Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)
D. Bartz, X. Pueyo, E. Reinhard (Editors)

An Interleaved Parallel Volume Renderer With PC-clusters

Antonio Garcia [1] and Han-Wei Shen [1]

[1]Department of Computer and Information Sciences, The Ohio State University, Columbus, Ohio, USA

Abstract

Parallel Volume Rendering has been realized using various load distribution methods that subdivide either the screen, called image-space partitioning, or the volume dataset, called object-space partitioning. The major advantages of image-space partitioing are load balancing and low communication overhead, but processors require access to the full volume in order to render the volume with arbitrary views without frequent data redistributions. Subdividing the volume, on the other hand, provides storage scalability as more processors are added, but requires image compositing and thus higher communication bandwidth for producing the final image. In this paper, we present a parallel volume rendering algorithm that combines the benefits of both image-space and object-space partition schemes based on the idea of pixel and volume interleaving. We first subdivide the processors into groups. Each group is responsible for rendering a portion of the volume. Inside of a group, every member interleaves the data samples of the volume and the pixels of the screen. Interleaving the data provides storage scalability and interleaving the pixels reduces communication overhead. Our hybrid object- and image-space partitioning scheme was able to reduce the image compositing cost, incur in low communication overhead and balance rendering workload at the expense of image quality. Experiments on a PC-cluster demonstrate encouraging results.

1. Introduction

Volume rendering is a process that directly projects three-dimensional scalar data into two-dimensional images without generating intermediate geometry. Since the amount of computation needed for data sampling and projection is usually large, it is difficult to perform volume rendering at an interactive rate. Intensive research has been conducted in the past decade to accelerate volume rendering. Among the existing techniques, parallel volume rendering is an effective approach that can potentially handle very large sized volume data and generate high resolution images.

Parallel volume rendering algorithms can be generally divided into two categories based on how the workload is distributed among the processors. One is the image-space partitioning scheme which subdivides the screen into small tiles and assigns each processor one or multiple tiles. The other is the object-space partitioning scheme which subdivides the volume into small subvolumes and each processor is responsible for rendering one or multiple subvolumes. Generally speaking, image-space partitioning techniques require less communication at the image compositing stage. This is because the screen tiles assigned to different processors usually do not overlap, thus no compositing is needed to assemble the final image. However, image-space partitioning can incur in higher data redistribution overhead when the viewer changes position, because different portions of the volume will be projected onto different image tiles. If the entire volume data is not replicated and stored locally with each processor, data must be redistributed among the processors when the view is changed. Most of image-space partitioning volume rendering techniques choose to replicate data to avoid the costly data redistribution operations.

Compared to image-space partitioning techniques, object-space partitioning techniques do not require data redistribution or replication when the view is changed because each processor is responsible for the same data blocks regardless of viewing parameters. This allows object-space partitioning scheme to exhibit a better storage scalability when increasing the size of the dataset and/or the number of processors. However, since a pixel can receive color contributions from several data blocks assigned to different processors, assembling the final image requires compositing the partial images from different processors, which incurs in higher communication cost. As a result, the communication overhead to perform image compositing in object-space partitioning tech-

© The Eurographics Association 2002.

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

niques can become a major bottleneck when the size of the rendered image is large.

To reduce the communication overhead at the image compositing stage and to increase the algortihm's scalability for large scale datasets, we devise a hybrid image- and object-space partitioning algorithm to perform parallel volume rendering. Our algorithm first divides the processors into several groups, and performs an object space partitioning to distribute the volume data among the processor groups. Within each processor group, we perform image-space partitioning by interleaving pixels and let processors render alternating rows and columns of pixels. To reduce the amount of data that is required by each processor, we use a volume interleaving scheme to distribute the volume data among the processors within each group and approximate the final rendering result. This approximation is directly related to the adjusting of a user-specified parameter called interleaving factor. Based on this factor, the behavior of our algorithm can be steered from performing pure image-space partitioning to pure object-space partitioning. The goals of our pixel and volume interleaving algorithm are: a) keep communication cost as low as that of image-space partition techniques, and b) maintain low memory overhead in each processor similar to object-space partition techniques. The cost of our algorithm is a moderate loss of image quality when the interleaving factor is high. We implemented our parallel algorithms using both ray casting and hardware texture mapping as the core rendering component, and have tested our implementation on several volume datasets to monitor rendering performance and image quality. In addition, we tested several screen sizes to study the cost of compositing and of load balancing.

This paper is organized as follows: section 2 reviews previous techniques, section 3 describes our algorithm in detail, section 4 discusses and compares the results of our tests, section 5 presents conclusions, and section 6 suggests new directions for extending our work.

2. Related Work

Volume Rendering as described by Derbin[1] has been realized in several ways, of which raytracing [2] and texture hardware [3] are at opposite ends when comparing rendering speed.

Ma *et al.*presents a divide-and-conquer algorithm for parallel volume rendering [4]. Their method is a pure object-space partitioning scheme, where each processor receives a partition of the volume. Once rendering is done on the partition, neighboring processors repeatedly exchange half of their current images for compositing until all processors hold a small tile of the final image. Scalability is a major advantage of this algorithm, but load imbalance is a drawback.

Other partition schemes as classified by Molnar *et al.*[5] subdivide the screen into tiles or contiguous arrays of pixels. Each tile is assigned to a different processor and rendering is done according to that tile. However, load imbalance is very high if the projected bounding box of the dataset does not fall in the tile or has a small contribution to the tile. This is a big problem for either object-space or image-space partition schemes. Either dividing the final image or the dataset does not guarantee the same rendering time for every processor because of data coherence or data randomness. Molnar [5], Neumann[6] and Lee [7] agreed that interleaving pixels or block of pixels distributes the work evenly among processors.

Interleaving has other uses such as described by Keller and Heidrich [8], where the idea of interleaving not only pixels but also sampling planes is described. The main idea was to generate an image with reduced or removed artifacts as if rendered with the methods proposed by Cullip and Neumann[3] or Wilson *et al.*[9], which use either image-aligned or object-aligned polygons to render the final image in back-to-front order. Keller and Heidrich [8] divided the full image into blocks of pixels, then performed rendering in a multipass fashion. Their results present far better quality than traditional 3D texture hardware rendering.

To further increase data scalability, one can use multiresolution techniques. In essence, multiresolution rendering techniques first create multiple levels of detail for the underlying dataset, where each level is a filtered then subsampled version of a higher resolution data. At run time, certain criteria are used to choose appropriate levels of detail for the volume when producing the final image. LaMar [10] and Weilder [11] are examples for 3D texture hardware and Danskin [12] for ray tracing. These techniques can greatly reduce, in the case of 3D texture hardware, the bottleneck that results from trasmitting data into texture memory or, in the case of ray tracing, the traversal of voxels. Because of the subsampling at each level, fine data features can be missed in the final image when low resolution of data are used.

Finally, PC-clusters are increasingly being used in massive parallel computations because of their low cost and high expandability. However, the interconnection network for communications between processors, though becoming faster and faster, is orders of magnitude slower than internal CPU and memory buses found in supercomputers, therefore, reducing the amount of bandwidth needed by global operations such as compositing and final assembly of the image is paramount. Samanta [13] focused on these goals for polygons.

In our work, we combine the benefits of Keller[8], Ma [4] and LaMar [10], and use raycasting [2] and texture hardware [3] as volume renderers to obtain an interleaved volume renderer.

3. Algorithm

3.1. Overview

Our algorithm is based on a combined pixel and volume interleaving scheme. Hereafter interleaving is referred to as distributing samples to processors in a round-robin fashion

© The Eurographics Association 2002.

with a predefined pattern. The purpose of using interleaving are twofold: reduce communication overhead among processors during the image compositing stage, and reduce the amount of data needed by each processor to perform rendering.

Given N processors, a screen with P^2 pixels, and a volume with S^3 samples, we first subdivide the processors into groups. The number of processors per group is determined based on a user-specified parameter, called interleaving factor, which will be explained later. Depending on how many groups, the volume is partitioned repeatedly along alternating dimensions into subvolumes. Each subvolume is assigned to one group.

Inside of a group, the screen is subdivided into sets of pixels. Each pixel set is assigned to a processor which becomes responsible for computing the pixels' final colors based on the assigned subvolume. The pixels are assigned by alternating rows and columns. In addition, the subvolume is also interleaved among the processors, so that each processor holds only a portion of the subvolume. Up to this point, everything has been done off-line. The on-line part of the algorithm includes the rendering and the global operations.

Rendering is performed locally on each processor and its result is a partial image that will serve as input for image compositing and global gathering. Image compositing is required to composite partial images from processors in different groups that have pixels corresponding to the same screen locations. Inside each group, image compositing is not necessary since no two processors will attempt to render the same pixels due to the interleaving effect. Once compositing is completed, gathering operations generate the final image. In the following, we explain each of the important stages of our algorithm in detail.

3.2. Interleaving Factor

The combination of an image-space partitioning and an object-space partitioning scheme is decided by an interleaving factor. This factor divides the workload into groups of processors. As this factor increases, the number of groups decreases and the number of processors per group increases. The number of groups (G), the number of processors per group (M) and the interleaving factor (i) are related as follows:

$$M = 2^i \tag{1}$$

$$G = N/M \tag{2}$$

Choosing 0 for i yields N groups with 1 member holding a partition of the volume. Choosing the highest number available for i, on the other hand, yields 1 group with N members. The later would suggest that each member holds the full volume to generate its pixels, but scalability is one concern that is treated in the next section. Choosing any other number for i yields G groups with 2^i members.

It is also important to mention, that a processor inside of a group will hold an identification number that it's local to the group. This local id is used to interleave volume and pixel samples, which will determine the processor's workload. The id results from a simple mod operation:

localID = globalID mod M

The globalID is a number that a processor is assigned initially.

3.3. Object-space volume partition

Once we determine the number of groups based on the interleaving factor, we partition the volume so that each group receives one subvolume. We use a Kd partitioning scheme similar to Ma [4] to subdivide the dataset. Starting with the longest dimension, processors with lower globalIDs are assigned the lower half of a partition, while processors with higher globlaIDs the upper half of a partition. This partition process is recursively performed for each dimension until the number of partitions equals the number of groups.

However, if every member of the group holds the entire subvolume with size E and the size of the full volume is S^3, the memory requirements will become prohibitively expensive as the interleaving factor i increases. This is because the splitting process uses the number of groups to create the partitions, thus:

$$E = S^3/G \tag{3}$$

But from (1) and (2)

$$E = S^3/(N/2^i) \tag{4}$$

$$E = 2^i S^3/N \tag{5}$$

In the extreme cases, $i = 0$ gives every processor S^3/N, which is optimal; while $i = log_2(N)$ gives every processor a subvolume with a size equal to S^3, which is certainly not optimal. To fix this problem, we use a volume interleaving scheme to ensure that every processor only holds S^3/N amount of data. We now explain the volume interleaving scheme.

3.4. Volume interleaving

Each processor will accommodate a subvolume that remains the same until the end of the rendering. To reduce the size of local volume stored in each processor, we interleave the subvolume assigned to each processor group in alternating

© The Eurographics Association 2002.

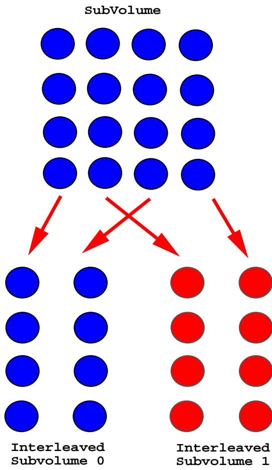

Figure 1: *Volume Interleaving of a subvolume's slice distributed to 2 processors. Starting from the lower left corner of the slice, a processor finds its initial sample. Processor 0 takes every other sample starting from the corner, while processor 1 takes every other sample but starting at an offset of 1 from the corner.*

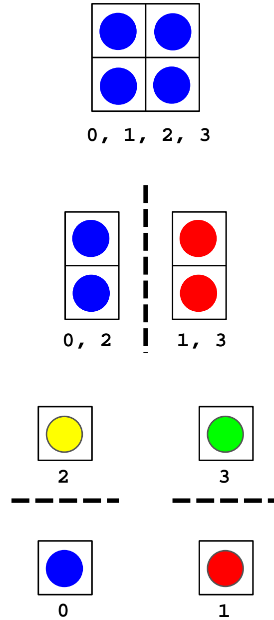

Figure 2: *Pixel interleaving with 4 processors that are in the same group. Starting with the horizontal dimension, even processors take the left side and odd processors take the right side. Next, the process repeats separately on each partition but this time the top or bottom side are distributed. At this point the splitting terminates since there is only 1 processor per partition.*

x, y, or z planes and distribute the partition results to the processors. The purpose of volume interleaving is to keep the size of data stored in each processor equal to S^3/N regardless of the interleaving factor, but at the expense of moderate rendering quality losses. Figure 1 shows an example in two dimensions of how samples are divided into 2 processors.

With nothing else but subsampling to create the interleaved subvolumes, rendering artifacts will be noticeable. To solve this problem, we filter the volume before interleaving. Filtering effectively distributes the value of each voxel to its neighborhood, which ensures that features from the original data will not be completely lost when volume interleaving is performed, Filtering also smoothes the intermediate subvolumes and reduces the high frequencies that can create discontinuities at rendering. Inevitably, since processors have different filtered data samples, it is possible that some processors hit volume features, while others miss the same features resulting in discontinuities. This is noticeable at edges, silhouettes and features that are in the size of a few voxels. However, our experimental results showed that these discontinuities were moderate.

3.5. Pixel Interleaving

Before rendering starts, we assign pixels to the processors within each group according to the interleaving factor. This part is view-dependent, because we calculate the projected bounding box of the subvolume, then divide along the longest dimension until the number of division stages reaches i, the interleaving factor. For instance, if the horizontal dimension of the projected bounding box is longer, the pixels in the columns are interleaved first, then the rows, then the columns and so on. If the vertical dimension is longer, then the interleaving sequence is rows, columns, rows and so on. Since each processor is responsible for a different pixel set, the partial image it generates has less resolution than the final image.

As we alternate between the horizontal and vertical dimensions of the screen, the processors are recursively splitted into two partitions: the first partition selects every other processor starting from the first, while the other partition takes the rest. This way the first partition gets either the left or bottom screen partition and the second partition gets either the right or top screen partition. The splitting uses the localID to assign pixels, so that the same pixel interleaving pattern is followed by one processor in each group. The reason for this distribution is to simplify the global operation gathering, which alternates between dimensions to collect data. Figure 2 shows an example of how 4 processors are arranged to select their pixels.

The number of pixels assigned to a processor equals $P^2/2^i$. However, this is an upper bound, since the projected bounding box of the subvolume of a processor is at most P^2.

© The Eurographics Association 2002.

In the extreme cases, the upper bound equals P^2 when $i = 0$, and P^2/N when $i = log_2(N)$.

3.6. Rendering

Once we have determined the pixel and volume interleaving patterns and distributed the data accordingly, each processor is ready to perform local rendering of its assigned subvolume. The renderer takes a 3D lattice of scalar values, a transfer function, a camera and a transformation matrix, and returns an array of pixels comprised each of 4 values: the 3 color channels, red, green and blue, plus the alpha channel, which holds accumulated opacity.

A couple of considerations must be taken into account when the interleaving factor is greater than 0: a) the transfer function must be corrected, and b) volume samples must be rendered at their original positions. The first is because, processors inside of a group hold a subvolume that has been filtered and interleaved, therefore, there are less samples taken than in the original subvolume. As rays traverse these interleaved subvolumes, opacity must be compensated to produce the correct color. We use LaMar's strategy [10] to vary the transfer function. The second consideration rests in the fact that all volume samples will contribute to the final image and need to remain at their original positions. Multiresolution techniques average samples and assign low resolution subvolumes to replace the original data at the same world position. In our case, that simple replacement will translate into pixel shifting and extreme blurriness for our images, therefore, we take into account the offsets of the initial positions when volume interleaving.

3.7. Compositing and Gathering

After rendering is completed, we perform a binary-swap image compositing similar to the method that Ma [4] proposed. According to the bounding box of the full volume, we start splitting either horizontally or vertically and alternate between the screen dimensions for the next splits. At every split, processors exchange half their partial image to composite and repeat the process $log_2(N) - i$ times. Therefore, in the extreme cases, when $i = 0$, the number of compositing stages needed is maximum and equals the original binary swap algorithm, which is a pure object-space partitioning scheme. On the other hand, when $i = log_2(N)$, no compositing at all takes place since it is a pure image-space partitioning scheme. For any other value of i, the number of compositing stages decreases as i increases.

Inside the groups, each processor has its own local ID. At the compositing stage this number comes into play, only processors in different groups with the same local ID need to composite since they hold the same volume and pixel interleaving pattern.

At the end of compositing, the tiles generated will resemble the numbering of the pixel interleaving section, thus

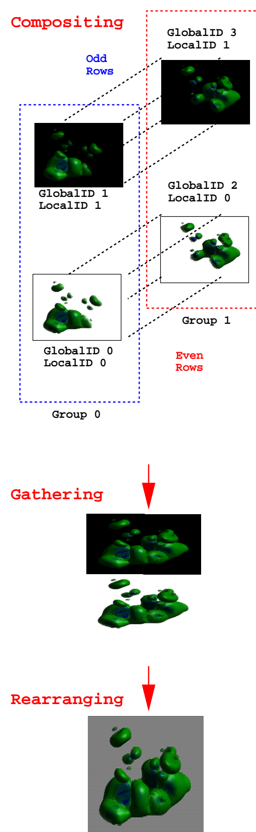

Figure 3: *Compositing, gathering and rearranging of 4 processors with interleaving factor 1. Processors 0 and 2 who are in different groups exchange and composite halves to produce correct pixels on the even rows, while at the same time processors 1 and 3 take care of odd rows. Since interleaving factor is 1, there are only 2 groups and therefore 1 compositing stage is needed. Next all tiles are collected and finally rearranged for display. The background colors are set for illustration purposes.*

gathering can and will alternate between screen dimensions. At each stage, gathering joins the tiles and repeats the process until 1 processor holds all pixels. Finally, if the interleaving factor is greater than 0, the pixels of the collected tiles must be rearranged since they do not represent contiguous set of pixels but interleaved ones. Figure 3 shows the traveling of pixels before they are displayed on screen.

4. Results

We performed experimental tests on a 8-node PC-cluster. Each node had a Pentium III (993.334Mhz) with a GEForce II graphics card and a CLAN 1000 network interface card using the virtual interface architecture (VIA). All machines were connected to a Fast Ethernet network. The Code

© The Eurographics Association 2002.

was written in C++, and it was linked to MPI for parallel communication and to OpenGL/GLUT [14] for hardware rendering. Three datasets were used in our experiments: Hipip (64x64x64), shockwave (128x128x128) and skull (256x256x256). Every sample in the dataset was represented by a 8-bit unsigned character. The transfer functions were lookup tables of 256 entries, and each entry has 4 floating-point RGBA values. All rendered pixels were 4-tuples (RGBA) of 8-bit values, but they became 3-tuples (RGB) of 8-bit values before gathering. When interleaving factor was not 0, every filtered dimension used the tent filter [15].

4.1. Rendering

We implemented our parallel algorithm using two different volume rendering techniques. One is a software-based raytracer, and the other is a hardware-based 2D-texture volume renderer. In the raytracer implementation, the processor casts a ray from every pixel to sample the subvolume. At every sample, the ray trilinearly interpolates the eight neighboring samples to obtain the sample density. Normals are calculated at every sample, and used for local lighting calculation to generate the sample colors. As the ray marches through the subvolume, samples are blended in a front-to-back order. We early terminate the ray once it has accumulated enough opacity. In our 2D-texture hardware renderer implementation, subvolume slices are loaded into texture memory and each texture is rendered over a simple quadrilateral. Rendering is done in back-to-front order and lighting is disabled. As expected, raycasting dominates the rendering time in the parallel raytracing program. Loading data into the hardware's texture memory and reading pixels from the hardware's frame buffer into main memory are the bottlenecks for texture hardware rendering.

Figure 4 shows the speedups for rendering one frame with varying interleaving factor and number of processors. The charts show the results obtained from rendering the skull dataset at 1024x1024 pixel resolution. The behavior is almost identical for the other datasets and different resolutions. We notice that for raytracing, speedups go beyond linear acceleration, which is an effect of volume interleaving. Even though processors have the same amount of data to render, the interleaved subvolumes are in a way lower resolution versions of the subvolumes at $i = 0$, thus rays traverse them faster.

Texture hardware rendering does not exhibit the same super-linear property. This is mainly because the speedup gained from rasterizing a slightly smaller number of polygons is negligible compared to other factors such as clearing or reading back the frame buffer. From Figure 4, we can see that the hardware texture rendering time reduced proportionally to the different number of processors.

Other factors for texture hardware are texture loading,

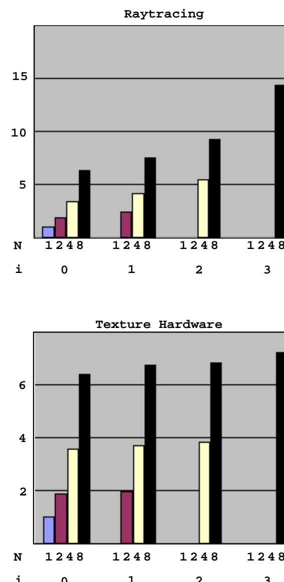

Figure 4: *Speedups for raytracing and texture hardware for the skull dataset at 1024x1024 pixel resolution. The horizontal axis represents different interleaving factors, and the vertical axis shows speedups. The multiple bars for each interleaving factor represent different number of processors*

which is affected by the size of data, and pixel read back, which is affected by the size of the screen. Our volume and pixel interleaving scheme keeps processors with the same amount of data and reduces the number of pixels read back from the frame buffer. Table 1 and 2 shows timings for all datasets and all screen sizes tested. Both stages scale well with the number of processors.

A study involving frames-per-second is certainly warranted. However, such a study must be considering the option of texture color table lookups turned on, so that transfer functions can be changed on the fly without reloading of textures; otherwise, the cost of texture loading will slow down rendering performance. This feature was not implemented in this study, but we ceratinly consider the possibility for future work.

Load imbalance is a major concern for parallel processing. Raytracing exhibit major load imbalances for purely object-space partition, when interleaving factor equals to 0. Most of our results indicated that as interleaving increased, load imbalance decreased. However, Figure 5 shows a change in this pattern. Hipip and Shockwave could be considered symmetric data, that is top and bottom view are similar, left and right view are similar and front and back view are similar; Skull, on the other hand, is not. This means that volume interleaving might generate subvolumes that have

© The Eurographics Association 2002.

Dataset	Num. of Processors	Interleaving factor 0	1	2	3
Hipip (64x64x64)	1	0.069042			
	2	0.036717	0.036425		
	4	0.019623	0.020353	0.019646	
	8	0.009849	0.011388	0.011471	0.009802
Shockwave (128x128x128)	1	0.538468			
	2	0.273060	0.270793		
	4	0.138192	0.141305	0.138130	
	8	0.069262	0.072650	0.072881	0.069118
Skull (256x256x256)	1	4.327917			
	2	2.173000	2.167407		
	4	1.076563	1.082972	1.075233	
	8	0.538956	0.541802	0.541316	0.537077

Table 1: *Timings for loading data slices into texture memory (in seconds)*

Image Size	Num. of Processors	Interleaving factor 0	1	2	3
256x256	1	0.005373			
	2	0.003560	0.002403		
	4	0.002084	0.001773	0.001390	
	8	0.001329	0.001267	0.001096	0.000908
512x512	1	0.021043			
	2	0.014088	0.010232		
	4	0.007948	0.006498	0.005034	
	8	0.004471	0.003764	0.003212	0.002396
1024x1024	1	0.067127			
	2	0.042891	0.022831		
	4	0.020285	0.013864	0.011250	
	8	0.009063	0.003810	0.006876	0.000848

Table 2: *Timings for reading pixels from the hardware's frame buffer onto memory (in seconds)*

unequal contributions to the final image. Figure 6 shows visually the skull's case.

4.2. Compositing and Gathering

Operations in this stage include compositing of partial images, conversion from 4-tuples (RGBA) to 3-tuples (RGB) of 8-bit values, gathering of tiles, and rearranging of pixels into their respective positions in the final image. The communication cost incurred by image compositing and gathering are affected by the interleaving factor. Figure 7 shows a general breakdown of the four stages involved to generate the final image. We can see, starting from the top, the fully interleaved schemes, i.e., when 2^i equals to the number of available processors, no compositing was needed and the conversion of partial images into tiles is optimal since partial images are non-overlapping portions of the screen. When the interleaving factor is small, our algorithm behaves more like a purely object-space partitioning algorithm, and thus has higher communication costs due to larger image transfer time. Figure 7 presents the results of Hipip dataset at one of the resolutions tested. Table 3 shows that the compositing time reduces when the interleaving factor increases. The same behavior was obtained from the other screen resolutions. These results provide the answer to our goals. As interleaving increases, the amount of compositing decreases faster with respect to gathering. This achieves behavior similar to pure image-space partitioning schemes. Furthermore, the fact that processors only allocate a portion of the volume, which remains the same regardless of interleaving, achieves behavior similar to pure object-space partitioning schemes.

Even though, it is possible to implement the rendering unit

© The Eurographics Association 2002.

Image Num. of Processors	Interleaving factor 0	1	2	3
1	0.000002			
2	0.088542	0.000002		
4	0.133079	0.043364	0.000003	
8	0.115259	0.068798	0.022516	0.000003

Table 3: *Compositing times as interleaving increases for 1,2,4 and 8 processors*

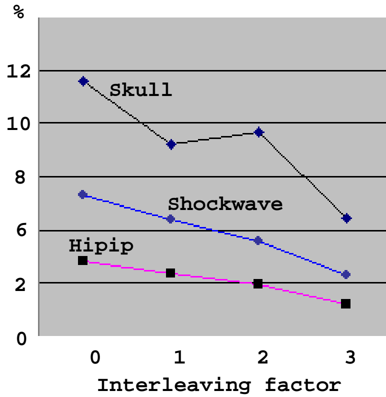

Figure 5: *Load imbalance percentages for the 3 datasets at 1024x1024 pixel resolution, 8 processors and varying interleaving factor*

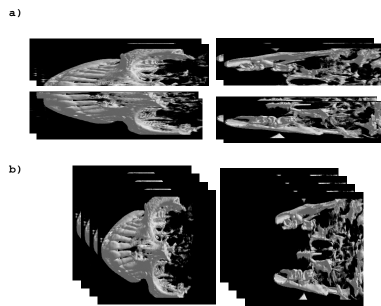

Figure 6: *Since Skull concentrates its data on one side, processors rendering on that side terminate their rays faster, while those on other side continue traversing until they exit the volume. Therefore, traversal of voxels is slower. Furthermore, the amount of empty space traversed is greater at b) with $i = 2$ than at a) with $i = 1$.*

such that partial images are comprised of two separate arrays: one for color channels and the other for opacity, so that before gathering we can simply drop the array of opacity and have no conversion; it is also possible to combine gathering and rearranging into one stage, such that they overlap

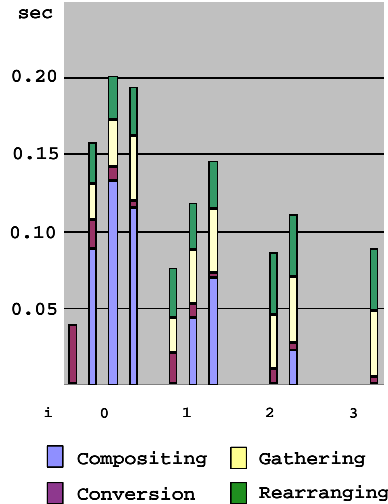

Figure 7: *Time breakdown of compositing, conversion, gathering and rearranging for Hipip at 1024x1024. The multiple bars for each interleaving factors represent different number of processors.*

in time. We chose contiguous data arrangements for simplicity and efficiency of network communication.

4.3. Image Quality

Although most features are still present in the rendering, even the small ones such as those in skull, minor image quality degradations are indeed noticeable. The degradation mostly happens at the edges and silhouettes because features can be missed by sampling rays as a result of volume interleaving. For small datasets such as Hipip 64^3, artifacts are more noticeable, but for large datasets we see that it becomes less of a problem. See the images on Figure 8, where each dataset's final image is shown side-by-side with interleaved counterparts. The projection size of the voxels determines how much the artifacts appear in the final image. The smaller their sizes the less processors interleave the feature. Hence, artifacts will be less noticeable.

© The Eurographics Association 2002.

5. Conclusion and Future Work

We present an algorithm that combines the benefits of scalability from object-space partitioning schemes and low-communication from image-space partitioning schemes. Our interleaving factor controls the combination of these two schemes and according to our results, low interleaving factors can effectively reduce the cost of compositing and keep quality close to that of a purely object-space partition.

In the implementation of our parallel algorithm, we have intentionally left out several known techniques to accelerate the rendering process, so that we can concentrate on the effects of the interleaving factor. Nevertheless, we feel that those techniques can be incorporated into our algorithm with no major changes, especially those that can skip regions that do not contribute much to the final image either because they have low opacity or because they are totally occluded by the accumulated partial image. We suspect that techniques that are data-dependent will incur in more load imbalance, but interleaving in conjunction with heuristics to better partition the data can present better load balancing.

Multiresolution techniques such as those by LaMar [10] and Danskin [12] with their corresponding acceleration techniques are also good candidates for extending our algorithm.

Volume interleaving remains an open topic for enhancement. The better interleaved subvolumes resemble the original subvolumes, the less artifacts will be present at edges and silhouettes. Filtering provided us with better results than simple subsampling, however, better approximations require further study.

Finally, the study of time varying volume data in parallel environments is an area that will certainly benefit from our algorithm, specially in the reduction of global operations overhead.

References

1. R. Derbin, L Carpenter, and P Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, 1988.
2. M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.
3. T. Cullip and U. Neumann. Accelerating volume reconstruction with 3d texture hardware. *Technical Report UNC Chapel Hill*, 1994.
4. K.-L. Ma, J.S. Painter, C.D. Hansen, and M.F. Krogh. Parallel volume rendering using binary-swap image composition. *IEEE Computer Graphics and Applications*, 14(4):59–68, 1994.
5. S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.
6. U Neumann. Communication costs for parallel volume rendering algorithms. *IEEE Computer Graphics and Applications*, 14(4):49–58, 1994.
7. T. Lee, C. Raghavendra, and J. Nicholas. Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. In *Proceedings of IEEE Parallel Rendering Symposium 95*, pages 55–62. IEEE Computer Society Press, Los Alamitos, CA, 1995.
8. A. Keller and W. Heidrich. Interleaved sampling. In *Proceedings of Eurographics Rendering Workshop*, pages 269–276, 2001.
9. O. Wilson, A. Van Gelder, and J. Wilhelms. Direct volume rendering via 3d textures. *Technical Report UCSC-CRL-94-19, University of California, Santa Cruz*, June 1994.
10. E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of Visualization '99*, pages 355–361. IEEE Computer Society Press, Los Alamitos, CA, 1999.
11. M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl. Level-of-detail volume rendering via 3d textures. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 7–13, 2000.
12. J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1992.
13. S. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *In Eurographics/SIGGRAPH Workshop on Graphics hardware*, pages 99–108, 2000.
14. M. Woo, J. Neider, T. Davis, and D. Shreiner. Opengl programming guide. Addison-Wesley, 1999.
15. K. Turkowski. Filters for common resampling tasks. *Graphics Gems I, Academic Press*, 1990.

© The Eurographics Association 2002.

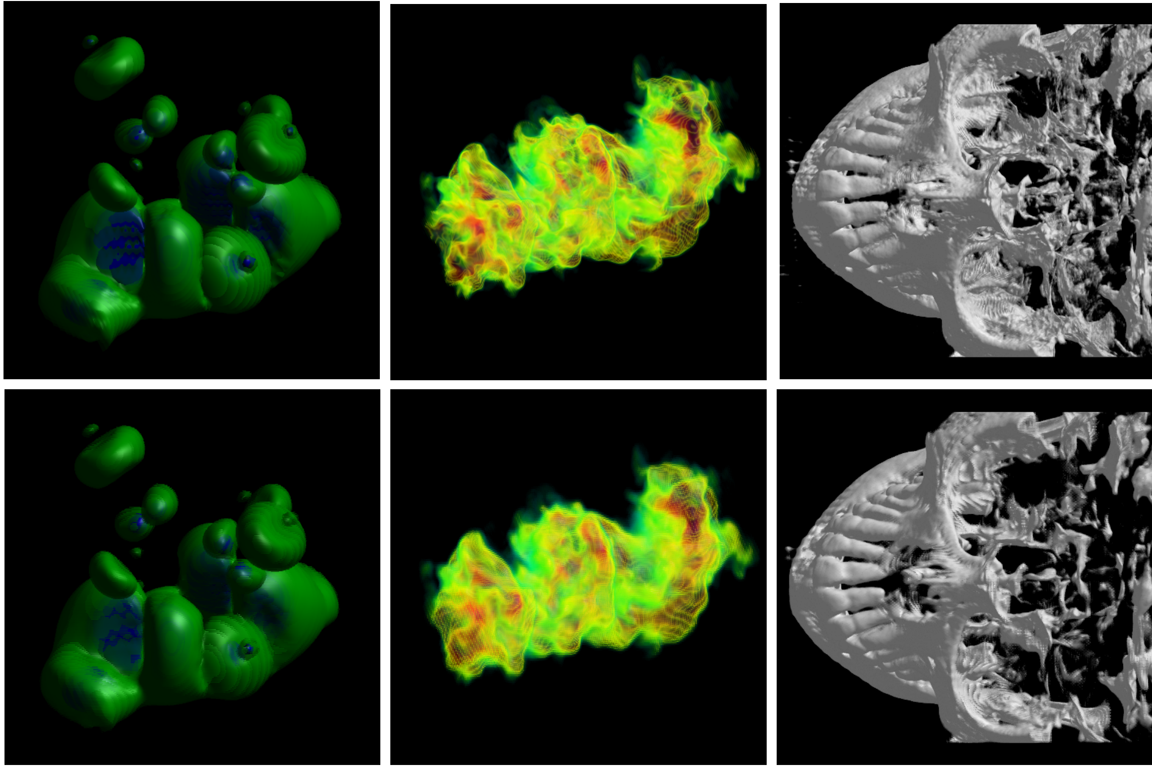

Figure 8: *Datasets: Left: Hipip(64x64x64), Middle: Shockwave (128x128x128), Right: Skull (256x256x256). The first row represent the original images, the second row goes as follows: Hipip with* $i = 1$*, shockwave with* $i = 2$ *and skull with* $i = 3$.

© The Eurographics Association 2002.