmull and Smith introduces multi-pass resampling for warping two-dimensional images, and the technique was first applied to volume rendering at Pixar [9]. The shear-warp factorization is a fast volume rendering algorithm [10]. It has a high performance of parallel computing approach when optimize this algorithm by using early-ray terminating and run-length encoding. In this research, we develop our volume rendering algorithm based on this shear-warp factorization.

Recently, two parallel volume rendering algorithms for the shear-warp factorization had been proposed. Lacroute [10] proposed a parallel shear-warp factorization algorithm for two types of shared-memory multiprocessors (SGI Challenge and Stanford DASH). Lacroute achieved 15 frames per second for 256 × 256 × 167 voxels on an SGI Challenge 32-processor shared-memory computer. Kentaro Sano et al. [11] present another parallel volume rendering algorithm. They implemented shear-warp factorization and achieved 12 frames per second for 256 × 256 × 256 on an IBM SP2 32-processor distributed-memory multiprocessor. This algorithm first partitions the slices and distributes them into each processor, then it uses the shear-warp factorization to render sub-volume to generate intermediate images. Finally, they use binary-swap compositing method to assemble the final image.

3. THE SERIAL VOLUME RENDERING OF SHEAR-WARP FACTORIZATION

In this section, we describe the serial volume rendering of shear-warp factorization. It is an object-order volume rendering algorithm and has three major steps. First, a three-dimensional shear is based on a factorization of the viewing transformation matrix. Then use projection to generate a distorted intermediate image. Finally, warping a two-dimensional image to form an undistorted final image [9].

Figure 1 illustrates the volume rendering of shear-warp factorization. In this figure, we denote the horizontal lines to represent slices of volume data that were sampled on a rectangular grid. In the first step, we shear the volume slices such that the viewing rays become perpendicular to the slices of the volume. Then the resampled slices are assembled together in a front-to-back order using the "over" operator to generate an intermediate image. Finally, the intermediate image must be transformed into the correct final image by applying an affine 2D warp.

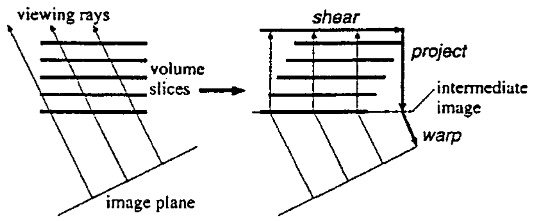


Figure 1: Serial shear-warp factorization.

4. AN EFFICIENT PARALLEL VOLUME RENDERING ALGORITHM BASED ON SHEAR-WARP FACTORIZATION

To implement the shear-warp factorization in distributed memory multiprocessor systems, an efficient method to minimize rendering time is important. We first employ an adaptive data partitioning method to distribute the volume dataset, then use shear-warp factorization to generate intermediate images, warping these images and reassembling them to form the final image. In the following subsections, we will discuss the methods used in data partitioning stage and image compositing stage.

4.1 Data partitioning

In the volume rendering of a distributed memory multiprocessor system, the first stage is partitioning the original three-dimensional volume datasets and distributing the subvolume to each processor element. If the data partitioning method is efficient, the computation time and rendering time can be reduced and the communication overhead can also be decreased.

In the data partitioning stage of Kentaro Sano et al. [11], they employ the slice volume partition method to distribute volume data among processors. They group volume slices onto a set of subvolumes. Then each processor will have several continuous volume slices. In each processor, it employs shear-warp factorization method to perform run-length encoding and resample of allocated subvolumes. Then each subvolume image is generated in parallel. The main advantage of this method is easy to implement in parallel because this is a simple volume data partitioning method. On the other hand, the disadvantage of this method is that the total size of the intermediate images generated from all processors is equal to that of the final image. They must use "over" [12] operation to decide the opacity and color in each image pixel as shown in Figure 2.

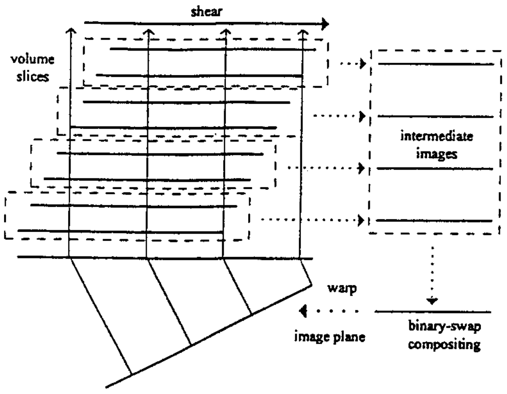


Figure 2: Slice data partitioning and binary swap compositing method

To avoid getting a large size of intermediate image in each processor, we can slice the volume datasets in each processor by vertical direction instead of slice partition as shown in Figure 3. In this figure, there are four processor elements. The volume dataset is sliced into four parts that are denoted by parallelogram of dash lines. Each part contains one piece of the slices. When shearing the volume datasets, the intermediate images will have overlapping areas and the intersection parts will be assembled, which are generated from each processor element. While the overlapping areas of intermediate images have the communication overheads, the compositing intersection parts of intermediate images have extra computation overheads. The communication time and computation time will be increased when it is sheared in a large angle. This is because each processor needs to exchange more intermediate image data resulting more scanlines and uses "over" [12] repeatedly to compute the color and opacity for each image pixel.

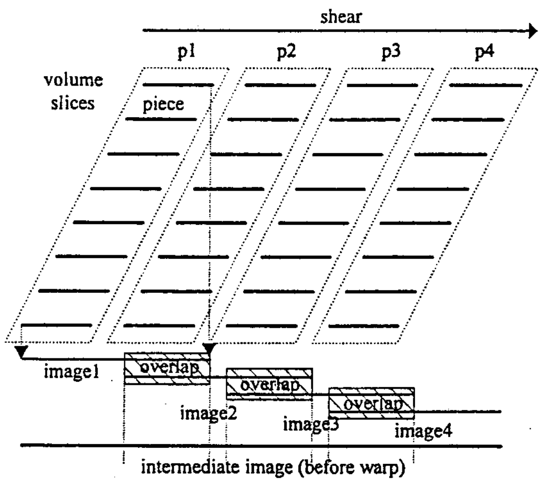


Figure 3. Volume data partition method

To avoid the communication overheads of overlapped areas, one can shear the volume datasets and then partition them by slicing orthogonally to volume slices according to the angle of viewing direction. Figure 4 shows the example of this method. From Figure 4, we can see that the intermediate image of each processor does not have overlapping area. The intermediate image can be warped independently. We employ the mathematics formula to decide the size of volume datasets which are partitioned efficiently and adaptively by the sheared angle. Note that the sizes of intermediate images in the processor elements are not equal. After the data partitioning is finished, we then use *shading table* to compute the shading of voxels and use shear-warp method to generate partial image rendering.

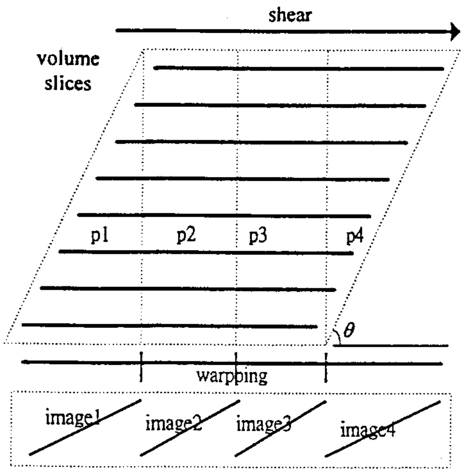


Figure 4. Example of adaptive data partition method.

To decide the size of subvolume dataset, we derive the mathematics formula in three cases: tanθ < $\sqrt[3]{2/p}$, tanθ = $\sqrt[3]{2/p}$, and tanθ > $\sqrt[3]{2/p}$, where θ is the angle for shearing and p is the number of processors. For case 1 (Figure 5(a)), the mathematics formula to decide the size of subvolume dataset is given as follows,

- The left- and right-side parts (denoted as (i) in Figure 5(a)):

$$I = n\sqrt{\frac{2}{p\tan\theta}}(\sqrt{p'} - 1) \qquad (1)$$

- The trapezoid parts (denoted as (ii) in Figure 5(a)):

$$I = n(\frac{2}{p} - \frac{1}{2}\tan\theta + 1 - \sqrt{\frac{2}{p\tan\theta}}) \qquad (2)$$

- The middle rectangle parts (denoted as (iii) in Figure 5(a)):

$$I = \frac{n}{p-4}(\frac{1}{2}\tan\theta - \frac{4}{p} - 1) \qquad (3)$$

Here I is the size of a partial image and $p' = \frac{n}{\tan\theta}$.

For case 2 (Figure 5(b)), the mathematics formula to decide the size of subvolume dataset is given as follows,

- The left- and right-triangle parts (denoted as) (i) in Figure 5(b)):

$$I = n\tan\theta \qquad (4)$$

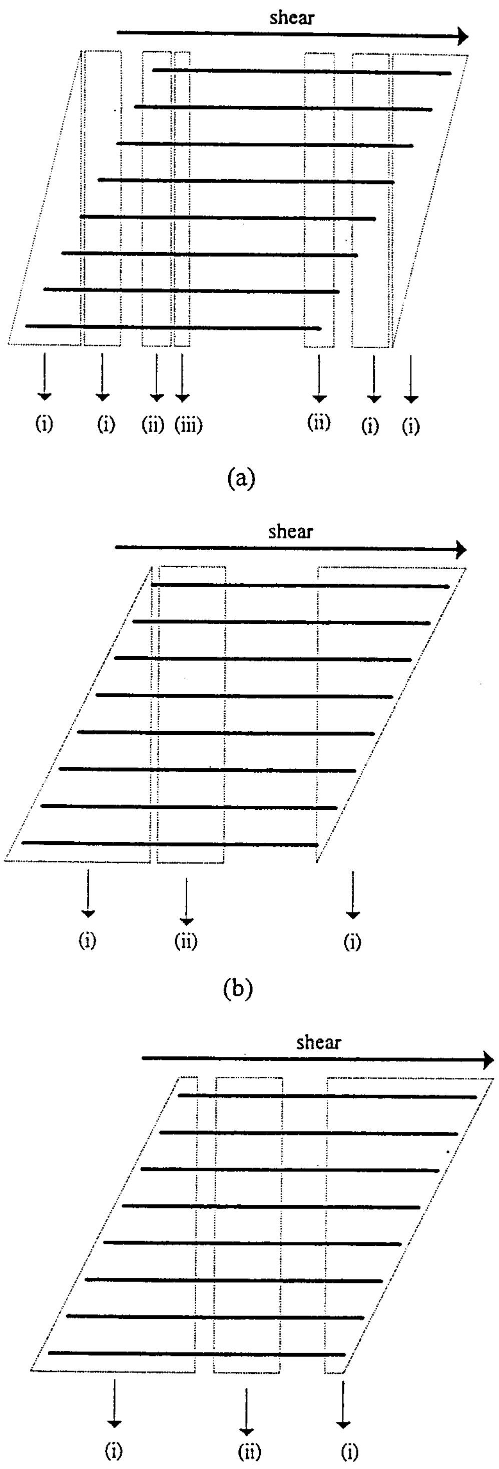


Figure 5: The three cases of mathematics formula

- The middle rectangle parts : (denoted as (ii) in Figure 5(b)):

$$I = \frac{n}{p-4}\left(\frac{1}{2}\tan\theta - \frac{4}{p} - 1\right) \qquad (5)$$

Here I is the size of a partial image.

For case 3 (Figure 5(c)), the mathematics formula to decide the size of subvolume dataset is given as follows,

- The trapezoid parts : (denoted as (i) in Figure 5(c)):

$$I = n\left(\tan\theta + \frac{1}{p} - \frac{1}{2}\tan\theta\right) \qquad (6)$$

- The middle rectangle parts : (denoted as (ii) in Figure 5(c)):

$$I = \frac{n\left(1 + \frac{2}{p} - 3\tan\theta\right)}{p-2} \qquad (7)$$

Here I is the size of a partial image.

4.2 Image Compositing

The image compositing stage reassembles the intermediate images, which are generated from individual processor elements, to form the final image. After the volume rendering, each processor element will have one subvolume image which contains partial intermediate image. Then, different compositing methods are used to combine these intermediate images to compose the final image.

Kentaro. Sano et al. [11] employ the binary-swap compositing method [13] to produce the final image. The binary-swap compositing method is a divide-and-conquer method. The main advantage of the binary-swap compositing method is that the image size decreases in the dividing process and the compositing process is more efficient. However, the number of processor elements is restricted in a power of two. It is not adaptable for the cases when the processor elements are not equal to a power of two.

In our image compositing method, since the rendered partial images are generated independently, the images do not have overlapping areas and intersecting parts. We only need to use a *merge method* to assemble them to the final image. By using the directives of message passing library such as MPI on a distributed memory multiprocessor system, the compositing time is reduced and fixed. Therefore, the advantages of our method are twofold: (1) it is not limited by the number of processor elements and (2) the time used is short and fixed.

5. PERFORMANCE EVALUATION

In this section, timing results for the parallel volume rendering implementation are presented. We also compare the proposed method with other volume rendering algorithms [11]. We first describe our volume rendering system architecture in Section 5.1. Then we discuss the performance results and comparison of our volume rendering algorithm in Section 5.2.

5.1 System Architecture

In our volume rendering system, we have implemented our parallel volume rendering algorithm based on shear-warp factorization on an IBM SP2 [14]. The IBM SP2 is supported from the National Center of High performance Computing in Taiwan. The IBM SP2 is a super-scalar architecture and the CPU model is IBM RISC System/6000 POWER2 which clock rate is 66.7 MHz. There are 40 IBM POWER2 nodes in the IBM SP2. Each node has 128KB 1st-level data cache, 32KB 1st-level instruction cache and the memory capacity is 128MB. Each node is interconnected based upon a low-latency, high-bandwidth interconnection network called the High Performance Switch (HPS).

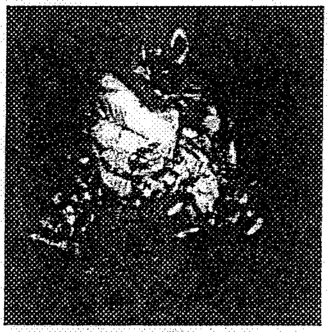
In the software, we have used C language and MPL message-passing library to implement our parallel volume rendering algorithm on the IBM SP2. The MPL message-passing library is the IBM's message passing library and it is similar to the MPICH, which is one of the MPI [15] message-passing libraries. MPI is a standard for message passing to send and receive data in a parallel architecture system. So our volume rendering algorithms are portable and can be implemented in different distributed memory multiprocessor systems.

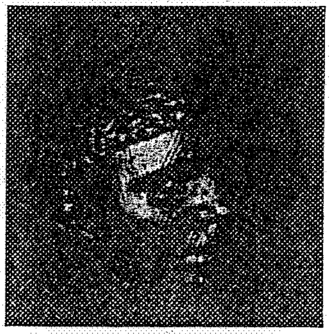
5.2 Experimental Results

To evaluate the performance of our parallel volume rendering algorithm, we use six different datasets as test samples. These datasets are supported from the Chapel Hill Volume Rendering Test Dataset. The dimensions and descriptions of these datasets are shown in Table 1. The first and the second samples are the "brain" dataset which is an MR scan of a human head volume data and it is resampled to two different resolution voxel sizes. The third, the forth, and the fifth samples are the "head" dataset which is a CT scan of a human cadaver head and it is resampled to three different resolution voxel sizes. The last sample is the "engine" dataset which is a CT scan of an engine block. The images are grayscale and contain 256 × 256 pixels. Figures 6 (a) - (c) show the three final images of our test datasets.

Table 1. Dimensions and descriptions for test datasets.

Dataset	Dimensions	Descriptions
Brain (small)	128 × 128 × 84	Applying a Gaussian filter; no further scaling is necessary
Brain (big)	128 × 128 × 109	Scaling 1.54x in the Z dimension
Head (small)	128 × 128 × 113	Applying a box filter; no further scaling is necessary
Head (middle)	256× 256 × 113	Scaling 2x in the Z dimension
Head(big)	256× 256 × 225	Applying a cubic bspline filter; no further scaling is necessary
Engine	256× 256 × 110	No scaling


(a)


(b)

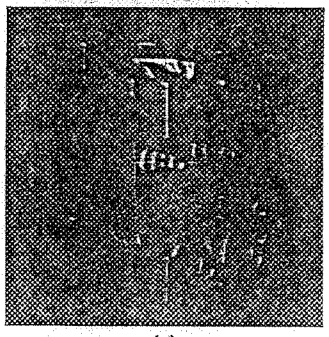
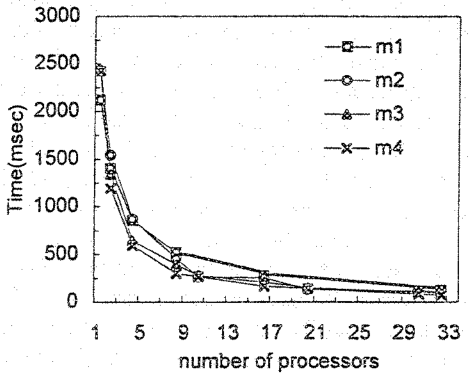
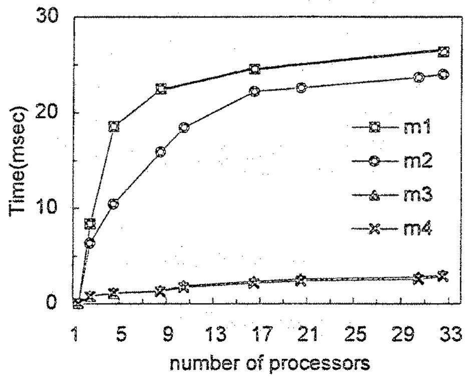

(c)

Figure 6: Test data sets for parallel volume rendering algorithm. (a) A CT scan "head" (128 × 128 × 110) (b) A MR scan "brain" (128 × 128 × 84) (c) A CT scan "engine" (256 ×256 × 110)



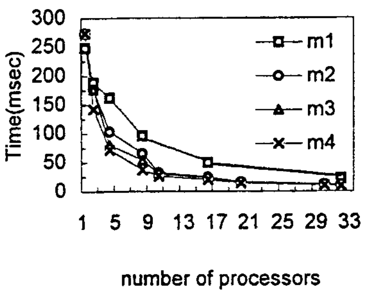
(a) Shear-warp time



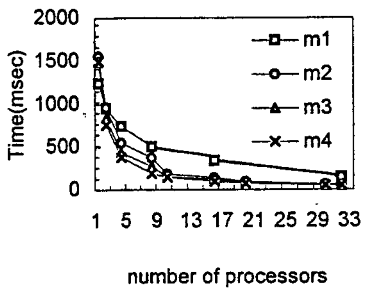
(b) Compositing time

Fig 7: The shear-warp time and compositing time on the 256 × 256 × 225 dataset (a CT scan "head")

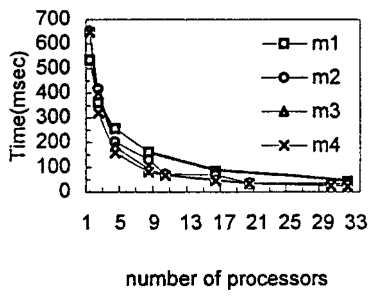
Figures 7(a) and (b) show the results of shear-warp time and compositing time. The shear-warp time contains shear time and warp time while the compositing time contains the "over" operation time and the merge time. In these figures, we compare four different volume rendering algorithms. The *m1* represents the algorithm that was reported by Sano et al. [12]. It uses slice data partition and binary-swap compositing method. Since the *m1* method uses the binary-swap compositing approach, it does not have the timing results when the number of processors is 10, 20 and 30. The *m2* represents the algorithm that was described in the second data partition method in Section 4.1. It uses volume data partition and compositing method. The *m3* and *m4* represent the algorithm that was described in the third data partition method in Section 4.1. It uses sliced volume data partition and merge compositing method. The *m4* method slices volume data partition with the mathematics formula and merge compositing method while *m3* does not use the mathematics formula. Figure 7(a) shows the timing results of shear-warp in different number of processors. The time of *m4* method is less than any other methods. This is because that it uses the mathematics formula to decide its partition size in each processor. The load balancing in the *m4* method is better than others. Figure 7(b) shows the compositing time of these four different algorithms. The compositing time of the *m3* and *m4* methods is lower than that of the *m1* and the *m2* methods. This is because the *m3* and *m4* methods only use merge method to assemble the partial images. However, the *m1* and *m2* method must use "over" operation to decide the color and opacity in the intersection parts of intermediate images. The timing results of shear-warp time and compositing time for other test datasets are similar to this case. Due to the page limitation, we only present the total rendering time of these four algorithms for test datasets shown in Table 1.



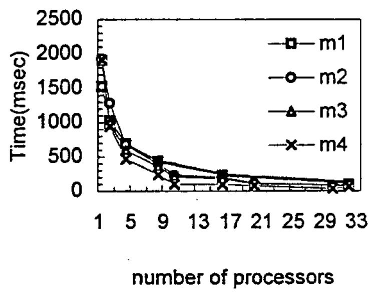
(a) Brain (small) dataset



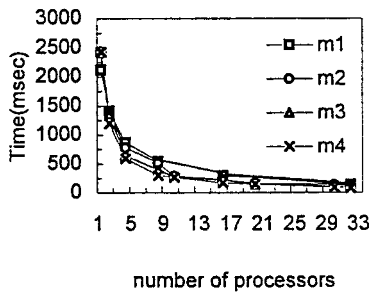
(b) Brain (big) dataset



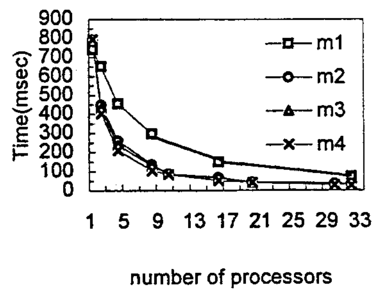
(c) Head (small) dataset



(d) Head dataset



(e) Head (big) dataset



(f) Engine dataset

Fig 8: The total rendering time of all sample datasets.

Figures 8(a) to 8(f) show the total rendering time for the six test datasets described in Table 1. The total rendering time contains the shear-warp time, the data communication time and the compositing time. We also compare four algorithms in each test data set. In any case, the *m4* method has a better performance than the other methods.

6. CONCLUSIONS

In this paper, we have presented an efficient volume rendering algorithm based on the shear-warp factorization and demonstrated its performance on distributed memory multiprocessor systems. As the experimental results show, our algorithm has a better performance based on shear-warp factorization in rendering medical images. We use the mathematics formula to decide the sizes of subvolume datasets. This method partitions volume data efficiently and distributes among processors. Each processor employs the shear-warp factorization to render subvolume and to generate intermediate image. Then we use the merge method to assemble the final image. Our volume rendering algorithm is attractive for its exploitation of volume data partition method. The experimental results show that the proposed algorithm is a viable means of achieving real-time volume rendering.

REFERENCES

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

[2] P. Lacroute, "Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization, " *IEEE Trans. on VCG.*, vol. 2, no. 3, pp. 218-231, 1996.

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

[4] W.M. Hsu, "Segmented Ray Casting for Data Parallel Volume Rendering," *Proc. 1993 Parallel Rendering Symp.*, pp. 7-14, San Jose, Oct. 1993.

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

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

[7] Westover, L. "Footprint evaluation for volume rendering", *Computer Graphics (SIG-GRAPH'90 Proceedings)*, vol.24, Dallas, pp. 367-376, 1990.

[8] Upson, C. and Keeler, M. "V-BUFFER: Visible volume rendering," *Computer Graphics (SIGGRAPH'88 Proceedings)*, vol. 22, Atlanta, pp. 59-64, 1988.

[9] Drebin, R. A., Carpenter, L. and Hanrahan, P. "Volume rendering, " *Computer Graphics (SIGGRAPH'88 Proceedings)*, vol 22, Atlanta, pp.65-74, 1988.

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

[11]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", *Parallel Rendering Symposium '97 (PRS97)*, October 20th-21st, 1997.

[12]T. Porter and T. Duff, "Compositing Digital Images, " *Proc. of SIGGRAPH'84*, volume 18, pp. 253-259, July 1984.

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

[14]IBM, IBM AIX parallel Environment, Paralllel Programming Subroutune Reference.

[15]MPI Fourm, MPI: A message-passing interface standard, may 1994.