Data Dependent Optimizations for Permutation Volume Rendering

Craig M. Wittenbrink
Kwansik Kim
Alex T. Pang*

UCSC-CRL-96-24
December 11, 1996

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Volume rendering or volume visualization is an algorithm for creating images from three-dimensional and four-dimensional data sets, without computing intermediate surface representations. Because of the inherent $O(N^3)$ run time, numerous approximations are used to provide interactivity. As compute platforms have become more capable in their operations per second, and in their memory capacities, the requirements of volume rendering become more and more advanced. The goal of application users is high fidelity renderings of medical, simulation, and remotely sensed data sets. Interactivity provides a tractable means for setting the many input parameters, but interactivity is challenging because of the run time complexity. We investigate the further optimization of massively parallel algorithm solutions, in an effort to have the efficiency of the parallel approaches reach that of sequential ones, without sacrificing fidelity. This paper discusses experiments in extending permutation warping on the MasPar MP-2 implementation to include data dependent coherency optimizations, and object space to screen space precompositing communication savings. We start with a description of the base algorithm, and then describe the extensions, showing results of nearly 400% speed improvement or a factor of 5 speedup.

Keywords: parallel volume visualization, SIMD, algorithms, octree, ray tracing

---

*Partially supported by a grant from ISCR-LLNL B291836 and NSF IRI-9423881, C.M.Wittenbrink is with Hewlett-Packard Laboratories, Palo Alto, CA

Keywords:

1 Introduction

Volume rendering algorithms calculate visualizations from sampled medical and simulation data, and researchers have sought to speed up the algorithms to make them more useful. Our algorithm uses permutation warping to achieve linear speedup and linear storage on data parallel machines. The algorithm supports arbitrary view directions, large data sets, large parallel machines, and high order filters, combined features not supported by other data parallel algorithms. We have derived a new one-to-one processor assignment that improves on the previously known solutions. On the MasPar MP-1, we have shown a speedup of 15.7 for a 16 k processor MP-1 over a 1k processor MP-1, and two frames/second with a 128^3 volume and trilinear view reconstruction. Supra linear speedups of over 20 are achieved with near neighbor filtering [1]. We have also shown that run time is constant across view angle, that the algorithm has tunable filter quality, and that one may achieve efficient memory implementation.

In this paper we investigate extensions to the algorithm to make it competitive in speed, while supporting superior filtering qualities to parallel variants of the shear warp factorization algorithm. The focus of this paper are extensions and implementation results on the MasPar MP-2. The ability to do load balancing, culling, and adaptive ray termination are possible with scalable, efficient massively parallel MIMD and SIMD algorithms. Implementation of remote frame viewing for interactive visualization has also been developed.

Even though the performance of sequential algorithms is impressive, there are always factors that drive for higher performance solutions including growing data set sizes, algorithm requirements, and user expectations. Data set sizes are increasing because of the increased sophistication of scanners, data fusion methods, and dramatic price reductions of disk and random access memory. Applications which have recently driven up the required memory capabilities are irregular grids from computational fluid dynamics and the visible human data sets. Algorithm requirements are a moving target, as users want increasingly sophisticated shading, classification, and interaction methods. As more capable algorithms and/or hardware are made available, users prefer higher and higher fidelity solutions, which may use improved gradient calculations, supersampling, and larger frame sizes.

User expectations are also changing, because of the real-time interactivity on surface shaded graphics, users are inconvenienced when an application doesn't achieve the same performance. The use of volume rendering in applications likely lags behind, because there is always a gap in performance between surface/wire frame graphics and volume rendering.

In the pursuit of the highest performance volume rendering solutions, three approaches have been taken: parallel algorithms on general parallel machines, parallel algorithms on special purpose graphics hardware, and special purpose volume rendering hardware. There is a place for all three of these approaches, and one may imagine that the best approach will differ depending on metric, state of the art, and price point. We have done work on general parallel machines [2, 3, 1] (MasPar, Proteus), and continue to research that approach. The highest performance in frames/second has been the parallelization of the shear warp approach of Lacroute [4, 5, 6, 7]. Volume rendering algorithms on special purpose graphics hardware include Cabral et al. [8]'s use of the SGI Reality Engine's Texture mapping hardware, SGI Infinite Reality Engine MIP Mapping hardware for 3D texture mapping, and special purpose volume rendering hardware includes Meagher's Octree Corporation volume rendering accelerator [9] and Pfister and Kaufman et al.'s Cube architecture [10].

In the pursuit of the highest performance solutions there are several challenges: achieve the highest performance, achieve the highest efficiency, create new capabilities, and create new applications. Imagine even without any innovation in algorithms, the highest performance will double every 18 months given the advancements in commodity computer hardware. If you simply had enough money to buy the latest, fastest shared address space multiprocessor, each year, you would be able to publish improvements. What balances this out is the growth in requirements, data set sizes, and expectations.

We have pursued improvements in massively parallel solutions. The parallel shear warp suffers from a lack of scalability and while the highest performance general purpose machine algorithm, is not efficiently parallelizeable beyond tens of processors. Our permutation warping approach has been proven to be scalable, but it's efficiencies do not match the shear warp approach. The efficiencies don't match because we use a trilinear reconstruction (7 multiplies), and the shear warp uses a bilinear reconstruction (3 multiplies) which means we're doing roughly 2.3 times the work to compute the more accurate filter. Other efficiency differences include data dependent optimizations, such as the thresholding and run length codings. The future requirements of scalability to thousands of processors and highest fidelity will make permutation warping an attractive alternative, and so we are researching the inclusion of data dependent optimizations. We give a short background on permutation warping, Section 2, then present our research methodology, Section 3, and research results, Section 4.

2 Background on Permutation Warping

Rendering is the creation of images from models, whether those models are geometric or volumetric. Medical imaging by rendering of slices and volumes of data, and by changing rendering parameters allows searching for different tissues or abnormalities. Development at the University of California, Santa Cruz uses the MasPar MP-2 4096 node machine, the Silicon Graphics Reality Engine 2 with 4 processors, and several Silicon Graphics Indigo-2 Extremes. We have also developed remote frame viewing solutions, in order to effectively use the remote MPP's [11].

We developed several solutions to various aspects of the rendering problem. Our solutions can be divided into primarily two areas, memory access efficiency, and network efficiency. We invented a cache architecture scheme, a storing of a subset of data for immediate access, which uses tiling to achieve efficiency with little hardware. Tiling is also called blocking, chuncking, and mosaicing–a recent example of such subdivision is in the Microsoft Talisman architecture [12]. In one study, we showed the utility of such a tiling scheme in working with tiles in images [13]. In another study, we showed how tiling improved the effectiveness of a massively parallel rendering algorithm [1]. To go to the next step in the development of tiling requires generalizing the tiler to work not just with common image processing operations and volume rendering, but with many classes of rendering algorithms. Another solution we developed is permutation warping for network efficiency. A permutation is a pairing, similar to choosing partners at a dance. If there is a one-to-one pairing, then the pairing is called a permutation. We showed that for spatial assignment in rendering, such a dance hall pairing can be computed, and therefore rendering requires simply a one-on-one tete-a-tete instead of a large conference [14, 1]. This holds for regular grids, and many data sets of interest are regularly gridded. We work with atmospheric circulation data, finite element trabecular bone data, and the numerous regularly gridded example data sets publicly available.

Our solutions have been published [15, 13, 14, 1], but their utility has not been widely seized upon, and collaboration with LLNL was started to port these solutions onto the world wide web for current biomedical studies [11]. It was hoped the use of the developed volume rendering tools would help to prove or disprove their efficiency. By generalizing tiling and permutation warping to more classes of rendering and showing implementations on more machines, it becomes more important to a wider audience of people. And, by investigating solutions for accelerated rendering on real problems, important directions in further parallel volume rendering research may be found.

Volume rendering is an algorithm that computes the interaction of light in a volume of light scattering particles. The algorithm uses geometric spaces between which transforms are performed: (Step 1) object space (OS) points , (Step 2) three dimensional screen space (SS) points, and (Step 3) two dimensional screen space points. The domain of points in each space is defined by bounding hulls. The three steps are: (1) the preprocessing stage (PPS), (2) the volume warping stage (VWS), and (3) the compositing stage (CS). The inputs to the algorithm are a scalar valued volume, a set of light sources and their positions, a viewing transform matrix, a classification function, used to convert derived values to densities, and a shading function, which calculates the lighting and illumination effects. The PPS calculates normals, opacities, and initial shaded intensities. Classification and

shading is discussed in detail in the background references [16, 17, 18, 19]. The VWS transforms the initial shading intensities and the opacities to the three dimensional screen space by resampling. The CS evaluates the view ray line integrals to get the two dimensional screen space pixel intensities. The final output is a two dimensional array of pixel values. For a more in depth discussion of volume rendering see Collected papers in Kaufman [20], Blinn [16], Kajiya and von Herzen [17], Levoy [18, 19], and our survey [13].

The array of output pixel intensities can be calculated many different ways indicated by the numerous input variables: volume data, light sources, view transform, classification function, and shading function. An illustrative categorization of possible algorithms is by viewing transform. Existing parallel algorithms may be grouped into four categories determined by their viewing transforms: backwards, multipass forwards, forwards splatting, and forwards wavefront. A backwards viewing transform is ray tracing [21]. Nieh and Levoy [22], Yoo et al. [23], Montani et al. [24], Neumann [25], Goel et al. [26], and Hsu [27] have developed backwards (ray tracing) volume rendering algorithms for parallel computers. Our permutation warping [2] approach computes a backwards mapping algorithm with optimal storage and deterministic communication on shared or distributed memory machines.

Figures 2.1 and 2.2 illustrate the transforms calculated by processors. The object space and screen space are separated, the object space on the left and the screen space on the right. A processor does permutation warping by: 2.1) Calculating processor assignments; 2.2) Calculating the reconstruction point; 2.3) Performing resampling and reading the values of its neighboring processors; (The number of neighbors used determines the filter order.) And, 2.4) Sending resampled values to screen processors. Figure 2.1 shows the aligned subimages in two-dimensions. In Step 3, a parallel product evaluation combines resampled intensities and opacities. Binary tree combining computes products for any associative (not necessarily commutative) operator. For details of binary tree compositing, see our work [28, 3].

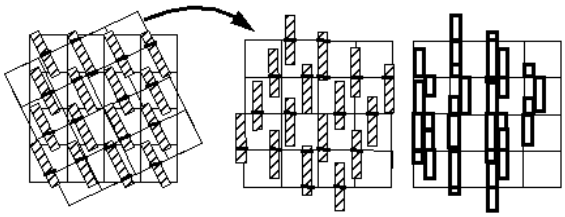

Figure 2.1: Volume in two dimensions showing the communication and depth sorting with a permutation warping assignment. The arrow shows the data communication from object space (OS) to screen space (SS). The final 2D volume on the right shows the communicated fragments properly sorted.

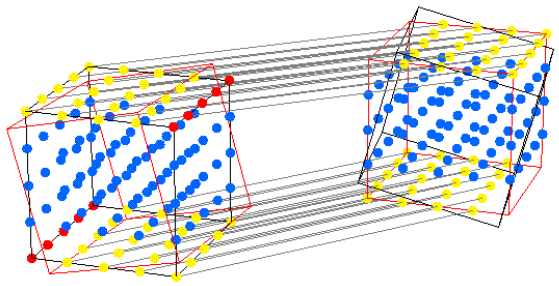

Figure 2.2: Volume transforms in parallel.

Several enhancements are possible for the permutation warping algorithm. Load balancing, region culling, and local subcube adaptive termination. We describe these enhancements, and their

expected improvement for the algorithm. Our prior algorithmic studies of permutation warping have shown that it is asymptotically time and space optimal for resampling on the EREW PRAM (exclusive read exclusive write parallel random access machine) [28, 2, 3, 14, 1]. The MIMD study that we carried out showed that there was poor load balacing for certain view angles. An effective load balancing strategy is a must for MIMD variants of the permutation warping algorithm, and the proposed work includes altering the mapping of permutations to virtual addresses which will allow the use of slackness [29], and also redistribution of processor's work. The permutation assignment can be done from virtual processor address to virtual processor address, and the virtual assignments can be dynamically altered to load balance the processing as it occurs.

The Lacroute et al. shear warp factorization algorithm, [4], has been able to reduce the amount of work necessary in computing a volume rendered output from regular volumes by nearly an order of magnitude. Parallel versions of this program have been presented [7], and the primary speedup is through straight forward parallelization of the sequential algorithm. Culling of the amount of volume that needs to be considered speeds up the algorithm. But, their studies show that the possible speedup to higher numbers of processors is limited because of the decomposition. Permutation warping can scale further as we demonstrated on the SIMD implementations, and it is also possible to use similar culling and compression to further improve efficiencies of our algorithm. Culling through run length encoding, and other compression schemes have been investigated, using the parallel permutation warping algorithm.

Because, in permutation warping, each subvolume is rendered as if a separate volume rendering job, some amount of adaptive termination along a ray may be performed. In addition, as the subvolume's contributions are computed, an amount of work may be reduced by doing front to back parallel evaluation and terminating once opacity has reached a threshold. The adaptive termination in the subvolume is a straightforward addition, while the adaptive termination across subvolumes can be done in a variety of ways. The improvement is expected to partially depend on the architecture.

3 Methodology

We have the following methodology with which to pursue our research for scalable permutation volume rendering: analytical evaluation of expected improvement followed by performance studies to provide evidence to support or deny analysis. We have attempted to get access to massively parallel MIMD (Multiple Instruction Multiple Data) platforms including those of our collaborators at Lawrence Livermore National Laboratories: a Meiko Scientific CS-2, and a Cray T3D. In addition, H.P. Convex platforms are available to us at H.P. Labs. There is also possible access to regional supercomputing centers, such as the San Diego Supercomputing center, but we have taken advantage of the machines at the University of California, Santa Cruz including a 4096 processor MP2 SIMD machine. Because this machine has thousands of processors, and a general interconnection network, it is suitable for studies in scalability, though may be dated for trying to compare numbers for raw performance. We hope to port our software to a variety of machines to investigate the scalability issues.

The analytical evaluation methodology provides an estimated performance for algorithm alternatives. We have used abstract parallel machines to quantify the relevant machine parameters. The first analysis was to investigate the known optimizations. The data coherency can be investigated in terms of how much performance improvement may be gained. There is volume coherency, ray coherency, and image coherency. That may all be exploited to reduce computation, and bandwidth.

The challenge for massively parallel algorithms is communication and storage. There is a tradeoff between the two. For example if the entire volume is stored on each processor, then there need not be any communication (but obviously an unacceptable amount of memory required). We assume that the unknown optimizations are those that involve reducing communication, and that permutation warping involves a significant advancement in communication efficiencies. For each coherency technique, we can hope to at best have communication not add any run time.

The timing numbers in the performance studies are derived from multipe runs of the program. On the MasPar, there is a hardware register available that captures the number of clock cycles,

and this can be read to compute the run time. The MasPar MP-2 at Santa Cruz has the following capabilities:

output from mpconfig:

```
MasPar DPU Model MP-2204 (64 rows, 64 columns)
ACU IMEM size: 4 MBytes
ACU CMEM size: 512 KBytes
PE memory size: 64 KBytes
IOCTLR in I/O slot 8 (8 MBytes)
IORAM unit #2: 128 MBytes in I/O slot 2
SSD: unit #0: 96 MBytes
```

output from mpi:

```
Version 4.00
MACHINE    TYPE      PE         PMEM    CMEM MODEL  UCODE    QUEUE RELEASE
ganesha    alpha-V3. 64x64     65536  475136 2204   0.19.228     1
jb-239-21
```

Also, necessary for our performance studies have been the use of image verification. Certain volume coherency adaptations have been found to affect the resulting image quality so comparison of computed images has been used to verify proper functioning, and identify artifacts of various schemes.

4 Implementation and Results

These are the algorithm variants that we have used to evaluate coherency acceleration on the MasPar (variant 0 is the baseline algorithm as described in [1]):

1. Octree compression of sub-volume with no attempt at load balancing.
2. Octree compression with dynamic load balancing.
3. Octree compression with static load balancing.
4. Pre-compositing before communication.

The performance of permutation warping can be improved by using compression techniques like octtree or run length encoding. A space partitioning octtree scheme has been implemented for an attempt to improve performance of permutation warping. The linear octtree[30] is a data structure that can be traversed linearly and spatial location can be computed quickly by simple computations. The minimal storage requirements for a linear octree node is the size of voxel plus the size of an octree code. We use 1 byte for voxel data and 32 bits for an octree code. One octree coding level require 4 bits including a padding character[30]. Therefore our data structure can handle a sub-volume of size 2^8 in each direction. For performance, we have an additional data structure that stores the origin of each octree region and its size in each direction of 3 dimensional space. The octree creationis done by checking if 8 consecutive codes with same parent have the same voxel intensities after sorting the octtree coded volume.

The octtree is constructed for the sub-volume of each PE with a specified threshold. If a node in the octtree is less than or equal to the threshold, it will be considered an empty node which does not contribute to the final rendering and these nodes are not stored in the local octtree.

If each PE has a node to be sampled, the sample point in the middle of the sub-volume for that node is computed. Then the sample point is transformed according to given viewing transformation and for that location the data value and corresponding intensity are computed. The resulting intensity is stored in the local array to be sent to the corresponding location in screen space processors which is done after the resampling.

Dynamic load balancing is done following way. If a PE doesn't have any more nodes to process due to its coherency while other PEs are still in the process of sampling octree nodes, it finds the neighbor with the most octree nodes and transfers N nodes. If a PE doesn't have any neighbor with more than N nodes to process, it will transfer a smaller number. This simple algorithm is not efficient when most PE's do not have many nodes while a few PEs have large numbers of nodes to process. It is unclear if

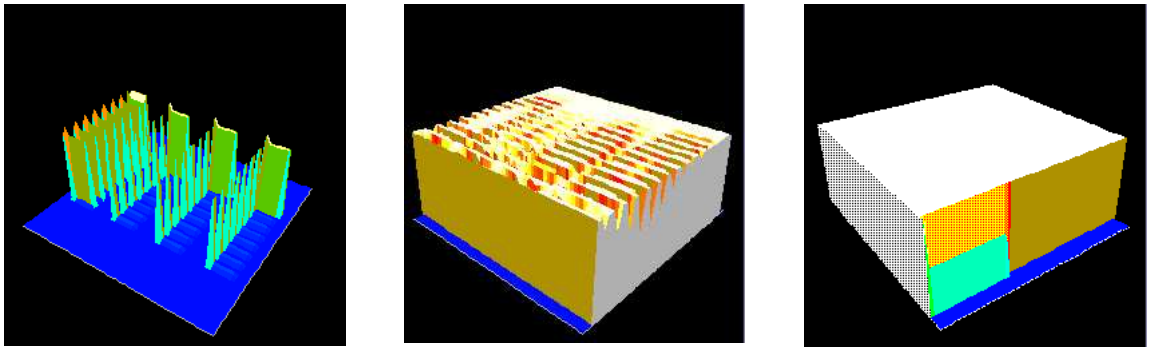

Figure 4.1: Distributions of number of octree nodes. Left: box64 before load balancing, Middle: brain64 before load balancing, Right: box64 after load balancing.

there exists an optimal dynamic load balancing algorithm on a SIMD architecture that will improve performance. We have to balance the time to communicate nodes with the time to interpolate a sample point. We haven't come up with an algorithm that prevents collisions in the dynamic load balancing on SIMD Architecture and improves performances of the permutation warping algorithm, though we have found marked improvement with static load balancing. If we spread all nodes evenly over all PEs dynamically, it greatly increases communication cost at run time. Figure 4 shows the distribution of the linear octree nodes over the 64 by 64 processor elements. The irregularity of the distribution is shown because some processor's sub-volumes are highly homogeneous or full of empty voxels while others are highly noisy and thus hardly compressed. The irregularities of the octree node numbers are highly dependent on the volume data content. We implemented the simple dynamic load balancing but the communication time overwhelmed the performance savings through compression and therefore implemented a static load balancing scheme. The static load balancing process should be done only once for each data set in the initialization process and when user changes volume classification parameters.

After the linear octree compression, we simply iterate a process that finds a processor with the minimum number of nodes and maximum number of nodes and moves the load properly. If the load difference is greater than the given slack number s, we move s nodes. Otherwise we move the difference between the maximum number of nodes and the target number of nodes for each processor which is simply the total number of nodes divided by the total number of processors available. The figure 5 shows the number of octree nodes before and after the load balancing. *Box64* data has 64^3 voxels of box shape data and *brain64* data has 64^3 data of an MR scanned volume. The rightmost figure shows the load balanced number of nodes for all processing elements. It still has some processors with low number of nodes but the balancing is optimal because the maximum among all processors can not be lowered further and only the maximum number counts on SIMD architecture.

Table 4.1 shows the performance of the variants of the octree optimized versions of permutation warping algorithms. The timing and octree numbers are for box64 data which is composed of 262144 voxels. The threshold is used to determine whether the given voxel is empty or not. The Total run time is an approximated wall clock time using the MASPAR hardware clock cycle. The "rotation time" includes the time for viewing transformation and re-sampling times. The version 0 is the base permutation warping algorithm without any coherency optimizations. It takes about 30 to 60 seconds for reading data, building octree and load balancing depending on the number of nodes compressed. The re-sampling was done using trilinear filter.

Version 0 is the baseline algorithm as described in [1]. Version 1 was run without any compression and load balancing. The run time is slightly increased because of the overhead in processing

Version	Number of Nodes	Max. Nodes per PE	Conden-sation	Removing Empty Voxels	Load Balan-cing	Avg. Rotation Time	Avg. Compositing Time	Tot. Run Time
0	262144	-(64)	-	-	-	0.12918	0.0103816	0.139563
1	262144	64	No	No	No	0.188704	0.0103834	0.199088
2	6166	2	1	Yes	Yes	0.0334324	0.0103829	0.0438151
3	29791	8	No	Yes	Yes	0.0331754	0.0103823	0.0435575
4	3765	48	Full	Yes	No	0.120899	0.0103808	0.131279
5	3765	1	Full	Yes	Yes	0.0172912	0.0103823	0.0276739

Table 4.1: Run times for 64^3 box64 data.

Version	Number of Nodes	Max. Nodes per PE	Conden-sation	Removing Empty Voxels	Load Balan-cing	Avg. Rotation Time	Avg. Compositing Time	Tot. Run Time
0	262144	-(64)	-	-	-	0.129179	0.0102961	0.139476
1	262144	64	No	No	No	0.188706	0.0102887	0.198993
3	28360	8	No	Yes	Yes	0.0321614	0.01029	0.0424513
5	28860	8	Full	Yes	Yes	0.0318081	0.0102895	0.0420975

Table 4.2: Run times for 64^3 brain64 data.

non-compressed linear octree data structures. Version 2 shows the timing for the octree compression with 1 level compressed, which means that the octree is compressed up to size of 2^3 voxels, and load balanced. Note that the number of nodes are greatly decreased because the empty voxels are also removed. The version 2 showed about 218 % faster speed. The version 3 has an octree that is not compressed at all but the empty voxels are removed and the load is balanced. It showed about 220 % improvement which is similar to version 2. This is due to the SIMD architecture of MASPAR. While the number of nodes to be re-sampled are much less in the version 2 than version 3, the communication time to send the sampled value to the output locations is increased if the node is a compressed large region of homogeneous voxels. While one processor is processing this large node, many processors with smaller nodes may be idle. The worst case is that communication time increases up to n^3 times where n is the size of sub-volume in each dimension for each processing element. In order to solve this problem, one might want to build an analytical model of communication time and re-sampling (or interpolation time). Using this model, one might want to device optimal load balancing algorithm. However, the exact modeling of communication is not possible because it depends on the viewing rotation. A partial and approximate solution to this problem would be to sort the array of nodes in decreasing size of its region for each processor so that the idling time is decreased.

Version 4 shows very little improvement even with full compression which shows the run time is determined by the processing element with the maximum number of nodes and thus we need the load balancing. Version 5 shows slightly more than 400 % faster performance with full compression and load balancing. Note that we might need to communicate some voxels' re-sampled values that are not communicated in the base algorithm and vice versa, which depends on the viewing rotation, volume data content, and load balancing. Also note that we have additional performance gains if the re-sampling points happens to be within the given octree node which is homogeneous and thus we do not need to interpolate. However, the probability that all 4096 processors' re-sampling points in MasPar are in homogeneous regions is relatively slim and unpredictable. It reveals another point with which we might want exploit with MIMD machines. The re-sample points are guaranteed to be within 1 voxel distance from the given octree node. Therefore we need to store voxels surrounding the octree.

Table 4.2 shows run times of versions of algorithm with *brain64* data which is 64^3 pre-shaded volume of MR brain scan data. It is down-sampled from pre-shaded $256 \times 256 \times 167$ original volume. The fact that we have down-sampled and pre-shaded volume makes the data highly noisy and thus we don't have much compression. Timing numbers for Brain64 data show similar speed up which is about 230 % faster than base permutation warping algorithm. It has been reported that 90 % of most volumes can be considered empty without sacrificing the resulting raycasted image quality [19]. From the numbers in the Table 4.1 and Table 4.2, we can conclude that we will get about 200 % to 400 % performance gains without losing image quality. With more compression with the linear octree, the performance gain is more. However, we might lose some filtering quality when we re-sample only one point for the octree node of large size region and the point is not within the octree node region. Figure 4.2 shows the pictures of box64 and brain64 data that are rendered with non-compressed and compressed volume data. Using better filtering and anti-aliasing in volume re-sampling should solve this aliasing problems. The left image of each is made with non-compressed data and the right image is with the compressed data. The box64 has some aliasing but those of brain64 data is hardly noticeable because the volume is pre-shaded and down-sampled and thus highly noisy. We haven't implemented shading and the popular lookup table based shading will be implemented in the future. As the cost of re-sampling goes high with shading and higher order filter, the performance gain should converges to 900 % (or 10 times speedup) without losing image quality.

As the size of the volume increases, the memory requirements for octree storage pushes the memory limit of MasPar which is 64 K per SIMD processor on our MP-2. The possible solution is that we can set the maximum storage for the additional voxels to store for re-sampling with an octtree and each node's surrounding voxels occupies part of that storage. This requires a slight increase in arithmetic calculations to store and retrieve voxels to interpolate data. For performance enhancements, it is important to devise an efficient retrieval algorithm that calculates the surrounding voxels for a node depending on the sampling location and node size. Schemes of overlapped octrees subnodes may also be tried. The memory required will still be a problem for worst case scenarios which means the optimal load balancing algorithm happens to put a combination of nodes whose total size of surrounding voxels exceeds the physical memory. In this case, we should either use a lower quality filter for re-sampling or sacrifice the performance through less optimal load balancing and communication. To save storage, we can also use the decoding calculations [30] without storing coordinate information for the octree region.

A simple extension of the interpolation (re-sampling) routine, that fetches all surrounding voxels when necessary, will solve the memory limitations. Let c be the average time required to fetch a voxel from another processor and e be the maximum number of re-sampling points that are not within the given octree node per physical processor. For example, the non-compressed box64 without empty voxels needs a maximum of 8 re-sampling points per processor. Therefore e will be 8 in worst case. The maximum number of voxels to be fetched for each re-sampling point is 7 out of 8 because the point is guaranteed to be within 1 voxel distance from the octree region. In most cases when the re-sampling points are located near the faces of octree region, e is 4. Therefore the speed will be slowed by $4 \times e \times c$ in average case. We haven't implemented this simple extension and one of our current research topics is to devise an efficient data structure for storing and retrieving surrounding voxels for load balanced octree nodes in order to achieve optimal performance.

We require 1 communication for each virtual processor in worst case. Let n^3 be the number of virtual processors for each physical processor. Then we need n^3 communications for each physical processor in the worst case on a SIMD architecture. We also implemented a version of program to reduce communication by pre-compositing local sub-volume in each physical processor and sending $O(n^2)$ voxels for each physical processor. Figure 4.3 depicts the boundaries of object space processors and screen space processors where we have 3 regions (in 2D for convenience) to be sent to other virtual screen space processors and 1 region that shares the same physical processor for object and screen space. For each piece of a sub-volume, including the shared region, voxels can be pre-composited before sending to other physical screen space processors or different locations of the same physical processor. Let m be the maximum number of pieces to be pre-composited locally and s^2 be the size of the composited image (in 3D space). In the worst case, s equals to n. Note that this approach increases the local compositing time up to m times in worst case although compositing

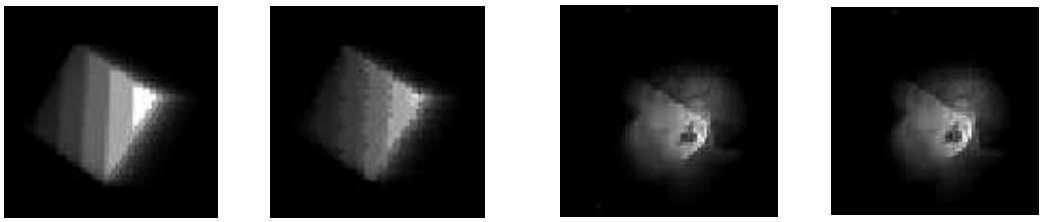

Figure 4.2: Rendered images of box64 and brain64 data.

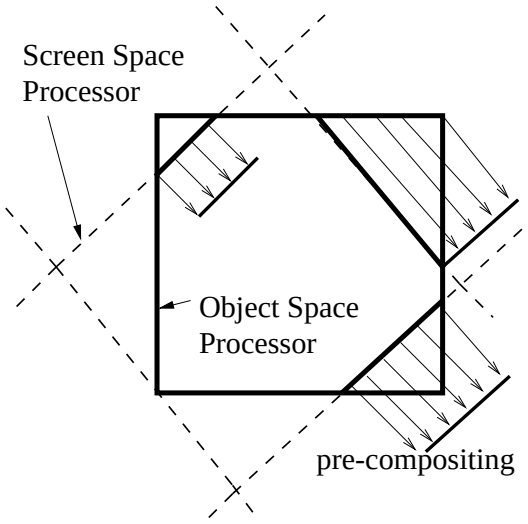

Figure 4.3: Pre-compositing voxel data for virtual processors.

time is usually 10 % of total running time. Therefore, the communication reduction is $n^3 - m \times n^2$ in worst case. On SIMD machine, processors with the worst case dominate the performance. Therefore n should be greater than m to get performance gains through pre-compositing. Let N^3 be the size of the entire volume and P be the total number of the available processors. Then n is $\frac{N}{P^{\frac{1}{3}}}$. From our experiment, n should be 128 or greater. The performance gains will be further exploited and increased when our program is able to handle larger size volumes and the algorithm is implemented on MIMD architectures.

5 Conclusion and Future Research

We have shown that performance of our permutation warping algorithm has been improved 220 % without loss of image quality and 400 % or speedup of 5 with minimal loss of image quality. The quality effect can be reduced by better filtering and anti-aliasing. This report also addressed several questions that direct our future research. One of our current research focuses is to devise

an efficient data structure and algorithm that can store and retrieve the surrounding voxels of load balanced octree nodes within the given hardware's memory limitation. As we further exploit issues in communication reduction though pre-compositing, using higher order filter, anti-aliasing and lookup table based shading, the performance gains are expected to converge to 900 % or speed up of 10 in the permutation warping implementation given the assumption of 90 % volume compressability. We also addressed issues that suggested further speed up in the implementation of our approach on MIMD architectures.

6 Acknowledgements

We would like to thank the support of the ISCR LLNL funding number B291836 of our project, our LLNL collaborators Karin Hollerbach and Nelson Max, and the involvement of students who worked on this project including Jeremy Story and Andrew Macginitie. A special thanks goes to Professors Patrick Mantey, Jane Wilhelms, and Allen Van Gelder for providing feedback, infrastructure, and support for our research.

References

[1] Craig M. Wittenbrink and A. K. Somani. Permutation warping for data parallel volume rendering. *Journal of Parallel and Distributed Computing*, 1995. submitted.

[2] Craig M. Wittenbrink and Arun K. Somani. Permutation warping for data parallel volume rendering. In *Proceedings of the Parallel Rendering Symposium*, pages 57–60, color plate p. 110, San Jose, CA, October 1993.

[3] Craig M. Wittenbrink and Michael Harrington. A scalable MIMD volume rendering algorithm. In *Proceedings IEEE 8th International Parallel Processing Symposium*, pages 916–920, Cancun, Mexico, April 1994.

[4] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458, Orlando, FL, July 1994.

[5] Philippe Lacroute. Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 15–22, Atlanta, GA, Oct 1995. IEEE. Use for a conference paper.

[6] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 7–14, Atlanta, GA, Oct 1995. IEEE. Use for a conference paper.

[7] Jaswinder Pal Singh, Anoop Gupta, and Marc Levoy. Parallel visualization algorithms: Performance and architectural implications. *IEEE Computer*, 27(7):45–55, July 1994.

[8] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings 1994 Symposium on Volume Visualization*, pages 91–98, Washington, D.C., Oct 1994. IEEE/ACM. Use for a conference paper.

[9] Donald Meagher. Fourth-generation computer graphics hardware using octrees. In *NCGA*, pages 316–325, Chicago, IL, Apr 1991. Nat. Comput. Graphics Assoc. Use for a conference paper.

[10] H. Pfister and Arie Kaufman. Real-time architecture for high resolution volume visualization. In *Proceedings of 8th Eurographics Workshop on Graphics Hardware Proceedings*, pages 72–80, Barcelona, Spain, September 1993.

[11] Craig M. Wittenbrink, Kwansik Kim, Jeremy Story, Alex T. Pang, Karin Hollerbach, and Nelson Max. A system for remote parallel and distributed volume visualization. In *accepted to the IS&T/SPIE Symposium on Electronic Imaging: Science and Technology*, San Jose, CA, January 1997. SPIE.

[12] Jay Torborg and James T. Kajiya. Talisman: Commodity realtime 3d graphics for the pc. In *Proceedings of SIGGRAPH*, pages 353–363, New Orleans, LA, August 1996. ACM.

[13] Craig M. Wittenbrink and A. K. Somani. Cache tiling for high performance morphological image processing. *Machine Vision and Applications*, 7(1):12–22, Winter 1993.

[14] Craig M. Wittenbrink and A. K. Somani. 2D and 3D optimal parallel image warping. *Journal of Parallel and Distributed Computing*, 25(2):197–208, March 1995.

[15] Robert M. Haralick, Arun K. Somani, Craig M. Wittenbrink, et al. Proteus: a reconfigurable computational network for computer vision. *Machine Vision and Applications*, 8(2):85–100, 1995.

[16] Jim Blinn. Light reflection functions for simulations of clouds and dusty surfaces. In *Computer Graphics*, pages 21–29, July 1982.

[17] J. T. Kajiya and B. Von Herzen. Ray tracing volume densities. In *Proceedings of SIGGRAPH*, pages 165–174, July 1984.

[18] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(5):29–37, May 1988.

[19] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[20] Arie Kaufman, editor. *Volume Visualization*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[21] J. Foley, A. vanDam, S.K. Feiner, and J.F. Hughes. *Computer Graphics Principles and Practice*. Addison Wesley Inc., Reading, MA, second edition, 1990.

[22] J. Nieh and M. Levoy. Volume rendering on scalable shared-memory mimd architectures. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 17–24, October 1992.

[23] T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, T. Cullip, J. Rhoades, and R. Whitaker. Achieving direct volume visualization with interactive semantic region selection. In *Proceedings IEEE Visualization '91*, pages 58–65, San Diego, CA, October 1991.

[24] C. Montani and R. Scopigno. Rendering volumetric data using the sticks representation scheme. In *Computer Graphics, San Diego Workshop on Volume Visualization*, pages 87–93, San Diego, CA, November 1990.

[25] Ulrich Neumann. Parallel volume rendering algorithm performance on mesh connected multicomputers. In *Proceedings on the Parallel Rendering Symposium*, pages 97–104, San Jose, CA, October 1993.

[26] Vineet Goel and Amar Mukherjee. An optimal parallel algorithm for volume ray casting. In *9th International Parallel Processing Symposium*, page to appear, Santa Barbara, CA, April 1995.

[27] W. H. Hsu. Segmented ray casting for data parallel volume rendering. In *Proceedings of the Parallel Rendering Symposium*, pages 7–14, San Jose, CA, October 1993.

[28] Craig M. Wittenbrink. *Designing Optimal Parallel Volume Rendering Algorithms*. PhD thesis, University of Washington, 1993.

[29] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[30] I. Gargantini. Linear octtrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20(4):365–374, December 1982.