論文 / 著書情報
Article / Book Information

Title	Implementation of global illumination within image-based distributed rendering architecture
Authors	Wenxiang Yang, Asahi Yamada, Suguru Saito
Citation	IEVC2024 proceedings, Page 1/4-4/4
Pub. date	2024, 3
Copyright	

Implementation of global illumination within image-based distributed rendering architecture

Wenxiang Yang Asahi Yamada Suguru Saito
Tokyo Institute of Technology

Abstract

With the increasing popularity of the metaverse and digital twins alongside ongoing releases of consumer VR equipment, the demands for a virtual communication space that can be shared by many people are growing. In this paper, we take an image-based distributed rendering architecture, that allows a variety of avatar models of multiple users to be utilized simultaneously, and extend it to enhance the realism of the virtual environment by introducing the global illumination effect based on screen-space rendering and deferred shading techniques. We present the measured performance results of the proposed architecture implemented on the TSUBAME3.0 grid computer.

Keywords: Distributed Rendering, Global Illumination, Deferred Shading

1. Introduction

With the increasing popularity of the metaverse and digital twins in recent years, various consumer-oriented VR equipment has been released. Researchers, engineers, and users are now exploring various ways of engaging in activities in a virtual space that is close to the real world. To realize such a virtual environment, a large number of various customized avatar models, reflecting individual users, need to be rendered in real time. In addition, a high level of realism is required to ensure immersion.

However, rendering such a huge variety of different avatars on user-side devices, many of which have a low performance, may impose too heavy an operational load. Server-side rendering is a potential solution. Fang et al. [1] proposed an image-based distributed rendering architecture that utilizes multiple server processes, where rendering is performed on the server-side and the image streams are delivered to users. However, since each avatar is rendered independently on different servers, the effects of global illumination are not taken into account.

In this paper, we modify Fang et al. [1] image-based distributed rendering architecture so that it allows global illumination effects using screen-space ray tracing [2]. We also introduce the deferred shading technique [3] to reduce unnecessary shading calculations.

2. Related Works

2.1. Distributed Rendering

Distributed rendering techniques are designed to handle rendering tasks that require a huge amount of computation. The key challenging with these distributed rendering technique is determining how to divide and then distribute the rendering tasks [4] [5]. Distributed rendering methods are categorized into screen-level division [6], sub-pixel level division [7], and time-based division [8]. The architecture adopted in this paper belongs to the screen-level division approach.

2.2. Real-time Global Illumination

Global illumination is a rendering technique that aims to handle direct and indirect light simultaneously in a physically accurate manner. Lambru et al. [9] categorized real-time global illumination methods into four types: those based on discrete 3D coordinate method, the voxel method, the reflective shadow map method, and the screen space method.

In this paper, as our objective is to perform distributed rendering based on images representing a scene, we utilize screen-space ray tracing(SSR), belonging to the fourth category, to achieve real-time global illumination.

2.3. Deferred shading

Deferred shading [3] splits rendering into two passes. The first pass renders the scene once and stores geometry information in G-buffers [10]. The second pass utilizes the G-buffers to perform shading in screen space.

In this paper, we introduce a distributed rendering technique that utilizes deferred shading to achieve approximate global illumination in screen space.

3. Method

3.1. Architecture

The distributed rendering architecture proposed in this paper is shown in Fig1. The system is composed of three types of server processes: “model rendering

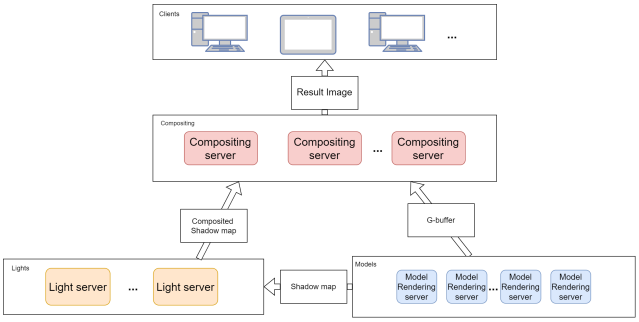


Figure 1: System architecture

server", "light server" and "compositing server". The model rendering server is responsible for generating G-buffers and a shadow map from the geometric data of the model it serves. The data structure of the G-buffers is shown in Fig2a. The model rendering server has multiple "main cameras" and "light cameras" that shoot user views and the views from the positions of lights, respectively. The main cameras generate model image data (G-buffers) from user viewpoints and the light cameras generate depth images from the lights view. This kind of depth image is called a shadow map (see Fig2b).

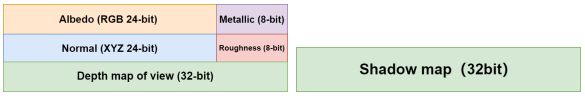
The light server is responsible for compositing shadow maps. Each light server corresponds to a single light source. It receives depth images from connected model rendering servers and then composites one shadow map by comparing the depth values of the corrected depth images pixel by pixel.

The compositing server composes model image data (G-buffers) from the model rendering server, performs global illumination and shadowing calculations with shadow maps from light servers, and delivers the results to the users.

3.2. Global illumination in the architecture

Calculating indirect lighting is a task that cannot be performed by for the model rendering server since the task requires information on all models in the scene, but the model rendering server holds only one model.

In contrast, since the compositing server receives G-buffers from model rendering servers, it is able to collect information about models in a scene from a particular viewpoint. In other words, although it is not complete, the compositing server has more scene information compared to the model rendering server. Therefore, in our architecture, the compositing server utilizes Screen-space Ray Tracing (SSR) to perform global illumination calculations.



(a) G-buffers (b) Shadow map

Figure 2: Data structure of G-buffer and shadow map

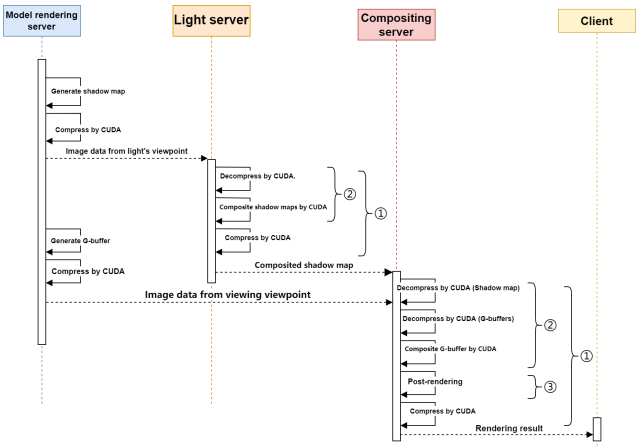


Figure 3: System workflow

3.3. Workflow

The workflow for generating one frame in this system is shown in Fig3. First, each model rendering server generates G-buffers for a user-specific view and shadow maps for lights and sends them to the compositing server and light servers. The light servers compose shadow maps and send them to the respective compositing server. The compositing server receives G-buffers from the model rendering servers and combines them by depth comparison. Following the deferred shading approach, it performs shading for direct and indirect light with the combined G-buffers, and shadowing by shadow mapping with the shadow maps from the light servers. Finally, it delivers the rendering result to the user.

G-buffer and shadow map data are compressed on the GPU using the CUDA nvCOMP library [11] before transmission to reduce the amount of transmitted data through the bus on a computer and the network among computers.

4. Results

We evaluated out architecture through experiments in which server processes were assigned to computation nodes on the TSUBAME3.0 [12] grid computer. The hardware specifications of one node in the grid computer are listed in Table 1. The installed OS was SUSE Linux Enterprise Server12 SP2. Since one node has four GPUs, we assigned up to four server processes to one node.

Table 1: Hardware specification of one node in the TSUBAME3.0 grid computer

Network	Intel Omni-Path HFI 100Gbps x4
CPU	Intel Xeon E5-2680 V4(14core) x2
Memory	DDR4-2400 DIMM 256GB
GPU	NVIDIA Tesla P100 (16GB,SXM2) x4

4.1. Experiment Conditions

Our prepared scene included 19 avatar models, one background model set, one light source, and three user viewpoints. Therefore, we set up 20 model rendering server processes, one light server process and three compositing server processes. The amount of data for the avatars and the background model are shown in Table 2. The avatar models were generated using VRoid Studio [13]. The resolution of the G-buffers the output of compositing server were set to 1024×1024, and the resolution of the shadow map was set to 2048×2048.

Table 2: Model data

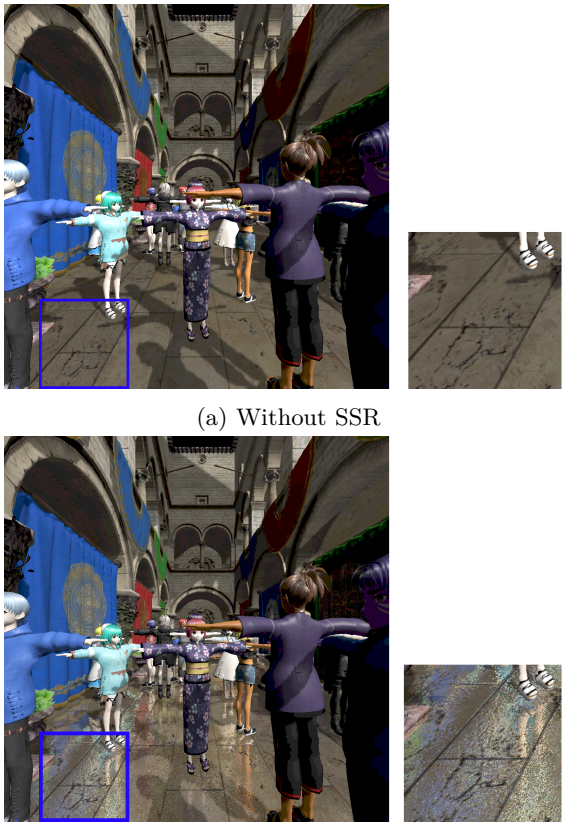
The average amount of avatar data	
average number of polygon	35119
average texture count	2
average file size(glTF)	14.46MB
The amount of background model data	
number of polygon	262267
texture count	69
file size(glTF)	52.7MB

4.2. Rendering results

We implemented indirect lighting with specular and glossy reflections and soft shadows. The rendering results of our experiments are shown in Fig4.

4.3. Time measurements

In this experiment, we measured frame generation, compositing and post-rendering execution times in the server processes. For the model rendering server, the frame generation time is the average of the sum of the G-buffers generation time and shadow map generation times for one cycle (for three main cameras and one light camera in this case). For the light server, the frame generation time is the sum of the decompression and compositing times of shadow map as indicated by ① in the center of Fig3. For the compositing server, the frame generation time is the time from decompression to the end of post-rendering as indicated by ① on the right in Fig3. Compositing time is the time of decompressing and compositing the received G-buffers and the received shadow map in the compositing server and the light server processes, respectively as indicated by ② in Fig3. Post-rendering time is the time of direct



(a) Without SSR

(b) With SSR

Figure 4: Rendering results under different conditions and indirect lighting and soft shadow generation on the compositing server as indicated by ③ in Fig3.

Tables3 and 4 show the results of the average of 1000 cycles. In the model rendering server process, the average frame generation time for avatar models was 4.13ms, while for the background model, it was 5.22ms. The model rendering server was able to generate about 200 frames per second for single view, whereas the compositing server could only generate about 50 frames per second. The average performance difference between using the SSR calculation and not using it in the compositing server was about 1.33ms, and the difference mainly occured in the post-rendering part. The post-rendering time accounts for approximately 75% of the frame generation time for the compositing server process. Since the deferred shading is executed in this part, it is the most time-consuming process in the whole system. The variation in time for the three compositing servers is due to the different view of the scene when varying the complexity in the image space.

5. Conclusion

In this paper, we extended an earlier image-based distributed rendering architecture to achieve global illumination by using SSR. Our experimental findings demonstrate the possibility of implementing the global

Table 3: Rendering without SSR

	Frame generation time①		Compositing time②		Post-rendering time③	
	Mean Value	SD Value	Mean Value	SD Value	Mean Value	SD Value
Compositing server 1	19.23ms	0.46ms	4.04ms	0.78ms	14.55ms	0.40ms
Compositing server 2	20.65ms	0.44ms	4.62ms	0.96ms	15.13ms	0.54ms
Compositing server 3	21.03ms	0.41ms	5.02ms	0.98ms	16.32ms	0.18ms
Light server	3.37ms	0.41ms	3.27ms	0.40ms		

Table 4: Rendering with SSR

	Frame generation time①		Compositing time②		Post-rendering time③	
	Mean Value	SD Value	Mean Value	SD Value	Mean Value	SD Value
Compositing server 1	20.31ms	0.48ms	4.53ms	0.85ms	15.72ms	0.37ms
Compositing server 2	22.35ms	0.20ms	4.58ms	0.82ms	17.38ms	0.11ms
Compositing server 3	22.53ms	0.72ms	4.78ms	0.98ms	17.62ms	0.14ms
Light server	3.34ms	0.36ms	3.30ms	0.36ms		

illumination in real-time aimed at distributed rendering architecture by sending material-specific G-buffers from model rendering server processes to compositing server processes, which may be a good choice for real-time processing.

Our implementation is still ongoing. The performance of the model rendering server is still not sufficient for providing multiple view images, and the post-shading part in the compositing server spends much time. Our future work will focus on improving the rendering performance in model rendering server and compositing server. We also plan to optimizate the data compression and transmission in the whole system.

References

[1] Fang, H., Okumura, N., Ishii, K. and Saito, S.: "Distributed rendering on grid computers for multiple users in shared virtual space", 2022 International Conference on Cyberworlds (CW), pp. 47–54 (online), DOI:10.1109/CW55638.2022.00016 (2022).

[2] McGuire, M. and Mara, M.: "Efficient GPU Screen-Space Ray Tracing", Journal of Computer Graphics Techniques (JCGT), Vol. 3, No. 4, pp. 73–85 (online), available from http://jcgt.org/published/0003/04/04/ (2014).

[3] Hargreaves, S. and Harris, M.: "Deferred shading", Game Developers Conference, Vol. 2, p. 31 (2004).

[4] Pineda, J.: "A Parallel Algorithm for Polygon Rasterization", SIGGRAPH Comput. Graph., Vol. 22, No. 4, p.17–20 (online), DOI: 10.1145/378456.378457 (1988).

[5] Akeley, K.: "Reality Engine Graphics", Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '93, New York, NY, USA, Association for Computing Machinery, p. 109–116(online), DOI: 10.1145/166117.166131 (1993).

[6] Samanta, R., Funkhouser, T., Li, K. and Singh, J.: "Sort-First Parallel Rendering with a Cluster of PCs (2000)".

[7] Eilemann, S., Steiner, D. and Pajarola, R.: "Equalizer 2.0 - Convergence of a Parallel Rendering Framework", CoRR, Vol. abs/1802.08022 (online), available from http://arxiv.org/abs/1802.08022 (2018).

[8] Ahrens, J., Law, C., Schroeder, W., Martin, K., Inc, K. and Papka, M.: "A Parallel Approach for Efficiently Visualizing Extremely Large", Time-Varying Datasets (2000).

[9] Lambru, C., Morar, A., Moldoveanu, F., Asavei, V. and Moldoveanu, A.: "Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines", IEEE Access, Vol. 9, pp. 125158–125183 (online), DOI: 10.1109/ACCESS.2021.3109663 (2021).

[10] Saito, T. and Takahashi, T.: "NC machining with G-buffer method", ACM Siggraph Computer Graphics, Vol. 25, No. 4, pp. 207–216 (1991).

[11] NVIDIA: nvCOMP, https://developer.nvidia.com/nvcomp (Last accessed date: 2023/9/15).

[12] Hardware configuration of tsubame3.0, https://www.t3.gsic.titech.ac.jp/hardware (Last accessed date: 2023/10/17).

[13] pixiv: Vroid Hub, https://hub.vroid.com (Last accessed date: 2023/10/17).