Extensions to Permutation Warping for Parallel Volume Rendering

Craig M. Wittenbrink
Computer Systems Laboratory
HPL-97-116 (R.1)
April, 1998

SIMD, MIMD,
MasPar,
volume visualization,
parallel algorithms

Biomedical volume visualization requires high quality and high performance, but the existing high performance solutions such as the Shear Warp algorithm, 3D texture mapping, and special purpose hardware have problems. Permutation warping achieves high fidelity for biomedical datasets of regular rectilinear volumes, using a one-to-one communication scheme for optimal O(1) communication on massively parallel computers. Extensions are presented including data dependent optimizations using octrees, arbitrary view angles flexibility, and multiple instruction stream multiple data stream (MIMD) implementation. A MasPar MP-2, single instruction stream multiple data stream (SIMD) (16,384 processor), implementation achieves 14 frames/second, using trilinear reconstruction on 128^3 volumes for 400% runtime improvement over our previous result. A Proteus MIMD (32 processor) implementation achieves 1 frame/second on the same data. Additionally the PermWeb software architecture is presented, that has shown as a proof of concept means to provide wide shared access to a powerful centralized rendered. All of these improvements make permutation warping an effective solution for biomedical volume visualization.

Extensions to Permutation Warping for Parallel Volume Rendering

Craig M. Wittenbrink
Hewlett-Packard Laboratories
Palo Alto, CA

April 8, 1998

Abstract

Biomedical volume visualization requires high quality and high performance, but the existing high performance solutions such as the Shear Warp algorithm, 3D texture mapping, and special purpose hardware have problems. Permutation warping achieves high fidelity for biomedical datasets of regular rectilinear volumes, using a one-to-one communication scheme for optimal O(1) communication on massively parallel computers. Extensions are presented including data dependent optimizations using octrees, arbitrary view angle flexibility, and multiple instruction stream multiple data stream (MIMD) implementation. A MasPar MP-2, single instruction stream multiple data stream (SIMD) (16,384 processor), implementation achieves 14 frames/second, using trilinear reconstruction on 128^3 volumes for 400% runtime improvement over our previous result. A Proteus MIMD (32 processor) implementation achieves 1 frame/second on the same data. Additionally the PermWeb software architecture is presented, that has been shown as a proof of concept means to provide wide shared access to a powerful centralized renderer. All of these improvements make permutation warping an effective solution for biomedial volume visualization.

Keywords: SIMD, MIMD, MasPar, volume visualization, parallel algorithms.

1 Introduction

Biomedical rendering demands the highest image quality, and good interactive performance. There are many classes of algorithms and implementations for volume visualization of regular rectilinear datasets. Such datasets include magnetic resonance imaging (MRI), computed tomography (CT), positron emission tomography (PET), and ultrasound. Such data is commonly used for clinical diagnosis and preoperative planning. In the last ten years, a technique for the direct visualization of these datasets–Volume Rendering [2]–has been explored in the computer science and medical communities. Initially developed for the visualization of natural phenomena such as clouds and planetary rings, the algorithms were quickly adapted to process medical volumes.

In medical diagnosis and planning, there has been resistance to using three-dimensional (3D) visualizations. There are questions about the quality of the renderings and the lack of dynamic range on cathode ray tube (CRT) displays. Researchers have downplayed the quality issues and continue to develop new ways in which to visualize volume data interactively. Recently, high-quality repeatable renderings have been defined by specifying a standard pipeline, for example, in the proposed OpenGL extensions for volume rendering [8].

Many avenues of research have proven to be useful to achieve high performance volume visualization. Examples include software acceleration techniques, specialized graphics and volume hardware, and 3D texture mapping architectures. Each of these solutions provides quality performance trade-offs in the rendering, as they are approximations to ray tracing.

Permutation warping is an algorithm for parallel processors that renders efficiently an exact ray tracing algorithm, is scalable to more processors and large volumes, and has simple data parallel or SIMD implementation. I have been researching extensions to permutation warping, including data dependent optimizations for up to a 5 times improvement, view angle flexibility–now with no restrictions for orthogonal views, and MIMD implementation–using large granularity methods. High quality requires higher performance, and to achieve this takes state of the art computers. State of the art computers are costly, but this is not an impediment to wide use of the permutation warping technique. Using centralized servers and distributed access can provide sharing of these techniques and a software architecture, PermWeb, has been designed and implemented for this purpose.

Section 2 describes the quality issues in the in the volume visualization pipeline. Section 3 reviews permutation warping and describes new developments, including data dependent optimizations using octrees, improved view angle flexibility with new matrix decompositions, and MIMD implementation. Section 4, describes the distributed architecture that provides easier access to special purpose renderers thus making them more available.

2 High Quality Rendering

When considered as a data processing pipeline, volume rendering [2] takes any data value input and passes it through the following stages: gradient, classification, shading, resampling, and compositing. With each stage, there are many algorithm design decisions. Also, user control of each stage may be useful when exploring the features of any given dataset. The following describes each stage, and briefly mentions how the existing work performs each stage citing any shortcomings.

Gradient The gradient calculation is the approximation of the partial derivative with respect to all three directions, $\nabla f(x, y, z) = \frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\frac{\partial f}{\partial z}$. Tradeoffs in various gradient calculations have been examined [1]. Gradient calculations are done as an offline process by the Shear Warp algorithm [6], Vizard [5], and 3D texture mapping [17]. EM-Cube [13] uses a skewed neighborhood for a 12-point gradient filter. The gradient is used both to classify the type of data, and to compute computer graphics shading when rendering the data. An accurate gradient operator requries a lot of computation, while simpler approximations are sensitive to noise because differentiation amplifies noise.

Classification A mapping from data values to material values is required in order to determine the proper light interaction. For example, bone has a different reflectivity, opacity, and color than skin. Classification is done as an offline process by Shear Warp, Vizard, and 3D texture mapping. Shear warp provides a partly accelerated classification using a min max octree, and EM-Cube does classification using a lookup table opacity map.

Shading Shading is the calculation of emitted intensity using the gradient orientation and classified material in the volume [7, 11]. Shading is done as a preprocess by Shear Warp, Vizard, and 3D texture mapping. Shear warp provides lookup table shading with octrees, but has degraded performance due to the nonscanline traversal of octrees. EM-Cube uses lookup table shading, and 3D texture mapping can do shading with a lookup table approach, but it requires multiple passes to reload the textures. Shading is complex, and only recently have quality artifacts from Levoy's technique of separate interpolation of color and opacity been discovered, as shown in Figure 1 [10].

Resampling Resampling is required to calculate the samples of opacity and intensity along the view rays. Because any particular view ray will not in general line up with the original dataset grid points, a reconstruction of the discrete dataset to a continuous dataset and a discretization to points along view rays are performed. Shear warping [6] resamples with a bilinear interpolation to shear the volume and then a warp of the skewed projection image using a bilinear interpolation. The EM-Cube architecture [13] performs sheared trilinear resampling for which results vary with view angle, and a warp of the skewed projection image similar to Shear Warp. Vizard uses a table lookup, and a multiply/addition for an approximate trilinear interpolation. 3D texture mapping uses trilinear interpolation, though does not correctly interpolate for perspective projection, because using a set of parallel planes to resample through the volume creates a different ray step size for different pixels. This varying step size incorrectly accumulates opacities that are computed for a fixed step size. Ray tracing typically uses a trilinear reconstruction filter [7].

Figure 2 shows a comparison of multipass and direct resampling, where the trilinear and nearest neighbor or zero order hold both perform better than a multipass linear resampling. A cube and a sphere are resampled, and the error between the sample and their analytically defined values is summed along view rays. Black is mapped to the maximum error, as shown in the color map at the bottom of the figure. Multipass resampling, bottom row, has greater error than trilinear interpolation, top row, for both test shapes. Figure 3 shows the difference between a zero order hold and trilinear reconstruction for rendering a bifurcation of an MRI angiogram.

Compositing Compositing combines the view samples along each view ray by calculating the over operator, given below. The opacity α defines the material presence at each point along the ray. Volume rendering takes into account the partial transparency of material along a view ray. The equations given use associated colors $\tilde{C} = \alpha C$ [10], which can be used for shading when classifying before interpolation.

$$\begin{aligned} \tilde{C}_{new} &= (1 - \alpha_{front})\tilde{C}_{back} + \tilde{C}_{front} \\ \alpha_{new} &= (1 - \alpha_{front})\alpha_{back} + \alpha_{front}. \end{aligned} \tag{1}$$

3 Permutation Warping

An algorithm that addresses the shortcomings, of slow or required preprocessing and inaccurate resampling is permutation warping [22]. A permutation is a one-to-one assignment of processor to processor, which for massive parallelism can be considered as a processor per voxel. I have shown that for spatial assignment in resampling for rendering, such a permutation can be computed, and therefore rendering takes 1 communication step for any equiareal view transform [22]. This holds for regular grids such as biomedical datasets from MRI, CT, and PET. Such an assignment is used to calculate a warping or resampling of the input volume data to data that lie upon view rays. In this section I briefly review the basic permutation warping algorithm for volumetric visualization, and then describe extensions for data dependent acceleration, view angle flexibility, and MIMD implementation.

Permutation warping calculates a 2D output image, by processing a 3D volume. 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, gradient, classification, and shading selections. The PPS calculates gradients, opacities, and intensities. The VWS transforms the intensities and the opacities to the 3D screen space by resampling. The CS evaluates the compositing to the 2D output image. Figure 4 shows pseudo-code for the permutation warping algorithm for no virtualization or for data parallel implementation.

Figure 5 illustrates the transforms calculated by processors. The object space and screen space are separated into the object space on the left and the screen space on the right. Yellow processors on the left are communicating with yellow processors on the right to compute the resampled 3D screen space. On the left and right the 3D screen space volume is red, and the 3D object space volume is black. Blue processors, for this slight rotation, do not perform any communication, but compute and store the result locally. Red processors are clipped outside of the 3D screen space, so do not interpolate at all. A processor does permutation warping by (See Figure 4): 2.1) Calculating processor assignments; 2.2) Calculating the reconstruction point; 2.3) Resampling after reading the values of the neighboring processors; And, 2.4) Sending resampled values to screen processors. In Figure 4, Step 3, a parallel compositing combines resampled intensities and opacities. Binary tree combining computes compositing as shown in our work [19, 22] and the related work by Ma et al. [9].

Our prior algorithmic studies of permutation warping have shown that it is asymptotically time and space optimal for resampling on the EREW PRAM (exclusive read exclusive write parallel random access machine) [19, 22, 23]. But, several enhancements are possible such as load balancing and data dependent acceleration.

Quality Comparison Gradient calculations are performed on the original sample points, and near-neighbor communication is used for high efficiency. Because a general purpose parallel processor is used, gradient calculations do not need to be part of an offline process as in Shear Warp, Vizard, and 3D texture mapping. A general purpose parallel processor has sufficient performance to compute gradients as needed. The gradient is not skewed as required for EM-Cube, because of the object space partitioning. Classification and shading can be done using a general function or through a lookup table.

And to save on compute time, permutation warping can also use preclassified/shaded values. Resampling is done in a one-pass resampling, using a local area support reconstruction filter. A one pass filter avoids multiple aliasing, but a multipass filter both distorts and aliases with each pass as shown in Figure 2. This one-pass resampling is an improvement over the two-pass resamplings used by Shear Warp and EM-Cube. The resampling is also an accurate reconstruction, and not an approximation as used by Vizard. 3D texture mapping does accurate trilinear resampling, but suffers from the shortcomings discussed in Section 2 where gradient, classification, and shading are done as an offline preprocess. If shading, classification, and gradients are assumed to be computed offline, then there are not sufficient computational resources to compute them interactively.

Related Parallel Work There are a large number of volume rendering approaches that have been tried, and because of the many variants, there is a large difference between algorithms. Accurate direct comparison of approaches is difficult, so I discuss some of the main techniques, and also discuss more closely related work done on the MasPar. Parallel volume rendering algorithms may be grouped into four categories as determined by their viewing transforms: backwards, multipass forwards, forwards splatting, and forwards wavefront. Ma et al. [9], Neumann [12], Goel and Mukherjee (G^*) [3], and Hsu (Hz) [4] have developed backwards (ray tracing) volume rendering algorithms. 3D texture mapping using Silicon Graphics' architectures [17] is also a backwards view transformation approach. Lacroute [6], Osborne et al. [13], and Vezina et al. (Vf, Vz) [18] have developed multipass forwards algorithms. Neumann [12] has developed a forwards splatting algorithm, and Schröder and Stoll [15] have developed a forwards wavefront algorithm. The Lacroute and Levoy Shear Warp algorithm [6] reduces the amount of work by an order of magnitude over octree encoding [7]. Culling of the volume, compressed data structures, and adaptive termination speed up the algorithm. Parallel versions [6] use screen space parallelization. But, their studies show that the possible speedup to higher numbers of processors is limited because of this screen space decomposition. A speedup of 12.5 is achieved on 32 processors over one processor, where 32 would be linear [6]. Figure 7 shows that permutation warping has better scaling properties. SIMD implementations [23, 22] achieve linear scalability on a 1K processor MP-1 to a 16K processor MP-1, and it is also possible to use similar culling and compression to further improve efficiencies. In direct comparison to other MasPar implementations of volume rendering [3, 4, 18], permutation warping [23, 22] has also been proven to be more scalable, Figure 7. Permutation warping has also demonstrated the highest performance achieved on the MasPar. Figure 6 shows that permutation warping on a 16,384 processor MasPar MP-1 achieves 10.1 Mvoxels/second performance versus Hsu's 8.8 Mvoxels/second (Hz). Further improvements have been achieved using octree encoding which is discussed in Section 3.1.

3.1 Data Dependent Optimizations

As presented in [19, 22, 23], permutation warping does not use data dependent acceleration methods. This means that the worst case performance is the same as the best case performance. All datasets will be rendered with the same runtime, irrespective of the density of the volume, classification function, or distribution of nonzero and heterogeneous voxels. The methods used to accelerate volume rendering using data dependencies include: adaptive ray termination [7], skipping of empty regions, leaping through homogeneous regions, and load balancing the varying workload.

We have implemented octree volume subdivision within the permutation warping algorithm [20]. The octree encoding is computed for each subvolume of the object space subdivision. The octree encodings investigated include the use of thresholding to cull regions of the volume. To demonstrate the encoding, consider the example in Figure 8. Processors are shown in a 2D layout. Regions are quadtree-encoded at each processor, PE0 to PE3. PE0 has a heterogeneous volume, resulting in a tree of depth two, with 13 nodes. PE1 and PE2 have empty space, and their quadtrees have a single null node. PE3 has a homogeneous region, and its quadtree is a single homogeneous node.

Octrees can be used to further compress the data by using thresholding and by considering all voxels below a given threshold to be effectively zero. This provides some immunity against noise in the creation of empty regions in the octrees, but does alter the output visualization. Considering Figure 8, the values in PE1 and PE2 may have been non-zero, but were below threshold when the octrees were created.

Using thresholding and the additional precalculated static load balancing of the data, we achieved a 260% to 404% improvement in runtime over the baseline permutation warping algorithm as presented in [22]. Table 1 shows the run times with the publicly available brain data set at two resolutions, 128^3 and 256^3. The times shown are captured using the cycle count facility in the MasPar MP-2 instruction set, and calculated as the average of multiple runs, taken at different view angles. The runtime for different view angles is the same. The MasPar MP-2 is a single instruction stream multiple data stream (SIMD) machine with 1K to 16K processors that have mesh and general router interconnections. Experiments presented were run on 16,384 and 4096 processor machines. The speedups were calculated against the baseline execution times of .282 and 2.199 seconds, respectively [20]. The majority of the processing takes place in the rotation or resampling phase of the algorithm, as shown in column three of Table 1. Figure 10 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.

octree	vol. size	rotation time	compositing time	avg. total time	speedup
yes	128^3	65	13	78	3.60
yes	256^3	368	68	436	5.04
no	128^3	269	13	282	-
no	256^3	2131	68	2199	-

Table 1: Data dependent run times trilinear reconstruction (milliseconds) on 16,384 processor MasPar MP-2, octree threshold 50, and speedup of octree algorithm over nonoctree permutation warping

Using thresholding to eliminate nearly zero valued voxels, and creating an octree creates approximations that degrade the image quality. Figure 11 shows images rendered with different thresholds, 0, 1, 5, 10, 50 that are considered to be empty voxel space. Table 3.1 shows the RMS errors, run times, and percentage of the run time over the baseline for the octree enhancements. Higher thresholds results in higher errors, but there is also a corresponding improvement in runtime. There are noticeable differences for very high thresholds, and the best quality is available by using no threshold. The performance improvements of 260% to 404%, Table 1 are impressive enough to warrant the tradeoff in image quality, especially when considered in the context of an interactive system. Even more improvements in speed and quality are possible by tuning the algorithm to use adaptive coding, compression, and adaptive refinement where the highest quality is selected when the user stops changing the viewpoint. With data dependent optimizations the scalability for greater numbers of processors is no longer linear, but is nearly linear. A 16,384 processor MP-2 achieves a speedup of 3.6 over a 4096 processor MP-2 when using the octree enhancements discussed. 4.0 would be linear, and the baseline permutation warp algorithm achieves a speedup of 3.9.

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: Quality performance tradeoffs, Runtimes (milliseconds) RMS error (of images as shown in Fig. 11, and percentage of the baseline permutation warping algorithm on a 16K processor MP-2.

3.2 View Angle Flexibility

In interactive volume visualization, one of the key advantages comes from changing the viewpoint to help explore the 3D nature of the data. What this requires is specifying arbitrary viewpoints. Many volume visualization algorithms have achieved tremendous speedups by limiting the possible viewpoints, as was done by Schröder and Stoll [15]. Restricting the viewpoints limits the types of communication that can occur. This guarantees high performance for certain view angles and/or architectures.

In permutation warping, there is no angular restriction nor performance penalty for views. The view transform, T, can be any equiareal transformation or any perspective transformation, if a two-pass warp is allowed. In 2D, an equiareal transformation is simply a two-by-two matrix that has a determinant equal to 1. Any rotation, translation, or shear of the volume under view may be performed.

When specifying this matrix as a rotation instead of as an arbitrary matrix, some numerical instabilities may arise. In earlier work, we got around this by limiting the rotation to a certain range. We essentially allowed rotation angles of 0 to 85 degrees. For example, to rotate by an angle θ, you specify the matrix:

$$T = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \tag{2}$$

When the permutation decomposition is calculated, several matrix values are $\tan\theta/2$.

$$T = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = \begin{vmatrix} 1 & -\tan\theta/2 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 \\ \sin\theta & 1 \end{vmatrix} \begin{vmatrix} 1 & -\tan\theta/2 \\ 0 & 1 \end{vmatrix} \tag{3}$$

You can avoid numerical problems in the tan function, by changing the angle interval. Figure 12 shows the separation of the rotation angle domain into two different tangent functions. The solid line shows $\tan\theta/2$, and this function goes towards positive and negative infinity at $+-\pi, +-3\pi, ...$, etc. By incorporating a 180-degree rotation and back rotation into the permutation decomposition, it is possible to always stay within a stable tangent evaluation. So, if the permutation decomposition were to be unstable, $p' = Mp$, then simply use

$$p' = M_R \begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix} p \tag{4}$$

where M_R is matrix M with θ substituted by $\theta - \pi$ or a 180 degree rotation.

Figure 12 shows the interval in which the angle theta blows up. The function $\tan(\theta - \pi)$ is stable in this region. Figure 12 shows the translated interval as a dashed line, and illustrates that the two functions have outputs between $+-1$. This is directly extended to 3D rotations, doing a conditional test for each angle of rotation to determine which interval to compute.

This new angle of rotation specification has been verified in both 2D and 3D permutation warping examples. Figure 13 shows the implementation, assuming θ lies in the 0 to 2π interval:

The change in the view angle specification provides view angle flexibility with no numerical dangers, constant communication complexity, and minimal performance impact to check the interval of angle. This is an important clarification for the permutation warping decomposition rotation angle specification. Permutation warping does not require transposing of the data or storing of multiple volumes to accommodate different viewpoints efficiently. With permutation warping, only one copy of the volume is stored, and the efficiency to render is the same no matter the visualization angle. Shear warping requires three copies of the volume or transposing of the data prior to resampling for scanline efficiency. This was the goal of the research: greater view flexibility with no quality or performance penalties.

3.3 MIMD implementation

As presented in [19, 22, 23], permutation warping is a general parallel algorithm, but has been evaluated only on single instruction stream multiple data stream (SIMD) machines. It is unclear whether permutation warping would be as efficient for other types of parallel architectures such as MIMD. SIMD and multiple instruction stream multiple data stream (MIMD) provide somewhat different challenges for implementation. In order to determine the effect upon the permutation warping approach for volume visualization, an experiment was performed to implement permutation warping on a general purpose distributed memory MIMD computer [19]. This section discusses the performance attained on the Proteus Supercomputer [16] and how to maintain full parallelism on scalable MIMD machines.

In data parallel machines such as the MasPar, the preferred solution is a one-step warping phase to align the volume to the view direction [22], but in high granularity machines such as Proteus [16], small messages create network congestion. Instead, an algorithm was developed that takes advantage of the high data to processor ratio and sends large messages.

To send large messages, processing occurs at a large granularity level. Figure 16 shows the source volume considered as a collection of subvolumes assigned to processors. The subvolumes are rendered in parallel to create subframes. And the subframes are composited to create a final image. Also, for MIMD implementation, it is essential to exploit the separate instruction streams. Specifically this algorithm has the same three steps as given in Figure 4 (PPS, VWS, and CS), but now the VWS and CS are intermixed. Figure 15 shows the processors warp, VWS, and composite, CS (partial).

The processors use the local data to create subframes, as given in Step 2.1. In Step 2.2, each processor then sends sections of its subframe to the output screen space subframe locations that are aligned. Aligned subframes are then combined through further compositing in Step 3. Subframes are composited in parallel using a new parallel product that doubles screen area as screen depths are combined. This approach provides full parallelism, and the end result is completely distributed across processors.

Figure 16 shows the MIMD algorithm with resampling and product calculations. Voxels are viewed from an arbitrary viewpoint (View) in the lower left. For each pixel, resampling occurs along the rays, then the samples are combined by compositing. Figure 16 also shows the parallel data flow and partitioning from Steps 1 through 3 for 16 processors or clusters. Each processor preprocesses its locally assigned subvolume in Step 1. The subvolume is assigned through a static object space partitioning of the volume. Figure 16 shows how subframes are created locally for new view directions. The subframe is sent to the aligned screen space, Step 2.2. Note that the subframe may be sent to several screen space processors. Four processors in the view direction result in two parallel product composites to calculate the final image.

Figure 16 gives a simplified view of the algorithm, but reveals, nevertheless, the full parallelism and data layout(s). Compositing is more efficient in the aligned dimensions of the machine, the screen space. Due to the physical interconnections used in MIMD machines, preferred communication patterns exist, such as in a mesh where communication along rows or columns is very efficient. To communicate subframes to their aligned positions, multiple rounds of communication are used. Permutation warping for four processors requires three rounds; in the first round, processor 0 talks to 1, 1 to 2, 2 to 3, and 3 to 0, etc. Each processor calculates a subframe, and then communicates to $P-1$ processors. The actual data sent in each phase changes with the viewing transform.

Eight processors require seven rounds of communication. This provides a constant number of communication periods with no conflicts. Each processor overlaps at most 18 screen space subvolumes because of the length of the diagonal of a subvolume in Step 2.2. Then, at most 18 messages are sent. The length of a diagonal through a subvolume defines the extent through which a subvolume may interesect a rigidly translated and rotated 3D screen space subvolume assignment. If a shared memory machine is used, processors write their results into the appropriate output buffers without conflict and with less congestion, because they will get cache ownership of the lines with which they are working. The aligned screen space is similar to the object space assignment. The object space coordinates of the screen space change with the view transform. Each processor sends the aligned samples they create to the processors that need them in Step 2.2.

The permutation *sends*, Step 2.2, can be overlapped with calculation, Steps 2.1 and 3, to hide the communication costs. Figure 16 also shows parallel compositing, Step 3. In any view direction there are remaining composites along each view ray, because the object space is carved up uniformly. In order to avoid idling of processors, every processor remains busy through this stage by binary tree compositing the frames. In

Volume Size	32 PE's Tril	32 PE's Zoh	1 PE tril	1 PE Zoh
32^3	161	150	241	97
64^3	291	203	1760	554
128^3	1046	498	13846	3870
256^3	4316	1411	95064	24523

Table 3: Proteus run times, all output images are 256x256, milliseconds

previous ray tracing techniques, processors are idled when there are more processors than the number of rays. Our parallel product maintains linear speedup until the compositing finally dominates, an improvement over image space partitioning and ray tracing methods.

The key aspects of the MIMD algorithm are: only local source data is used–no concurrent reads are necessary; the communication requirements are the same, irrespective of the viewpoint; the communication has no conflicts; the algorithm scales input size and number of processors; and, the filter quality is as high as desired.

Experiments were performed on the Proteus Computer to evaluate the MIMD version of the permutation warp algorithm. Figure 14 shows Proteus, a scalable MIMD parallel computer [16] with 32 to 1024 processors. A prototype group, or eight clusters, with 32 Intel i860's has been implemented. Figure 14 shows the physical layout of 32 processors with clusters labelled $C0$ to $C7$. The interconnection network is a bit-serial crossbar with single link transfer rates of 250 Mbits/second. The total memory capacity of the 32 processor system is 64 Megabytes (eight Mbytes per cluster).)

Performance experiments used two different reconstruction filters, a first-order hold (trilinear interpolation) and a zero-order hold (nearest neighbor). Refer to Figure 3 for examples and our work [22] that compares zero-order holds, first-order holds, and multipass shearing. Images created were eight bits/pixel. All measurements were taken using multiple runs of the code, and averaging their execution time. Table 3 shows the Proteus volume rendering algorithm's runtime versus volume size. The output image is 256^2 for all volume sizes. Speedup is given in Table 4. Proteus provides a speedup of 22 for 32 processors over a single processor using trilinear resampling. More speedup is achieved for larger volume sizes because of increased cache and communication efficiencies, that hide fixed startup latencies.

The performance of the Proteus implementation is two frames/second for 128^3 volumes and 0.7 frames/second for 256^3 volumes. Volumes up to 512x512x128 byte voxels were processed. The voxel/second rate is 12 Mvoxels/second for zoh reconstruction filter and 4 Mvoxels/second for a trilinear reconstruction. Key implementation features were the use of scanline processing in both resampling and compositing for cache efficiency, and optimized memory to memory framebuffer transfer for host interaction. The implementation also showed that large granularity messages were required, and that the parallelization strategy worked for MIMD architectures.

Volume Size	Trilinear	Zoh
64^3	6.05	2.73
128^3	13.24	7.80
256^3	22.03	17.38

Table 4: Proteus speedup for 32 processors

4 PermWeb Software Architecture

Because high fidelity with large data at interactive rates has not been achieved by any hardware, the initial solutions will use high performance hardware. Such high performance hardware is of high cost, but this does not mean it cannot be widely available. The most cost effective solution is to have a powerful centralized compute resource that solves the problem well, and is remotely shareable. In other words, supporting biomedical visualization with parallel rendering requires wide access. One means for easily accessing a parallel rendering system is through the use of World Wide Web (WWW) front-ends.

Figure 17 shows an overview of the software architecture of our PermWeb system [21] URL *http://www.cse.ucsc.edu/research/slvg/mp-render.html*. The system design makes use of as many off-the-shelf WWW components as possible, and was implemented as a proof of concept for remote parallel rendering access. This system has been interfaced to the MasPar, and also to workstation servers. The software architecture uses processes shown as ellipses, data files shown as cylinders, and custom source code as rectangles. The four main processes are the *Web Server*, *Render Request*, *Render Server*, and *Child Render Work*. Only the last three are built from custom software, and many components of the programs are built from freeware.

The custom software developed is also indicated by the source files shown in Figure 17. The rectangular source files include: render.html, tcpclient.c, tcpserv.c, myppmtogif.c, and render.c. The dashed lines show the correlation of source files to programs. The front-end, because of the use of standard WWW browsers, is coded in hypertext markup language (HTML). This language specifies a hypertext document, and has embedded images. Figure 18 shows the appearance of a front-end.

The tcpclient.c program is invoked as a common gateway interface (CGI) program, so it may run on a different platform than the web server that provides access to the *render.html* forms interface. The processes *Render Server* and *Child Render Work* are from the same program that runs continually, waiting for requests to create a volume rendering. The volume data is in the data file *bone_volume.vol*, which is stored in memory for fast access by the rendering server. The results are always compressed before being sent back to the client (*myppmtogif.c*), and the server requires a volume renderer (*render.c*).

A brief explanation of PermWeb's operation highlights features of the software architecture. Refer to the numbered circles. In (1) a user makes a request via the WWW to the *Web Server*. The *Web Server* returns the *render.html* front-end forms interface. In (2) the user selects the desired parameters and chooses a button "render", which invokes the CGI process. Typically, CGI processes are found in the cgi-bin directory, and may be referred to as cgi-bin scripts. In (3), the cgi-bin script contacts the *Render Server*, which (4) causes the *Render Server* to fork off *Child Render Work*. In this way, a single server handles requests from many users and controls how many connections are accepted. In (5), *Child Render Work* accesses the source volume, *bone_volume.vol*, renders it into an image, compresses the image to the GIF format, (6) sends the result through the socket to *Render Request*, and exits.

Render Request upon receipt of the image, in (7) writes it to a file, *result.gif*, and in (8) returns the appropriate uniform resource locator (URL) to the requesting client. The page returned to the client includes the URL for the image, and is typically in the form of a WWW page with the cgi-bin script as the page: http://WebServer2/cgi-bin/Render Request/?params. *WebServer2* is indicated if in fact a different server is used than for the initial request. The parameters indicate the location where the actual viewing parameters are passed to the server. The system is simple, efficient, and uses WWW technology including front-end clients, servers, protocols, to provide access to rendered information. The advantage of the system is a small amount of development code for general and flexible use. One of the key components to achieve acceptable performance is the use of Lempel-Zif compression of the result image. In addition to the HTML front-end, Java has been investigated as a means to provide more interactivity to users.

The current performance of the MasPar access is on the order of many seconds. Depending on the loading of the MasPar, which is a queued system, there may be a delay to get in the execute queue. The file read time of the source volume dominates the total execution time, which could be fixed by using either parallel I/O or memory caching of the volume using a continually running server. The current performance of the shear warp is typically a few seconds. The precompressed, run-length encoded, source volumes used by shear warp help in reducing the file read latency. The anticipated performance, given full optimization of the system software, would be 5-10 frames per second. To get better performance may require a web browser plug-in for a more direct connection to the rendering server, bypassing the multiple step socket paths used with the current system.

5 Conclusions and Future Research

I have shown that for high quality rendering the resampling process is crucial for accurate data reconstruction. One way to do very high quality resampling is by using a one-pass resampling, and permutation warping does a one-pass resampling. This provides a quality improvement over other approaches, including Shear Warp, EM-Cube sheared interpolation, Vizard, and 3D texture mapping. New results were presented in making the view flexibility arbitrary when specifying the view with angles. A new decomposition was used to avoid numerical instability in the tan() function used in the angle processor communication assignments. New results were described in data dependent acceleration techniques, that allowed for a 260% to 400% improvement in runtime over not using data dependent acceleration on a MasPar MP-2. The MasPar MP-2 can achieve (with 16,384 processors) 14 frames/second on 128^3 volumes using trilinear interpolation. Octrees were used to take advantage of empty space and data coherency, but required load balancing to achieve high efficiency. The octree encoding allowed an object space partitioning to achieve higher efficiencies similar to the Shear Warp algorithm, while maintaining scalability and quality advantages. High quality rendering is important for biomedical visualization, and permutation warping quality was demonstrated.

Additional research was performed to investigate the applicability of permutation warping to MIMD computing. Large granularity communication is more efficient on such an architecture. An alternative multiple permutation communication was implemented on the Proteus distributed memory MIMD computer. Proteus can achieve (with 32 processors) computation rates of one frame/second (12 Mvoxels/second) on 128^3 volumes. Results showed that volume rendering is efficiently parallelizeable using object space partitioning, parallel product, and extended permutation warping for volume rendering on scalable MIMD architectures.

A performance of 39 Mvoxels/second has been achieved on a 16,384 processor MasPar MP-2, but other researchers have achieved higher rates on current MIMD machines. Lacroute achieved 221 Mvoxels/second on the same MRI brain dataset that we used for our studies, and Palmer et al. [14] achieved 3.5 Billion voxels/second on the visible female dataset. Permutation warping does achieve better scalability: 3.6 (versus ideal 4) of a 16,384 processor versus a 4096 processor MasPar MP-2 using octree encodings; and a 22 (versus ideal 32) for a 32 processor Proteus. Lacroute's speedup is 12.5 (versus ideal 32). The results are not directly comparable, because the machines used do not have comparable performance, and were released in different years. I have shown evidence that octree-permutation warping is scalable, and of the highest performance of published results on the MasPar, but further work needs to be done to directly compare the approach to Shear Warp on the same computer.

I have detailed the PermWeb software architecture for remote volume rendering using many standardized WWW components, and a few custom components. The key to the performance of the system is image compression, and in-memory rendering and transfer. Performance issues were investigated to reduce the latency of the rendering and improve the system. Such an implementation is a proof of concept that a powerful centralized server can be widely accessed. A high performance, high quality, volume rendering system such as octree-permutation warping accessed via PermWeb is a good solution for biomedical applications.

Future work is researching the accuracy of the lighting model for medical visualization. Physically based lighting models are being investigated, which are more accurate than those derived for atmospheric phenomena. Additional topics are development of coupled compression renderers, parallel compression algorithms, heterogeneous platform parallel rendering, and further data dependent optimized parallel rendering.

6 Acknowledgements

I would like to thank Michael Harrington, Karin Hollerbach, Nelson Max, Kwansik Kim, Jeremy Story, and Andrew Macginitie. I would also like to thank Claudio Silva, Barthold Lichtenbelt, and several anonymous reviewers for greatly improving the manuscript. This project has been partially funded by Lawrence Livermore National Labs, grant ISCR B291836, the Navy Coastal Systems Center developmental grant, the NASA Graduate Student Researcher's Program, the Applied Physics Laboratories-University of Washington, and the Office of Naval Research REINAS. Thanks to NASA-Goddard for providing access to a 16K processor MasPar MP-2. Portions of this paper were published as [19, 20, 21].

References

[1] Mark J. Bentum, Barthold B.A. Lichtenbelt, and Thomas Malzbender. Frequency analysis of gradient estimators in volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):242–254, September 1996.

[2] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Computer Graphics*, pages 65–74, August 1988.

[3] Vineet Goel and Amar Mukherjee. An optimal parallel algorithm for volume ray casting. *The Visual Computer*, 12(1):26–39, 1996. short versions appear in IPPS'95 and Fifth Symp. on Frontiers of Massively Parallel Computation'94.

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

[5] Gunter Knittel and Wolfgang Strasser. VIZARD: Visualization accelerator for realtime display. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 139–146, Los Angeles, CA, August 1997. ACM.

[6] Phillippe Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, September 1996. a short version of this paper appears in the 1995 Parallel Rendering Symposium.

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

[8] Barthold Lichtenbelt. Design of a high performance volume visualization system. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 111–120, Los Angeles, CA, August 1997. ACM.

[9] K. L. Ma, J. Painter, C. D. Hansen, and M. F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(4):59–67, 1994.

[10] Thomas Malzbender, Craig M. Wittenbrink, and Michael E. Goss. Opacity-weighted color interpolation for volume sampling. Technical Report HPL-TR-97-31, H.P. Laboratories, April 1997.

[11] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

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

[13] Randy Osborne et al. EM-cube: An architecture for low-cost real-time volume rendering. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 131–138, Los Angeles, CA, August 1997. ACM.

[14] Michael E. Palmer, Stephen Taylor, and Brian Totty. Exploiting deep parallel memory hierarchies for ray casting volume rendering. In *Proceedings of the Parallel Rendering Symposium*, pages 15–22, Phoenix, AZ, October 1997. IEEE.

[15] P. Schröder and G. Stoll. Data parallel volume rendering as line drawing. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 25–32, Boston, MA, October 1992.

[16] A. K. Somani et al. Proteus system architecture and organization. In *Fifth International Parallel Processing Symposium*, pages 276–284, April 1991.

[17] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via 3D textures. In *ACM/IEEE Symposium on Volume Visualization*, pages 23–30, San Francisco, CA, October 1996. See also technical report UCSC-CRL-96-16.

[18] G. Vezina, P. A. Fletcher, and P. K. Robertson. Volume rendering on the MasPar MP-1. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 3–8, October 1992.

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

[20] Craig M. Wittenbrink and Kwansik Kim. Data dependent optimizations for permutation volume rendering. In *Proceedings of IS&T/SPIE Visual Data Exploration and Analysis V*, page In press, San Jose, CA, January 1998. SPIE. Available as Hewlett-Packard Laboratories Technical Report, HPL-97-59-R1.

[21] Craig M. Wittenbrink, Kwansik Kim, Jeremy Story, Alex T. Pang, Karin Hollerbach, and Nelson Max. A system for remote parallel and distributed volume visualization. In *In Proceedings of IS&T/SPIE Visual Data Exploration and Analysis IV*, pages 100–110, San Jose, CA, February 1997. SPIE. Available as Hewlett-Packard Laboratories Technical Report, HPL-97-34.

[22] Craig M. Wittenbrink and A. K. Somani. Time and space optimal parallel volume rendering using permutation warping. *Journal of Parallel and Distributed Computing*, 46(2):148–164, November 1997. Available as Technical Report, Univ.

of California, Santa Cruz, UCSC-CRL-96-33. Portions appeared as C.M. Wittenbrink and A.K. Somani. Permutation Warping for Data Parallel Volume Rendering, in Proceedings of the Parallel Rendering Symposium, pages 57–60, Oct. 1993.

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

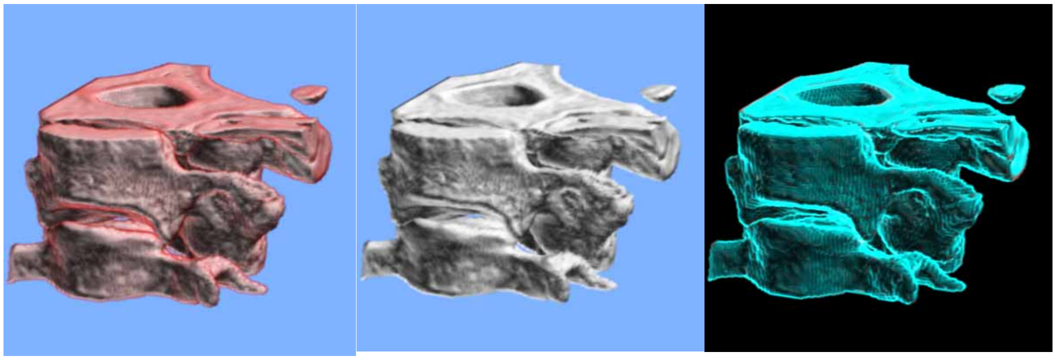

Figure 1: Human vertebra rendered using raycasting with preclassification. Left: separate interpolation of color and opacity. Middle: Opacity weighted interpolation of colors. Right: normalized difference image.

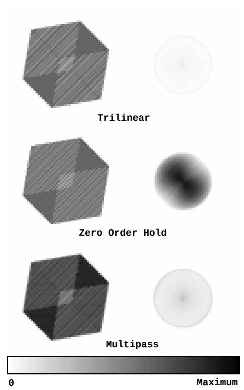

Figure 2: Error in volume resampling.

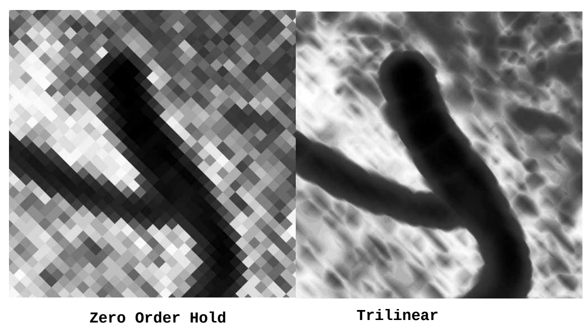

Figure 3: Zero-order hold compared to trilinear reconstruction.

```
1) PPS, preprocessing stage, gradient, classification, shading
2) 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) CS, compositing stage, composite to 2D output image
```

Figure 4: Permutation warping data parallel algorithm pseudo-code

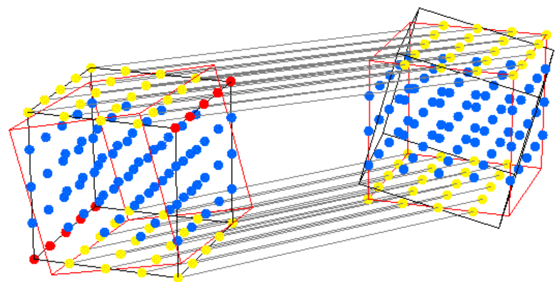

Figure 5: Volume transforms in parallel.

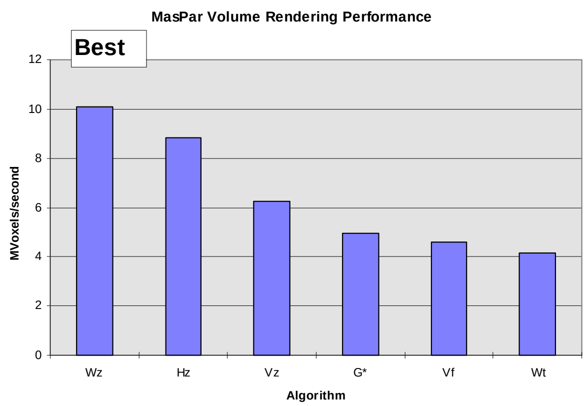

Figure 6: Volume rendering algorithm's performance on a 16k MasPar MP-1, rendering a 128^3 volume.

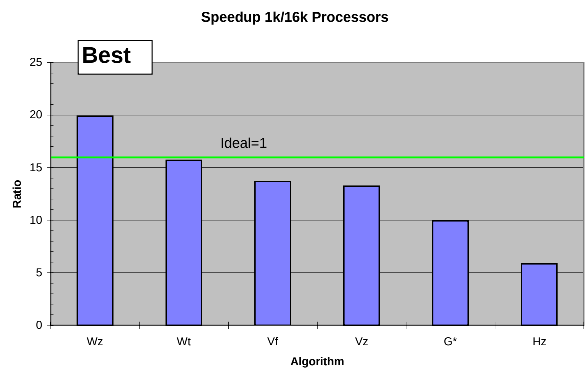

Figure 7: Volume rendering algorithm's speedup, of a 16k to a 1k processor MasPar MP-1.

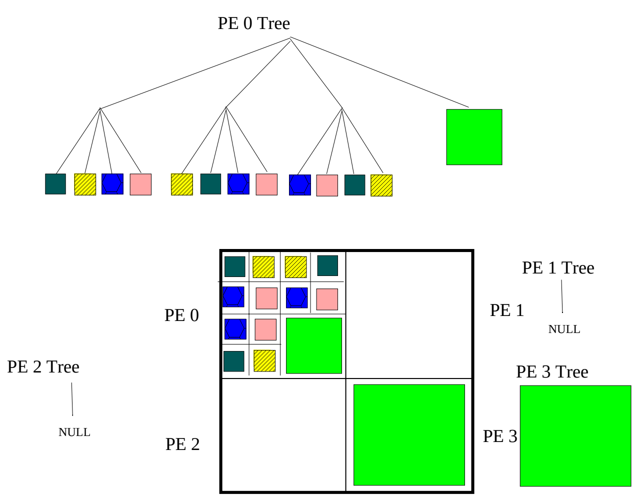

Figure 8: Octrees (quadtree) in 4 processor machine.

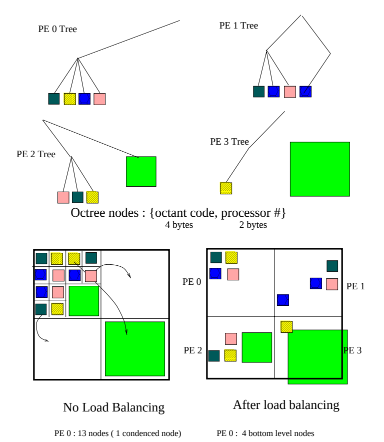

Figure 9: Load balanced octrees.

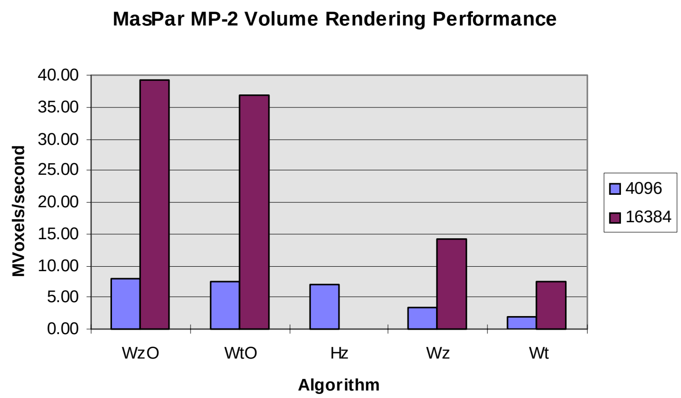

Figure 10: Octree encoded permutation warping performance (WzO and WtO) in Mvoxels/second versus the baseline permutation warping algorithm (Wz and Wt) and Hsu (Hz).

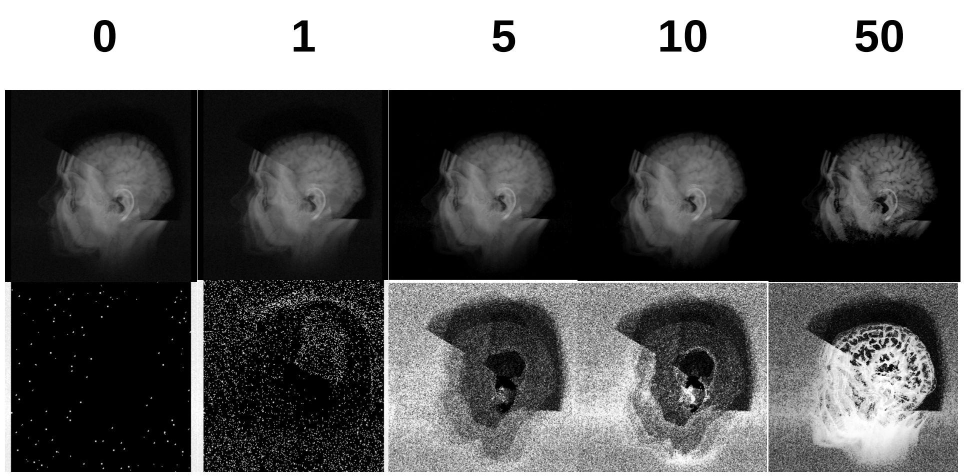

Figure 11: Outputs from octree accelerated algorithm with different thresholds from the left 0, 1, 5, 10, 50.

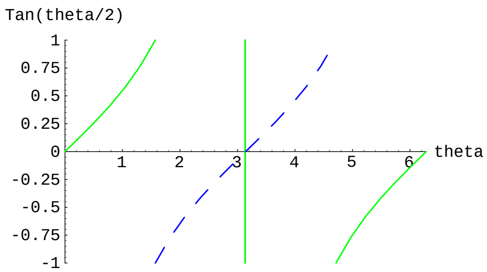

Figure 12: $\tan(\theta/2)$ and $\tan(\frac{\theta-\pi}{2})$ plotted versus angle.

```
if (t > pi/2 && t < 3 pi / 2 ) { /* translated interval */
       s = t-pi      /* 180 degree rotation */
       x = -x        /* rotation by reflecting coordinates */
       y = -y
       p.x = rint(shear(x,y,-tan(s/2.0))); /* round and shear */
       p.y = rint(shear(y,p.x,sin(s)));
       p.z = rint(shear(p.x,p.y,-tan(s/2.0)));
    }
    else { /* simply use tan(theta/2) */
       p.x = rint(shear(x,y,-tan(t/2.0)));
       p.y = rint(shear(y,p.x,sin(t)));
       p.z = rint(shear(p.x,p.y,-tan(t/2.0)));
    }
```

Figure 13: View angle interval translation pseudo-code.

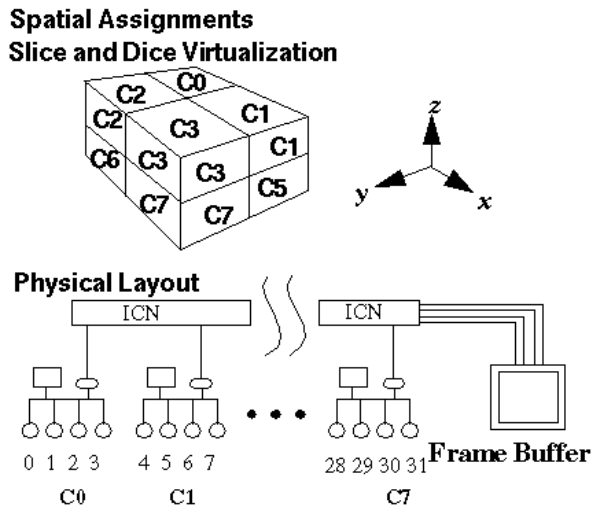

Figure 14: Volume virtualization for proteus

```
1) Preprocessing stage opacities and colors calculated
2) VWS/CS(partial), volume warping stage, processors in parallel:
      2.1) Warp and composite local data, subframes calculated
      2.2) Permutation sends of object space subframes to screen space
3) CS(cont.) Parallel compositing of subframes
```

Figure 15: Permutation warping MIMD-algorithm pseudo-code

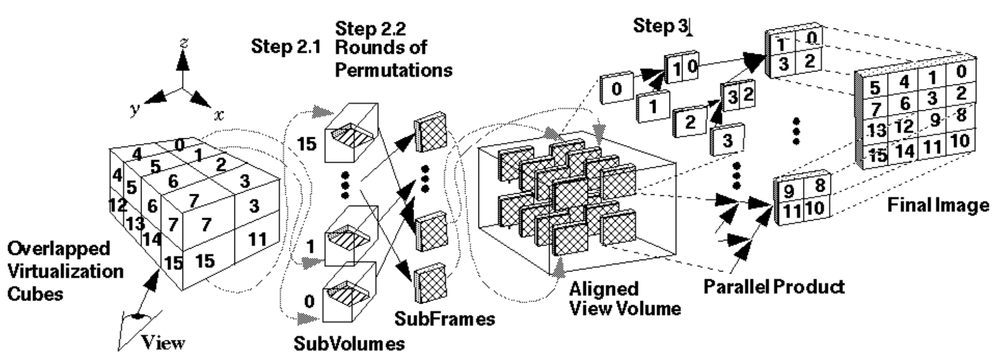

Figure 16: Parallel algorithm flow: object space to resampled view volume to final image

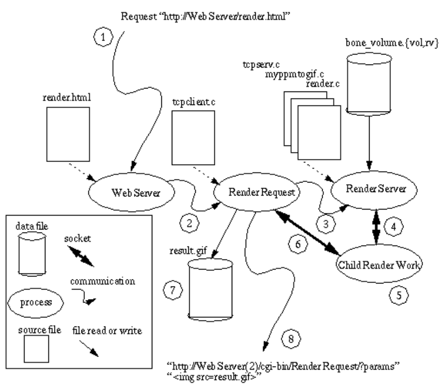

Figure 17: PermWeb architecture

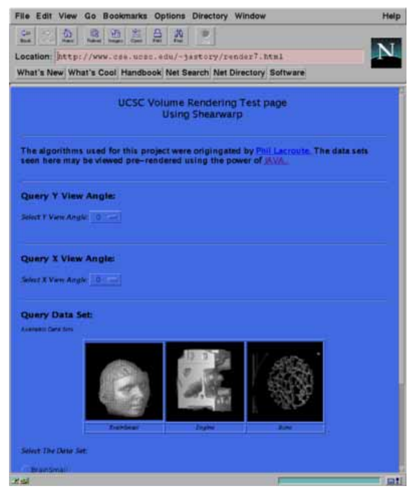

Figure 18: PermWeb HTML front-end.