Data Dependent Optimizations for Permutation Volume Rendering

Craig M. Wittenbrink, Kwansik Kim*
Computer Systems Laboratory
HPL-97-59 (R.1)
January, 1998

E-Mail: craig@hpl.hp.com
ksk@cse.ucse.edu

parallel volume visualization, SIMD, algorithms, octtree, ray tracing

We have developed a highly efficient, high fidelity approach for parallel volume rendering that is called permutation warping. Permutation warping may use any one pass filter kernel, an example of which is trilinear reconstruction, an advantage over the shear warp approach. This work discusses experiments in improving permutation warping using data dependent optimizations to make it more competitive in speed with the shear warp algorithm. We use a linear octree on each processor for collapsing homogeneous regions and eliminating empty space. Static load balancing is also used to redistribute nodes from a processor's octree to achieve higher efficiencies.

In studies on a 16384 processor MasPar MP-2, we have measured improvements of 3 to 5 times over our previous results. Run times are 73 milliseconds, 29 Mvoxels/second, or 14 frames/second for 128^3 volumes, the fastest MasPar volume rendering numbers in the literature. Run times are 427 milliseconds, 39 Mvoxels/second, or 2 frames/second for 256^3 volumes. The performance numbers show that coherency adaptations are effective for permutation warping. Because permutation warping has good scalability characteristics, it proves to be a superior approach for massively parallel computers when image fidelity is a required feature. We have provided further evidence for the utility of permutation warping as a scalable, high fidelity, and high performance approach to parallel volume visualization.

Internal Accession Date Only

*University of California, Santa Cruz, California
To be published in the *Proceedings of the SPIE, The International Society for Optical Engineering, Visual Data Exploration and Analysis V,* Conference 3298, January 1998.

Data dependent optimizations for permutation volume rendering

Craig M. Wittenbrink[a] and Kwansik Kim[b]

[a]Hewlett-Packard Laboratories, Palo Alto, CA

[b]Computer Science Department, University of California, Santa Cruz, CA

ABSTRACT

We have developed a highly efficient, high fidelity approach for parallel volume rendering that is called permutation warping. Permutation warping may use any one pass filter kernel, an example of which is trilinear reconstruction, an advantage over the shear warp approach. This work discusses experiments in improving permutation warping using data dependent optimizations to make it more competitive in speed with the shear warp algorithm. We use a linear octree on each processor for collapsing homogeneous regions and eliminating empty space. Static load balancing is also used to redistribute nodes from a processor's octree to achieve higher efficiencies.

In studies on a 16384 processor MasPar MP-2, we have measured improvements of 3 to 5 times over our previous results. Run times are 73 milliseconds, 29 Mvoxels/second, or 14 frames/second for 128^3 volumes, the fastest MasPar volume rendering numbers in the literature. Run times are 427 milliseconds, 39 Mvoxels/second, or 2 frames/second for 256^3 volumes. The performance numbers show that coherency adaptations are effective for permutation warping. Because permutation warping has good scalability characteristics, it proves to be a superior approach for massively parallel computers when image fidelity is a required feature. We have provided further evidence for the utility of permutation warping as a scalable, high fidelity, and high performance approach to parallel volume visualization.

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

1. INTRODUCTION

Volume rendering[1] algorithms calculate visualizations from sampled medical and simulation data, and researchers have sought to speed up the algorithms to make them more useful. 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 the best approach differs depending on metric, state of the art, and price point. We have done work on general parallel machines called permutation warping.[2,3] Permutation warping has been proven to be scalable. In fact, it has been shown to have superior scalability to all published results of algorithms implemented on the MasPar MP-1. See Figure 2, Wz and Wt at left of plot. Our zero order hold algorithm (Wz) also turns out to have the absolute highest performance, as shown in Figure 1, over 10 MVoxels/second on a MasPar MP-1. We present in this paper, new results that are nearly 3 to 5 times faster than our prior results on the MasPar. But demonstrate here, how our previous results compare to other researchers.

The zero-order hold (Wz), simply reads the voxel value needed through the general interconnection network, which requires general communication, and results in supralinear scalability, Figure 2, and is also fast, Figure 1. The closest competitor in speed is Hsu's algorithm (Hz),[4] followed by Vezina et al. (Vz),[5] Goel et al.(G*),[6] Vezina et al.'s first order hold (Vf), and then permutation warping trilinear interpolation (Wt). And, even though the zero order hold (Wz) provides the highest absolute performance, the trilinear permutation warping (Wt) shows linear scalability, and uses superior filtering to the other algorithms (Goel et al. use a trilinear filter as well, but do not have good scalability.) For example, on the MasPar MP-1, we have shown a speedup of 15.7 for a 16 k processor MP-1

Other author information: Partially supported by a grant from ISCR-LLNL B291836 and NSF IRI-9423881 (E-mail: craig@hpl.hp.com ksk@cse.ucsc.edu)

*We scaled Goel et al.'s results on a 8192 processor machine which used 4096 PE's by 4X for the performance plotted.

over a 1k processor MP-1 (16 is ideal), 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.[3] 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.

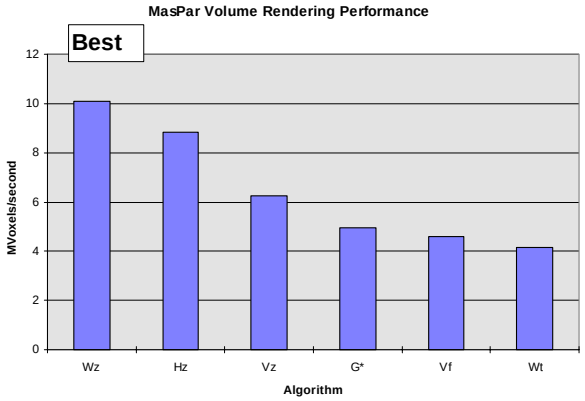

Figure 1. Volume rendering algorithms performance on a 16k MasPar MP-1, rendering a 128^3 volume.

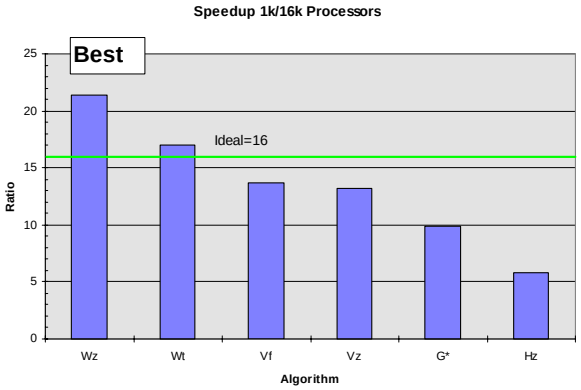

Figure 2. Volume rendering algorithms speedup, comparing performance of runtimes on a 1024 processor MasPar MP-1 to a 16384 processor machine.

Recently, the sequential and parallel MIMD shear warp algorithms of Lacroute and Levoy[7,8] have taken advantage of data dependent optimizations to achieve the highest performance published on general purpose parallel machines. The algorithm exploits runlength coding, image and object space bookkeeping, and bilinear filtering. But, their studies show that the speedup to higher numbers of processors is limited because of the screen space decomposition. Our algorithm research investigates extensions to permutation warping to make it more competitive in speed, while supporting its superior filtering qualities. Permutation warping is highly scalable as we demonstrated on the SIMD implementations[2,3] (Figure 2), and it is also possible to use data dependent encoding and compression to further improve efficiencies of our algorithm. The focus of this paper are extensions and implementation results on the MasPar MP-2. The ability to do load balancing and coherency adaptation are possible with scalable, efficient massively parallel MIMD and SIMD algorithms. We demonstrate the application of linear octrees[9,10] to encode volumetric data, and also demonstrate static load balancing techniques that provide a 3 to 5 times speedup over our previous permutation warping run times. This result is achieved primarily through the skipping of empty voxels, which are quite prevalent in datasets such as medical MRI and CT studies, but performance will vary with dataset and classification. We investigated alternatives using dynamic and static load balancing, compression, precompositing, and adaptive sampling. We demonstrate through our performance timings, that it is possible to greatly speedup permutation warping using data dependent optimizations; that on the MasPar this is most efficiently done through a preprocessing to create a statically balanced load; and, the filter quality can be preserved with efficient massively parallel algorithms. On a 16,384 processor MasPar MP-2, we can achieve 14 frames/second for a 128^3 volume or 2 frames/second for a 256^3 volume. Scalability of a 16K MP-2 over a 4K MP-2 shows near linear scalability 3.6 (4 is linear). Figure 3 shows the performance of our new octree encoded volume rendering algorithm Wittenbrink and Kim zero-order hold octree (WzO) and Wittenbrink and Kim trilinear octree (WtO). Performance is given for a 16,384 processor MP-2, rendering a 256^3 MRI brain dataset. The zero-order hold octree version (WzO) achieves 39.3 Mvoxels/second, the trilinear algorithm (WtO) achieves 36.8 Mvoxels/second, which is 2 times the performance of our previously published permutation warping algorithm (Wz) which achieves 14.2 Mvoxels/second (zero-order hold). Results are also shown for the 4K MP-2, where we render a 128^3 dataset for closer comparison, and we showed results superior to Hsu[4] (Hz) who also provides MP-2 results. A current technology SIMD architecture would be able to provide 30-60 frames/second with larger volumes, assuming the VLSI processors and interconnects could be updated. The MasPar MP-2 was produced in 1992, 5 years ago. We give a short background on permutation warping, Sect. 2, discuss octree encoding in Sect. 3, and then present our research results, Sect. 4.

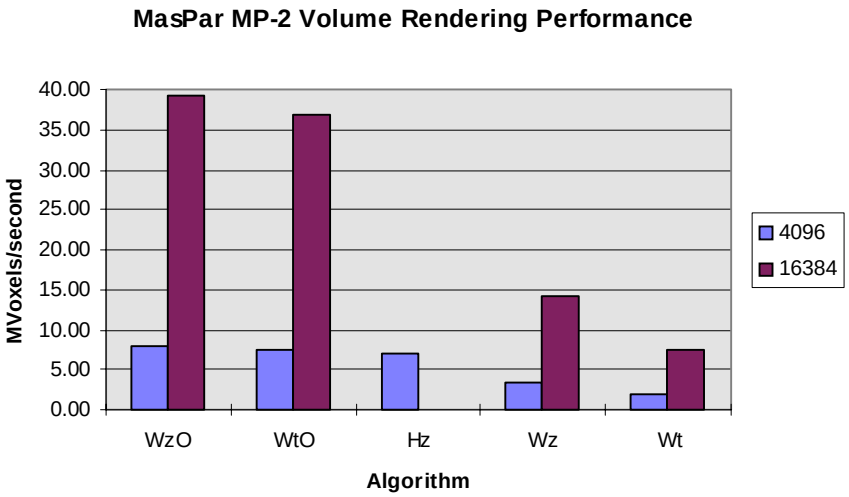

Figure 3. Octree encoded permutation warping performance (WzO and WtO) in Mvoxels/second versus the baseline permutation warping algorithm (Wz and Wt) and Hsu (Hz).

2. BACKGROUND ON PERMUTATION WARPING

Permutation warping[2,3] uses a one-to-one pairing for processor to processor communication to efficiently resample a data volume in the view transform. A permutation communication pattern can be optimally embedded into a variety of interconnection networks used in massively parallel computers, such as the hypercube. Volume rendering is an algorithm that computes the interaction of light in a volume of light scattering particles. 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.[1] 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.

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. Existing parallel algorithms may be grouped into two categories determined by their viewing transforms: backwards[6,4] and multipass forwards.[7,8,5,1] Our permutation warping[2,3] approach computes a backwards mapping algorithm. Figure 4 shows pseudo-code for the permutation warping algorithm.

```
1.0) PPS, preprocessing stage, classify, compute opacity, shading
2.0) VWS, volume warping stage, processors in parallel:
      2.1) Calculate processor assignments, pick screen space processor
      2.2) Calculate reconstruction point, inverse view transform
      2.3) Perform resampling of opacity and intensity
      2.4) Send resampled opacity and intensity to screen space processors
3.0) CS, compositing stage, combine ray intensities and opacities with
     parallel product
```

Figure 4. Permutation Warping Algorithm Pseudo-code

Figure 5 illustrates an example transform calculated by processors. The 3D 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. In Step 3, a parallel binary tree combining computes ray compositing.

The permutation is calculated using an isomorphic operator, M, whose form is covered in Wittenbrink and Somani.[3] Any neighborhood reconstruction filter may be used such as trilinear, and cubic. The pattern given by Fig. 5 is only one example, because for any equiareal transform, the mapping M is provably one-to-one. Virtualization is performed by tiling the volume address, and dividing the volume in all three orthogonal dimensions. Each processor gets a subvolume of data to work on, and run time is nearly constant across view angle because of this virtualization.[3]

3. LINEAR OCTREE PERMUTATION WARPING

We have applied *linear octrees*[9,10] to compress and encode the volume data. The linear octree is a data structure that can be traversed linearly and spatial locations can be computed by simple computations. Another advantage of the linear octree is that it requires a small amount of storage. For each octree node, we store the octant code for the spatial location and size in terms of the sub-volume, and we store the processor number for load balancing.

Each subvolume of the total volume is encoded into an octree. Figure 10 shows a simple example, where there are four processors, and therefore four octrees (quadtrees) created. The tree structures to represent the necessary data are shown. The octree is constructed for the sub-volume of each PE with a specified threshold, for data to be considered empty. If a node in the octree 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 octree. The use of an octree within each processor allows for compression of the source data. We also use the octree to control adaptively, the resampling process. Processor 0 has 12 bottom level (non-condensed) nodes and one 1 level condensed node which means the region has 4 voxels (pixels) that have the same intensity. Condensation is the method used to build octrees as all nodes are encoded, and then nodes that have all of the 8 octants for a higher level filled are condensed to one node. Processor 0 would have the following list of nodes 00_{green}, 01_{yellow}, 02_{blue}, 03_{pink}, 10_{yellow}, 11_{green}, 12_{blue}, 13_{pink}, 20_{blue}, 21_{pink}, 22_{green}, 23_{yellow}, and $3X_{\text{lime}}$. The $3X_{\text{lime}}$ node represents the condensation of 4 other nodes that will have the same value, lime. Processor 1 and 2 do not have any nodes because all voxels are below the given threshold and compressed down to one node and removed. Processor 3 has one homogeneous region and thus is compressed down to 1 node, XX_{lime}. Each PE has its own sub-volume that is one quarter of the entire volume (image). Therefore, some PEs will have highly homogeneous sub-volumes while others will have highly heterogeneous sub-volumes.

Octrees nodes are shuffled between processors, to adjust their work loads. The adaptation is done by preprocessing. The total number of nodes for all PEs and the target number of nodes per PE is first determined. We simply find PEs with maximum and minimum loads. We calculate a target load per PE, and balance processor's work. Figure 11 shows the load balancing scheme. PE 0 has the maximum number of nodes, 13, and PE1, has 0 nodes. We attempt to achieve an average $\lceil 14/4 \rceil$ nodes or 4 nodes per PE, and nodes are sent to neighbors. The result after load balancing is processors PE0, PE1, and PE2 have 4 nodes, and PE3 has 2 nodes. Such an encoding is used for permutation warping as shown in the following pseudo code, Fig. 6.

Data are processed to create an octree within each processor's subvolume. The octree nodes are then thresholded and condensed to higher level octree nodes. Empty nodes are eliminated. Load balancing is performed by exchanging nodes and volume data among neighbors. We implemented a simple dynamic load balancing method, but the communication time overwhelmed the performance savings through compression. We, 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. Then nodes are sorted by their level, so as to process similar octree nodes in each processor. The number of virtual processors in screen space changes depending on their level.

Each processor considers octree nodes assigned to them through the load balancing, and maintained on a local list, *octreeList*. The while loop picks octree nodes off of the list, and calculates the point and virtual processor, π, represented by the node. The screen space processor is calculated by the permutation assignment $\pi' = M\pi$. The point location of that virtual processor is calculated $p = T^{-1}\pi'$. An interpolation is performed at that point. For SIMD processing we have found it to be most efficient to keep a copy of the original voxels to simplify the interpolation, though with obvious memory overhead. Then all virtual processor locations within that octree node are selected, the *for* loop; their screen space partner computed, $\pi'_j = M\pi_i$; and the prior interpolated result, *value*,

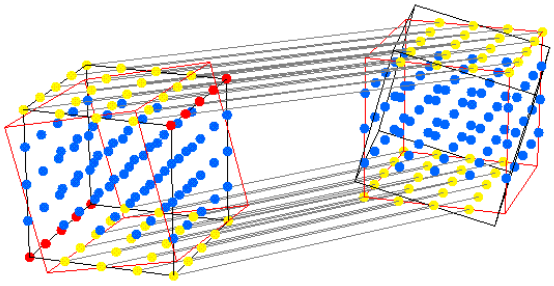

Figure 5. Volume transforms in parallel.

```
create octree in local subvolume
condense and threshold
load balance among near neighbors
sort nodes locally by octree level
currentNode = head(octreeList)
while(currentNode < endOfLocalList)  {
   π = pointRepresentedByNode(currentNode)
   π' = Mπ  // permutation assignment
   p = T^{-1}π'
   value = interpolateOctree(p)
   for all 3D voxels π_i to be affected  {
      π'_j = Mπ_i // permutation assignment
      send(value, π'_j)
   }
}
```

Figure 6. Octree Accelerated Permutation Warping Algorithm Pseudo-code

Table 1. Compared Performance (MVoxels/second) on MP-2 4K and 16K processors.

	Algorithm				
processors	WzO	WtO	Hz	Wz	Wt
4096	8.04	7.45	6.97	3.44	1.92
16384	39.25	36.81	-	14.20	7.48

sent. Following this resampling, the data are aligned along view rays, and alpha compositing is efficiently computed by scan operators.

4. RESULTS

The algorithm in Sect. 2 was implemented in MPL on the MasPar MP-2. Performance timings were taken, and features were evaluated. Various parameters including dataset size, number of processors, and filter quality were investigated. Performance timings were measured using the *dpuTime()* utilities that output a register timer for cycle accurate timings. Multiple runs were taken for a particular view angle, and many view angles were taken. The median run times are given in this paper. We cover the best results of our studies, the quality performance trade-offs , and implementation difficulties. Other detailed results are given in Sect. 5.

Figure 3 and Table 1 show the performance of the algorithm in comparison to related work. Variants are the Wittenbrink and Kim zero-order hold Octree (WzO), Wittenbrink and Kim trilinear Octree (WtO), Hsu zero-order hold (Hz), Wittenbrink et al. zero-order hold baseline (Wz), and Wittenbrink et al. trilinear permutation warping (Wt). The octree accelerated parallel algorithm (WzO) outperforms other published volume renderers on the MasPar.[4,2,6,5] Mvoxels/second is not a good metric to compare data dependent algorithms, but gives a good indication of the improvement over brute force processing of all voxels. In Table 1 results are computed from rendering a 256^3 volume except for Hsu's (Hz)[4] and the 4096 processor octree results (WzO,WtO) which are from rendering a 128^3 volume. We encountered memory limitations due to the creation of a linear octree, and storage of the original volume. This storage was maintained to simplify the SIMD instructions in order to get the highest performance, and only the 16,384 processor machine could process a 256^3 volume with the octree algorithm variants (WzO, WtO).

The performance numbers presented are impressive, but come with some cost. The octree algorithm is accelerated by eliminating or combining nodes. Figure 7 shows the results of five different thresholds used for eliminating empty regions. All voxels below the threshold were considered to be empty, and were eliminated. The thresholds were 0, 1, 5, 10, and 50. Figure 7 shows the histogram equalized errors differencing the baseline algorithm's results from that of the trilinearly interpolating octree code (WtO). A 256^3 volume was used.

Table 2. Quality performance tradeoffs, Runtimes (milliseconds) RMS error (of images as shown in Fig. 7, and percentage of the baseline permutation warping algorithm on a 16K processor MP-2.

threshold	0	1	5	10	50
RMS error	0.868	0.922	2.741	2.968	7.218
run time (milliseconds)	1,867	1,751	804	710	456
percentage of baseline	83%	78%	36%	32%	20%

Table 2 shows the percentage improvements over the baseline permutation warping algorithm as the threshold is set higher and higher. Runtime goes from 83% of the baseline to 20% of the baseline. The root-mean-square (RMS) error is given also in the table to provide magnitudes for the enhanced (histogram equalized) error shown in Fig. 7. Improved performance comes with increased interpolation error.

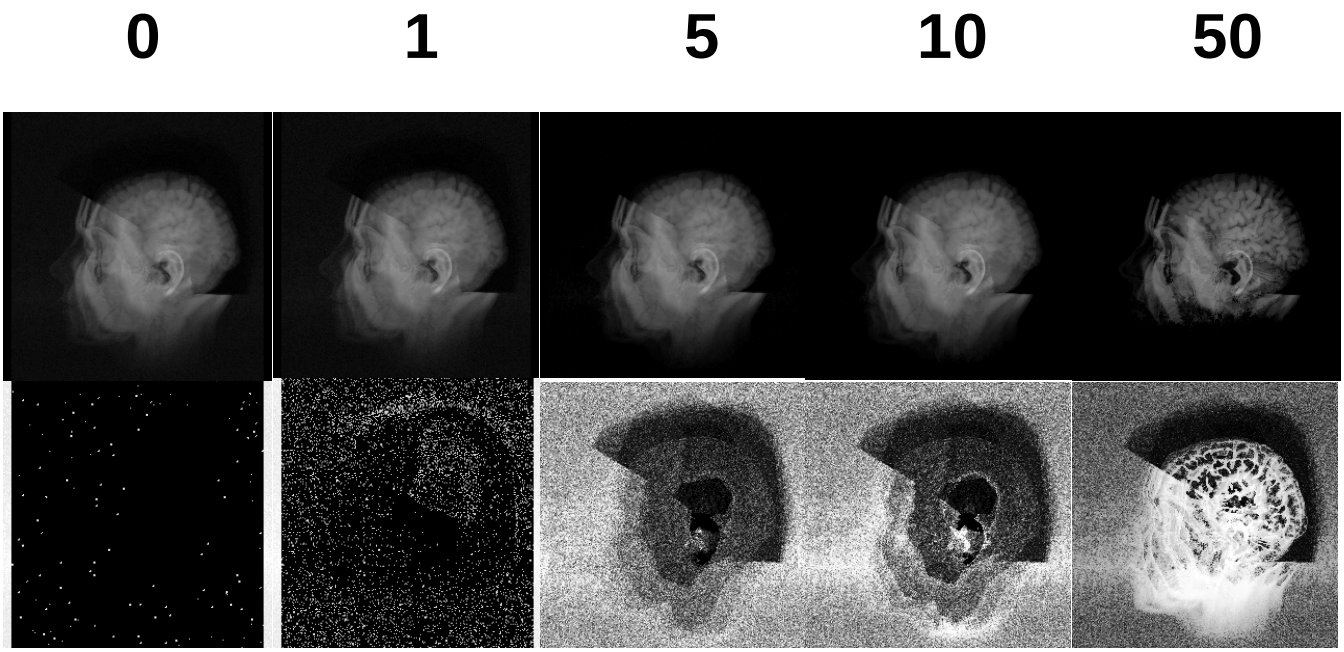

Figure 7. Outputs from octree accelerated algorithm with different thresholds from the left 0, 1, 5, 10, 50.

An interesting result is the scalability. We measured performance timings on a 16,384 processor MasPar and a 4096 processor MasPar for all of the variants. Table 4 and Table 6 provide the speedup calculations, computed from the runtime provided in Table 3 and 5. The speedup for octree accelerated algorithms is 3.6 versus an ideal linear speedup of 4.0, Table 4. The speedup is reduced compared to the baseline's speedup of 3.85, Table 6. And for smaller volumes using high thresholds may do worse, 2.9, because of the lack of virtualization. With datasets of at least 128^3, the scalability remains constant with threshold, and perhaps with larger datasets the scalability will be improved. But, the scalability is high, and shows the effective parallelism of this approach. The baseline algorithm achieves linear scalability with larger volumes, 3.90, and supralinear scalability with a nearest neighbor filter, 4.12, as was shown in Wittenbrink and Somani.[3]

5. OTHER RESULTS

We have analyzed the impact of the following algorithm techniques: non condensed linear octree processing, one level condensed linear octrees, thresholding empty voxels, and load balancing. We collected timings for several variants. We found that skipping empty voxels provided the opportunity for speedup, but effective load balancing, and sorting of octree nodes was necessary to insure a speedup over the base algorithm. Condensation of octree nodes to higher levels was a smaller impact than simply skipping empty space. In detailed examination of the load balancing we

Table 3. Performance (milliseconds) of Octree accelerated algorithm on MP-2 4K and 16K processors

		64^3					128^3					256^3				
proc.	int.	0	1	5	10	50	0	1	5	10	50	0	1	5	10	50
16K	zoh	44	44	21	20	14	301	294	126	116	73	1686	1582	742	658	427
	tril	49	48	22	22	14	335	327	137	126	78	1867	1751	804	710	456
4K	zoh	145	139	66	61	40	1090	1055	461	421	261	-				
	tril	161	154	71	66	42	1214	1175	504	459	282	-				

Table 4. Speedups of octree accelerated algorithm on MP-2 4K/16K

	64^3					128^3				
interpolation	0	1	5	10	50	0	1	5	10	50
zoh	3.30	3.20	3.17	2.98	2.90	3.62	3.59	3.66	3.62	3.58
tril	3.32	3.21	3.22	3.01	2.95	3.63	3.60	3.68	3.65	3.60

found that the time to load balance was a function of the data and machine size. The load balancing took 10,000 to 16,000 iterations for all data sizes on a 16,384 processor MasPar and 2800 to 3600 iterations on a 4096 processor Maspar. The number of linear octree nodes is nearly the same across all processors, and load varied only by 4 nodes typically. Worst inbalances were 17 nodes for a 16K MP-2 and 13 nodes for a 4K MP-2 with the largest dataset and the largest threshold.

For the results presented in Sect. 4 static load balancing was used as demonstrated by the example in Fig. 11. If load balancing is performed during rendering, it is called dynamic load balancing. Dynamic load balancing may be done in many different ways, but to taylor the algorithm to the more efficient near neighbor processing mesh of the MasPar, we evaluated the following approach. If a PE does not 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 reads N nodes to process. If a PE doesn't have any neighbor with more than N nodes to process, it transfers a smaller number. This approach has not demonstrated an improvement, because of overheads in communication, and stalls due to SIMD conditionals. The primary difficulty is the inability to efficiently migrate higher workloads to processors with less work.

If we spread all nodes evenly over all PEs dynamically, it greatly increases communication cost at run time. Figure 13 shows the distribution of the linear octree nodes over the 64 by 64 processor elements on a 4096 processor MasPar. The MasPar is a 2D array architecture. The distribution is irregular because of the noisy and empty volume regions. The octrees depend directly on the volume data.

The static load balancing used for our results was computed in a preprocess. In actual computations on the MasPar, Figure 13 shows the number of octree nodes before and after load balancing with a simplified test dataset. *Box64* data has 64^3 voxels of box shape data. The right figure shows the load balancing. Some processors have fewer nodes, but the balancing is optimal because the maximum among all processors can not be lowered further and only the maximum counts for SIMD runtime. There are overheads to statically load balance. For our implementation the load balancing took nearly 30 seconds on a 128^3 volume and 1.9 minutes on a 256^3 volume. The volume is unchanged for recalculating new viewpoints.

Table 5. Performance of baseline (milliseconds) on MP-2 4K and 16K processors

proc.	int.	64^3	128^3	256^3
16K	zoh	24	153	1182
	tril	41	288	2243
4K	zoh	83	621	4874
	tril	145	1107	8738

Table 6. Speedups of baseline on MP-2 4K/16K

int.	64^3	128^3	256^3
zoh	3.54	4.05	4.12
tril	3.56	3.85	3.90

Table 7. Compositing time's percentage of total rendering time on 16K MP-2.

	threshold	
dataset	0	50
64^3	9.6%	32.3%
126^3	4%	17.1%
256^3	3.7%	15%

The algorithm was also broken up into the resampling and the compositing time. We resample to a complete resolution screen space volume, therefore the compositing time is not dependent on threshold, only volume size. The compositing stage's percentage of the total run time, therefore increases with higher thresholds because the resampling time is reduced. Table 7 shows that the compositing time is a small fraction of the total time, typically 4%, and up to 15% for larger volumes.

Because the memory limit of the MasPar is 64Kbytes per SIMD processor, we were limited in the approaches we could evaluate. Care was taken to encode the octree as efficiently as possible, and our data structure uses 11 bytes per octree node, 7 bytes for the code and 4 bytes for the spatial location and levels. Interpolation was also challenging to implement, as octree nodes require neighboring values for proper interpolation. Such overlapped storage is expensive, but can be replaced with additional communication. Additional storage can be recovered by not encoding the spatial location of nodes, 3 bytes, and reconstructing the location from the octree code.

Each subvolume can be precomposited before communication to the screen space. Figure 8 shows three regions within an object space processor that are composited to images before being sent to the three corresponding screen space processors. We were unable to craft an efficient precompositing method on the SIMD MasPar.

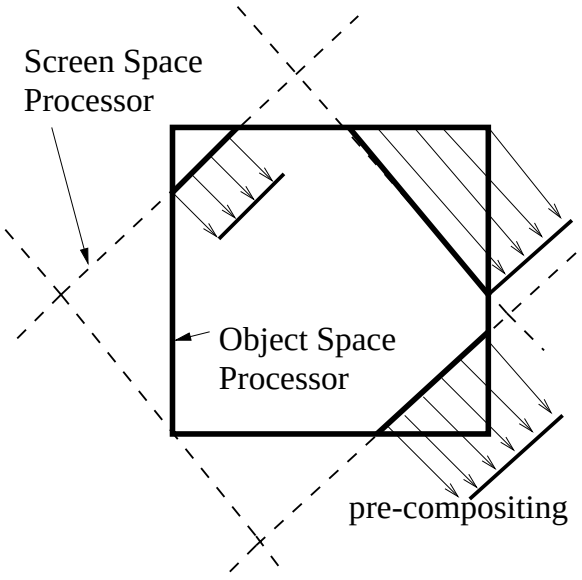

Figure 8. Pre-compositing voxel data for virtual processors.

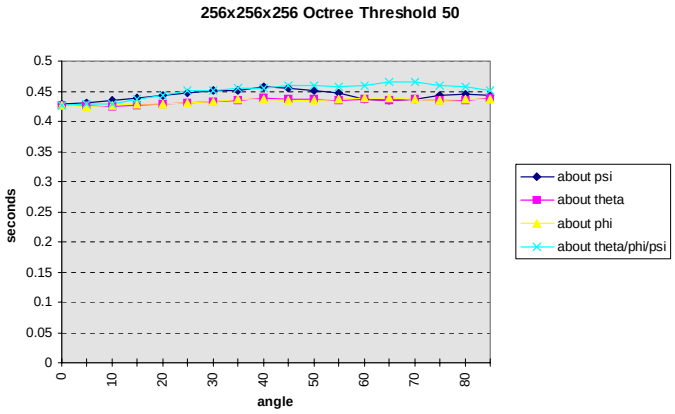

Figure 9. Nearly constant run time for all different viewing angles for the octree accelerated algorithm (WtO).

Permutation warping has constant run time for different view angles.[2,3] Figure 9 shows that the octree algorithm

(WtO), trilinear reconstruction, also has constant runtime over view angle, as different view angles in rotation about the 3 major axes, and about all 3 axes are shown.

A restriction of the MasPar is the slow file access if parallel I/O has not been installed. We had access times for the 128^3 dataset of 39 seconds to read the volume, 15 seconds to build condensed octrees. The 256^3 volume required 148 seconds to read and 4 minutes to condense. While these parts of the program were not optimized, writing compressed images using GIF encoding was examined to support a world wide web client (WWW).[11] The initial compression results were slower than sending the image to an R8000 based SGI for compression, highlighting the relative age of the MasPar and DEC front end.

6. CONCLUSIONS AND FUTURE RESEARCH

We have shown that performance of our permutation warping algorithm has been improved 3 to 5 times using linear octree encodings with static load balancing. The savings result from collapsing of homogeneous regions into octree nodes and elimination of empty space. Many other alternatives were investigated for achieving performance improvements as well. We investigated dynamic load balancing, precompositing, and memory conservation. We were surprised by the large increase in efficiency available on our implementation, but it was not without considerable effort. The SIMD implementation on the MasPar MP-2 is limited by the small amount of memory per processor, 64 K Bytes, and by the SIMD execution stream, where the worst case dominates performance. Schemes that have provided coherency acceleration on single processors or MIMD processors are very difficult to adapt to SIMD machines. For this reason, the dynamic load balancing, and precompositing were difficult to implement and achieve a performance improvement. Precompositing has many conditions which add to the SIMD instruction stream, while dynamic load balancing drove the communication costs higher than the improved efficiences in redistributed work.

The most encouraging result of the research is that coherency adaptations are effective for permutation warping. We have shown this through our implementation on a SIMD and massively parallel computer. It is important to compare our approach to the state of the art, and other existing work. The filter differences between permutation warping and shear warp are important for image fidelity, a feature some users are not willing to give up. A direct comparison of shear warp and permutation warping is not easy, as a trilinear resampling filter takes more work than the shear warp multipass bilinear filtering, hence the improved image quality. Shear warp resampling is inferior to direct resampling, but may be compensated for somewhat by oversampling of the original dataset, and restriction of classification functions allowed to those that do not introduce high frequency features. We have shown that data dependent optimizations are useful for permutation warping. Because permutation warping has good scalability characteristics, it may prove to be a superior approach for massively parallel computers when image fidelity is a required feature. We achieved a performance of 427 milliseconds per frame, 39 Mvoxels/second, or 2 frames/second on the MRI brain 256^3 volume dataset on a 16,384 processor MasPar MP-2. We achieved a performance of 73 milliseconds per frame, 29 Mvoxels/second, or 14 frames/second on the 128^3 dataset. A scalability of 3.6 over the 4,096 processor MP-2 was shown. Constant run time across view angle was shown, and performance quality trade-offs were evaluated.

Future work is necessary to evaluate other algorithm variants, such as MIMD implementations. We are also working on different variations for volume shading and compositing. A fast shading method like lookup table based shading might be difficult to implement because of the memory limitations on MasPar hardware. Improved gradient operators may also be a direct advantage of permutation warping because of the overlapped storage and slice and dice virtualization. Scanline operating algorithms have difficulty operating on large 3D neighborhoods efficiently. The key limitations of the implementation are the use of orthographic projection, a limit of 256^3 sized volumes, simplified shading, preprocessing, and only MasPar implementation. None of these are limitations of the algorithm, and the demonstrated improvements, even on a single architecture, provide evidence for the utility of permutation warping as a scalable, high fidelity, and high performance approach to parallel volume visualization.

7. 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. Thanks to NASA Goddard for providing access to a 16,384 processor MasPar MP-2 with Prof. Alex Pang's NASA affiliation and sponsorship. 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. R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Computer Graphics*, pp. 65–74, Aug. 1988.
2. C. M. Wittenbrink and A. K. Somani, "Permutation warping for data parallel volume rendering," in *Proceedings of the Parallel Rendering Symposium*, pp. 57–60, color plate p. 110, (San Jose, CA), Oct. 1993.
3. C. M. Wittenbrink and A. K. Somani, "Time and space optimal parallel volume rendering using permutation warping," *Journal of Parallel and Distributed Computing* **In Press**, 1997. Available as Technical Report, Univ. of California, Santa Cruz, UCSC-CRL-96-33.
4. W. H. Hsu, "Segmented ray casting for data parallel volume rendering," in *Proceedings of the Parallel Rendering Symposium*, pp. 7–14, (San Jose, CA), Oct. 1993.
5. G. Vezina, P. A. Fletcher, and P. K. Robertson, "Volume rendering on the MasPar MP-1," in *Proceedings of 1992 Workshop on Volume Visualization*, pp. 3–8, Oct. 1992.
6. V. Goel and A. Mukherjee, "An optimal parallel algorithm for volume ray casting," *The Visual Computer* **12**(1), pp. 26–39, 1996. short versions appear in IPPS'95 and Fifth Symp. on Frontiers of Massively Parallel Computation'94.
7. 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, ACM, (Orlando, FL), July 1994.
8. P. Lacroute, "Analysis of a parallel volume rendering system based on the shear-warp factorization," *IEEE Transactions on Visualization and Computer Graphics* **2**, pp. 218–231, Sept. 1996. a short version of this paper appears in the 1995 Parallel Rendering Symposium.
9. I. Gargantini, "Linear octtrees for fast processing of three-dimensional objects," *Computer Graphics and Image Processing* **20**, pp. 365–374, Dec. 1982.
10. J. Foley, A. vanDam, S. Feiner, and J. Hughes, *Computer Graphics Principles and Practice*, Addison Wesley Inc., Reading, MA, second ed., 1990.
11. C. M. Wittenbrink, K. Kim, J. Story, A. T. Pang, K. Hollerbach, and N. Max, "A system for remote parallel and distributed volume visualization," in *In Proceedings of IS&T/SPIE Visual Data Exploration and Analysis IV*, pp. 100–110, SPIE, (San Jose, CA), Feb. 1997. Available as Hewlett-Packard Laboratories Technical Report, HPL-97-34.

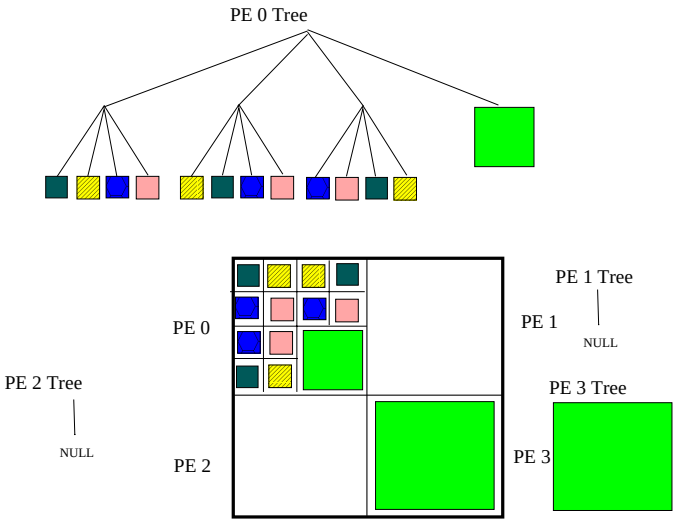

Figure 10. Octree per virtual subvolume (image) in a four processor machine. The 2 dimensional quadtree is shown for convenience. The voxels and corresponding trees are shown as an example of our octree based compression.

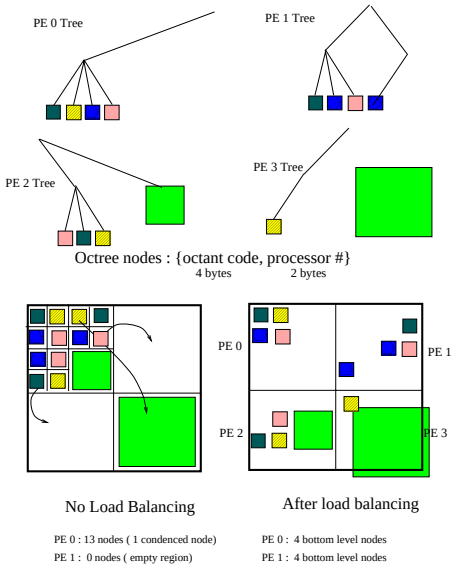

Figure 11. Precomputed static load balancing for adjusting octrees in each processor. The load is balanced in terms of number of nodes per PE and thus the maximum number of interpolation is minimized on a SIMD architecture.

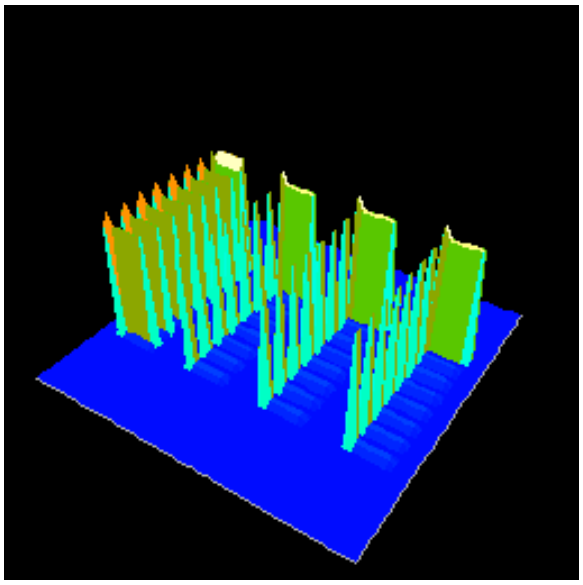

Figure 12. Distributions of number of box64 octree nodes before load balancing.

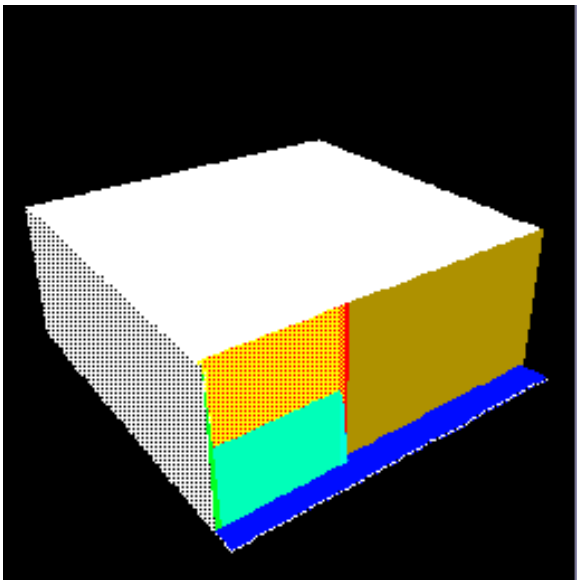

Figure 13. Distributions of number of octree nodes after static load balancing.