Distributed Out-of-Core Approach for In-Situ Volume Rendering of Massive Dataset

Jonathan Sarton, Yannick Rémion, Laurent Lucas

Distributed out-of-core approach for in-situ volume rendering of massive dataset

J. Sarton, Y. Remion, and L. Lucas

Université de Reims Champagne-Ardenne, CReSTIC, France
{jonathan.sarton,yannick.remion,laurent.lucas}@uni-reims.fr

Abstract. This paper proposes a method that allows a fluid remote interactive visualization of a terabytes volume on a conventional workstation co-located with the acquisition devices, leveraging remote high performance computing resources. We provide a study of the behavior of an out-of-core volume renderer, using a virtual addressing system with interactive data streaming, in a distributed environment. The method implements a sort-last volume renderer with a multi-resolution ray-guided approach to visualize very large volumes of data thanks to an hybrid multi-GPUs, multi-CPUs single node rendering server.

Keywords: Interactive visualization · large volume data · out-of-core · multi GPUs · biomedical imaging

1 Introduction

Several scientific fields rely on the visualization of data represented in the form of 3D regular voxel grids. The characteristics of this representation are particularly well suited to the architecture of modern GPUs. These have become essential to benefit from an interactive visualization of this type of data, with a good rendering accuracy. However, the increase in the size of volume data in science, and particularly in biomedical imaging, is very rapid. The latter is faster than the increase in the physical capacity of the on-board memory on GPUs. One solution to address this problem is to design an out-of-core approach that allows on-demand data streaming of small chunks (bricks) of a large volume to the GPU during interactive visualization. Although these methods are now effective, data transfers to the GPU are still a bottleneck and can be restrictive to allow a fully pleasant interactive visualization. High-performance computing environments can help to reduce this bottleneck. In addition to offering an increase in computing power, useful for interactivity and visualization quality, they increase storage capacities on several GPUs and allow to distribute the load of transfers. The presented method propose a distributed out-of-core management that offer interactive data streaming on heterogeneous node, to allow scientists to visualize large volume data directly after their acquisition from their usual workstation.

Contributions: The use of an out-of-core approach with a virtual addressing system based on a page table for multi-resolution volume is particularly well

suited for the visualization of very massive volume data [7]. However, to our knowledge, any work has been carried out on the possibilities of using this structure in high-performance computing environments for all stages of the pipeline allowing in-situ visualization. In this paper, we present a distribution method to use such out-of-core data structure in distributed environments, on single multi-CPUs, multi-GPUs compute nodes. We address the distribution of virtual addressing space in very large volumes, in line with a distributed sort-last rendering approach for volume ray-casting. Our system is composed of a:

- GPU-based multi-resolution volume ray-casting with a ray-guided approach to render large volume data ;
- complete out-of-core pipeline from disk to GPU with a multi-level, multi-resolution page table hierarchy, completely managed on GPU ;
- multi-GPUs sort-last volume renderer and a distributed strategy of the out-of-core solution on a multi-CPUs, multi-GPUs single node server;
- remote solution to display, on a thin client, a high frame rate coming from a high-performance rendering server.

2 Related work

Volume ray-casting, introduced and improved by Levoy [9,10], is nowadays the most intuitive and efficient approach to apprehend direct volume rendering and solve the volume rendering equation [12]. Its implementation on GPUs was first proposed in 2003 [17,8] and then improved in 2005 [20,19], taking advantage of hardware advances and introducing optimization techniques.

Afterward, some work has focused on out-of-core approaches for rendering of large volumes that exceed the amount of memory available on the GPU and even on the CPU. Gobbetti *et al.* [6] were among the first to propose an out-of-core approach for volume ray-casting on GPU, based on a multi-resolution octree. Crassin *et al.* [3] also use an octree and provide a very efficient ray-guided pipeline for rendering large voxelized scenes. While many works in this context have focused on a tree structure to address an entire multi-resolution volume with out-of-core storage, Hadwiger *et al.* [7] introduced a hierarchical page table for efficient virtual addressing. They show that their structure is more suitable than an octree for very large volumes of data. Fogal *et al.* [4] provide a detailed analysis of ray-guided out-of-core approaches on GPU. For more details, we can also refer to a complete state of the art [1]. Parallel volume rendering has also been studied to distribute the computational workload over different units and thus operate on multi-GPUs architectures for efficient rendering [21]. Molnar *et al.* [13] proposed a classification of parallel rendering methods. Although some work in multi-GPUs volume rendering is focused on a sort-first approach [14], most methods are oriented towards a sort-last approach [15,11]. In addition to providing a computational scaling, the latter also provides a memory management scaling. This aspect is the main bottleneck in an on-demand streaming context for interactive rendering of very large volumes of data. Finally, there are very few methods that combine multi-GPUs rendering with an out-of-core

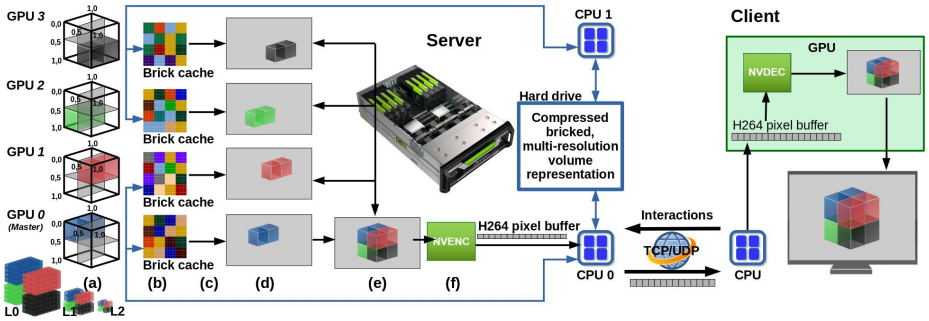


Fig. 1: **Pipeline overview.** Example of our remote out-of-core rendering pipeline on a 4-GPUs and 2-CPUs server. This pipeline includes the following steps: (a) Virtual multi-resolution volume distribution (b) Out-of-core virtual addressing structure and brick cache (c) Ray-guided volume renderer (d) Local rendering pixel buffer (e) GPUs communications for sort-last compositing (f) H.264 GPU compression.

approach. Fogal *et al.* [5] propose an out-of-core volume renderer with an MPI-based compositing to render large datasets on multi-node clusters up to 256 GPUs. Beyer *et al.* [2] present a distributed version on 4 GPUs of an octree-based virtual addressing system to visualize a very large volume.

Compared with these works, we propose to introduce a distribution method of an out-of-core approach based on an efficient GPU virtual addressing and caching system [18] on multi-GPUs, multi-CPUs single node server. Combined with a ray-guided volume renderer with sort-last compositing for interactive visualization on datasets exceeding terabytes.

3 Our method

3.1 Out-of-core approach

Figure 1 summarizes the proposed pipeline. It includes an out-of-core data management in order to handle very large volumes that can exceed the amount of GPU and CPU memory. It is based on a virtual addressing system using a multi-level multi-resolution page table hierarchy and a brick cache on GPU. The underlying structure, implemented in GPU texture memory, is fully maintained on the GPU to optimize the communication load with the CPU [18]. This system is able to interactively address a whole multi-resolution volume decomposed into small voxel bricks entirely stored on disk with compression. The bricks required for interactive visualization are streamed to the GPU cache when requested by the application. On the GPU, the navigation in the multi-resolution volume is performed in a normalized virtual volume. The full parallel GPU implementation of our virtual addressing structure management includes a brick request manager. This one takes care of sending to the CPU a list containing the IDs

of the bricks required by the interactive application during its whole execution. The streaming of these bricks to the GPU cache is performed asynchronously with the GPU visualization step in order to maintain a constant interactivity for the user. A dedicated CPU thread manages the lists of the required bricks to provide them to the GPU after reading and decompressing. Unlike [7] and [18], we are interested here in the distribution of this out-of-core management solution on a heterogeneous node that includes several GPUs and CPUs with an approach including remote rendering allowing a complete in-situ visualization pipeline.

3.2 Sort-last distributed ray-casting

The bottleneck of a distributed renderer combined with an out-of-core system lies in the heavy loads of the voxel bricks, streamed to the different rendering units, and in the updates of their associated GPU caches. In this context, a sort-last approach is more appropriate than a sort-first approach to minimize these issues. By pre-determining a sub-domain of the volume on each GPU, this allows limited CPU/GPU and GPU/GPU data transfers and avoids cache flushes.

In association with our sort-last approach, we propose to use a multi-resolution ray-guided volume renderer that efficiently and naturally integrates our out-of-core loading method. This approach ensures to load only the useful bricks on the different GPUs, according to the sampling along the rays in the volume. Our renderer is also suitable for the multi-resolution volume representation by choosing an adapted level of detail for each sample based on its distance to the camera. This allows us to adapt the sampling step, and thus reduce the number of texture memory lookup for areas containing less detail.

In our system, the compositing step, involved by the sort-last rendering, is implemented on GPU with a basic approach applying the OVER operator [16] in front-to-back. At each rendering pass, the local pixel buffer of each GPU is communicated to the pre-defined master GPU, in charge of the compositing. We propose to initiate these transfers from the master GPU, with peer-to-peer communications using CUDA Unified Virtual Addressing system if possible, i.e. if the GPUs are in the same addressing space as the master GPU. Otherwise, our strategy switches to explicit transfers via the CPUs.

3.3 Multi-GPUs virtual addressing

The volume partitioning is done on a normalized virtual volume, used on each GPU to navigate through all the multi-resolution volume and to virtually access to any voxels (Fig. 1(a)). This distribution only consists of restricting the addressing range in the normalized virtual volume. This method makes it possible to implicitly distribute the entire multi-resolution representation of the volume. The virtual addressing structure, used to ensure access to all voxels of a very large volume, is distributed as follows. Each GPU has its own instance of this structure with its own brick cache, completely independent of other GPUs. However, a single virtualization configuration is used for all the GPUs. Thus, the size

of the bricks, blocks in the page table and the number of virtualization levels in the structure do not differ from one unit to the other. We also implement a single common CPU brick cache. We use a multi-threaded strategy with OpenMP on the different CPU(s) of the server to communicate with all the GPUs. For each of them, the corresponding CUDA context is linked to one and only one OpenMP thread. The affinity of each thread is essential and is achieved with respect to the CPUs locations according to the server topology. Thus, a thread in charge of communicating with a GPU is necessarily placed on a physical core of the CPU that has a direct physical link with that GPU. At the initialization step, each GPU is responsible for creating all the necessary resources, especially allocating its brick cache and its page table. Then, each OpenMP thread first communicates to its associated GPU the addressing subspace of the volume on which it must perform its part of the rendering. Each thread can then independently manage the requests and the streaming of the bricks to its associated GPU during the whole duration of the execution of the interactive application.

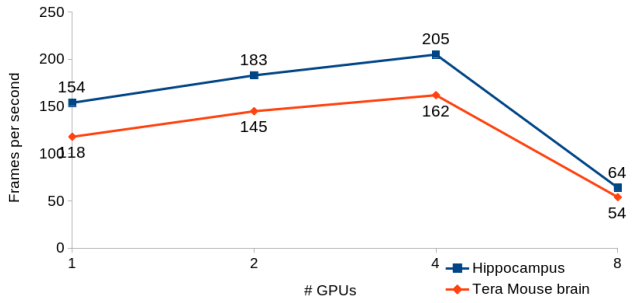
3.4 Remote rendering

In order to exploit resources offered by high-performance computing environments from lightweight devices, we propose to integrate a remote visualization system in our method. Thus, after each rendering and compositing pass, the resulted pixel buffer is directly compressed into an H.264 stream on a designated master GPU of the rendering server node. Then, the compressed buffer is copied to the CPU before being sent over the network. A thin client that receives the content of this H.264 stream, decompress it and simply display the result in an interactive display environment. This configuration allows us to provide a single pipeline that can be seen as an in-situ solution in the sense that the visualization can be done on a standard PC in place of the data acquisition. However, this requires sending all the acquired data to the remote rendering server where they are then handled for a pre-processing step. This step only includes the creation of the bricked multi-resolution representation of the 3D volume. The creation of page table entries is done on the fly as needed during the visualization. The initialization of our virtual addressing structure on the different GPUs only consists of allocating the necessary texture memory space for the different cache levels.

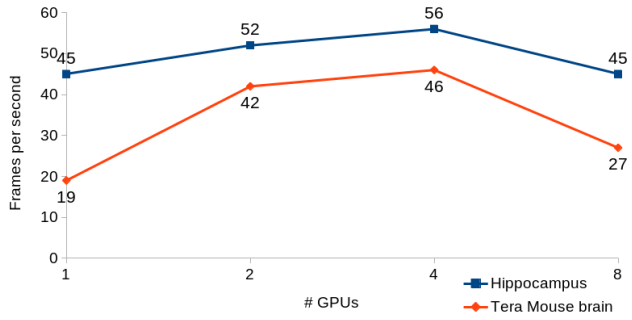
4 Results

The results presented here, were obtained on a NVidia Quadro VCA rendering server. It consists of a single computing node containing eight Quadro P6000 GPUs with 24 GB of VRAM, two CPUs Intel Xeon E5-2698 2.2 GHz and 256 GB of RAM. The display is performed on a full HD viewport 1920×1080. To illustrate the results, we used two datasets :

- *Hippocampus:* a $2160 \times 2560 \times 1072$ volume with grayscale 16-bits voxels (11.8 GB) of a primate hypocampus from a light sheet microscope.



(a)



(b)

Fig. 2: **Out-of-core multi-GPUs volume ray-casting FPS.** Average FPS according to two different zoom level for both data sets. (a) For a medium zoom at an intermediate level of detail (see figure 3(a, c)). (b) For a high zoom level with a highly detailed full-screen volume (see figure 3(b, d)). The averages are calculated on measurements taken over the entire duration of a scenario of several camera rotations around the volume.

– *Tera Mouse Brain:* a $64000 \times 50000 \times 114$ volume with RGBA 32-bits voxels (1.45 TB) of a histological slices stack of a mouse brain.

The evaluation is based on these two datasets with 64^3 voxels bricks. In addition, the bricks of the *Tera Mouse brain* volume are stored on disk, cached on CPU and transferred to the GPU with 256^3 voxels, which includes 4^3 bricks of 64^3 voxels.

4.1 Display frequency

Figure 2 shows the average FPS, based on the number of GPUs used, for two different zoom level. These results were obtained by measuring the number of

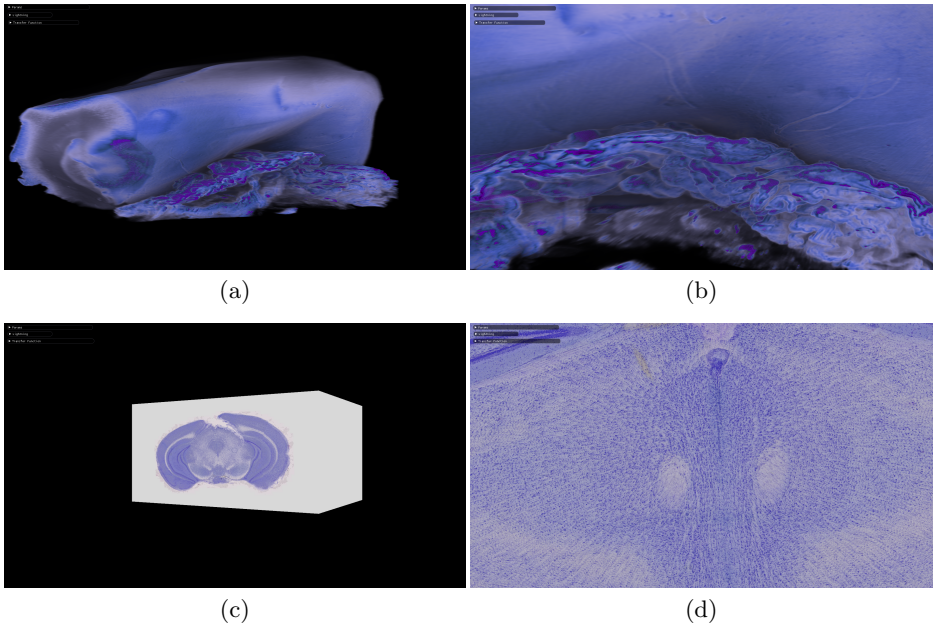


Fig. 3: **Rendering illustrations.** Rendering of the *Hippocampus* dataset (a, b) and the *Tera Mouse brain* dataset (c, d) at two different level of detail.

frames per second over the entire duration of an interactive scenario involving several rotations of an orbital camera around the volume. The results are, among other things, dependent on the defined transfer function. Here, the transfer functions are chosen to provide an interesting visualization independently for both datasets.

First, we can note that the behavior of our system is the same for both datasets and, the number of FPS are almost the same. This allows us to show that the performance of our approach does not depend on the input size of the volume to be visualized. The difference between the FPS for the two datasets is actually due to the different transfer function used. With a single GPU, we already achieve interactive visualization time. We also see the gain brought by the multi-GPUs system for the global rendering time. However, there is a significant drop in performance beyond four GPUs. It is due to the topology of the server node where four GPUs are connected to one CPU while the other four are connected to another CPU. According to this topology, four GPUs are not in the same address space as the master one. In this scenario, it is not possible to take advantage of direct GPU to GPU communications with CUDA peer-to-peer and UVA, for these GPUs. The communications with the master GPU are therefore made by explicit transfers, passing through the CPU and crossing a QPI port between the two processors. In order to validate this analysis, figure 4, provides details of the different steps required for rendering. We can see the impact of

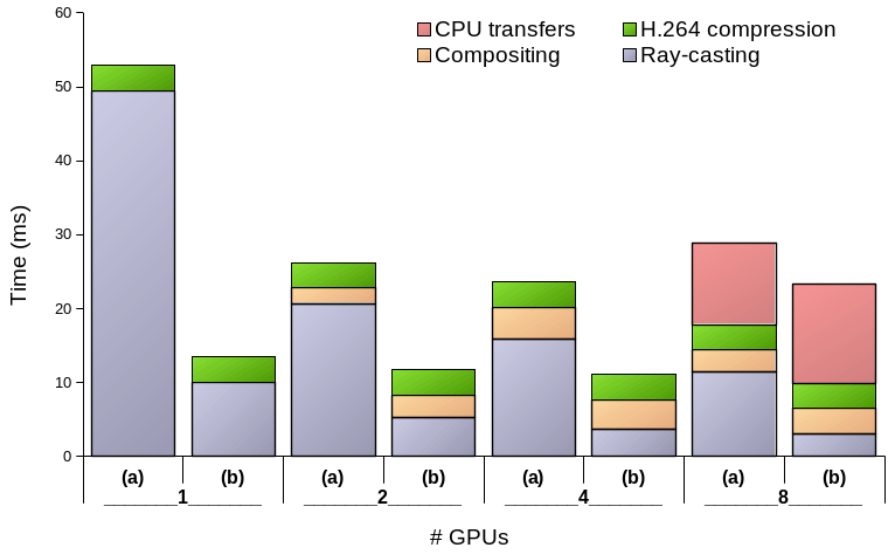


Fig. 4: **Analysis of the different step of the rendering.** These measurements represent the time required for the different rendering steps according to the number of GPUs for the *Tera Mouse brain* dataset. (a) For a high zoom with full-screen volume display at a high level of detail (see figure 3(d)). (b) For a medium zoom with an intermediate level of detail (see figure 3(c))

these transfers on the overall rendering time. This behavior could be fixed by using compute nodes that provide direct communication between all GPUs, with the NVLink technology of NVidia for instance.

4.2 Data loading

Figure 5 shows the results obtained by evaluating the data loading time in our multi-GPUs out-of-core management context. This evaluation is done on the loading of all the needed bricks of a complete HD view, in a worst-case scenario, from empty GPU and CPU caches. All the data are therefore only present on the server disk and stored with compression. The trend of these curves shows that the data loading time is greatly reduced by the number of GPUs used for rendering, from few seconds for a single GPU to one second or less with eight GPUs. Indeed, our sort-last rendering solution allows distributing the bricks loading on the different GPUs. Moreover, these values are once again mostly the same for both datasets although their dimensions are very different.

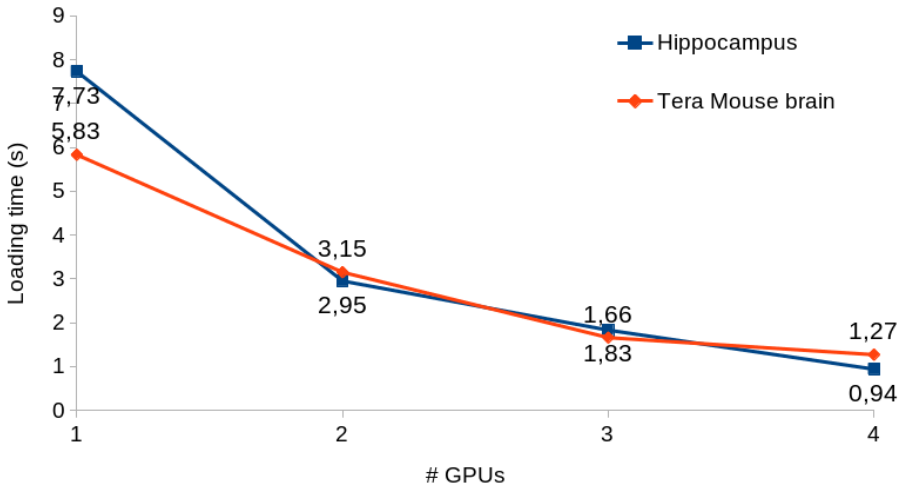


Fig. 5: **Loading time of a complete HD viewport in a worst-case scenario.** These measurements represent the loading times of all the bricks required to generate the complete HD viewport presented in figure 3(b) and 3(d), based on the number of GPUs used, for the two datasets. They are obtained in a worst-case scenario, with empty GPU and CPU caches.

5 Conclusion and Future Work

In this paper, we have presented a method that proposes a solution to allow in-situ visualization of high dimension volume data. We use an out-of-core approach, based on an efficient virtual addressing system and a GPU caching mechanism, to handle very large volumes of data, in a context of remote parallel volume rendering. The distribution of our out-of-core model is combined with a sort-last ray-guided volume renderer in a multi-GPUs single node environment. This method reduces the main bottleneck of on-demand data streaming, from mass storage to GPU, used by out-of-core approaches to handle data exceeding GPU and CPU memory. The overall data block loading response is accelerated by distributing the charge across all GPUs. In addition, the balancing of the rendering workload allows a smooth interactive visualization. These results are observed on data exceeding terabyte on a server with 8 GPUs. The use of a multi-resolution page table, introduced by Hadwiger for out-of-core addressing of large volumes of data on GPUs, is particularly well suited for very large volumes of data. Moreover, we can conclude with this paper that this structure is also efficient and particularly well suited for the use in a distributed environment on a multi-GPU multi-CPU single node and that its behavior allows to visualize volumes exceeding the terabyte remotely on standard PC. The proposed out-of-core structure opens up a broad application framework allowing complete support of the all pipeline from data acquisition through processing to

interactive visualization. We could then consider performing all these steps with our virtual addressing structure to propose an efficace in-situ solution allowing to remotely visualize the data during their acquisition and not only after their acquisition.

Acknowledgments

This work is supported by the French national funds (PIA2'program "Intensive Computing and Numerical Simulation" call) under contract No. P112331-3422142 (***3D**Neuro**Secure*** project). We would like to thank all the partners of the consortium led by Neoxia, the three French clusters (Cap Digital, Systematic and Medicen), Thierry Delzescaux and the Mircen team (CEA, France) for the two datasets as well as the Centre Image of the University of Reims for the VCA server used.

References

1. Beyer, J., Hadwiger, M., Pfister, H.: State-of-the-Art in GPU-Based Large-Scale Volume Visualization. Computer Graphics Forum **34**(8), 13–37 (2015). https://doi.org/10.1111/cgf.12605, `http://dx.doi.org/10.1111/cgf.12605`
2. Beyer, J., Hadwiger, M., Schneider, J., Jeong, W.K., Pfister, H.: Distributed terascale volume visualization using distributed shared virtual memory. In: 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV). pp. 127–128 (Oct 2011). https://doi.org/10.1109/LDAV.2011.6092332, ****
3. Crassin, C., Neyret, F., Lefebvre, S., Eisemann, E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games. pp. 15–22. ACM (2009), `http://dl.acm.org/citation.cfm?id=1507152`
4. Fogal, T., Schiewe, A., Kruger, J.: An analysis of scalable GPU-based ray-guided volume rendering. In: 2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV). pp. 43–51 (Oct 2013). https://doi.org/10.1109/LDAV.2013.6675157
5. Fogal, T., Childs, H., Shankar, S., Krüger, J., Bergeron, R.D., Hatcher, P.: Large Data Visualization on Distributed Memory multi-GPU Clusters. In: Proceedings of the Conference on High Performance Graphics. pp. 57–66. HPG '10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2010), `http://dl.acm.org/citation.cfm?id=1921479.1921489`, *
6. Gobbetti, E., Marton, F., Guitián, J.A.I.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. The Visual Computer **24**(7-9), 797–806 (Jun 2008). https://doi.org/10.1007/s00371-008-0261-9, `http://link.springer.com/article/10.1007/s00371-008-0261-9`
7. Hadwiger, M., Beyer, J., Jeong, W.K., Pfister, H.: Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2285–2294 (Dec 2012). https://doi.org/10.1109/TVCG.2012.240
8. Kruger, J., Westermann, R.: Acceleration Techniques for GPU-based Volume Rendering. In: Proceedings of the 14th IEEE Visualization 2003

(VIS'03). pp. 38–. VIS '03, IEEE Computer Society, Washington, DC, USA (2003). https://doi.org/10.1109/VIS.2003.10001, http://dx.doi.org/10.1109/VIS.2003.10001
9. Levoy, M.: Display of surfaces from volume data. IEEE Computer Graphics and Applications **8**(3), 29–37 (May 1988). https://doi.org/10.1109/38.511
10. Levoy, M.: Efficient Ray Tracing of Volume Data. ACM Trans. Graph. **9**(3), 245–261 (Jul 1990). https://doi.org/10.1145/78964.78965, http://doi.acm.org/10.1145/78964.78965
11. Marchesin, S., Mongenet, C., Dischler, J.M.: Multi-GPU Sort-last Volume Visualization. In: Proceedings of the 8th Eurographics Conference on Parallel Graphics and Visualization. pp. 1–8. EGPGV '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008). https://doi.org/10.2312/EGPGV/EGPGV08/001-008, http://dx.doi.org/10.2312/EGPGV/EGPGV08/001-008
12. Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics **1**(2), 99–108 (Jun 1995). https://doi.org/10.1109/2945.468400
13. Molnar, S., Cox, M., Ellsworth, D., Fuchs, H.: A sorting classification of parallel rendering. Ieee Computer Graphics and Applications pp. 23–32 (1994)
14. Moloney, B., Ament, M., Weiskopf, D., Moller, T.: Sort-First Parallel Volume Rendering. IEEE Transactions on Visualization and Computer Graphics **17**(8), 1164–1177 (Aug 2011). https://doi.org/10.1109/TVCG.2010.116
15. Müller, C., Strengert, M., Ertl, T.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. The Eurographics Association (2006). https://doi.org/http://dx.doi.org/10.2312/EGPGV/EGPGV06/059-066, https://diglib.eg.org:443/xmlui/handle/10.2312/EGPGV.EGPGV06.059-066
16. Porter, T., Duff, T.: Compositing Digital Images. In: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques. pp. 253–259. SIGGRAPH '84, ACM, New York, NY, USA (1984). https://doi.org/10.1145/800031.808606, http://doi.acm.org/10.1145/800031.808606
17. Roettger, S., Guthe, S., Weiskopf, D., Ertl, T., Strasser, W.: Smart Hardware-Accelerated Volume Rendering p. 9 (2003)
18. Sarton, J., Courilleau, N., Remion, Y., Lucas, L.: Interactive Visualization and On-Demand Processing of Large Volume Data: A Fully GPU-Based Out-Of-Core Approach. IEEE Transactions on Visualization and Computer Graphics pp. 1–1 (2019). https://doi.org/10.1109/TVCG.2019.2912752
19. Scharsach, H.: Advanced GPU Raycasting. pp. 69–76 (2005)
20. Stegmaier, S., Strengert, M., Klein, T., Ertl, T.: A simple and flexible volume rendering framework for graphics-hardware-based raycasting. pp. 187–241 (Jun 2005). https://doi.org/10.1109/VG.2005.194114
21. Zhang, J., Sun, J., Jin, Z., Zhang, Y., Zhai, Q.: Survey of Parallel and Distributed Volume Rendering: Revisited. In: Computational Science and Its Applications – ICCSA 2005. pp. 435–444. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (May 2005). https://doi.org/10.1007/11424857_46, https://link.springer.com/chapter/10.1007/11424857_46