Multi-user predictive rendering on remote multi-GPU clusters

J. Randrianandrasana, A. Chanonier, H. Deleau, T Muller, P. Porral, M. Krajecki, Laurent Lucas

▶ **To cite this version:**

J. Randrianandrasana, A. Chanonier, H. Deleau, T Muller, P. Porral, et al.. Multi-user predictive rendering on remote multi-GPU clusters. IEEE VR International Workshop on Collaborative Virtual Environments (3DCVE), 2018, Reutlingen, Germany. pp.1-4, 10.1109/3DCVE.2018.8637114 . hal-02977423

HAL Id: hal-02977423
https://hal.univ-reims.fr/hal-02977423v1

Submitted on 24 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Multi-user predictive rendering on remote multi-GPU clusters

J. Randrianandrasana[1]* A. Chanonier[2] H. Deleau[1] T. Muller[4,3] P. Porral[4,3] M. Krajecki[1]
L. Lucas[1]

[1]Université de Reims Champagne-Ardenne, CReSTIC EA3804
[2]PSA Peugeot Citroën
[3]PSL-Research University, MINES ParisTech, Centre for Robotics
[4]United Visual Researchers

ABSTRACT

Many stages of the industry workflow have been benefiting from CAD software applications and real-time computer graphics for decades allowing manufacturers to perform team project reviews and assessments while decreasing the need for expensive physical mockups. However, when it comes to the perceived quality of the final product, more sophisticated physically based engines are often preferred though involving huge computation times. In this context, our work aims at reducing this gap by providing a predictive rendering solution leveraging the computing resources offered by modern multi-GPU supercomputers. To that end, we propose a simple static load balancing approach leveraging the stochastic nature of Monte Carlo rendering. Our solution efficiently exploits the available computing resources and addresses the industry collaboration needs by providing a real-time multi-user web access to the virtual mockup.

Index Terms: Human-centered computing—Collaborative interaction; Computer Graphics—Distributed/Network graphics; Computer Graphics—Raytracing, Predictive Rendering

1 INTRODUCTION

During the last two decades, virtual reality has become a promising technology widely embraced by the industry. In the automotive industry workflow, many stages including the vehicle design process, virtual prototyping and manufacturing are already benefiting from virtual reality thus resulting in a shorter time-to-market and a higher product quality [1, 15]. At the same time, the increasing need of mobility and synergy all along the industry pipeline has led to research efforts targeting delocalized collaboration [5] thus opening the way to new perspectives such as networked virtual reality. However, real-time visualization tools including CAD software applications and virtual reality are still facing many challenges when the key concerns are focused on the aesthetic of the final product. Despite recent technological advances, real-time rendering can only be obtained at the price of numerous approximations on *i*) the underlying lighting and material models and *ii*) on the algorithms used to compute light-matter interactions. In practice, such visualization tools can only be used for illustrative purposes and thus, more sophisticated physically based solutions belonging to the predictive rendering field must be used when it comes to the perceived quality of the product. In contrast to the usual rendering methods, predictive rendering attempts to accurately resolve the light transport, allowing one to predict how a virtual scene would look like under the conditions defined by the virtual model [14]. To achieve this expected level of accuracy, predictive-oriented approaches make an exclusive use of spectrally defined materials and light inputs to prevent well-known surface metamerism exhibited by traditional RGB/XYZ-based renderers. However, physically accurate results can only be guaranteed if the spectral and polarized natures of light are both carefully accounted for through unbiased rendering algorithms such as Monte Carlo *path tracing*. As one can expect, such a simulation requires tremendous computational resources preventing any real-time use even with cutting-edge hardware, especially when targeting high definition resolutions. As a consequence, expensive physical mockups are still preferred by decision makers when assessing the perceived quality of the vehicle.

Contribution We propose to fill the gap between the illustrative real-time environments enabling user collaboration such as virtual reality and the computationally expensive simulations such as predictive rendering, with a predictive rendering environment taking full advantage of modern clusters to drastically reduce the rendering time. Our work mainly targets multi-GPU supercomputers frequently found in high-end industries. Additionally, the system enforces remote collaboration all along the industry pipeline by providing a remote web access enabling real-time multi-user interactions with the virtual mockup in a synchronized fashion.

This paper is structured as follows: First, a review of previous work related to remote collaborative high performance graphics is given in section 2, along with the previous work related to distributed rendering. Then, section 3 describes each logical component involved in our solution as well as their relationship. The experimental settings, the results as well as the system limitations are detailed in section 4. We conclude our paper and highlight the future research directions in section 5.

2 RELATED WORK

Remote collaborative high-performance graphics Many research efforts addressing delocalized collaboration have been carried out. Among the previous work addressing both remote collaboration and remote high-performance graphics, Renambot et al. proposed SAGE [10] as a scalable framework enabling collaborative visualization, mainly focused on local collaboration through tiled display systems. The framework supports distributed rendering on remote clusters and high resolution streaming to a variable number of displays. Although remote shared visualization has been partially addressed through the Visualcasting extension [8] by Jeong, important issues such as multi-user interactions at the application level were not addressed. Repplinger et al. proposed DRONE [11] as a flexible framework enabling distributed physically based rendering and supporting various deployment scenarios including remote multi-user visualization and interactions. However, as the project has become outdated, the implementation does not benefit from recent advances including modern cluster interconnections such as InfiniBand and new internet standards such as HTML5 and WebRTC opening the way to ubiquitous cross-platform solutions. Commercial and free systems addressing remote visualization such as NICE DCV and VNC are also available. While these solutions

*e-mail:rasolonjatovo.randrianandrasana@univ-reims.fr

can take advantage of high-performance environments, their support for collaborative scenarios remains poorly limited in terms of mutual feeling of presence and multi-user management as all users share an unique mouse cursor and any user can preempt the remote control anytime irrespectively of the ongoing interaction flow. Other remote visualization solutions such as OTOY, OnLive and PolyStream achieve low latency streaming but do not target collaboration.

Distributed rendering As the computational load may significantly vary between different image areas, one must pay a special attention to the load balancing strategy when designing a distributed rendering system. Otherwise, load imbalances can easily occur and hence penalize the system scalability. The load balancing strategies can globally be classified into three categories [12]. Static load balancers evenly divide the image space and assign each image area to a distinct compute unit. With static balancing, the assignment layout remains the same during the entire simulation. Common approaches consist in dividing the image space into homogeneous tiles or non-contiguous pixel sets spread over the image. Such strategies can be easily implemented but usually result in load imbalances in case of homogeneous tiles [6] and in poor local performances in case of non-contiguous pixel sets [13]. Semi-static load balancers split the image space into nonuniform tiles in order to balance their rendering costs [2, 12]. The layout is updated between each frame based on a high-definition timing map reporting the rendering time of tiny contiguous pixel sets. Unfortunately, the required high-definition timing is not suitable for GPU rendering as the operations are simultaneously performed on wide pixel batches. Dynamic load balancers follow a producer-consumers pattern [7, 13]. The image space is divided into a set of equal size tiles much larger than the compute unit count. When rendering a new frame, each compute unit is assigned one or several tiles from the pool and request new tiles to render as it becomes idle. This process repeats until the whole frame is rendered. While these approaches result in evenly balanced workloads, the central task pool can become a bottleneck as all nodes request new tasks simultaneously hence penalizing the system scalability. As an alternative, we propose an efficient static balancing scheme taking advantage of the stochastic nature of Monte Carlo rendering.

3 System description

As depicted in Figure 1, the whole rendering solution is hosted on the remote cluster and each cluster node is assigned a unique role among the three component classes described below. At the core of the system, the rendering process is carried out by the compute nodes. In this purpose, each compute node runs an instance of the predictive rendering engine. The renderer consists of a spectral path tracer running on the GPU. The server acts as a facade in charge of the compute nodes coordination. This component hides the underlying distributed rendering logic by exposing only a few set of functionalities to the client application which therefore remains entirely agnostic of the actual deployment setup. The server acts as a gateway between the client and the compute nodes. Scene update commands issued by the client such as camera moves are broadcasted to all the compute nodes which in turn update the concerned scene entities on the GPU. The server is also responsible for the image assemblies and post-processing before their delivery to the client application. It is important to note that in a predictive context, the rendering of a single frame may require many seconds to minutes to obtain noise-free images, even within a high-performance environment. Thus, our system follows a *progressive rendering* approach to provide the user with the current simulation progress. The client is a traditional GUI application built around a central event loop. In contrast to standalone applications, the displayed image is here requested to the server at each display refresh and any interaction with the virtual scene such as camera motions and material picking trig-

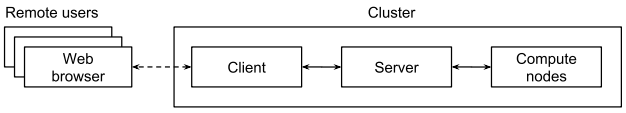

Figure 1: Overview of the solution

gers a server object update. Finally, users interact remotely with the client GUI application through the USE Together infrastructure [9]. With this middleware technology, several users can simultaneously share a native desktop environment or a specific application window from any device running a web browser and supporting the WebRTC standard. In contrast to the server and compute nodes which run on Linux OS, the client application is currently deployed on a distinct node running a Windows OS as USE Together only supports those systems at the moment. However, as USE Together is expected to support Linux OS in the near future, it will remove the unnecessary client-server communication and will allow having both streaming and multi-user management processes directly on the server.

3.1 Distributed rendering

Monte Carlo stochastic rendering is able to capture a scene image as seen from an observer by sampling light paths connecting the observer and the light sources of the scene. The final color of each pixel is then obtained by weighting the color estimates of all the rays passing through the pixel. In a predictive context though, hundreds to thousands of light paths per pixel are required to produce high-quality noise-free images. Our load balancing approach exploits this specificity by distributing the light path sampling tasks of a pixel over several compute nodes. We shall now use the term *compute group* to refer to a set of nodes computing colorimetric estimates for the same image pixels. Following this static path-space division approach yields a naturally well-balanced workload. Moreover, as no runtime communication for task assignment is needed, work starvation is avoided, hence maximizing rendering occupancy. The pixel value is finally obtained at each display request by weighting the current estimates of all the nodes of the group. As the data traffic linearly increases with the number of participating nodes, this compositing step can significantly reduce the system reactivity. However, by combining this static path-space division approach with a static image-space division strategy, interactive compositing rates can be maintained. Thus, our system first divides the image space following an interleaved pixel scheme and then assigns each resulting partition to a distinct compute group. This partitioning step ensures a well-balanced load but can severely affect the performance of modern ray tracers which take advantages of the spatial coherence of the rays to intersect [13]. One must therefore pay attention to the image level-of-division to apply to ensure both local rendering performance and system responsiveness. Figure 2 outlines our hybrid approach through a typical use case of the system. Each compute node starts an instance of the renderer at startup time. The client issues an initialization command to the server (1) supplying the scene to load and some setting parameters such as the rendering resolution. The image space is divided following an interleaved scheme as illustrated on the right side of Figure 2. The server broadcasts the initialization parameters to every compute node and each compute group is assigned a distinct set of pixel partitions to compute. In the depicted example, each compute node embeds two GPUs and the image-space division is defined to eight interleaved partitions. Four compute groups are then constituted, every node of a group computing estimates for the same pixels. Each node then loads the scene on the GPUs and starts an infinite rendering loop (2). The initialization step is completed. The client issues an image request to the server whenever a display update is needed (3). The server forwards the request to all the groups (4) which in return send their respective pixel partitions in their current states of rendering (5). All the pixel partitions are then put together to yield the full image

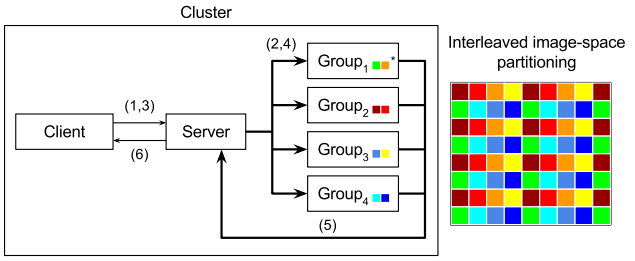

Figure 2: The image space is divided into interleaved pixel partitions (each partition is assigned a distinct color). Each compute group is assigned a distinct set of partitions to compute.

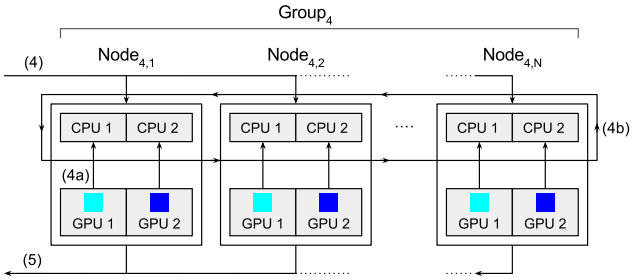

Figure 3: Collaboration of the nodes of a group during an image request

which is ultimately delivered and displayed on the client application (6). Figure 3 highlights the interactions occurring within a compute group when the server is issued an image request. First, a CPU local copy of the pixel estimates residing on the GPU is done by each node of the group (4a). Then, a collective compositing pass involving all the nodes of the group is performed (4b). This compositing step results in average estimates for each pixel value of the local partitions and is implemented with the Direct-Send compositing algorithm proposed by Eilemann et al. [4]. A careful implementation of this algorithm yields a well-balanced computing load over the whole group and partially enables communication and computing parallelization. Each node finally sends its locally weighted pixel values to the server. All the communications occurring between the server and the compute nodes are implemented with the MVAPICH2 MPI library and therefore efficiently exploit modern network interconnections such as InfiniBand.

3.2 Collaborative web access

Lucas et al. [9] proposed USE Together as a WebRTC-based middleware enabling remote users collaboration through the web. With this infrastructure, delocalized co-worker can remotely share a native desktop environment or a specific application window, with all user interactions broadcasted in real-time. The remote video stream is continuously produced by capturing the target GUI application or the entire desktop content through native OS API calls (see Figure 4). The captured stream is therefore augmented with the different user cursor locations. In this way, all users can locate each other in the shared workspace therefore enhancing their mutual feeling of presence and the way they collaborate. Finally, the video stream is compressed before being sent to every peer using WebRTC MediaStream dedicated channels. To that end, USE Together takes advantage of H.264 low latency encoding capabilities of modern graphics hardware by leveraging NVIDIA NVENC encoding API with its low-latency high-performance preset. User inputs are streamed through WebRTC Data channels and processed in real-time within a synchronized multi-user environment. In this environment, a user can seamlessly take the control anytime he does a specific action (mouse click and/or keyboard use) as long as the system is not

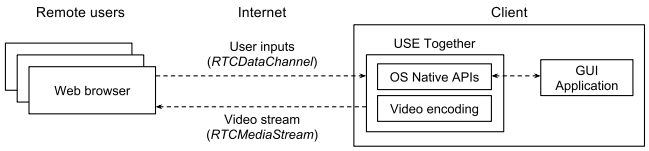

Figure 4: Overview of the collaborative web infrastructure

already used by a third party (another user) of course. In this case, user inputs other than those associated with the active user action are simply ignored. By following this approach and assuming minimum client download bandwidths of 3 Mbps (ADSL/3G/4G) for HD streaming, USE Together provides a collaborative environment through a user experience very close to a native desktop from any device running a web browser supporting the WebRTC standard.

4 Results

The experiments were conducted with three simultaneously connected users located at hundreds of kilometers from each other with various device types (two laptops respectively equipped with an NVIDIA Geforce GT 650M and an Intel HD4000 chip and a tablet embedding a Tegra K1 chip) and through different network infrastructures (ADSL and 4G). The system used for testing was the ROMEO supercomputer from the University of Reims Champagne-Ardenne. The cluster is composed of 128 nodes embedding 2 Intel Xeon E5-2650 with 8 cores per CPU and 2 NVIDIA Tesla K20Xm GPUs. The cluster is fully interconnected with InfiniBand ConnectX-2 QDR offering a 40 Gbps theoretical network bandwidth. The scenes used for the tests are shown in Figure 5. The car exterior and interior views respectively consist of >5M and >2M triangles and around 20 physically measured materials. The car interior view particularly requires a considerable amount of rendering time as the lighting is highly indirect. Sponza is made of 260K triangles and consists of a single Lambertian material. Many indirectly illuminated areas can also be found in this scene. HDR environment maps have been used to illuminate all the scenes and the rendering resolution was set to 1920×1080. Table 1 shows the rendering times of the different scenes for different node counts using the proposed load balancing approach where the image-space partitioning is used up to 4 nodes. Therefore, the path-space division approach is additionally combined to the image-space division when rendering with more than 4 nodes. Note that in our benchmark setting, the rendering of an image is considered as complete when its root mean square error (RMSE) computed against a high-quality reference image has dropped below a user-specified threshold. The RMSE thresholds were respectively fixed to 0.025, 0.3 and 0.1 for the car exterior view, the car interior view and Sponza.

Table 1: Rendering times (seconds) of the test scenes for different node counts

Node count	1	2	4	8	16	32	64
Car exterior view	199.6	97.9	53.6	26.1	13.1	6.6	3.4
Car interior view	334.3	162.9	81.7	40.4	20.3	10.1	4.9
Sponza	461.1	220.9	112.2	55.7	28.3	13.8	7.1

Figure 6 reports the scaling efficiency of the load balancing scheme where the scaling efficiency is defined as the ratio of the ideal linear rendering time to the effective rendering time. As can be seen, our hybrid load-balancing strategy shows a good scalability behaviour (90% to 110%) and is able to exploit almost all computing resources we were granted access to. The slightly reduced scaling efficiency for the car exterior view can be attributed to the impact of the interleaved image-space partitioning which may vary between different scene types. Moreover, this setup achieves interactive compositing rates close to 15 frames per second on the server node,

Figure 5: Scenes used for the tests: A car exterior view (left), a car interior view (middle) and Sponza (right).

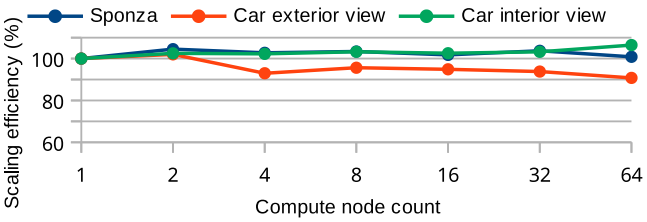

Figure 6: Scaling efficiency of the load balancing approach. Image-space division is used up to 4 compute nodes, then path-space division is additionally applied.

up to 64 rendering nodes. Finally, the proposed architecture results in a globally smooth experience for all the users by providing streaming rates and latencies compatible with a remote collaborative experience.

Limitations The system requires the replication of the whole scene into the GPU memory of each node which can reveal critical when dealing with large scene models. Therefore, it would be desirable to investigate on out-of-core scene management techniques to handle such cases.

5 Conclusion and Future Work

In this paper, we have proposed a collaborative predictive rendering environment taking full advantage of the computational power offered by modern multi-GPU clusters. To make an efficient use of the remote computational resources, we have proposed a static load balancing method leveraging the stochastic nature of Monte Carlo rendering by distributing the light path sampling process. By combining this approach to a usual image-space partitioning scheme, a good scalability can be achieved with a simple static load balancing scheme which cannot be obtained with traditional static schemes. The proposed architecture not only efficiently exploits the computational resources, but also promotes co-worker mobility and collaboration by providing a remote web access allowing real-time multi-user interactions with the virtual mockup. In comparison to the usual remote visualization tools, the system emphasizes collaboration with an enhanced feeling of presence and a transparent multiple user input management. As a result, organization members can easily access the computational resources of the system and work on centralized data from any device running a web browser supporting the WebRTC standard. We are now looking forward to extend the system to cover other useful collaboration use cases. As an example, leveraging the computing resources of modern clusters to allow the remote users to work on different views of the same shared model seems to be a promising research direction [3].

Acknowledgments

This work was supported by the ROMEO Computing Center and the Image Center of the University of Reims Champagne-Ardenne. The authors also thank OPEXMedia for their support on USE Together.

References

[1] L. P. Berg and J. M. Vance. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual Reality*, 21(1):1–17, 2017.

[2] B. Cosenza, C. Dachsbacher, and U. Erra. Gpu cost estimation for load balancing in parallel ray tracing. In *GRAPP/IVAPP*, pp. 139–151, 2013.

[3] C. Desprat, J.-P. Jessel, and H. Luga. 3devent: a framework using event-sourcing approach for 3d web-based collaborative design in p2p. In *Proceedings of the 21st International Conference on Web3D Technology*, pp. 73–76. ACM, 2016.

[4] S. Eilemann and R. Pajarola. Direct send compositing for parallel sort-last rendering. In *Proceedings of the 7th Eurographics conference on Parallel Graphics and Visualization*, pp. 29–36. Eurographics Association, 2007.

[5] C. Fleury, N. Férey, J.-M. Vézien, and P. Bourdot. Remote collaboration across heterogeneous large interactive spaces. In *Collaborative Virtual Environments (3DCVE), 2015 IEEE Second VR International Workshop on*, pp. 9–10. IEEE, 2015.

[6] B. Freisleben, D. Hartmann, and T. Kielmann. Parallel raytracing: a case study on partitioning and scheduling on workstation clusters. In *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, vol. 1, pp. 596–605 vol.1, Jan 1997. doi: 10.1109/HICSS.1997.667407

[7] T. Ize, C. Brownlee, and C. D. Hansen. Real-time ray tracer for visualizing massive models on a cluster. In *EGPGV*, pp. 61–69, 2011.

[8] B. Jeong. *Visualcasting—Scalable real-time image distribution in ultra-high resolution display environments*. University of Illinois at Chicago, 2009.

[9] L. Lucas, H. Deleau, B. Battin, and J. Lehuraux. *USE Together, a WebRTC-Based Solution for Multi-user Presence Desktop*, pp. 228–235. Springer International Publishing, Cham, 2017. doi: 10.1007/978-3-319-66805-5_29

[10] L. Renambot, B. Jeong, R. Jagodic, A. Johnson, J. Leigh, and J. Aguilera. Collaborative visualization using high-resolution tiled displays. In *ACM CHI Workshop on Information Visualization Interaction Techniques for Collaboration Across Multiple Displays*, pp. 1–4, 2006.

[11] M. Repplinger, A. Löffler, D. Rubinstein, and P. Slusallek. Drone: a flexible framework for distributed rendering and display. *Advances in Visual Computing*, pp. 975–986, 2009.

[12] G. Tamm and P. Slusallek. Web-enabled Server-based and Distributed Real-time Ray-Tracing. In E. Gobbetti and W. Bethel, eds., *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2016. doi: 10.2312/pgv.20161182

[13] I. Wald, T. J. Purcell, J. Schmittler, C. Benthin, and P. Slusallek. Realtime ray tracing and its use for interactive global illumination. In *Eurographics 2003 - STARs*. Eurographics Association, 2003. doi: 10.2312/egst.20031091

[14] A. Wilkie, A. Weidlich, M. Magnor, and A. Chalmers. Predictive rendering. In *ACM SIGGRAPH ASIA 2009 Courses*, p. 12. ACM, 2009.

[15] P. Zimmermann. Virtual reality aided design. a survey of the use of vr in automotive industry. *Product Engineering*, pp. 277–296, 2008.