Multi-Display Ray Tracing Framework

Luciano A. Romero Calla[1], Bipul Mohanto[2], Renato Pajarola[1], and Oliver Staadt[2]

[1]University of Zurich, Switzerland [2]University of Rostock, Germany

Abstract

We present a framework that will provide a highly efficient and scalable multi-display ray-tracing based rendering system capable of utilizing multiple GPU devices to produce high-quality images. Our system integrates advanced technologies, including MPI, CUDA, CUDA IPC, OptiX 7.6, and C++, resulting in a cutting-edge solution for interactive rendering.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Graphics processors;***

1. Introduction

We have developed a framework for multi-display rendering using advanced technologies such as MPI (Message Passing Interface), CUDA (Compute Unified Device Architecture), CUDA IPC (Inter-Process Communication), OptiX 7.6, and the C++ programming language.

Our framework has two main components: the ray tracing module and the multi-display module. Any ray tracing algorithm implements the ray tracing module, like a path tracing algorithm or a simple ray-caster with only primary and shadow rays. Its implementation is independent of the multi-display module that cares about synchronization across multiple nodes and handles multiple GPUs as well as displays.

2. Related Work

We can divide the related work into rasterization and ray-tracing based approaches. Among the latest rasterization works, we find [DP22], which extend [DP19] to handle load balancing and LOD compared to *Equalizer* [ESP20]. *Equalizer* [ESP20] is a framework for scalable, parallel rendering and data distribution for large scale visualizations. Another relevant work is [DK11], which extends OpenGL to implement a distributed framework for high-performance visualization systems.

Our framework belongs to the second group of ray-tracing based approaches. In this group, we find [HWU*20] that presents a framework for rendering large tiled display walls as a display service. [UWA*19] proposed a distributed frame buffer approach and extended the API from OSPRay [WJA*17]. Finally, not related to multi-display rendering but in the scope of distributed rendering, in [JvSv21] a data-distributed solution to path-tracing massive scenes across multiple GPUs has been proposed.

3. Our Framework

Our framework contains two independent components: the multi-display and the ray-tracing modules. Figure 1 shows an overview of a running process configured for two nodes and two GPUs per node. The Display processes run the multi-display module. Its implementation extends the OpenGL-based viewer from the *gproshan* framework [RF] to handle multi-display configurations using MPI to communicate and synchronize the processes and using CUDA IPC to allow access to the GPU buffer from a ray-tracing (RT) process.

Figure 1: *Framework architecture.*

An RT & Display process initialize and run a ray-tracer implementation per GPU, which defines the ray-tracing module of our framework, independent of the multi-display module. It handles all the render tasks for the process running on the same GPU and their respective displays. In the following section, we introduce a *Variable Rate Path Tracer* as a specific implementation of a ray-tracing module.

The first setup to run some basic experiments for the general framework consists of two nodes with an Intel Core i7-10700K processor, 32GB of RAM, and NVIDIA GeForce RTX 3090 with 24GB of memory and a GeForce RTX 3080 with 10GB of memory, respectively in each node. The setup includes four monitors on the first node and three on the second one, all with a resolution of 2160×1440 pixels.

Table 1 shows the results of our multi-display framework, which has two configurations for *per process* and *per GPU* rendering.

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

Per process means that all the processes are RT & Display process pairs, while *per GPU* implies that only one Display process is an RT process per GPU device.

Scene	Triangles	Monitors	GPU Memory	GPU usage %	FPS	Rendering
San Miguel	9980699	4	5719 MiB	72 %	68	per gpu
San Miguel	9980699	4	18171 MiB	70 %	73	per process
Sponza	262267	4	2410 MiB	66 %	67	per gpu
Sponza	262267	4	4933 MiB	67 %	72	per process
San Miguel	9980699	4, 3	3968 MiB, 3382 MiB	69 %, 55 %	74	per gpu
San Miguel	9980699	4, 3	**exceeds memory on second node**			per process
Sponza	262267	4, 3	2411 MiB, 1825 MiB	64 %, 50 %	64	per gpu
Sponza	262267	4, 3	4939 MiB, 3375 MiB	66 %, 49 %	76	per process

Table 1: *FPS for a ray tracer with primary and shadow rays.*

4. Variable Rate Path Tracer

Path tracing [Kaj86] is one of the best-known algorithms for high-end rendering. However, the long convergence times make it infeasible for real-time applications. In this framework, besides hardware acceleration, we exploit a space-variant human vision constraint to limit the number of rays and make interactive path tracing feasible. The human visual system has the highest visual acuity around 5.2° of the visual axis, also known as the fovea [MIGS22]. From that fovea, the visual acuity abruptly declines towards the periphery.

In this experiment, we contemplate a VRPT (variable-rate path tracer) to limit the number of rays outside the foveated region. First, we calculated the Euclidean distance from the gaze point and divided the screen space (Figure. 2) into three regions. Next, we derived a relationship to determine the foveated, intermediate, and peripheral pixels using MAR (Minimum Angle of Resolution) and Snellen chart's distance. Finally, we used a discriminant function to limit the number of rays for each region.

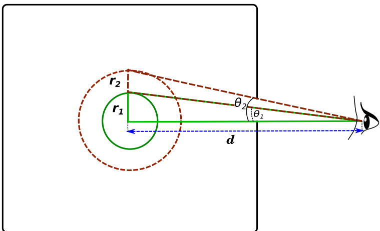

Figure 2: *We used a Snellen distance (d= 60 cm) from the eye position. Further, we measured the physical pixel size to define the foveated (r_1) and intermediate (r_2) region's radius concerning the viewing angle θ_1 and θ_2, respectively. The peripheral region covers the rest of the pixels outside the intermediate zone.*

To test the performance of our VRPT, we used a 12th Gen Intel(R) Core(TM) i9-12900KS processor, with 2 NVIDIA GeForce RTX 3090, 24GB of memory each, and we rendered 24.88 million ($3840 \times 3 \times 2160$) pixel images. The user can interact with a static scene using the framework's mouse-based interface. We measured the 100 conjugative seconds for the average framerate, see Figure 3. The average framerate for uniform path tracing is *3.72fps*, whereas the variable rate path tracing achieved framerates on average of *16.62fps*, which is 4.45× faster.

5. Conclusions

The variable rate path tracing implementation is currently limited to diffuse shaders only. Furthermore, the simple pinhole camera model has been used for ray generation, and the recursion depth

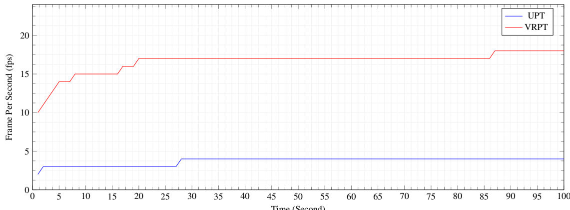

Figure 3: *Framerate for 100 conjugative seconds. The blue line represents the uniform path tracing, and the brown line represents our variable rate path tracing algorithm.*

is with hard truncation. The future study involves a full feature path tracing algorithm with robust variable sampling rate distribution.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 813170.

References

[DK11] Doerr K.-U., Kuester F.: CGLX: A scalable, high-performance visualization framework for networked display environments. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (3 2011), 320–332. 1

[DP19] Dong Y., Peng C.: Screen Partitioning Load Balancing for Parallel Rendering on a Multi-GPU Multi-Display Workstation. In *Eurographics Symposium on Parallel Graphics and Visualization* (2019), Childs H., Frey S., (Eds.), The Eurographics Association. doi:10.2312/pgv.20191111. 1

[DP22] Dong Y., Peng C.: Multi-gpu multi-display rendering of extremely large 3d environments. *The Visual Computer* (12 2022). doi:10.1007/s00371-022-02740-7. 1

[ESP20] Eilemann S., Steiner D., Pajarola R.: Equalizer 2.0 - Convergence of a parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics 26*, 2 (2 2020), 1292 –1307. doi:10.1109/TVCG.2018.2870822. 1

[HWU*20] Han M., Wald I., Usher W., Morrical N., Knoll A., Pascucci V., Johnson C. R.: A virtual frame buffer abstraction for parallel rendering of large tiled display walls. In *IEEE Visualization Conference (VIS)* (2020), pp. 11–15. doi:10.1109/VIS47514.2020.00009. 1

[JvSv21] Jaroš M., Říha L., Strakoš P., Špeťko M.: Gpu accelerated path tracing of massive scenes. *ACM Transactions on Graphics 40*, 2 (4 2021). doi:10.1145/3447807. 1

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings ACM SIGGRAPH* (1986), ACM SIGGRAPH, pp. 143–150. 2

[MIGS22] Mohanto B., Islam A. T., Gobbetti E., Staadt O.: An integrative view of foveated rendering. *Computers & Graphics 102* (2022), 474–501. doi:10.1016/j.cag.2021.10.010. 2

[RF] Romero Calla L. A., Fuentes Perez L. J.: gproshan: geometry processing and shape analysis framework. URL: https://github.com/larc/gproshan. 1

[UWA*19] Usher W., Wald I., Amstutz J., Günther J., Brownlee C., Pascucci V.: Scalable ray tracing using the distributed framebuffer. *Computer Graphics Forum 38*, 3 (2019), 455–466. doi:10.1111/cgf.13702. 1

[WJA*17] Wald I., Johnson G., Amstutz J., Brownlee C., Knoll A., Jeffers J., Günther J., Navratil P.: Ospray - a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 931–940. doi:10.1109/TVCG.2016.2599041. 1

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.