Eurographics Symposium on Parallel Graphics and Visualization (2010)
J. Ahrens, K. Debattista, and R. Pajarola (Editors)

Cross-Node Occlusion in Sort-Last Volume Rendering

Stéphane Marchesin[1] and Kwan-Liu Ma[1]

[1]University of California, Davis

Abstract

In the field of parallel volume rendering, occlusion is a concept which is already widely exploited in order to improve performance. However, when one moves to larger datasets the use of parallelism becomes a necessity, and in that context, exploiting occlusion to speed up volume rendering is not straightforward. In this paper, we propose and detail a new scheme in which the processors exchange occlusion information so as to speed up the rendering by discarding invisible areas. Our pipeline uses full floating point accuracy for all the intermediate stages, allowing the production of high quality pictures. We further show comprehensive performance results using this pipeline with multiple datasets and demonstrate that cross-processor occlusion can improve the performance of parallel volume rendering.

1. Introduction

Volume rendering is a multi-purpose tool for data exploration and visualization thanks to its good ability at depicting internal data features. However, the images produced using volume rendering usually require a lot of computation, and often processor time is spent on areas which do not contribute to the final pictures, namely the occluded areas. In this paper, we introduce a parallel rendering pipeline making use of inter-processor occlusion. Taking advantage of cross-processor occlusion allows us to obtain substantial performance improvements, especially on dense scenes, which are common in the area of volume rendering. We also make use of a two-level parallel rendering scheme with load balancing at the lower level. We demonstrate and benchmark a floating point rendering pipeline running on a CPU cluster. Although CPU clusters are not the most appropriate for high performance rendering, GPU clusters are not always available. Furthermore, CPU clusters are often the only way to achieve in-situ visualization, which is useful for extremely large simulations where the sheer size of the data prevents moving it to a separate, dedicated visualization cluster.

2. Related works

Volume rendering as introduced by Sabella [Sab88] is an efficient technique for data exploration and visualization. As opposed to surface rendering, it allows the depiction of all the internal data features in a simultaneous fashion, and thus produces pictures conveying more information. At the core, volume rendering entails numerically computing an approximation of the so-called volume rendering integral, which computes the final color of each ray going through a volume. This is an expensive process, especially since it has to be computed for each pixel of the screen.

Much research was pursued to speed up compute-intensive volume rendering algorithms. Volume rendering of large datasets on a single machine can be achieved using simplification-based techniques. These techniques usually create a number of level-of-details for the data and use these levels to achieve real-time visualization. In this context, bricking as introduced by Weiler *et al.* [WWH*00] is a method for improving the efficiency of volume renderers, which consists in splitting the volume into equally-sized bricks. This entails a number of advantages: empty bricks can be culled, and multi-resolution methods can be used, for example by lowering the level of detail for bricks which are further away from the observer. Weiler *et al.* [WWH*00] and Lamar *et al.* [LHJ99] use data-dependent multi-resolution textures which reduce the requirements for texture memory, and are thereby able to render large datasets. Lamar *et al.* [LHJ03] propose an efficient error computation technique for 3D data. Their technique uses a histogram for each data brick, and takes advantage of this histogram to quickly compute error values for a given brick. This is particularly useful to evaluate the simplification error and the visibility of a given brick without iterating all its voxels. Röttger *et al.* [RKE00] introduced the notion of pre-integration which greatly enhances the quality of volume rendered pictures. This work was conducted in the context of unstructured datasets, and was later extended by Engel *et al.* [EKE01] to support structured datasets. Grimm *et al.* [GBKG04] intro-

© The Eurographics Association 2010.

duce an efficient CPU-based volume rendering framework, including many techniques which improve performance.

Guthe *et al.* [GS04] use a two-pass algorithm to achieve culling of invisible areas. In a first pass, they consider the data at a low resolution and determine the invisible bricks; in the second pass, only the visible volume parts are rendered.

In order to increase the size of the datasets that one can handle, volume rendering has been parallelized. In the field of parallel rendering, techniques were classified into three categories by Molnar *et al.* [MCEF94] (sort-first, sort-middle and sort-last) according to the place of the sorting phase in the graphics pipeline. When the sorting is done prior to transforming and rasterizing the primitives, the approach is of the sort-first kind. If sorting is done between the transform and rasterization phases, the approach is called sort-middle. Finally, if sorting is done at the end of the pipeline, the approach is of the sort-last kind. However, only sort-first and sort-last are relevant in the field of parallel volume visualization. In this paper, we will focus on sort-last techniques which are the most efficient for very large datasets.

Much research has gone into the issue of improving the performance of sort-last rendering. In particular many techniques have been proposed to recompose the partial images together. Direct send as introduced by Hsu [Hsu93] is an efficient algorithm for compositing these images. This algorithm allows using all the processors for compositing and therefore achieves good scalability. Ma *et al.* [MPHK94] propose the binary swap algorithm which uses a hierarchical communication scheme to improve the performance on large scale workloads while still involving all the processors in the compositing. Stompel *et al.* [SML*03] propose a scheme which focuses on software rendering and minimizing the impact of the sort-last communication stage. The authors take into account the footprint of the rendered data and use it to load balance the compositing jobs between the cluster nodes. Yu *et al.* [YWM08] improve the binary swap algorithm by exploiting a hybrid 2 and 3-swap mechanism which allows using an arbitrary number of compositing processors. Peterka *et al.* [PGR*09] further improve this algorithm by proposing the Radix-k algorithm which combines multiple direct-send stages. Strengert *et al.* [SMW*04] implement an efficient sort-last volume rendering framework using level-of-detail, and report interactive frame rates on a cluster of Myrinet-connected machines. Lombeyda *et al.* [LMS*01] are able to achieve interactive volume rendering performance (more than 25 frames per second) using dedicated rendering and compositing hardware. Peterka *et al.* [PYRM08] evaluate the efficiency of software-based volume rendering Blue Gene/P for in-situ visualization on a large number of processors. Wylie *et al.* [WPLM01] show that using a sort-last technique, it is possible to handle large datasets using a static data distribution. However, when using a static data distribution in a sort-last scheme, load imbalance can happen between the nodes as not all the nodes have the same amount of work to be done. For example, level-of-detail based methods will result in such load imbalance. Advanced techniques using occlusion and empty space culling can also result in such imbalance. Therefore, load-balancing sort-last techniques have been developed. Wang *et al.* [WGS04] achieve dynamic load balancing in the context of software based volume rendering. Finally, dynamic load balancing of volume data can be achieved using a hierarchical KD-tree decomposition as described in [MMD06, MSE06].

The issue of occlusion between different rendering nodes of a sort-last cluster has been studied before. Cox *et al.* have analyzed the issue of depth complexity in parallel polygonal rendered scenes [CH92]. In particular, the authors have proven [CH93] that an evenly distributed depth between the nodes was the worst situation for inter-node occlusion snooping in a parallel rendering system. In the field of isosurface extraction and rendering, Gao *et al.* [GS01] propose a multi-pass occlusion algorithm which only computes and renders the visible parts of an isosurface on a rendering cluster. To exploit occlusion in volume rendering, Gao *et al.* [GHSK03] propose the use of plenoptic opacity functions which allow determining opaque bricks and therefore culling away the hidden voxels. However, this approach is conservative and requires a pre-computation step.

Although inter-node occlusion has already been widely exploited for parallel isosurface occlusion and volume rendering already exploits culling of invisible volume parts at a coarse granularity, to the extent of our knowledge no work has been conducted which handles exact occlusion in the case of parallel volume rendering. Volume rendering can actually benefit greatly from occlusion culling as the process is computationally intensive, and therefore culling out invisible and occluded regions is a substantial source of performance improvement.

3. Parallel rendering pipeline

Our parallel rendering pipeline is based on a direct send implementation. We now detail its two main capacities: two-level high quality hybrid volume rendering and cross-processor occlusion.

In the rest of the paper, we use the following terminology: a *processor* is the smallest possible computation unit; a *node* is a machine connected to a network, containing one or multiple *processors*; a cluster is a collection of multiple *nodes* connected by a shared network.

3.1. Hybrid parallel volume rendering

Our rendering pipeline uses two-level hybrid parallelism. This scheme is depicted on Figure 1 and works as follows:

- The first level of parallelism takes place between the different processors of a node. Inside a node, sort-last rendering is not used, instead we employ a brick-based rendering scheme for performance reasons. In this scheme, the

© The Eurographics Association 2010.

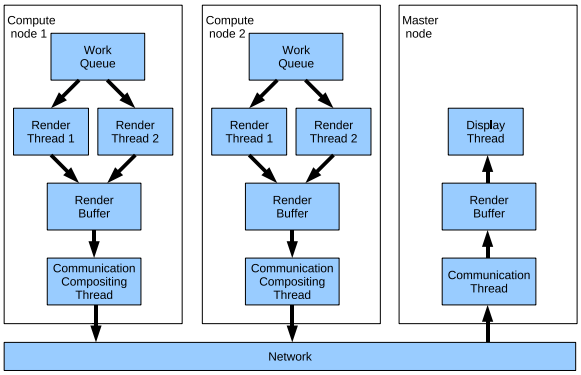

Figure 1: *Hybrid multi-core/multi-node parallelism with two dual-core rendering nodes and a master node. The render threads repeatedly take a brick from the work queue and draw it onto the frame buffer. After all bricks have been rendered, the communication threads handle the compositing stage while another frame starts being rendered on the processors.*

processors use a shared queue of work to achieve load-balanced rendering of volume bricks. This queue contains a front-to-back sorted list of bricks, and the different processors take work away from that queue and render the corresponding bricks to a shared frame buffer. Notice that this is not as trivial as one could think, as race conditions can happen if all processors render to the shared buffer directly. For example if we consider two bricks B1 and B2 with a back-to-front dependency between B1 and B2, one processor P1 could finish rendering brick B1 before processor P2 finishes brick B2, even though processor P2 started first and therefore the work queue order was respected. We solve this by having each processor render each brick to a private buffer and using a completed work queue; a given volume rendered brick is only blended from the private buffer onto the shared buffer once all the bricks it depends on are in the completed work queue. Once the brick is rendered, it is also placed in the completed work queue.

- The second level of parallelism exists between the nodes and uses sort-last rendering to dispatch the work. The data is partitioned between the nodes following a KD-tree decomposition. We use a space-coherent decomposition as it is less likely to result in evenly-distributed depth in the rendered scenes, as was proved in [CH93], and therefore provides the better inter-node occlusion speedups. Notice that although intra-node load balancing happens, no load balancing happens between separate nodes, as the cost of moving data over the network is very high. An optimized implementation of direct-send compositing is used to combine the intermediate pictures. Essentially, each of the nodes is seen as a single entity from the point of view of the direct send compositing system.

The rendering is done on the processors by a volume ray-caster which uses pre-integration and shading with pre-computed gradients. In order to cull invisible data, we make use of bricking. The memory layout is also brick based: each of the data bricks (including the associated gradient) uses a separate linear area of contiguous memory which ensures cache-friendly memory accesses as each brick is rendered. We use tri-linear interpolation of the scalar and gradient values which provides good rendering quality, although it results in longer rendering times, mainly because of the increased memory access costs. The rendering itself works as follows: First a processor takes a brick from the work queue. The footprint of this brick is projected onto the screen. For each pixel of this footprint a ray is cast through the brick and the result is accumulated to the private render buffer. Notice that for completely transparent samples, it is not necessary to compute the shading function, nor the interpolated gradient value. Therefore, as a performance optimization we skip this step if the look-up into the pre-integration table for the entry and exit scalar values results in a completely transparent sample. After the whole brick has been rendered to the private buffer, the dependencies with other bricks are checked and the brick is written back to the shared buffer.

The compositing stage of our parallel pipeline is an optimized direct-send implementation. We make use of the bounding rectangle optimization, by which the visible footprint of the current node's data is projected into screen space and only this specific area is shared with other processors. We also experimented with the use of the LZO compression as a means of reducing the amount of network data.

Most of the existing sort-last rendering pipelines use a quantization of intermediate pictures into four 8-bit RGBA values before sending them over the network and before compositing. Even if the rest of the rendering is done at full floating point accuracy, this means that a quantization step happens once per node. In the case of sort-last rendering, this has the unwanted side-effect of lowering the final rendering quality as the number of quantization steps increases with the number of rendering nodes. This in turn means that the quality of the final pictures lowers as the scale of the rendering job increases, which makes high-scale parallel visualization less interesting. Instead, we decided in this paper to use floating point quantities for all the intermediate computations of our rendering pipeline, and only use a single quantization step at the very end of the pipeline into 8-bit RGBA values for final display. Therefore, all the intermediate pictures used for rendering and compositing are stored in full floating point accuracy. However, this incurs an additional cost; in particular, floating point accuracy quadruples the network bandwidth usage when compared to 8-bit values, and the image compositing process also requires four times the memory bandwidth as for 8-bit data. Notice that we quantize the composited images to 8 bit and drop the alpha channel at the last step of direct-send, right after compositing and just before sending the final pictures to the master

© The Eurographics Association 2010.

node for display. This quantization operation could of course be done on the master node, but doing so on the rendering nodes lowers network bandwidth usage for the last communication stage while maintaining the same final quality.

3.2. Cross-processor occlusion

When rendering 3D images, occlusion commonly happens between different areas. Figure 2 depicts the case of sort-last rendering where the rendering from two processors occludes the rendering from two other processors. Although such information can be trivially exploited to cull away rendering in the sequential rendering case, we want to extended it to the sort-last rendering case. To do so, we make use of cross-processor occlusion at two different levels matching the two rendering levels seen previously in Subsection 3.1:

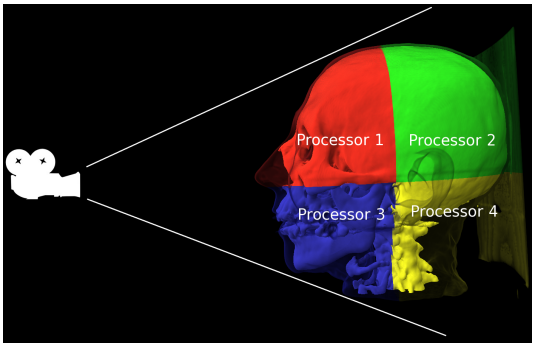

Figure 2: *Occlusion between different nodes in a sort-last context where each processor is assigned a separate part of the dataset. In this example, rendering from processors 1 and 3 occludes rendering from processors 2 and 4, respectively.*

- Inside a given node, all processors render in a front-to-back fashion to a shared buffer. Since rendering is sorted front-to-back, it is trivial to add cross-processor occlusion at this level by considering the opacity value of the current pixel in the shared buffer and discarding the ray computation if the opacity is maximal. Before casting a ray for each pixel of a given brick, the render threads read the current render buffer opacity. If it is currently opaque, the pixel computation is skipped entirely. This is depicted on Figure 3.
- Between different nodes, processors exchange occlusion information and use it to cull away rendering work which would otherwise result in invisible contributions. The occlusion information exchange is realized by having the render threads send messages to the communication threads as they find opaque pixels. After a number of such messages (we use a threshold of 100 messages in our implementation), the communication thread in turn sends the occlusion information to other nodes. To do this, the communication thread packs together the occlusion information from multiple pixels by iterating the shared render buffer and creating occlusion spans (horizontal lines of occluding pixels encoded using the screen-space coordinates of the first pixel and the length of the span). These occlusion spans are in turn sent to all concerned nodes, i.e. nodes which render a part of the data which falls into the footprint of the spans. The other communication threads then receive this occlusion information and update the node's render buffer accordingly, by writing a maximal alpha value for every pixel in the span footprint. The occlusion information is then available to the render threads, and will be used next time they look at the render buffer. Figure 4 depicts this situation.

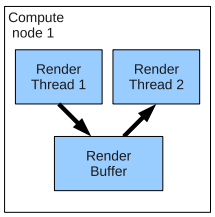

Figure 3: *Propagating occlusion information inside a single node. One processor writes the opacity information to the shared buffer, and other processors can read this information directly and discard their rendering work accordingly.*

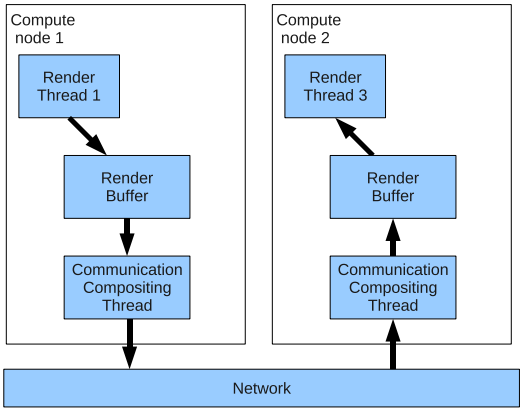

Figure 4: *Propagating occlusion information across different nodes of a cluster. The communication thread receives messages from the render thread. After a number of messages, the communication thread reads the occlusion information from the render buffer and sends it in the form of a list of occluded spans to other nodes. This information is picked up by the communication thread which writes it to the render buffer by making the corresponding pixels opaque. The render threads can then read the occlusion information from the render buffer the same way as with intra-node occlusion.*

4. Results

We first benchmarked our volume rendering system on a single node using a simple scene ($256 \times 256 \times 256$ voxels and a

© The Eurographics Association 2010.

step size of 0.5 voxels) to assess the scalability of the intra-node volume rendering and load-balancing technique. We have observed good scalability on a small-size shared memory system with two Xeon E5345 2.33 Ghz processors (for a total of 8 cores): on 1 core, we get a rendering time of 1981 ms, on 2 cores, we get a time of 994 ms (a speedup factor of 1.99); on 4 cores we get 498 ms (a speedup of 3.97); on 8 cores, 249 ms (a 7.95 speedup factor) is observed. Using one of the dual Opteron 252 2.6 Ghz processor nodes from our visualization cluster, we go from 2140 ms on 1 processor to 1074 ms on 2 processors, which is a speedup factor of 1.99. This technique is therefore suitable for use inside a single node of a cluster as it scales well for a small number of processors, ensures intra-node load balancing and does not incur additional communication costs as more processors are added to the node. On top of this, skipping gradient interpolation on fully transparent contributions leads to a speedup factor of approximately 1.5 times (timings on 4 Xeon cores go from 498 ms per frame using naive tri-linear interpolation down to 330 ms per frame when skipping gradient interpolation when possible, for reference rendering using nearest neighbour interpolation takes 219 ms on the same scene).

We have implemented our parallel renderer in C++ and OpenGL. Tests were conducted using 64 processors (32 dual processor nodes) of a Linux cluster. The configuration of a single node is given on Table 1. All renderings were performed at a 1024×768 screen resolution. To compare performance on the visualization cluster, we have experimented with three different datasets: the first one is the time step 1354 of the entropy variable from the supernova dataset (864^3 voxels), the second one is the beetle dataset (512^3 voxels) and the last one is a head CT (256^3 voxels). All these datasets were ray-cast using a 0.25 voxel step size. Figure 13 shows the final renderings obtained using these datasets.

Component	Type
CPU	2×Opteron 252, 2.6Ghz
Memory	4GB
Interconnection	Gigabit Ethernet
Network card	Broadcom BCM5704

Table 1: *Hardware configuration of a cluster node*

Figures 5, 6, 7 and Table 2 show the scalability of our high quality volume rendering pipeline for three different datasets with and without LZO compression of the direct-send communications. For the larger dataset, we obtain a speedup factor of 45 over 64 processors using compression and of 40 without enabling compression. Notice that both the smaller datasets (512^3 beetle and 256^3 head CT) see a 15% frame rate improvement with the addition of compression, which is not seen with a bigger dataset. This is explained by the fact that rendering for these datasets using 64 processors is limited by the 1Gb network bandwidth. Notice that the gap gained from compression increases with the number of nodes, as in that case the network becomes the bottleneck.

Figures 8, 9, 10 and Table 2 demonstrate the speed improvement from using cross-node occlusion information to

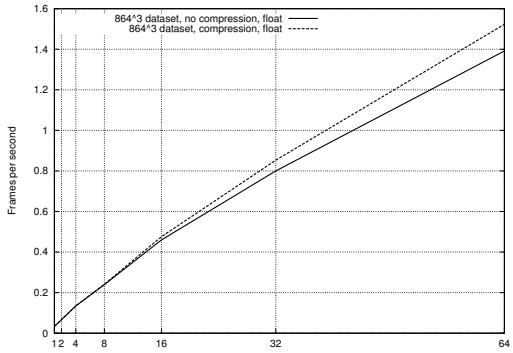

Figure 5: *Comparison of performance with and without compression using the* 864^3 *supernova dataset.*

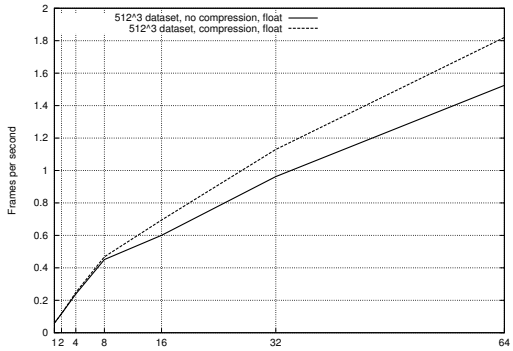

Figure 6: *Comparison of performance with and without compression using the* 512^3 *beetle dataset.*

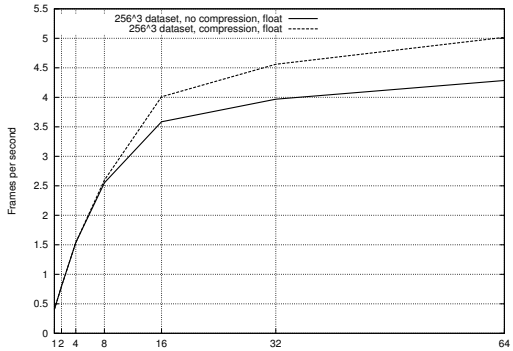

Figure 7: *Comparison of performance with and without compression using the* 256^3 *head CT dataset.*

© The Eurographics Association 2010.

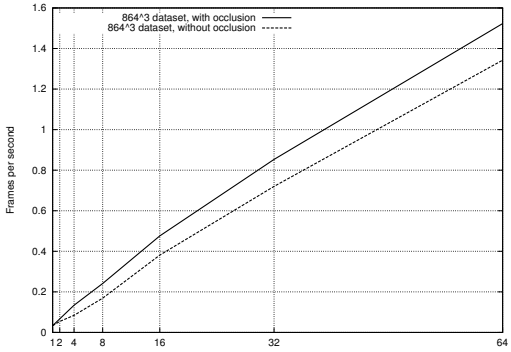

Figure 8: *Comparison of performance with and without cross-processor occlusion using the* 864^3 *supernova dataset.*

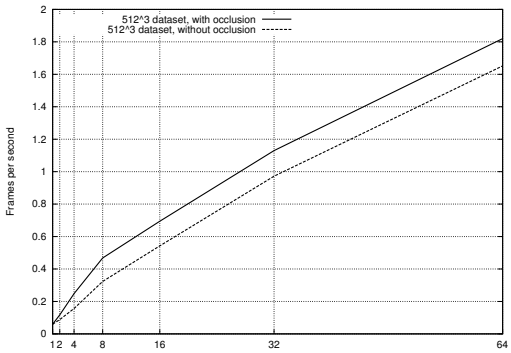

Figure 9: *Comparison of performance with and without cross-processor occlusion using the* 512^3 *beetle dataset.*

cull away invisible areas. Notice that, as opposed to the use of compression, occlusion leads to consistent speedups for any number of processors; in particular, cross-node occlusion contributes to helping performance with as few as two processors. Using cross-node occlusion, we observe consistent speed-ups despite the lack of global load balancing. Intuitively, one could think that because of the lack of global load balancing the speed of the slowest node should remain a limiting factor despite the use of cross-node occlusion, and

Dataset	Supernova	Beetle	Head
With compression, without occlusion	40	25.5	9.5
Without compression with occlusion	41	27.6	10.5
With compression and occlusion	45	30	12.3

Table 2: *Speedup factors using different datasets on 64 processors.*

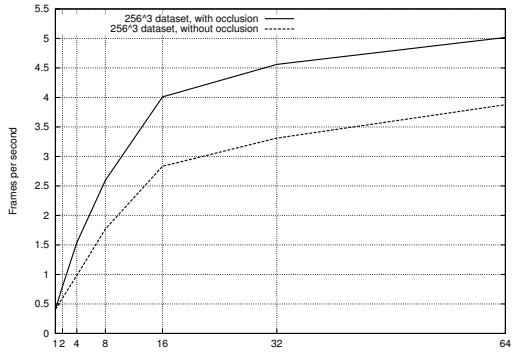

Figure 10: *Comparison of performance with and without cross-processor occlusion using the* 256^3 *head CT dataset.*

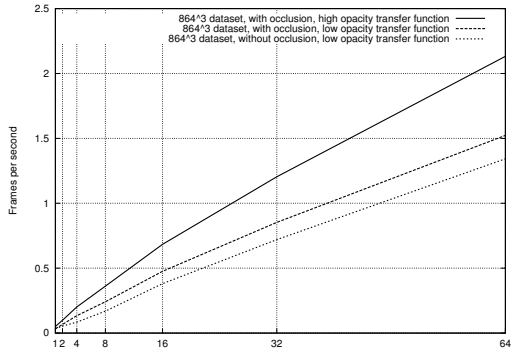

Figure 11: *Impact of the transfer function opacity on the final rendering performance using the* 864^3 *supernova dataset. Two transfer functions are used: one with multiple transparent isosurfaces at* 8% *opacity and one with multiple transparent isosurfaces at* 15% *opacity. For reference, results without cross-processor occlusion are also shown.*

would therefore prevent any type of speedup from our algorithms. However, the nodes which finish their work last will benefit more from the occlusion information than the other nodes, and therefore our scheme can improve unbalanced situations. The impact of the nature of the transfer function on the final results is shown in Figure 11. We measured the performance and scalability of our technique using the same dataset and two transfer functions featuring the same transparent isosurfaces but with different opacities (respectively of 8% and 15%). This figure shows that our cross-processor occlusion algorithm is highly sensitive to different transfer functions, in particular when the opacity changes. In this case, increasing the opacity of the isosurfaces from 8% to 15% increases the frame rate by 30%. The amount of occlusion information exchanged between the processors is shown on Figure 12. These curves are not regular because as the number of nodes increases, the data subdivision fol-

© The Eurographics Association 2010.

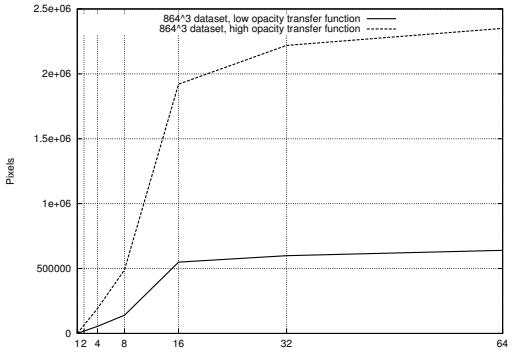

Figure 12: *Amount of occlusion information exchanged over the network (in pixels) for the Supernova dataset using the two different transfer functions.*

lows a KD-tree pattern alternating the splitting dimensions; whenever the added splitting direction is parallel to the viewing plane, this results in a lot of additional communication between the nodes to exchange the corresponding occlusion information.

5. Conclusions and future works

We have presented a high quality volume rendering pipeline making use of multi-level parallelism, intra-node load balancing and inter-processor occlusion to improve the frame rendering times.

We think our work could see many relevant future enhancements. First, we would like to extend the occlusion framework to support a distributed level of detail approach: if an area is partially occluded and therefore contributes only little information to the final pictures, a lower level of rendering detail can definitely be used while keeping a controlled level of rendering quality.

Second, we would like to experiment with strategies to compute the contributions for the areas most likely to occlude other parts of the data first. This is not taken into account in our current algorithm and would lead to better efficiency in our rendering pipeline.

Finally, we would like to conduct larger scale testing, in particular using a bigger number of processors per node and more importantly a better interconnection network. We have shown that our intra-node volume rendering algorithm scales up to 8 processors, and we think that multi-level volume rendering algorithms like the one we presented can lead to better scalability at large scales because they are better suited to the actual hardware architecture. However, increasing the number of nodes may require improvements to the communication schemes used, especially for the occlusion information exchange stage. Instead of sending occlusion information to all the potentially occluded nodes, we would like to find scalable, hierarchical schemes which gradually propagate the occlusion information throughout the whole cluster without incurring too much additional communication.

6. Acknowledgments

This research was supported in part by the U.S. National Science Foundation through grants OCI-0325934, OCI-0749217, OCI-0749227, and OCI-0905008, and the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777. The supernova dataset was provided by John Blondin at North Carolina State University.

References

[CH92] COX M., HANRAHAN P.: Depth complexity in object-parallel graphics architectures. In *Proceedings of the Seventh Workshop on Graphics Hardware, Eurographics Technical Report Series, ISSN* (1992), pp. 1017–4656.

[CH93] COX M., HANRAHAN P.: *Evenly Distributed Depth is the Worst for Distributed Snooping*. Tech. rep., 1993.

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), ACM Press, pp. 9–16.

[GBKG04] GRIMM S., BRUCKNER S., KANITSAR A., GRÖLLER M. E.: Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In *Proceedings IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics* (Oct. 2004), D. Silver T. Ertl C. S., (Ed.), pp. 1–8.

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility culling using plenoptic opacity functions for large volume visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 45.

[GS01] GAO J., SHEN H.-W.: Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics* (Piscataway, NJ, USA, 2001), IEEE Press, pp. 67–74.

[GS04] GUTHE S., STRASSER W.: Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics 28*, 1 (Feb. 2004), 51–58.

[Hsu93] HSU W. M.: Segmented ray casting for data parallel volume rendering. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering* (New York, NY, USA, 1993), ACM Press, pp. 7–14.

[LHJ99] LAMAR E., HAMANN B., JOY K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the IEEE Visualization conference* (1999), D. Ebert M. G., Hamann B., (Eds.), pp. 355–362.

[LHJ03] LAMAR E. C., HAMANN B., JOY K. I.: *Efficient Error Calculation for Multiresolution Texture-Based Volume Visualization*. Springer-Verlag, Heidelberg, Germany, 2003, pp. 51–62.

[LMS*01] LOMBEYDA S., MOLL L., SHAND M., BREEN D., HEIRICH A.: Scalable interactive volume rendering using off-the-shelf components. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics* (Piscataway, NJ, USA, 2001), IEEE Press, pp. 115–121.

© The Eurographics Association 2010.

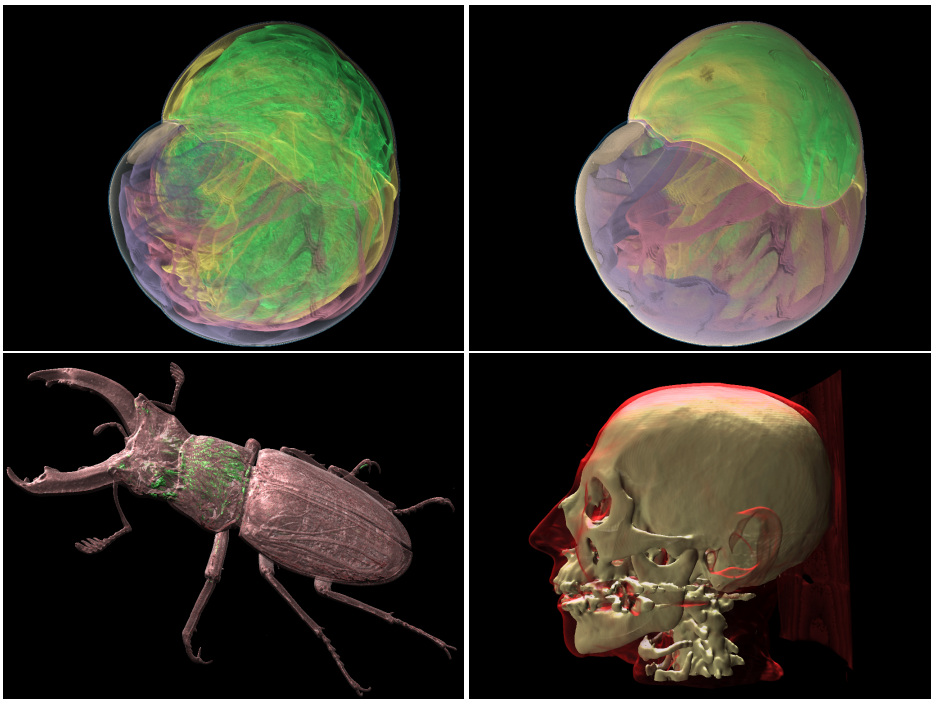

Figure 13: *Sample renderings of the* 864^3 *supernova dataset with both transfer functions,* 512^3 *beetle dataset and* 256^3 *head CT dataset used for the performance comparisons.*

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *IEEE Comput. Graph. Appl. 14*, 4 (1994), 23–32.

[MMD06] MARCHESIN S., MONGENET C., DISCHLER J.: Dynamic Load Balancing for Parallel Volume Rendering. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2006), Eurographics Association, pp. 43–50.

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl. 14*, 4 (1994), 59–68.

[MSE06] MÜLLER C., STRENGERT M., ERTL T.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2006), Eurographics Association, pp. 59–66.

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A configurable algorithm for parallel image-compositing applications. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), ACM, pp. 1–10.

[PYRM08] PETERKA T., YU H., ROSS R., MA K.-L.: Parallel volume rendering on the ibm blue gene/p. In *Proceedings of Eurographics Parallel Graphics and Visualization Symposium (EGPGV 2008)* (April 2008), pp. 73–80.

[RKE00] RÖTTGER S., KRAUS M., ERTL T.: Hardware-accelerated volume and isosurface rendering based on cell-projection. In *VIS '00: Proceedings of the conference on Visualization '00* (Los Alamitos, CA, USA, 2000), IEEE Computer Society Press, pp. 109–116.

[Sab88] SABELLA P.: A rendering algorithm for visualizing 3d scalar fields. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), ACM, pp. 51–58.

[SML*03] STOMPEL A., MA K.-L., LUM E. B., AHRENS J., PATCHETT J.: SLIC: Scheduled linear image compositing for parallel volume rendering. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Washington, DC, USA, 2003), IEEE Computer Society, p. 6.

[SMW*04] STRENGERT M., MAGALLÓN M., WEISKOPF D., GUTHE S., ERTL T.: Hierarchical visualization and compression of large volume datasets using gpu clusters. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2004), pp. 41–48.

[WGS04] WANG C., GAO J., SHEN H.-W.: Parallel multiresolution volume rendering of large data sets with error-guided load balancing. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2004), pp. 23–30.

[WPLM01] WYLIE B., PAVLAKOS C., LEWIS V., MORELAND K.: Scalable rendering on pc clusters. vol. 21, IEEE Computer Society Press, pp. 62–70.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3d textures. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization* (New York, NY, USA, 2000), ACM Press, pp. 7–13.

[YWM08] YU H., WANG C., MA K.-L.: Massively parallel volume rendering using 2-3 swap image compositing. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (Piscataway, NJ, USA, 2008), IEEE Press, pp. 1–11.

© The Eurographics Association 2010.