at the University of Washington, Seattle, in the department of electrical engineering and the department of computer science and engineering. He earned his MSEE and Ph.D degrees in electrical engineering from the McGill University, Montreal, Canada, in 1983 and 1985, respectively. He also worked as Scientific Officer for Govt. of India, New Delhi from 1974 to 1982. During this period he designed and developed an anti-submarine warfare system for Indian Navy. Professor Somani's research interests are in the area of fault tolerant computing, computer interconnection networks, computer architecture, parallel computer systems algorithms. Currently he is involved in the following projects: i) high integrity system design specifically addressing the issues related to cache memory faults and performance analysis tools for such systems; ii) resource management, fault tolerance, and interworking of LAN/MAN/WAN in ATM networks; and iii) Optical WDM networks design and role of wavelength converters in such networks. He was the chief architect of the Meshkin architecture for fault tolerant computing and Proteus multiprocessor architecture designed and built at the University of Washington to study the effectiveness of coarse-grain processing mechanism. He is also developing an analysis tool, Himap, to facilitate hierarchical modeling and analysis of systems using fault trees, markov chains, and stochastic petri nets.

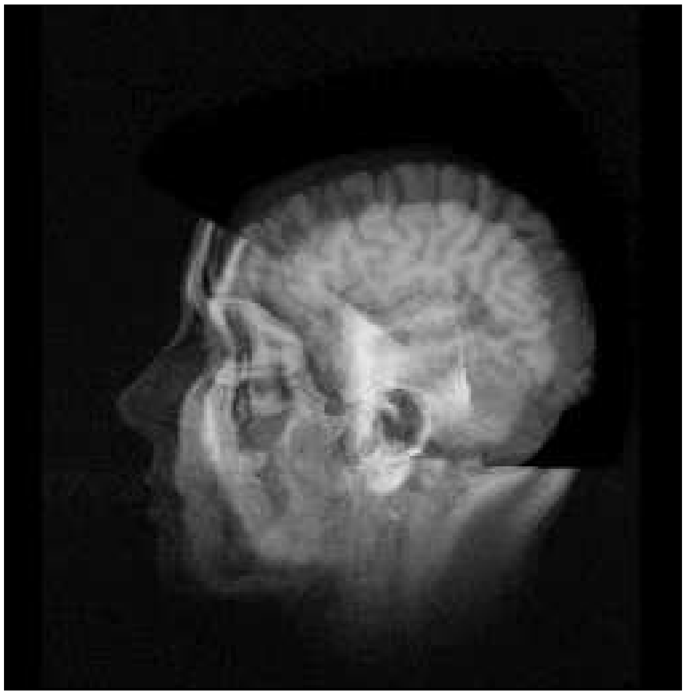

FIGURE 18 MRI 256^3 data set as rendered by permutation warping

Author’s Biographies:

Craig M. Wittenbrink is a research engineer at Hewlett Packard Laboratories and a Research Associate at the University of California at Santa Cruz. He received a B.S. in 1987 in electrical engineering and computer science from the University of Colorado, Boulder, an M.S. in 1990 and Ph.D. in 1993 in electrical engineering from the University of Washington, Seattle. He was a digital hardware design engineer with Boeing Aerospace from 1987 to 1989 developing hardware for computer image generators. As a graduate researcher from 1989 to 1993, he was one of the computer architects and co-authored a patent for the UW-Proteus Supercomputer. From 1994 to 1996 he was a postdoc at the University of California, Santa Cruz working in environmental visualization. He co-chaired the 1995 Parallel Rendering Symposium (PRS), and is now a member of the PRS steering committee. He is a member of ACM, Siggraph, and the IEEE Computer Society.

Arun K. Somani is currently David C. Nicholas Professor of Electrical and Computer Engineering at Iowa State University. Prior to that, from 1985 to 1997, he was a joint faculty member

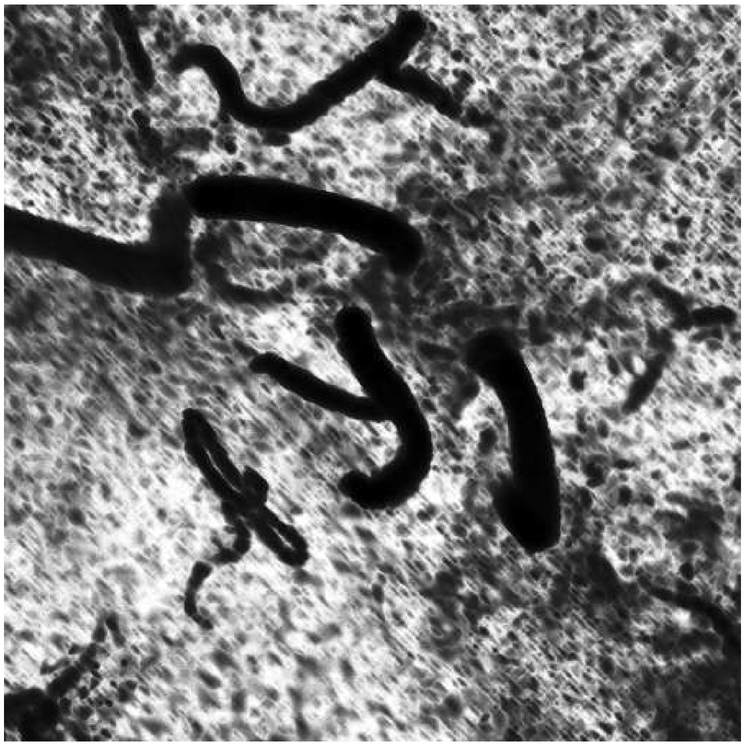

FIGURE 16 Data with Histogram equalization to Show Noise

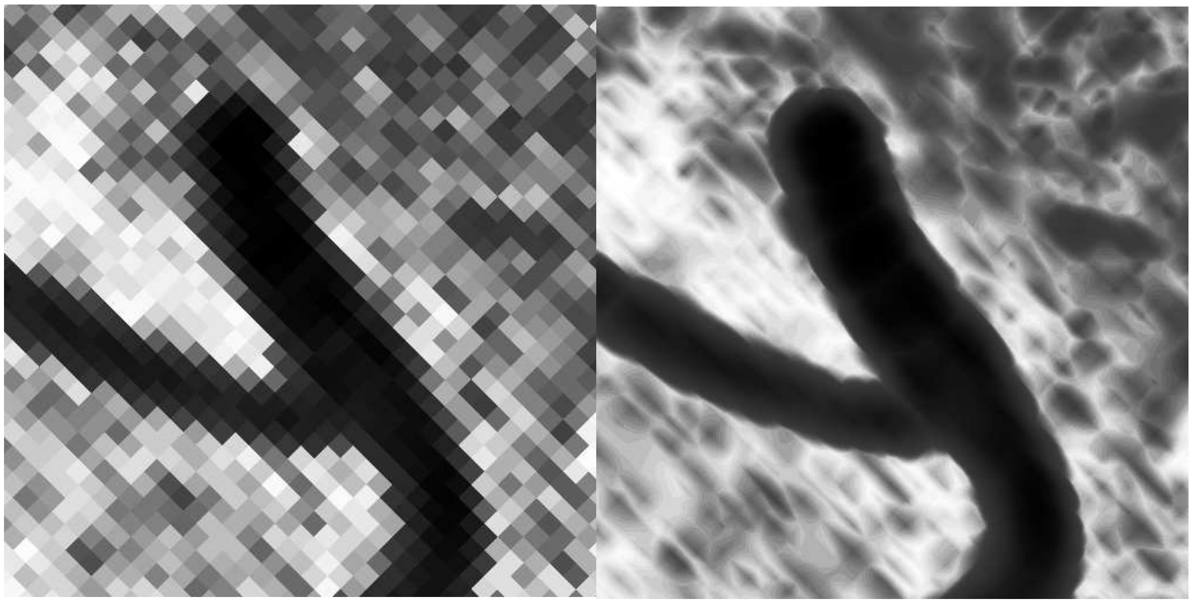

FIGURE 17 8X magnification Zero Order Hold/Trilinear

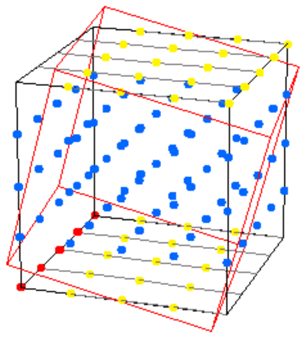

FIGURE 14 Volume Transforms with OS and SS Merged

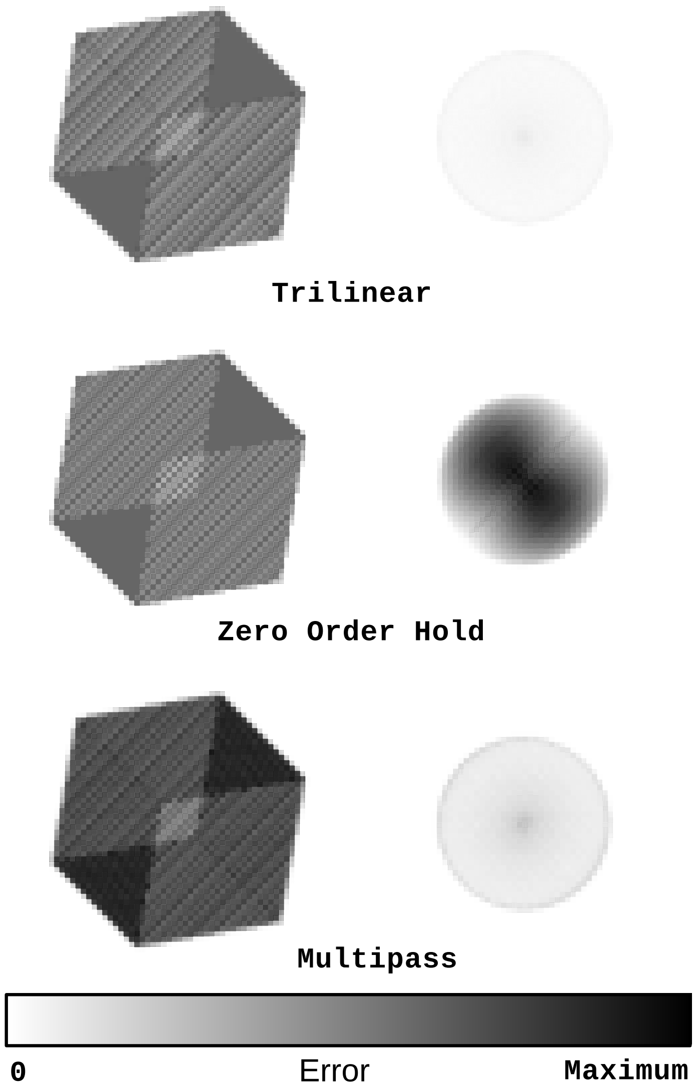

FIGURE 15 Error for 45x45x45 rotations,
Top: Trilinear; Middle: Zero Order Hold; Bottom: Multipass.

[33] Westover, L. Footprint Evaluation for Volume Rendering. *Proc. SIGGRAPH*. ACM, in *Computer Graphics*. **24**, 4 (Aug. 1990), 367-376.

[34] Wilhelms, J., and Gelder, A.V. A Coherent Projection Approach for Direct Volume Rendering. P*roc. SIGGRAPH*. ACM, in *Computer Graphics*. **25**, 4 (1991), 275-284.

[35] Wittenbrink, C. M., and Somani, A. K. 2D and 3D optimal parallel image warping. *J. of Parallel Distrib. Comput*. 25, 2 (March 1995), 197-208. Short version in *Proc. Seventh International Parallel Processing Symposium*, IEEE Computer Society, Newport Beach, CA, 1993, pp. 331-337.

[36] Wittenbrink, C. M. Designing Optimal Parallel Volume Rendering Algorithms. Ph.D. dissertation, University of Washington, 1993.

[37] Wittenbrink, C. M., and Somani, A. K. Permutation Warping for Data Parallel Volume Rendering. *Proc. Parallel Rendering Symposium ‘93*. IEEE Computer Society and ACM, San Jose, CA, 1993, pp. 57-60, color plate pp. 110.

[38] Wittenbrink, C. M. and Harrington, M. A Scalable MIMD volume rendering algorithm. *Proc. Eighth International Parallel Processing Symposium*. IEEE Computer Society, Cancun, Mexico, 1994, pp. 916-920.

[39] Wolfram, S. *Mathematica: A System for Doing Mathematics by Computer. Second Edition*. Addison Wesley, Redwood City, CA, 1988.

[40] Wright, J. R., and Hsieh, J. C. L. A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data. *Proc. IEEE Visualization ‘92*. IEEE Computer Society, Boston, MA, 1992, pp. 340-348.

[41] Yoo, T. S., Neumann, U., Fuchs, H., Pizer, S. M., Cullip, T., Rhoades, J., Whitaker, R. Achieving Direct Volume Visualization with Interactive Semantic Region Selection. *Proc. IEEE Visualization ‘91*. IEEE Computer Society, San Diego, CA, 1991, pp. 58-65.

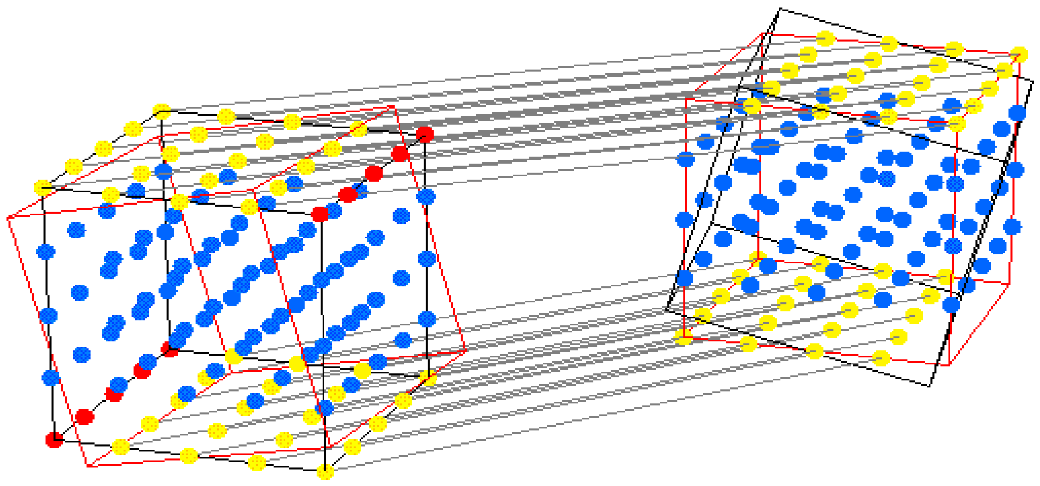

FIGURE 13 Volume Transforms in Parallel

[17] Lacroute, P., and Levoy, M. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *Proc. SIGGRAPH*. ACM, Orlando, FL, 1994, pp. 451-458.

[18] Levoy, M. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*. **8**, 5 (May 1988), 29-37.

[19] Levoy, M. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*. **9**, 3 (July 1990), 245-261.

[20] Ma, K.-L., Painter, J.S., Hansen, C.D., and Krogh, M.F. A Data Distributed, Parallel Algorithm for Ray Traced Volume Rendering. *Proc. Parallel Rendering Symposium '93*. IEEE Computer Society and ACM, San Jose, CA, 1993, pp. 15-22, color plate pp. 105.

[21] Montani, C., Perego, R., and Scopigno, R. Parallel Volume Visualization on a Hypercube Architecture. *Proc. of 1992 Workshop on Volume Visualization.* IEEE Computer Society and ACM, 1992, pp. 9-16.

[22] Neumann, U. Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers. *Proc. Parallel Rendering Symposium '93*. IEEE Computer Society and ACM, San Jose, CA, 1993, pp. 97-104.

[23] Nieh, J., and Levoy, M. Volume Rendering on Scalable Shared-Memory MIMD Architectures. *Proc. of 1992 Workshop on Volume Visualization.* IEEE Computer Society and ACM, 1992, pp. 17-24.

[24] Paeth, A. W. A Fast Algorithm For General Raster Rotation. *Proc. Graphics Interface.* Canadian Information Processing Society, 1986, pp. 77-81.

[25] Porter, T., and Duff, T. Compositing Digital Images. *Proc. SIGGRAPH*. ACM, in *Computer Graphics*, **18**, 3 (July 1984), 253-259.

[26] Pratt, W. K. *Digital Image Processing*. John Wiley & Sons, Inc., New York, NY, 1978.

[27] Schröder, P., and Salem, J.B. Fast Rotation of Volume Data on Data Parallel Architectures. *Proc. IEEE Visualization'91*. IEEE Computer Society, San Diego, CA, 1991, pp. 50-57.

[28] Schröder, P., and Stoll, G. Data Parallel Volume Rendering as Line Drawing. *Proc. of 1992 Workshop on Volume Visualization*, IEEE Computer Society and ACM, 1992, pp. 25-32.

[29] Singh, J. P., Gupta, A., and Levoy, M. Parallel Visualization Algorithms: Performance and Architectural Implications. *IEEE Computer.* **27**, 7 (July 1994), 45-55.

[30] Smith, A. R. Planar 2-Pass Texture Mapping and Warping. *Proc. SIGGRAPH*. ACM, in *Computer Graphics.* **21**, 4 (July 1987), 263-272.

[31] Tanaka, A., Kaneyama, M., Kazama, S., and Watanabe, O. A Rotation Method For Raster Image Using Skew Transformation. *Proc. IEEE Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society, 1986, pp. 272-277.

[32] Vezina, G., Fletcher, P. A., and Robertson, P. K. Volume Rendering on the MasPar MP-1. *Proc. of 1992 Workshop on Volume Visualization*. IEEE Computer Society and ACM, 1992, pp. 3-8.

Bibliography

[1] Bell, G. Ultra Computers: A Tera Flop Before Its Time. *Communications of the ACM*, **35**, 8 (Aug. 1992), 27-47.

[2] Blank, T. The MasPar MP-1 Architecture. *Proc. of Compcon 90.* IEEE, 1990, pp. 20-24.

[3] Blinn, J. F. Light Reflection Functions for Simulations of Clouds and Dusty Surfaces. *Proc. SIGGRAPH*. ACM, in *Computer Graphics*. **16**, 3 (July 1982), 21-29.

[4] Cameron, G.G. and Undrill, P.E. Rendering Volumetric Medical Image Data on a SIMD Architecture Computer. *Proc. of the Third Eurographics Workshop on Rendering.* Eurographics, Bristol England, 1992, pp. 135-145.

[5] Chow, C. and Ng. W.-Y. Fast Data Parallel Rotation of Digital Volume Images. Proc. of the 1994 International Symposium on Speech, Image Processing, and Neural Networks. Hong Kong, April 1994, 760-763.

[6] Drebin, R. A., Carpenter, L., and Hanrahan, P. Volume Rendering. *Proc. SIGGRAPH*. ACM, in *Computer Graphics.* **22**, 4 (Aug. 1988), 65-74.

[7] Elvins, T. T. Volume Rendering on a Distributed Memory Parallel Computer. *Proc. IEEE Visualization '92*, IEEE Computer Society, Boston, MA, 1992, pp. 93-98.

[8] Foley, J., van Dam, A., Feiner, S.K., and Hughes, J.F., *Computer Graphics Principles and Practice, Second Edition.* Addison Wesley Inc., Reading, MA, 1990.

[9] Gibbons, A., and Rytter, W. *Efficient Parallel Algorithms.* Cambridge University Press, Cambridge, England, 1988.

[10] Goel, V., and Mukherjee, A. An Optimal Parallel Algorithm for Volume Ray Casting. *Proc. Frontiers '95. The Fifth Symposium on the Frontiers of Massively Parallel Computation*, IEEE Computer Society, McLean, VA, 1994, pp. 238-245.

[11] Hanrahan, P. Three-Pass Affine Transforms for Volume Rendering. *Proc. of the 1992 Symposium on Volume Visualization*, in Computer Graphics, **24**, 5 (Nov, 1990), 71-78.

[12] Hsu, W. H., Segmented ray casting for data parallel volume rendering. *Proc. 1993 Parallel Rendering Symposium.* IEEE Computer Society and ACM, San Jose, CA, 1993, pp. 7-14.

[13] Kaba, J., Matey, J., Stoll, G., Taylor, H., and Hanrahan, P. Interactive Terrain Rendering and Volume Visualization on the Princeton Engine. *Proc. IEEE Visualization '92*, IEEE Computer Society, Boston, MA, 1992, pp. 349-355.

[14] Kajiya, J. T., and Von Herzen, B.P. Ray Tracing Volume Densities. *Proc. SIGGRAPH*. ACM, in *Computer Graphics*. **18**, 3 (July 1984), 165-174.

[15] Kaufman, A., Editor. *Volume Visualization*. IEEE Computer Society Press, Washington, D.C., 1991.

[16] Kruskal, C. P., Rudolph, L., and Snir, M. The Power of Parallel Prefix. *Proc. International Parallel Processing Symposium.* IEEE Computer Society, 1985, pp. 180-185.

positing), shading, or reconstruction filters, shows that permutation warping achieves high efficiency with flexibility on general machines. Special purpose machines cannot offer this flexibility in shading, combining, and filter choices. Changing viewpoints has been thought to be inefficient, and low quality filters in shearing methods with large memory requirements have been used for efficiency or data duplication used for ray tracing, but we have shown efficiency with a zoh and foh (trilinear filter) reconstruction filters. Our approach generalizes to higher order filters. A three dimensional decomposition was introduced that generalizes the work in pure shears [24][31][27], and improves our direct resampling approach [35]. Our implementation on the MasPar allows rendering with changing viewpoints of five frames/second and two frames/second for higher quality trilinear reconstruction on 128^3 byte volumes. This improves on previous results [4][6][17][27][28][29][32] because of the better filters used, and we discussed the filter differences.

Straight forward extensions to our techniques include coherency accelerations such as adaptive ray termination and adaptive quadrature. Perspective projection can be added as a follow on warp to our permutation warp for two passes versus four of Vezina et al. [32]. The MasPar implementation can also be modified to up sample or down sample, least expensively as an up sample of the transformed image or down sample while warping. Our new three dimensional decomposition will be useful for five pass pure shear and one shift multipass approaches versus eight passes [27][37]. We are continuing to investigate our new decompositions' utility for both sequential and parallel algorithms. Data dependent optimization may be incorporated, similar to Singh et al.'s parallel shear warp factorization [29], and we are working on data dependent optimizations for variants of our permutation warping algorithms.

Acknowledgments

We thank Professors R.M. Haralick, Linda Shapiro, David Salesin, Tony DeRose, and Steve Tanimoto for their help and feedback. We would also like to thank the conference (PRS) reviewers whose comments were invaluable in improving the full length paper. MasPar engineers Jon Becher and Rex Thanakij provided valuable machine support as did Helen Asher who coordinated the MasPar Challenge contest by which a 16k processor MasPar MP-1 was made available.

total compositing time when $R < P \le S$ is $O(S/P + \log P_z/S_{\text{frame}})$ which equals $O(S/P + \log P)$. Compositing achieves linear speedup for $P = O(S/\log S)$.

Each stage PPS, VWS, and CS is $O(S/P)$ for $P = O(S/\log S)$, therefore linear speedup is achieved over the fastest sequential algorithms which are $O(S)$ (Definition 1, Definition 4). $O(S)$ storage is used for optimal space complexity (Definition 3). The lower bound for computation on an EREW PRAM is $\log n$ for n inputs and the run time of the fastest algorithm $P \ge S$ is $\log W$ which meets the lower bound (Definition 2). ■

The amount of parallelism for the volume rendering algorithm can be broken down into four regions. The first region is parallel $1 < P \le R$, or fewer processors than rays. Processors are assigned subcubes that they preprocess, warp, and composite. Each processor stays busy through the parallel product calculations. Run time is $O(S/P)$. The second region is work efficient parallel product $R < P < S$ and $P = O(S/\log S)$, or more processors than rays with a bound. Now some processors become idle during the final steps of compositing. When $P = S$ half of the working processors are idled upon each step. Run time is $O(S/P + \log P)$. This region is where our algorithm is superior to known competitors, achieving linear speedup beyond the number of rays. The third region is non work efficient because processors become idled during compositing and efficiency starts to drop off of linear speedup, $O(S/P + \log P)$.

The fourth region is fully parallel with $P \ge S$ providing the fastest algorithm possible, with multiple processors per voxel, but compositing dominates so run time is $O(\log W)$, determined by the number of sample points along a ray. Storage is $O(S/P)$ for all ranges $P \le S$, and communication is the same order as the computation.

7.0 Conclusions and Future Work

We presented an EREW PRAM algorithm for volume rendering, and demonstrated its efficiency on a parallel machine. Our general reconstruction filter approach provides for time/quality tradeoffs not possible in previous data parallel approaches for improved parallel volume rendering. We believe our algorithm can be ported to nearly all massively parallel general purpose computers. To support this we have ported the algorithm to our UW-Proteus Supercomputer, and demonstrated a speedup of 22 on 32 processors [38]. This fact, and the ability to change combining rules (comproach [36][37][38]. A fully parallel compositing approach was developed in parallel by Ma et al. [20].

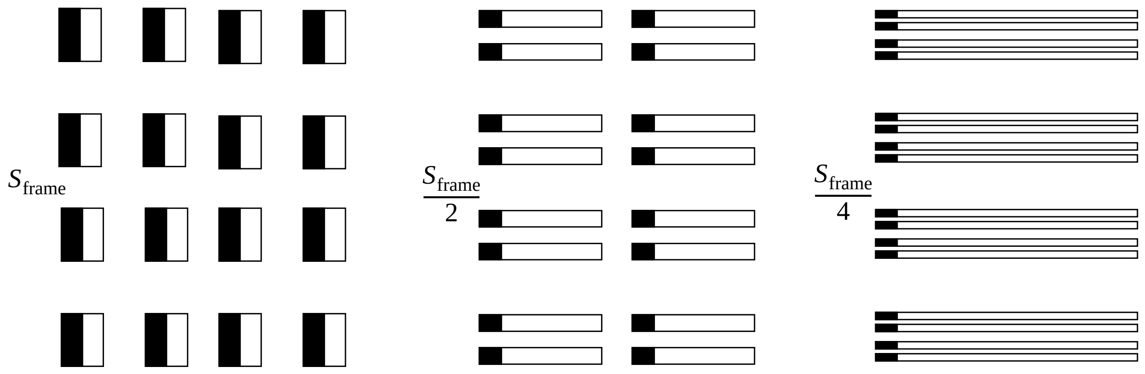

FIGURE 12 Halving of Frames During Parallel Product for Compositing

Theorem 3: Parallel Volume Rendering is an optimal parallel algorithm by definition 1, 2, 3, 4, and 5 for $P = O(S/\log S)$ processors on CREW and EREW PRAMs.

Proof: The preprocessing stage is point operations requiring only neighboring data for time $O(S/P)$, for S sample points and P processors. By *Theorem 1* warping is calculable with exclusive reads and is $O(1)$ for $P = S$, our result in [35]. For $P < S$ partitioning object space subcubes are warped in data parallel or overlap fashion. If rigid body transforms are used the volumes' extents remain constant with virtualization. The algorithm calculates compositing through a product evaluation. A parallel product evaluation [16], is work efficient up to $P = O(n/P + \log P)$ processors in the view depth dimension, FIGURE 6 and FIGURE 12 by *Theorem 2*. The subframes are combined through a parallel product evaluation, starting with $S_{\text{frame}} = R/(P_x P_y)$ samples at each processor, halved at each increment, FIGURE 12, where R is the number of rays in the image.

For $1 < P \le R$ each processor remains busy for all compositing and time is $O(R(P_z - 1)/P) = O(S/P)$. For $R < P \le RW$, processors are idled at some point during compositing. Two terms are, one, for all processors busy which equals $S_{\text{frame}} - 1$, and, two, for when some processors are idled. The subframes are halved until there is one sample in the subframe. There are $\log P_z - \log S_{\text{frame}}$ remaining composites, enough single sample composites to combine all P_z samples. The time for compositing the subframes is $S_{\text{frame}} - 1 + \log P_z - \log S_{\text{frame}}$ for $S_{\text{frame}} < P_z$. The

(See FIGURE 11.) The time complexity is the depth of the tree which is $\log W$. If done sequentially there are $W - 1$ compositing evaluations or $O(W)$. ■

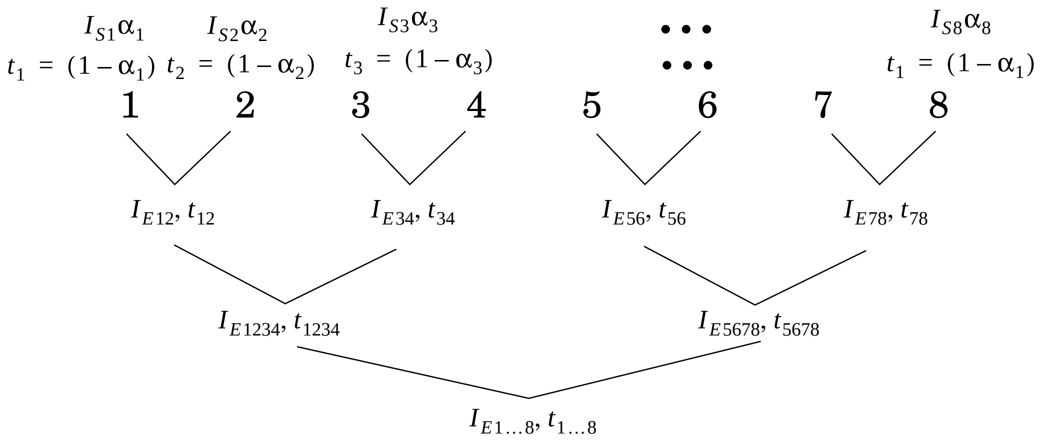

FIGURE 11 Fully Parallel Compositing

Constant factors for additions and multiplications using binary tree compositing are $(3W - 2 - \lceil \log W \rceil)$ multiplications and $(2W - 2)$ additions to take α_i, and I_{Si} to I_{ray} when W is a power of two. The most efficient sequential method, ignoring data dependent optimizations is back-to-front where no incremental transparency or opacity updates are performed giving $(2W - 1)$ multiplications and $(2W - 2)$ additions. (See our work in [36] for details.)

Similar results for the associativity, as well as the logarithmic combine time of the ray compositing, *Theorem 2*, can be found throughout the literature [28][12][8][9], and we do not claim novelty, but we include it here for completeness. *Theorem 3* results from *Theorem 1, Theorem 2,* and *Definitions 1* to *5*, and the fact that we have developed a fully parallel compositing ap-

Proof: As we showed in [35], a translation of all whole number grid positions will be the same, and therefore their location will be rounded by the same amount in the orthogonal basis directions. If the grid positions were unique before they are unique after translation and rounding. ■

Equiareal (including nonscaling affine) warps allow a one-to-one nonlinear processor assignment, and further, using this mapping insures that filtering takes place in local neighborhoods.

Theorem 1: two dimensional and three dimensional equiareal warps T, defined by those transforms whose $|\det(T)| = \pm 1$, can be decomposed into pure shears. The sequence of shears followed by rounding, $\pi' = M\pi$, is one-to-one on the natural numbers and results in a point whose inversed position is always within distance C, or $|\pi - T^{-1}(M\pi)| < C$, providing for a processor assignment on the EREW PRAM to calculate any two or three dimensional equiareal transform in constant time

Proof: Similar to our Theorem 1 in [35], and as equiareal warps are decomposable to pure shears which are essentially translations, from Lemma 1, the translation and rounding are one-to-one. A succession of one-to-one transformations is also a one-to-one transformation. As the inverse transform is linked to the forward transform used to compute the permutation warp, the error is bounded by the number of translation and rounding steps, and given a finite number of these, they are bounded within a constant distance. ■

Lemma 2: Compositing is associative $(I_{E1} \text{over} I_{E2}) \text{over} I_{E3} = I_{E1} \text{over} (I_{E2} \text{over} I_{E3})$.

Proof: Can be shown by algebraic reduction and induction. Essentially compositing is defined by two equations $I_{Eij} = I_{Ei} + I_{Ej} t_i$ and $t_{ij} = t_i(1 - \alpha_j)$ where $t_i = (1 - \alpha_i)$. Compute for three images with different associative groupings given in the lemma, and the resulting intensities will be $I_{E123} = I_{E1} + I_{E2} t_1 + I_{E3} t_1 t_2$ and $I_{E123} = I_{E1} + (I_{E2} + I_{E3} t_2) t_1$ that are the same. ■

Theorem 2: Parallel compositing is $O(\log W)$ and sequential compositing is $O(W)$, where W is the number of sample points along a view ray.

Proof: $I_{\text{ray}} = I_{E1} \text{over} I_{E2} \text{over} I_{E3} \ldots I_{EW}$ by Lemma 2 can be combined through any associativity. Assign two sample points to each processor, composite, and the number of points is halved. Continue this process of halving the number of sample points until the final ray intensity is calculated.

TABLE VII Non-Optimized: Machine Size Speedup for 1K/16K MasPar MP-1

Filters	vol size	Mean	Min.	Max
	128^3	19.9	19.2	35.3
	256^3	20.5	19.6	36.3
Hsu’s Zero Order Hold [12]	128^3	5.84	NA	NA

TABLE VIII Non Optimized: 4K Processor MP-2 Timings for 128^3 Volume Rendering, Seconds

Filter	Mean	Min	Max
First Order Hold	1.124	0.8894	1.231
Zero Order Hold	0.5609	0.3383	0.7511

6.0 Complexity Analysis

We now derive the communication, storage, and run time complexity for permutation warping for data parallel volume rendering. We show that volume rendering is a member of Nick’s class, or is ideally parallelizeable. First, some definitions:

Definition 1: Nick’s Class (NC) is the class of computable and efficiently parallelized algorithms, defined as parallel algorithms that use a polynomial number of processors, $O(n^{k_1})$, and take poly-logarithmic time, $O(\log^{k_2} n)$ [9], where the input size is n and variables k_1 and k_2 are constants.

Definition 2: Optimal speedup. Linear speedup of the parallel program over the fastest known sequential program.

Definition 3: Optimal run time. A lower bound dependent upon the model of computation. $O(\log P)$ on the PRAM for P processors.

Definition 4: Optimal space complexity. $O(n)$ on the order of the input size.

Definition 5: Optimal efficiency. Work efficiency, or time for the parallel algorithm times the number of processors equals the time for the fastest sequential algorithm.

Next we introduce *Lemma 1*, *Theorem 1*, *Lemma 2*, and *Theorem 2* to prove that our algorithm has the properties of definitions 1 through 5, in *Theorem 3*.

Lemma 1: Arbitrary translation and rounding is one-to-one.

For a scalability analysis, a more general implementation which works on any size volumes, and doesn't have the register optimization was used. TABLE VII and TABLE VIII give speedup and timings in seconds for 16k processor MP-1, 1k processor MP-1, and 4k processor MP-2. TABLE VII shows the speedup of the 16k processor machine over the 1k processor machine, which ideally would be 16 because of the additional processing power. Scalability studies using fewer processors than are available are not instructive, and we are limited in the MasPar between a 1k and 16k processor machines available. The table shows that without much virtualization, in 32^3 volumes, the speedup is about 10. With larger volumes the virtualization compensates for overheads in the memory and the network, and the speedup approaches 16. As discussed in our previous work [35], the permutation warping is used only for the trilinear interpolation, or first order hold, and the near neighbor reconstruction, or zero order hold, is done using general communication. What the speedup numbers show for the zoh is that because of the greater congestion in the network, supralinear speedups are obtained, because the larger machine provides less virtualization for a given volume size, resulting in proportionally less congestion. The algorithms do provide linear speedup in problem size (number of voxels), shown by the run times (FIGURE 10, TABLE VII, and TABLE V), and linear speedup in machine size (TABLE VII), empirically supporting our claims of algorithm scalability. Scalability of other approaches, such as the multipass shear warp, and line drawing are good, but sacrifice filter quality as shown in FIGURE 15. Scalability of image order ray tracing approaches such as Hsu's and Goel et al.'s are not competitive and for a 1K to 16k Hsu's algorithm yields only 5.8 versus our 20 fold speedup for zero order/near neighbor resampling (16 would be linear). Goel et al. achieve speedups of 5.0 for 512/4096 pe's for 128^3 data sets (8 would be linear), and speedups of 9.4 for 64/4096 pe's for 64^3 data sets (64 would be linear) on the MasPar MP-1.

TABLE VII Non-Optimized: Machine Size Speedup for 1K/16K MasPar MP-1

Filters	vol size	Mean	Min.	Max
Trilinear	32^3	9.88	9.19	14.4
	64^3	14.5	13.6	21.5
	128^3	15.7	14.6	23.4
	256^3	15.8	14.8	23.8
Zero Order Hold	32^3	8.83	7.20	17.1
	64^3	17.1	15.4	31.5

linear reconstruction. Volumes of size 128^3 can be rendered in 4.8 frames/second with a zoh and 2.0 frames/second with a trilinear filter.

TABLE VI Rotation Only, From [32][27][12] Milliseconds

	Computer	vol size	Time	Speedup Permutation Warp vs. Other
Vezina et al. [32] zoh 4 pass	16k pe MP-1	128^3	49	0.241
	16k pe MP-1	256^3	390	0.243
Vezina et al. [32] foh 4 pass	16k pe MP-1	128^3	139	0.277
	16-k pe MP-1	256^3	1107	0.278
Schröder et al. [27] foh 5 pass	64k pe CM-200	128^3	268	1.320
	32k pe CM-200	128^3	511	2.516
	16k pe CM-200	128^3	1033	5.087
Hsu [12], segment ray cast[a]	16k pe MP-1	128x128x112	238	1.172
Goel et al. [10][b] no reordering	8k pe MP-1	128^3	860/2	2.12
Goel et al. reordering	8k pe MP-1	128^3	1700/2	4.19
Cameron et al. [4]	1k pe DAP 510	128x128x64	100	0.493

a. Compositing done as part of volume reorganization plus stage II initialization overhead.
b. Goel et al.'s algorithm was assumed to scale linearly to 16k nodes for a speedup comparison.

If we are cautious, we may compare directly to other results researchers have obtained on other data parallel machines, realizing that optimizations, generations, implementations, and test cases may vary. The closest comparisons are to [10][12][27][28][32] who use similar voxel sizes and architectures. Comparison of resampling times TABLE VI shows that our direct filters cost more than some multipass forward algorithms [4][32] but are superior to others [27]. The forward wavefront approach [4][28] trades view angle freedom for high performance and a slight speedup over our work. Backwards or ray tracing approaches, such as Goel et al.'s [10], Hsu's [12], and Nieh et al.'s [23] (MIMD shared memory) are image order algorithms that aren't scalable as shown for the Hsu's SIMD algorithm in TABLE VII. For similar algorithms, those with backwards viewing transforms, our permutation warping achieves up to a five times speedup depending on the machine and/or algorithm. We have through permutation warping provided improved quality and view angle freedom for data parallel machine's volume rendering algorithms.

TABLE V Optimized: 16k Processor MP-1 Rendering and Warping only Times in Seconds, for many volume sizes

Filter	vol size	Mean Rendering Time	Mean Warping Only Time
Trilinear	32^3	0.0133	0.0122
	64^3	0.068.7	0.0667
	128^3	0.507	0.502
	256^3	3.998	3.977
ZOH	32^3	0.00768	0.00659
	64^3	0.0304	0.0284
	128^3	0.208	0.203
	256^3	1.623	1.602

FIGURE 10 and TABLE V give the mean run times across all angles for various volume sizes. Note that the performance is tightly bounded and predictable. Communication congestion is low for the data parallel permutation warp. Congestion is defined as conflicts within the interconnection network which result in a delay in communication. See our discussion in [35]. There is no congestion if a processor is available for every sample point. Using the rotation speed of 0, 0, 0 degrees in TABLE IV as zero congestion the congestion is 19% to 29% of the run time for permutation warping, foh. The congestion is 40% to 43% for the backwards algorithm, or zoh, that is not using the permutation assignment. The router start-up penalty and/or the rule overhead account for the rest of the difference. The effectiveness of direct warps lies in the performance filter tunability. The zero order hold takes from 73% (32^3 volumes) to 146% (256^3 volumes) less time than the first order hold, and can be used for interactive performance in viewing the larger volumes. The trilinear interpolation, or first order hold, has comparable performance to the multipass warps but is more accurate as we discussed in Section 4.0.

Our optimized implementation on the MasPar allows rendering with changing viewpoints of 130 frames/second for 32^3 volumes to 32^2 images and 75 frames/second for higher quality tri-

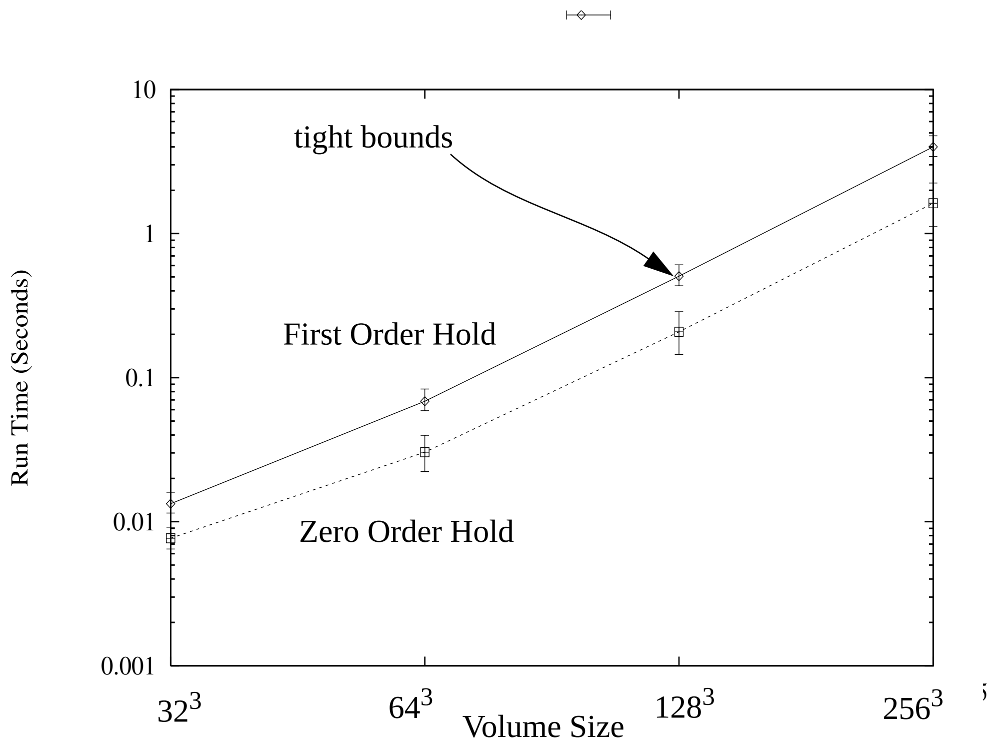

FIGURE 10 Optimized 16K Processor MP-1, Run Times (Log) Versus Volume Size (See TABLE V)

TABLE IV Optimized: 16k Processor MP-1 128^3 Rendering Times in Seconds, Rotation on Multiple Axes

Filter	Rotation Axes	0	20	40	60	80
FOH	*x*	0.434	0.482	0.496	0.505	0.533
	y	0.434	0.485	0.494	0.498	0.537
	z	0.435	0.499	0.508	0.512	0.543
	x, *y*, and *z*	0.434	0.502	0.508	0.521	0.607
ZOH	*x*	0.145	0.186	0.196	0.205	0.237
	y	0.145	0.187	0.192	0.199	0.237
	z	0.146	0.206	0.213	0.222	0.240
	x, *y*, and *z*	0.145	0.207	0.212	0.224	0.253

The over operator can be done similarly using the Scan operator to create the proper transparency at each processor, and then doing a parallel addition by ScanAdd.

We present two sets of run time measurements, optimized timings on the 16k machine and non-optimized timings on the 1k and 16k MP-1 and 4k MP-2. We examine the effect of volume size, machine size, and view angle on the run time. The comparisons done with other algorithms use an optimized version of the program which uses power of two volume sizes, and explicit register use in the MasPar MPL code. Measurements given are the average of multiple runs at each angle. FIGURE 9 shows the run times to render a 128^3 byte volume to a 128^2 image versus resampling angle. The zero order hold is most efficiently calculated without using permutation warping, as we showed in [35]. TABLE IV shows FIGURE 9's run times for some of the angles. The rotation only times are given in FIGURE 9 and TABLE V also showing how the resampling for warping takes the majority of the time. The warping time is shown in the figure a small fraction below the run time. The many lines for each filter show rotation about x, rotation about y, rotation about z, and rotation about x, y, and z. Each time represents rendering from the original data, and not an incremental rotation. Run time is nearly constant for a wide choice of viewing angles, a desirable property of our algorithm resulting from the communication efficiency.

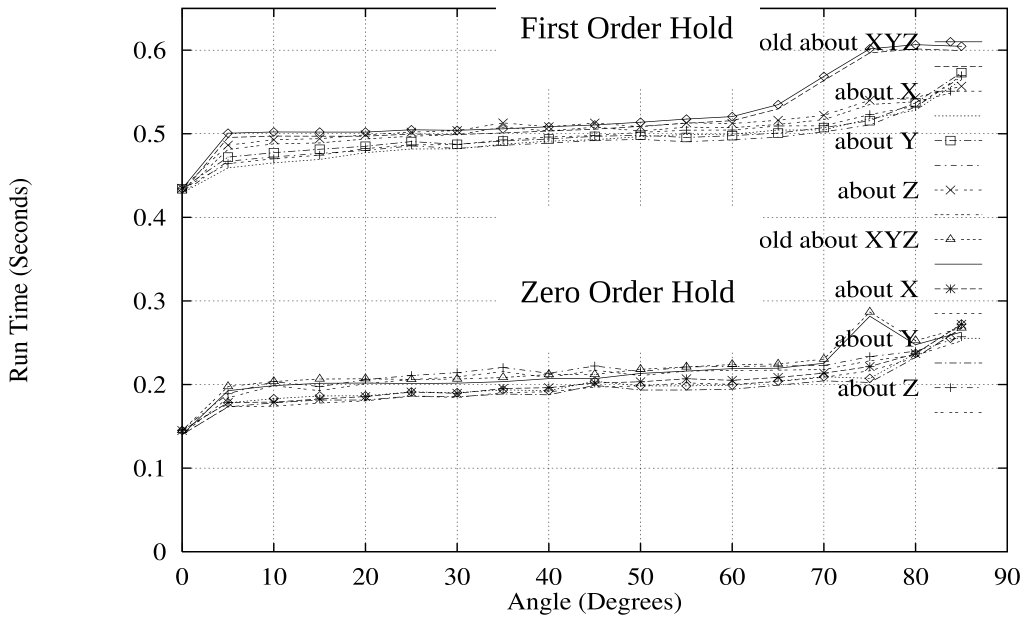

FIGURE 9 Optimized 16K Processor MP-1, Volume Rendering and Warping Run Times, Nearly Constant Run Time Versus Angle

approaches are repeated linear interpolations, the percent resolution error for the multipass warp may be up to 132%, and the percent interpolation error is 11.1% compared to 44% and 3.7% for a direct warp (approach as in Pratt [26]). Exact characterization of the multipass warp filtering error is difficult because of repeated aliasing which causes the transfer function to be highly nonlinear. FIGURE 16 shows the noise inherent in the MR angiography data, shown by the intensities scattered throughout the volume. FIGURE 17 shows the 256x256x32 data rendered at 512^2 zooming (x8) in on the bifurcation of FIGURE 16. Fast traversal is possible with the zoh of FIGURE 17 (left) and a more accurate trilinear filter is used to render FIGURE 17 (right). The filter difference on these medical images is readily apparent.

5.0 MasPar MP-1 Performance Study

Performance measurements were taken on the MasPar MP-1 and MP-2 [2]. The MasPar computers used for the performance study were 1024 and 16384 SIMD processor MP-1s and a 4096 processor MP-2. The larger machine's peak performance is 26,000 MIPS (32 bit integer) and 1,200 MFLOPS (32 bit floating point). The architecture supports frame buffers through VME frame grabbers, HIPPI connection, or through MasPar Corporations's frame buffer (not available at time of study). Image display in the current implementation is done on the X host. The processors are interconnected through a mesh with 23,000 Mbytes/second peak aggregate bandwidth and a multistage crossbar router with 1,300 Mbytes/second peak aggregate bandwidth. The array controller provides a software accessible hardware timer that accurately captures the elapsed run time.

Our implementation in MPL, a C like parallel language, uses the slice and dice virtualization discussed in Section 3.0. Only volumes with equal sides were used for the study, but this is not a limitation of the algorithm. The neighboring processors do not need to be accessed in the resampling step because of a one voxel overlap of volume storage on each processor. The overlap allows a random access to replace a costly case decision in the SIMD language. The storage overlapping does not restrict the size of volumes that can be processed in practice, because dynamic memory allocation has a small overhead and a 2^n request of memory will use a 2^{n+1} block. Therefore a $2^n + m$ request will use the same space as well as long as m is much less than 2^n. We take advantage of the MasPar instruction ScanMax. Once each processor composites its subcube, ScanMax composites across z in segments to complete each parallel product in one instruction.

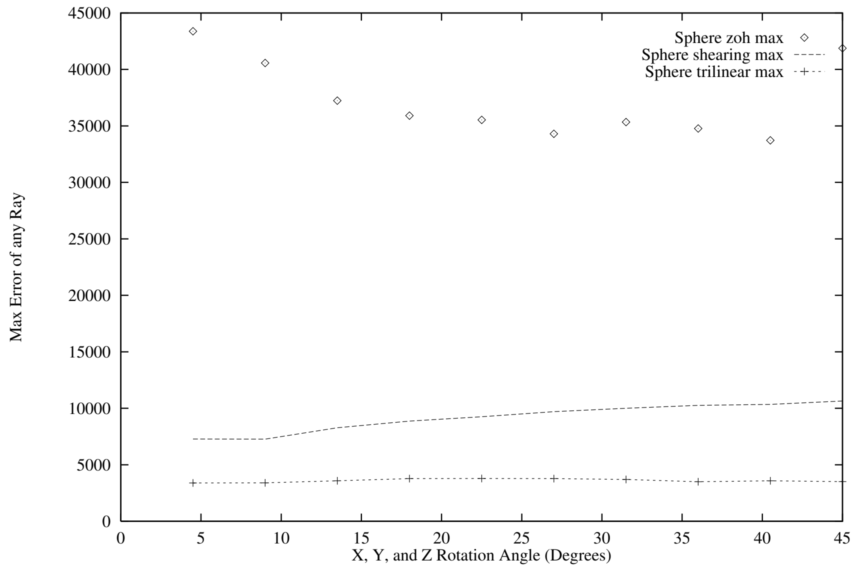

FIGURE 8 Maximum Error in Reconstruction of Sphere

TABLE III Absolute summed error on rays for 45, 45, 45 degree rotation (See FIGURE 15)

	Cube		Sphere	
	mean	max	mean	max
zoh	48372	131070	3977	41886
multipass	65028	217719	802	10648
trilinear	48309	172078	259	3501

The mean error varies little with different view angles. The maximum error does vary with view angle, with the largest errors resulting when view rays nearly glance off of the cube, or when the rotation angle is greater for the sphere. FIGURE 15 reveals that rays passing near the edge of the cube and sphere have more error for the multipass approach, that the multipass approach has the greatest error in the cube resampling, and that error is evenly spaced across the rays for the trilinear reconstruction.

The trilinear is clearly better than shearing, but the zero order hold is the same as the trilinear for the cube and worse than trilinear and shearing for the sphere. By assuming the resampling

TABLE II

Mean of the Measured Absolute Summed Error means over all rays for the resampled cube and sphere, angles five to 45 degrees.

	Cube	% worse than trilinear	Sphere	% worse than trilinear
zoh	48370	0%	4079	1468%
multipass	64068	32%	775	198%
trilinear	48385	0%	260	0%

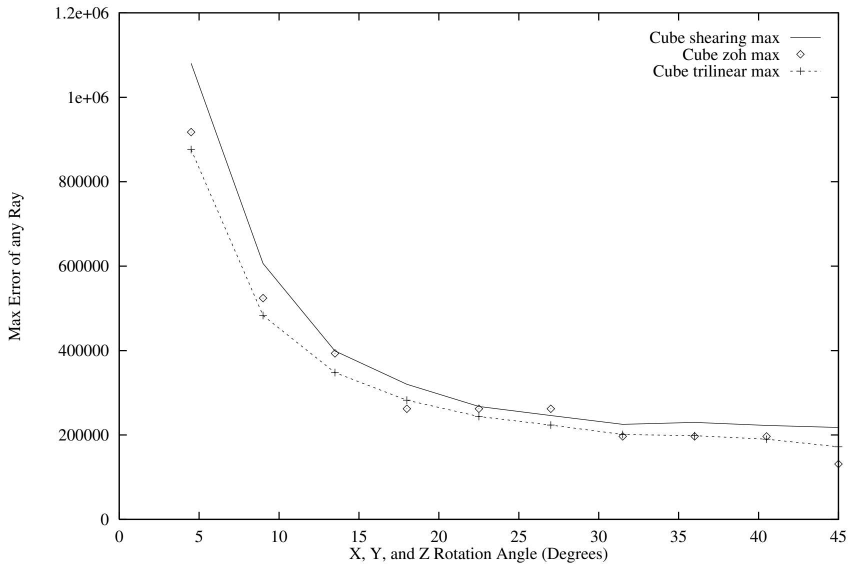

FIGURE 7

Maximum Error in Reconstruction of Cube

differencing each sample point for a rotated viewpoint with an analytically defined cube or sphere. Absolute error values were summed on each view ray. We compare three filters: a zero order hold (zoh), a first order hold (trilinear), and a multipass filter using linear interpolation.

FIGURE 7 and FIGURE 8 show the maximum error (intensity sums) for the three approaches versus rotation angle (degrees) about all axes simultaneously for resampling the cube and sphere volumes. Maximum error is the summed intensities. The mean error for all rays in the image remains the same for different view angles (angles five to 45 degrees) TABLE II. FIGURE 15 shows how error is placed in the image, with error ramped from the maximum errors in the pseudo colored images with TABLE III showing the mean and max errors for the 45, 45, 45 degree rotations. The range of error is from zero to 217719 (multipass shear filter highest error) for the cube images and zero to 41886 for the sphere images (zoh filter highest error).

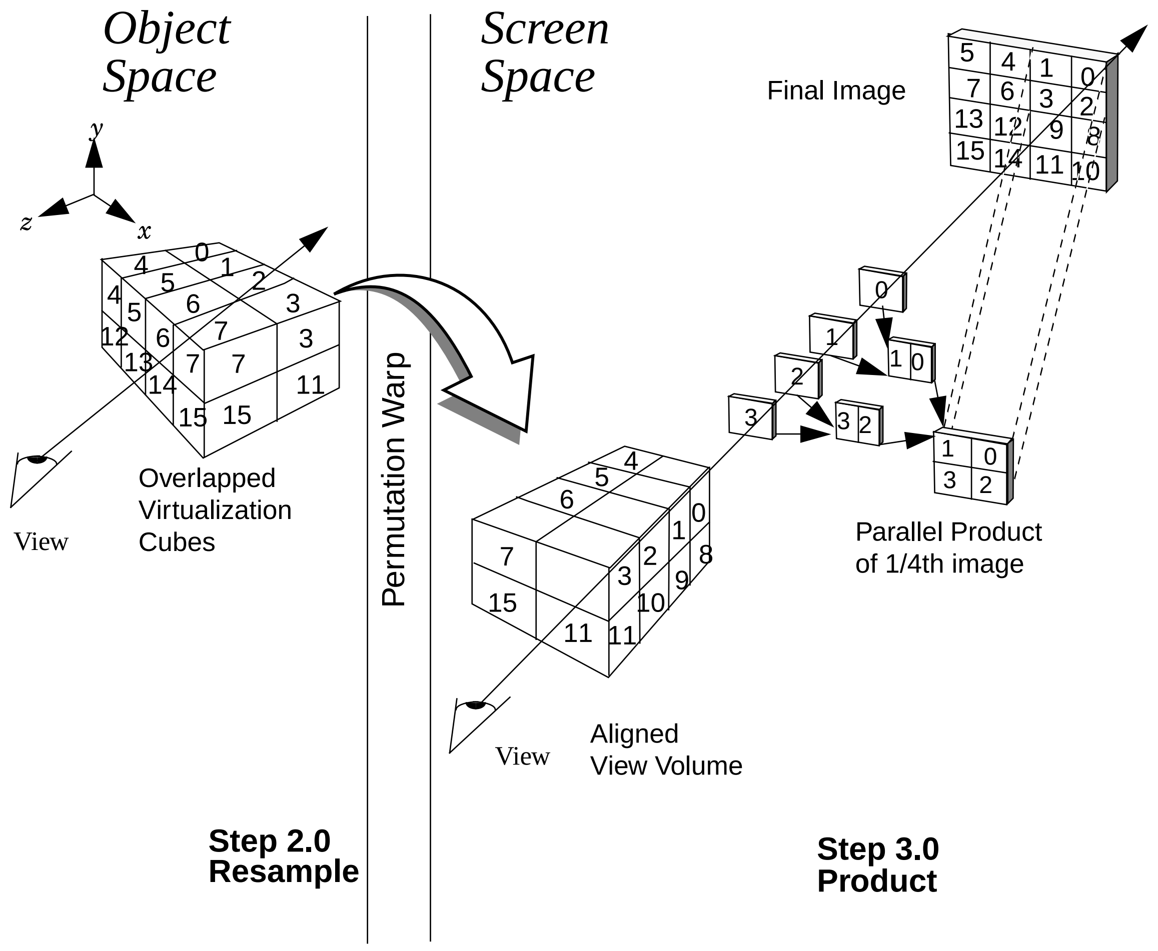

FIGURE 6 Steps of Virtual_Permutation_Volume_Render, Virtualized Subvolumes to SubFrames to Final Image

evaluation so they all remain busy. FIGURE 6 shows how the processors start with 1/4 of the screen, then get 1/8, and finally 1/16. FIGURE 6 also shows how the permutation warp (Step 2) computes a fully realigned volume that may be composited. Note especially how the view ray is coincident with the volume edge after the warp, and therefore coincident with the samples. The SS samples being calculated are unique, and in the EREW PRAM there will be no conflicts, but because of virtualization processors may receive more than one message. The density of messages across the network is the same if the slice and dice virtualization is used and communication remains efficient. The aligned view volume in Figure 6 may contain many zero valued voxels depending on the view, but the final compositing stage is less than 10% of the total calculation so the effect is small. An alternative approach where the communication is done on the subcube/processor integer grid is also possible.

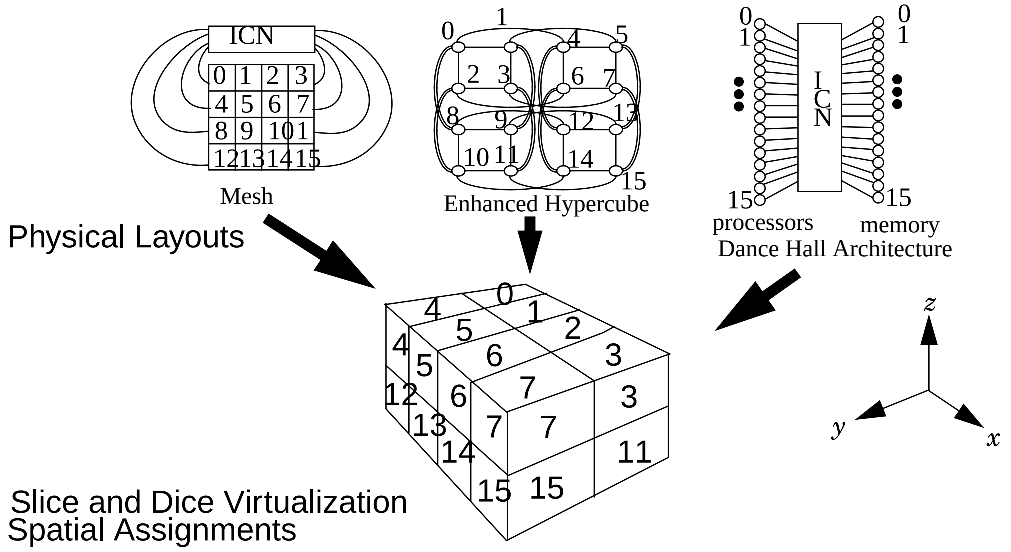

FIGURE 5 Spatial Volume Virtualization For a Variety of Architectures

4.0 Time/Quality Trade-offs

Multipass shears, [24][27][31][32], and direct warping, Section 2.0 [35], are not equivalent. A shear filtering approach has more resolution error and interpolation error than a comparable direct filter, because each resampling discards the prior data. We used two test objects to calculate the reconstruction error: a cube of intensity 65535 and a sphere whose intensities are zero at the edge and linearly ramp up to 65535 at the center. The sphere and cube were centered within 128^3 volumes of 16 bit voxels with a diameter/width of 64. A Sun Sparc 2 was used to calculate the comparison to ease implementation of the shearing approach. The errors were calculated by

For arbitrary centered rotations the inverse T^{-1} is easily calculated because rotation $R(\psi, \phi, \theta)$ is orthogonal, meaning $T^{-1} = T^{\mathrm{T}}$, and translations are inversed by negating their values,

$$\begin{aligned} T^{-1} &= (T(c_x, c_y, c_z)R(\psi, \phi, \theta)T(-r_x, -r_y, -r_z))^{-1} \\ &= (T(-r_x, -r_y, -r_z))^{-1}(R(\psi, \phi, \theta))^{-1}(T(c_x, c_y, c_z))^{-1} \quad . \\ &= T(r_x, r_y, r_z)(R(\psi, \phi, \theta))^{\mathrm{T}}(T(-c_x, -c_y, -c_z)) \end{aligned} \qquad \textbf{(EQ 10)}$$

The rotation matrix and a translation matrix are given in (EQ 11) and (EQ 12), and the transpose of (EQ 11) is composed with the translations for calculating the inverse with the minimum number of calculations.

$$R(\psi, \phi, \theta) = \begin{bmatrix} (\cos\phi\cos\psi) & (-\cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta) & (\cos\psi\cos\theta\sin\phi + \sin\psi\sin\theta) & 0 \\ (\cos\phi\sin\psi) & (\cos\psi\cos\theta + \sin\phi\sin\psi\sin\theta) & \cos\theta\sin\phi\sin\psi - \cos\psi\sin\theta & 0 \\ (-\sin\phi) & (\cos\phi\sin\theta) & (\cos\phi\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(EQ 11)}$$

$$T(c_x, c_y, c_z) = \begin{bmatrix} 0 & 0 & 0 & c_x \\ 0 & 0 & 0 & c_y \\ 0 & 0 & 0 & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(EQ 12)}$$

3.0 Data Parallel Virtualization

When there are fewer processors than sampling points, we emulate v virtual processors on $P < v$ physical processors. To virtualize we make an assignment of P processors to the S voxels. Object space voxel points are assigned to processor id’s by an address tiling. A slice, row major addressed volume coordinate is transformed to a sliced and diced coordinate by permuting bits of the address $\langle t|k|r|i|s|j\rangle \rightarrow \langle t|r|s|k|i|j\rangle$, where $\langle t|k\rangle$ is the slice, $\langle r|i\rangle$ is the row, $\langle s|j\rangle$, r, s, t is the processor, and i, j, k is the voxel address within a processor. Such virtualization is amenable to a wide variety of architectures such as mesh [2], enhanced hypercube, and multistage interconnection networks. FIGURE 5 shows how machines with 16 processors are virtualized into a three dimensional volume. Each dimension gets approximately $P^{1/3}$ divisions, for P_x, P_y, and P_z assignments of processors in the x, y, and z dimensions respectively.

The algorithm is the same as that in FIGURE 3, except now processors have more points to iterate over, S/P points each. In step 1, 2, and 3 a for loop is added to compute S/P points, and during compositing the screen space assigned to each processor shrinks after each parallel product one shifting pass (due to $c_{21} = 1$) versus the eight passes from [27], which we use in our permutation calculation. It could also be used for a three multipass warping provided the data could move in two directions operating on scanframes, or a five pass three dimensional transform working on scan lines improving Paeth [24], Tanaka et al.'s [31], and Schröder et al.'s [27] algorithms.

Most viewing transforms are rigid body transformations. As additional examples, we show how arbitrary rotation, and then arbitrary translation and rotation are decomposed into $\overline{M}$ matrices. Reflection can be easily added. (EQ 7) shows 3D rotation as a concatenation of rotation about each axis x, y, and z [FOLE90] p. 215.

$$R_z(\psi)R_y(\phi)R_x(\theta) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \text{(EQ 7)}$$

This transformation is decomposed into pure shears. (EQ 8) gives a decomposition of $R_x(\theta)$, or rotation about x by θ, into pure shear matrices. Rotation about y and z are done likewise with the 2D decomposition developed in (EQ 2), and 9 matrices result.This is how the solution was derived prior to our derivation of the 5 pass given in (EQ 5).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\tan\theta/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \sin\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\tan\theta/2 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(EQ 8)}$$

After each shear operation the point coordinate being operated upon is rounded to an integer coordinate maintaining the one-to-one assignment. The operation for the right most matrix in (EQ 8) results in $\overline{M}_\eta = \mathrm{Round}(y - z\tan\theta/2)$. Because only one coordinate is affected, and no scaling is used, rounding chooses a unique coordinate.

The inverse used in determining the reconstruction point is numerically stable. In fact equiareal transformations are by definition invertible. For arbitrary centered rotation the transform is a product of translation matrices, $T(x, y, z)$, and the rotation matrix, $R(\psi, \phi, \theta)$. We rotate about the point (r_x, r_y, r_z) and center the rotation in the output about (c_x, c_y, c_z). The aggregate transformation given in (EQ 9) is decomposed using (EQ 5) and contracted into operations on single coordinates, and used to calculate M.

$$T = T(c_x, c_y, c_z)R(\psi, \phi, \theta)T(-r_x, -r_y, -r_z) \quad \text{(EQ 9)}$$

$$\det\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\right) = a_{11}a_{22} - a_{12}a_{21} = \pm 1 . \quad \textbf{(EQ 1)}$$

The other four equations are found by setting

$$T = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & b_1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ b_2 & 1 \end{bmatrix}\begin{bmatrix} 1 & b_3 \\ 0 & 1 \end{bmatrix} \quad \textbf{(EQ 2)}$$

The solution using a_{ij} ‘s as given is $b_1 = (a_{11} - 1)/a_{21}$, $b_2 = a_{21}$, and $b_3 = (a_{22} - 1)/a_{21} = (a_{12}a_{21} - a_{11} + 1)/(a_{11}a_{21})$. A special case is rotation, where

$$T = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad \textbf{(EQ 3)}$$

and $b_1 = (\cos\theta - 1)/(\sin\theta) = -\tan\theta/2$ by insertion and reduction by the half angle formula, $b_2 = \sin\theta$, and $b_3 = (\cos\theta - 1)/(\sin\theta) = b_1$. This derivation shows how to calculate the result given in [24] [31].

The same approach is used for three dimensional equiareal transforms solving a system of ten equations with nine unknowns in (EQ 4) and (EQ 5),

$$\det(A) = \pm 1 \quad \textbf{(EQ 4)}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & b_{12} & b_{13} \\ 0 & 1 & b_{23} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ c_{21} & 1 & 0 \\ c_{31} & c_{32} & 1 \end{bmatrix}\begin{bmatrix} 1 & d_{12} & d_{13} \\ 0 & 1 & d_{23} \\ 0 & 0 & 1 \end{bmatrix} . \quad \textbf{(EQ 5)}$$

The system appears to be over constrained, but can in fact be solved. The symbolic solution from Mathematica (TM) [39] is,

$$c_{31} = a_{31}, \qquad c_{32} = \frac{a_{31} - a_{22}a_{31} + a_{21}a_{32}}{c_{21}}, \qquad d_{12} = \frac{a_{22} - 1}{c_{21}} + \frac{a_{32}}{a_{31}} - \frac{a_{21}a_{32}}{c_{21}a_{31}}, \qquad d_{23} = \frac{c_{21} + a_{23}a_{31} - a_{21}a_{33}}{a_{22}a_{31} - a_{21}a_{32}},$$

$$d_{13} = \frac{a_{23}}{c_{21}} - \frac{a_{23}a_{31} + a_{21}a_{33} - c_{21}}{c_{21}(a_{22}a_{31} - a_{21}a_{32})} + \frac{a_{33}}{a_{31}} - \frac{a_{21}a_{33}}{c_{21}a_{31}} ,$$

$$b_{23} = \frac{a_{21} - c_{21}}{a_{31}}, \; b_{13} = \frac{a_{11}}{a_{31}} - \frac{c_{21}a_{11}a_{32} + c_{21}a_{31}a_{12} - a_{31}}{a_{31}(a_{22}a_{31} - a_{21}a_{32})}, \; b_{12} = -\frac{1}{c_{21}} + \frac{a_{31} + c_{21}(a_{12}a_{31} - a_{11}a_{32})}{c_{21}(a_{22}a_{31} - a_{21}a_{32})} . \quad \textbf{(EQ 6)}$$

As c_{21} is not specified, we assign it to be equal to one. The solution above allows direct solution for a three pass nonscaling transform (EQ 5), or five single coordinate shears passes and volume rotation. There are no conflicts. Each line connects only two processors shown by the parallel nature of all of the lines. The object space processor bounding box is upright in the object space, and the forward T warped version is also rotated in the screen space. The screen space processor bounding box is upright in the screen space and rotated in object space. Of course, all processors are both object space processors π and screen space processors π' with $\pi[i, j, k] = \pi'[i, j, k]$. This is shown by those processors who interpolate for themselves, the processors in the interior where communications arcs are not drawn.

We have further qualified the transforms, T, beyond our work in [35] that permutation warping can be used for. They are the equiareal transforms defined by $\det(T) = \pm 1$ (determinant). Here we develop a solution for any 3D transform of this type. The processor assignment is calculated by the transform $\pi' = M(\pi)$. This permutation transforms points p_{π}, whose coordinates are a tuple of integers, to another point, $p'_{\pi'}$ whose coordinates are also integers. An integer coordinate field is mapped to another integer coordinate field, and the point $p'_{\pi'}$ when inversed by T^{-1} to $p_{\pi'} = T^{-1}(p'_{\pi'})$ is within π's neighborhood. Obviously M and T are closely related. The distances satisfy $|(p_{\pi'})_x - p_{\pi_x}| < C_x$, $|(p_{\pi'})_y - p_{\pi_y}| < C_y$, and $|(p_{\pi'})_z - p_{\pi_z}| < C_z$, a working definition of a neighborhood.

$M = \overline{M}_1\overline{M}_2\ldots\overline{M}_\eta$ is a concatenation of pure shear, translation, and round operators. Rounding is used to snap real values to integers. Shears are non angle preserving affine transforms. A pure shear is nonscaling and preserves distance. Any affine transform, T, with $\det(T) = 1$ is nonscaling, or equiaffine. This includes shears, rotations, and translations, that are all isometries. By allowing $\det(T) = \pm 1$, reflections can also be calculated for equiareal transforms. An isometry is a one-to-one and onto transform that preserves distances.

Next, we derive some new results that further define and clarify permutation warping for two and three dimensional transforms. The general solution to a two dimensional equiareal transform is calculated by solving a system of five equations with three unknowns. The unknowns are the coefficients in the three pass shearing operation. The knowns are the components of the transform matrix T. An equiareal transform by definition has

2.1) Calculating processor assignments $\pi' = M(\pi)$; the logical connection is shown by the dotted line in FIGURE 4.

2.2) Calculating reconstruction point $p_{\pi'} = T^{-1}(p'_{\pi'})$; the inverse transform is shown by a solid line and the point is shown as an asterisk (*).

2.3) Performing resampling of α_p and I_{Sp}, reading the values of I_{Sp} and α_p of its neighboring processors. The number of neighbors used determines the filter order. FIGURE 4 shows possible neighbors of $p_{\pi'}$ as darkened cubes.

2.4) Sending resampled values to screen processors π'.

In **Step 3**, a parallel product evaluation combines resampled intensities and opacities. Binary tree combining computes products for any associative (not necessarily commutative) operator $\otimes$, $I_1 \otimes I_2 \otimes \ldots \otimes I_W$ [16][25]. Compositing ($I_i \text{over} I_j$) is associative. Numerical integration is also associative.

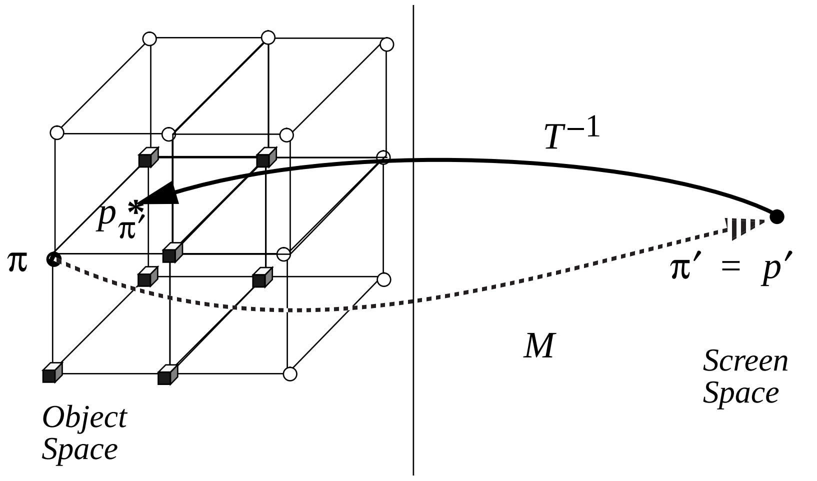

FIGURE 4 Transforms and Communications in Permutation Warping for a Single Voxel

2.1 Processor Assignment by Permutation Warp

Paeth [24], Tanaka et al. [31], Schröder et al. [27], and we [35] have used pure shear matrix decomposition of rotation to create efficient resampling algorithms. A pure shear is a non scaling transform of a single coordinate. The technique is a refinement of multipass filtering where the transform is restricted to rotation. We use the shears only to calculate the processor assignment, and use a better filter. While Schröder and Salem used a one to one assignment [27] to calculate multipass resampling, we are interested in calculating a direct one pass resampling. A permutation warp M calculates $[i', j', k']$ given $[i, j, k]$ and a transform T. To understand why we go through the extra work of calculating M, FIGURE 13 shows communications taking place in parallel for a numbers) are assigned sample points $p[x, y, z] \in \Re$ requiring $P = S$ processors where S is typically n^3 voxels. Our algorithm consists of the following three steps (as introduced in FIGURE 2):

```
$I_{\text{ray}}[\ ] \leftarrow$ Permutation_Volume_Render($V$, $\Gamma$, $T$, classify, shading) {
    1.0) PPS, at each processor $\pi$ calculate $\alpha_p$, $I_{Sp}$ in parallel.
    2.0) VWS, Processor $\pi$ does in parallel:
        2.1) Calculate processor assignments $\pi' = M(\pi)$
        2.2) Calculate reconstruction point $p_{\pi'} = T^{-1}(p'_{\pi'})$
        2.3) Perform resampling of $\alpha_p$ and $I_{Sp}$
        2.4) send resampled values to SS processors $\pi'$
    3.0) CS, calculate ray intensities $I_{\text{ray}}$ with parallel product.
    }
```

FIGURE 3 Permutation Warping Parallel Volume Rendering Algorithm

In **Step 1**, processors classify and shade reading neighboring data as necessary.

In **Step 2**, each processor resamples the opacities, α_p, and intensities, I_{Sp}, to be aligned with the view rays. If done in a straight forward fashion this would require many rounds of communication, but we have developed a permutation warp that requires only one communication [35]. We resample in the object space (OS) near where the points lie, and then send the resampled data to its screen space position. The challenge to doing this is using a rule, M, to calculate processor assignments for the viewing transform. For arbitrary equiareal view transformations there is always a one-to-one mapping, and therefore no multipass resampling nor accumulated rounding errors.

FIGURE 4 illustrates the transforms calculated by a single processor π. The object space and screen space are separated, the object space on the left and the screen space on the right. A processor π does permutation warping by:

A network efficient parallel algorithm is the *multipass forwards* approach [4][6]-[13][17][27][32][40]. Multipass forwards algorithms use a decomposition of the viewing transform into shears for low network congestion, but they suffer from lower filter quality and view angle restrictions. Of recent interest is the shear warp multipass forwards algorithm combined with data dependent optimizations [17][29]. FIGURE 15 illustrates the sampling error for analytical functions sampled by direct resampling and also by multipass resampling. The multipass resampling errors are larger than the single pass resampling errors, and the multipass forwards filter difficulties are only compensated by restricting viewpoints and over sampling [17].

The *forwards wavefront* approach [4] works easily on SIMD machines with simple interconnection networks, such as a rings [28], and gives better filter quality than the multipass methods. Limitations are similar to the multipass forwards methods where view angles are restricted, and the filter quality is not as good as a backwards viewing transform because of a post projection resampling also used in the multipass forwards algorithms of [29] and [27]. Perspective projection is not possible, and the technique suffers from network congestion as well [28].

Forwards splatting algorithms using Westover's technique [7][22][33][34] have been described as easily parallelizeable. Splatting techniques suffer from ordering noise because of the unavoidable overlap in the splatted kernels. Similar to ray tracing, general viewpoints require random accessing, in this case of the screen, resulting in congested writes. An implementation without view angle freedom by Elvins [7] uses sequential compositing limiting speedup. Neumann [22] has demonstrated an algorithm on mesh machines.

2.0 Permutation Warping

Our permutation warping solution is essentially a data parallel processor assignment technique that provides a general approach for efficient parallel transform algorithms. Permutation warping is better than prior parallel algorithms because it is memory efficient, processor efficient, general, and accurate. The algorithm, FIGURE 3, calculates the same image as FIGURE 2, but gives specific memory layout and communication requirements necessary for the exclusive read exclusive write parallel random access machine (EREW PRAM). Processors $\pi[r, s, t] \in \mathrm{N}$ (natural or whole ing more processors than rays, so provides truly massive parallelism which other solutions have not provided. We present algorithm analysis results, where the EREW PRAM (exclusive read exclusive write parallel random access machine [9]) storage complexity is $O(S)$ for $S = n^3$ samples; run time is $O(S/P)$ for P processors when $P = O(S/\log S)$; and run time is $O(\log W)$ for W sample points along a ray and $S/\log S < P \leq S$.

1.1 Related Work

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 [8]. Nieh and Levoy [23], Chow and Ng [5], Yoo et al. [41], Montani et al. [21], Neumann [22], Goel et al. [10], and Hsu [12] have developed backwards (ray tracing) volume rendering algorithms for parallel computers. Nieh and Levoy use a shared memory machine (Stanford DASH) where arbitrary memory requests are satisfied by the system. Memory congestion and storage overhead are the primary disadvantages, but the architectural strength of the DASH gives nearly linear speedup. Chow et al. used a processor assignment technique by iterative methods, which results in a variable run time and requires dynamic load balancing. Yoo et al. implemented a backwards algorithm on Pixel Planes 5, a distributed memory machine, and because of network congestion elected to replicate the data set on every processor. This results in high performance, but limits the amount of data that can be rendered. Montani et al. encountered similar difficulties on the nCUBE, where clusters of processors get copies of the data set, and data must also be sent on request resulting in both memory limitations and network congestion. Goel et al.'s algorithm uses multiple passes of hypercube communication, and also idles processors as compositing occurs. Hsu's algorithm fully distributes the volume, but bottlenecks result when going from the three dimensional frames to the two dimensional screen, preventing linear speedup. Our *permutation warping* [37] approach computes a backwards mapping algorithm with optimal storage and deterministic communication on shared or distributed memory machines.

and implemented. We have developed a superior approach, that does not use general communication, but uses a permutation of communication to solve the warping phase of the algorithm.

TABLE I Terms in algorithm

p	point in original volume space OS
$\vec{N}_p$	normal at point p in OS
$\vec{E}_p$	direction to eye at point p in OS
$\vec{L}_{p\gamma}$	direction to light source at point p
α_p	opacity at point p in OS
I_{Sp}	shading intensity at point p
p'	point in volume screen space SS_{final}
$\alpha_{p'}$	opacity at point p' in SS_{final}
$I_{Sp'}$	resampled shading intensity at point p' in SS_{final}
T	transform $OS \rightarrow SS_{\text{final}}$
$I_{p'_w}$	intensity of ray at point p'_W in 2D image space SS created from intensities, and opacities along the W ray at all points $p'_{w \in W}$.
B_V	bounding hull of volume data
$B_{V'}$	bounding hull of transformed volume data
B_{SS}	bounding hull of 2D screen space image

Previous parallel volume rendering algorithms, described in Section 1.1, have restricted platforms [17][22][23][29], data set sizes [17][21][29][41], filter quality [4][6][17][22][27][28][29][32], view angle freedom, or correctness [7][22][33][34]. Users shouldn't have to sacrifice functionality to achieve higher performance with parallel computers. Our permutation warping solution uses unrestricted viewpoints and filters, with efficient storage, linear run-time speedup, and problem and generation scalability (for generation scalability see [1]). Unrestricted viewpoints are achieved with provable one-to-one communication. For this reason we call our algorithm *permutation warping for parallel volume rendering.* This paper extends our work on parallel image warping [35]. In this paper, we present new decompositions for our permutation assignment, quantified error of competing multipass shears, provide a new virtualization technique that keeps run time constant across view angle, develop the crucial parallel compositing needed for volume rendering, and show empirical results for the full volume rendering algorithm, not just warping, on the MasPar MP-1 and MP-2 [2]. Our algorithm also scales to us-

To briefly summarize the algorithm, the inputs to the algorithm are a scalar valued volume, V, of $S = n^3$ points (or $S = n_1 n_2 n_3$ for that matter), a set of light sources and their positions, Γ, a viewing transform matrix, T, a classification function, used to convert V and derived values to densities, α, and a shading function, which calculates the lighting and illumination effects. The PPS calculates normals $\vec{N}_p$, opacities α_p, and initial shaded intensities I_{Sp}. The VWS transforms the initial shading intensities I_{Sp} and the opacities α_p to the three dimensional screen space by re-sampling. The CS evaluates the view ray line integrals to get the two dimensional screen space pixel intensities, $I_{p'_W}$. The final output is a two dimensional array of pixel values, I_{ray}. For a more in depth discussion of volume rendering see Blinn [3], Kajiya and von Herzen [14], Levoy [18][19], and our survey [36].

```
$I_{\text{ray}}[\ ] \leftarrow$ Transparency_Volume_Render($V, \Gamma, T,$ classify, shading) {
Step 1    PPS    $\vec{N}_p = \text{normal}(V)$
                 $\alpha_p = \text{classify}(V, \vec{N}_p)$    $\forall (p \in B_V)$
                 $I_{Sp} = \text{shading}(\vec{N}_p, \vec{E}_p, \vec{L}_{p\gamma}, \alpha_p, I_\gamma)$
Step 2    VWS    $\alpha_{p'}, I_{Sp'} = T(\alpha_p, I_{Sp})$    $\forall (p' \in B_{V'})$
Step 3    CS     $I_{p'_w} = \int_{p'_1}^{p'_2} t(l) I_S(l) \alpha(l) dl$    $\forall (p_W' \in B_{SS})$
          }
```

FIGURE 2 Data Parallel Volume Rendering Algorithm

FIGURE 16 and FIGURE 17 show an example volume rendering of magnetic resonance angiography images from data collected from a time of flight process that images human's blood flowing through the brain. The zoomed in views show a bifurcation in an artery using two different qualities of filters. FIGURE 18 shows an example of volume rendering of magnetic resonance imaging of the brain tissues and skull.

Volume rendering is a trivial problem for parallelizing if multiple copies of the source volume are stored. But, this solution is very expensive for massive parallelism, as there must be a volume stored for each processor. It is also not clear how to scale to using more processors than there are rays. Other parallel approaches that use approximations to the algorithm, such as multipass shearing have been proposed, and implemented, with a degradation in the image quality, a three times increase in the amount of required storage, and restriction in the scalability. Even brute force techniques, that require general communication for volume redistribution have been tried direction to the eye for several voxels along the view ray, $w = 1, 2 \ldots W$. The function $\text{Shading}(\vec{E}, \vec{N}, \vec{L}_\gamma, \alpha, I_{L\gamma})$ can calculate either surface analogies (flat, Gouraud, and Phong shading [8]), or particle analogies (phase functions [3][14]). Lighting is a complex phenomenon, and there are physically based models and empirical models, used for different effects. Shading classifies densities into visual properties. Directional lighting effects result from the volume normals $\vec{N}_p$ calculated by approximating gradients at all points, p, within the bounding box of the object space (OS) volume, B_V. A normal is a perpendicular vector to a surface, so may be understood as the direction about which light reflects on a mirror, for example.

The second stage of the algorithm (VWS) computes the viewing transformation T. Because voxels are discrete, transformation requires resampling. Spatial resampling is called warping [30]. The OS points p are transformed to screen space (SS) points $p' = T(p)$. Warping and resampling of the opacities and shading intensities creates SS opacities, $\alpha_{p'}$, and intensities, $I_{Sp'}$. The OS data consist of S sample points. The SS data is r rays of W sample points or $S' = rW$ total points in SS. Resampling is used in nearly all raster graphics applications to map geometry and data to discrete pixel or ray positions. In volume rendering, there are literally volumes of resamples to be performed.

The final stage of the algorithm (CS) combines SS intensities and opacities, creating a two dimensional image $I_{\text{ray}}[\ ]$. The intensities along view rays are attenuated by the transparency. FIGURE 1 shows the line path l for a view ray, and summation of transmitted intensities along l is expressed as an integral. The transparency integral is (Step 3) in FIGURE 2, and simplified evaluation is known as image compositing [25].

To understand our expression of the algorithm, some basic geometry review is required. For each algorithm step, points lie in geometric spaces between which transforms are performed: (Step 1) object space (OS) points p, (Step 2) three dimensional screen space (SS) points p', and (Step 3) two dimensional screen space (SS_{final}) points p'_{W}. A point transformed from $OS \to SS \to SS_{\text{final}}$ is transformed as $p \to p' \to p'_{\text{W}}$. The domain of points in each space is defined by bounding hulls B_V, $B_{V'}$, and B_{SS}. Screen spaces are the geometric spaces in which the objects to be rendered to a computer graphics screen are defined, and bounding hulls are the definitions of the limits of these spaces.

1.0 Introduction

Volume rendering is memory and compute bound. Researchers have used parallelism to speedup transparency volume rendering. The goal in parallelism is linear speedup and no storage overhead. Linear speedup is parallel run time t_P on P processors that is a fraction of the best sequential run time t_S, or $t_P = O(t_S/P)$. The goal in storage efficiency is $O(S)$ storage with S sample points, equivalent to the sequential algorithm's storage. To understand how we have parallelized volume rendering, we first introduce the single scattering volume rendering algorithm. FIGURE 1 shows a two dimensional slice of a volume of varying density. Light sources illuminate particles that reflect light to the eye. Assuming low particle reflectivity, the transfer equation is easier to solve because only a single scattering (reflection) occurs [3][14][19].

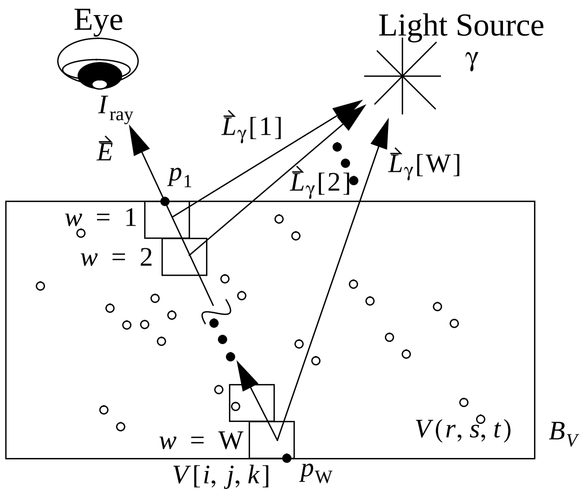

FIGURE 1 Single Level Scattering Particle Model

FIGURE 2 gives the volume rendering algorithm with terms defined in TABLE I. Inputs are sample points V, a set of light sources Γ, a viewing transformation T, a classification function, and a shading function. The algorithm is grouped into three steps: preprocessing stage (PPS), volume warping stage (VWS) and compositing stage (CS).

The first step (Step 1 of FIGURE 2) of the algorithm (PPS) classifies voxels to opacities $\alpha(r, s, t)$, and calculates shading intensities, I_{Sp}. The intensities are attenuated by the opacity. Opacities define how opaque or how much light is blocked in each part of the volume. The shading intensity, or amount of light reflected, depends upon the particle light interaction. FIGURE 1 shows the light source directions $\vec{L}_\gamma[x, y, z]$ used to calculate the illumination bouncing in the $\vec{E}$

Running Head: Permutation Warping for Volume Rendering

Author to whom proof should be sent:

Craig M. Wittenbrink

Hewlett-Packard Laboratories,
1501 Page Mill Road
Palo Alto, CA 94304-1126
craig_wittenbrink@hpl.hp.com
phone: (415) 857 2329
FAX: (415) 852 3791

Abstract

In this paper we present a data parallel volume rendering algorithm with numerous advantages over prior published solutions. Volume rendering is a three-dimensional graphics rendering algorithm that computes views of sampled medical and simulation data, but has been much slower than other graphics algorithms because of the data set sizes and the computational complexity. Our algorithm uses *permutation warping* to achieve linear speedup (run time is $O(S/P)$ for P processors when $P = O(S/\log S)$ for $S = n^3$ samples), linear storage ($O(S)$) for large data sets, arbitrary view directions, and high quality filters. We derived a new processor permutation assignment of five passes (our prior known solution was eight passes), and a new parallel compositing technique that is essential for scaling linearly on machines that have more processors than view rays to process ($P > n^2$). We show a speedup of 15.7 for a 16k processor over a 1k processor MasPar MP-1 (16 is linear) and two frames/second with a 128^3 volume and trilinear view reconstruction. In addition we demonstrate volume sizes of 256^3, constant run time over angles 5 to 75 degrees, filter quality comparisons, and communication congestion of just 19% to 29%.

Accepted to Journal of Parallel and Distributed Computing
c/o Sartaj Sahni, Algorithms Editor
Computer and Information Sciences Department
CSE 301, PO Box 116120
University of Florida, Gainesville, FL 32611

Time and Space Optimal Data Parallel Volume Rendering Using Permutation Warping[1]

Craig M. Wittenbrink
Board of Computer Engineering,
University of California
Santa Cruz, CA 95064
craig@cse.ucsc.edu

Arun K. Somani
Dept. of Computer Science and Engineering
Dept. of Electrical Engineering,
University of Washington
Seattle, WA 98195
somani@cs.washington.edu

UCSC-CRL-96-33

Dec. 31, 1996

Updated Sep 10, 1997

current addresses:

Craig M. Wittenbrink, Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304-1126, craig_wittenbrink@hpl.hp.com, phone: (415) 857 2329, FAX: (415) 852 3791.

Arun K. Somani, 223 Coover Hall, Dept. of Elect. and Comp. Engr., Iowa State University AMES, IA 50011 Tel: (515) 294-0442 Fax: (515) 294-8432 Email: arun@iastate.edu

1. Portions of the text originally appeared in the IEEE/ACM Parallel Rendering Symposium ‘93, San Jose, CA, 1993, pages 57-60, color plate page 110. This Research was supported in part by the NASA Graduate Student Researcher’s Program via NASA Headquarters.