

Accelerating path tracing rendering with Multi-GPU in Blender cycles

1st Jiuyi Chen

*Shenzhen Institute of Advanced
Technology, CAS
University of Chinese Academy
of Sciences
Shenzhen, China
chen.jy@siat.ac.cn*

2nd Ling Chen

*Shenzhen Institute of Advanced
Technology, CAS
University of Chinese Academy
of Sciences
Shenzhen, China
xjy109200@siat.ac.cn*

3rd Zhibin Yu

*Shenzhen Institute of Advanced
Technology, CAS
Shenzhen, China
zb.yu@siat.ac.cn*

Abstract— Metaverse is one of the current hottest research fields, which not only requires rendering techniques such as path tracing but also rendering engines such as Blender. However, rendering scenes by path tracing is a complex and computationally intensive process, especially for large scenes, causing catastrophically slow rendering using only a single GPU.

To improve rendering performance, we use multiple GPUs to achieve parallel rendering in Blender Cycles for path tracing. First, we distribute the workload based on pixel blocks. Each GPU renders one of the image fragments to finally combine the rendering results into a complete image. Second, we propose a Dynamic Workload Adjustment Algorithm (DWAA) to solve the workload imbalance problem. DWAA takes advantage of the fact that path tracing requires multiple samples, so it increases or decreases the work area of each GPU for the next ray sample according to the rendering time of each GPU's last path-tracing sample, which can reduce GPU idle time and improve overall rendering performance.

To evaluate the Multi-GPUs rendering approach and DWAA, we use two different scale scenes - Babershop and Junkshop, and test them with different GPU clusters consisting of 2, 3, and 4 NVIDIA A100 GPUs respectively. The results show that compared to a single GPU, 2 GPUs can accelerate up to 1.98x, 3 GPUs up to 2.94x, and 4 GPUs up to 3.9x, which has high scalability and the parallelism is close to linear increase.

Keywords— Multi-GPU, Rendering, Path-Tracing, Blender Cycles, Performance Optimization.

I. INTRODUCTION

Scene rendering refers to the conversion of a description of a 3D scene or object into a visually realistic 2D image. It can be used not only in animation, 3D movies, 3D games, and virtual reality, but also in architectural visualization, medical imaging, aerospace, and other engineering fields. For ray tracing rendering, each pixel of the image is emitted independently for light propagation to obtain the final pixel color value, which has a high degree of parallelism, but it is a computationally complex and expensive task. Even if the GPU is used to perform the rendering task, the rendering time is still very long, making it difficult to meet the real-time requirements. To further improve rendering performance, it is very necessary and important to build distributed GPU rendering architectures. Moreover, GPU clusters are increasingly explored and used in scientific research and industrial markets [1]–[4], especially for deep

learning and 3D rendering where the efficient computation of GPU clusters is highly needed. We therefore propose to use multiple GPUs in Blender Cycles to render the modeled 3D scenes in parallel to proportionally speed up the whole rendering process.

Blender is a very useful and mature 3D content creation suite that is not only open source and cross-platform, but also offers a wide range of essential tools including 3D modeling, video editing, two built-in rendering engines for rendering - Blender Internal and Blender Cycles [5], [6]. Blender Cycles can use GPU rendering or CPU+GPU hybrid rendering to increase rendering efficiency. It also supports distributed rendering for high performance computing, which is necessary to utilize the computational resources of multiple GPUs to improve rendering performance. So the Multi-GPU rendering experiments in this paper are conducted in Blender Cycles engine.

Unfortunately, workload imbalance is always a difficult problem for GPU clusters [7]–[9]. An unreasonable workload allocation method can lead to low workload balancing, resulting in a large number of idle GPU cycles and poor GPU utilization, which further leads to overall performance degradation. For Multi-GPU rendering, an image needs to be divided into an equal number of image segments evenly by the number of GPUs. Accordingly, each GPU is responsible for rendering one of the image pixel fragments and storing the rendering result into a buffer, and finally combining all the pixel values in the buffer to get the final displayed image.

This distribution method can cause workload imbalance easily, because even though each GPU renders an equally sized image fragments, the amount of rendering tasks for each rendering area cannot be determined, leading to a GPU that has already completed its task waits for another GPU. To solve this, we propose Dynamic Workload Adjustment Algorithm to adjust the rendering tasks of each GPU so that the rendering time of each GPU is as close as possible, thereby improving the overall rendering performance. Specifically, taking 2 GPU rendering as an example, it compares the rendering time of the last path tracing sampling of the 2 GPUs. If the rendering time of GPU0 is longer than that of GPU1, it increases the

working area of the next ray sampling of GPU1, and vice versa for GPU0.

To evaluate the multi-GPU rendering approach and Dynamic Workload Adjustment Algorithm (DWAA), we use two scenes of different sizes - Babershop and Junkshop, and test them with different GPU clusters consisting of 2, 3, and 4 NVIDIA A100 GPUs respectively. The results show that for Junkshop scene, the average speedup ratio is 1.91x for 2 GPUs, 2.81x for 3 GPUs, and 3.54x for 4 GPUs with DWAA compared to a single GPU. Similarly, 2 GPUs can accelerate up to 1.98x, 3 GPUs up to 2.94x, and 4 GPUs up to 3.9x for Babershop scene, demonstrating the necessity of our method and its high scalability. More importantly, the GPU rendering parallelism is close to linear increase.

II. THEORETICAL KNOWLEDGE

A. The Rendering Equation

Kajiya established the theory of rendering equation in 1986 [10], which describes the relationship between the radiance of a certain point on the surface of an object in a certain direction and the brightness of the incident radiation after the light energy transmission in the scene reaches a steady state. It is based on the physics of light, and the law of conservation of energy, which perfectly describes the propagation of light energy through the scene. The famous algorithms for global illumination, including Path Tracing [11], Metropolis Light Transport [?], etc. are based on the rendering equations. The mathematical representation of the rendering equation is as:

$$L_o(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \int_{\Omega} f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i) (\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \quad (1)$$

where $\vec{\omega}_o$ is direction of outgoing ray, $\vec{\omega}_i$ is direction of incident ray, Ω is the unit hemisphere in direction of normal vector $\vec{n}$ centered at x , $L_o(x, \vec{\omega}_o)$ is spectral radiance emitted by the source from point x in direction $\vec{\omega}_o$, $L_e(x, \vec{\omega}_o)$ is spectral radiance emitted from point x in the direction of $\vec{\omega}_o$ if point x can be self-luminous, $f_r(x, \vec{\omega}_i, \vec{\omega}_o)$ is bidirectional reflectance distribution function of the image (BRDF) in point x from direction $\vec{\omega}_i$ to direction $\vec{\omega}_o$, which determines the material of the object, $L_i(x, \vec{\omega}_i)$ is spectral radiance coming inside to x in direction $\vec{\omega}_i$, and $\vec{\omega}_i \cdot \vec{n}$ is the angle between $\vec{\omega}_i$ and the surface normal $\vec{n}$.

Undoubtedly, the rendering equation looks confusing, and due to the existence of complex integral calculations, solving the pixel values for ray tracing using the equation is extremely complex and computationally intensive. Thus, how to optimize the computation of this rendering equation becomes an important research direction for later global illumination technology.

B. Path tracing

Path tracing is an approximation method that use Monte Carlo to approximate the computation of complex integrals in the equation. Its basic idea is to emit a ray from the viewpoint. When the ray intersects the surface of the object, it continues to sample one direction according to the material properties of the surface, and emits another ray, and so on, until the ray hits

the light source (or escapes the scene), and then uses Monte Carlo method [13] to calculate its contribution as the color value of the pixel.

The Monte Carlo (MC) method for estimating the integral in equation by the law of large numbers can be written as:

$$\int_{\Omega} f(x) d\Omega \approx \frac{1}{N} \sum_D f(x) \quad (2)$$

where $f(x)$ stands for $f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i) (\vec{\omega}_i \cdot \vec{n})$, D is set of N samples uniformly distributed in Ω . The larger N is, that is, the larger the number of samples, the closer the approximation value is to the real value. Therefore, in order to reduce the noise in the rendered images of path tracing, a relatively large sampling value is required. In this paper, we use a suitable and generally applicable value - 1024, that is each pixel emits rays in random directions to perform 1024 path tracings, and the 1024 results are finally averaged to obtain the final pixel value.

Fortunately, the Blender Cycles rendering engine has a built-in path tracing algorithm based on the Monte Carlo to solve rendering equation, called Quasi-Monte Carlo [14], [15], which has a better convergence rate compared to the original method.

III. MULTI-GPU RENDERING METHODOLOGY

To use multiple GPUs for parallel rendering of path tracing, it is necessary to reasonably divide the rendering tasks to each GPU. We give priority to dividing the workload based on image space, i.e., split frame rendering (SFR) [16]–[18]. Thus, we divide the image into a number of work areas according to the number of GPUs on a single-node server, and distribute them to each GPU to perform their own rendering tasks. Unfortunately, even though each GPU is responsible for an equal image area size, the amount of rendering tasks for each rendering area is not determinable, which can easily cause load imbalance problems. To solve this problem, we realize that because path-tracing requires multiple samples to reduce noise (we set the total number of samples to 1024), we can dynamically adjust the workload distribution for the next path-tracing sample based on a comparison of the rendering time of the previous sample, so that each GPU is always working, reducing the idle time of each GPU and thus improving the overall rendering performance.

A. Task Distribution

When the image is rendered in Blender Cycles, the scene image modeled using Blender with a resolution of $width \times height$ is decomposed into tiles which is equal to the number of GPUs. Since the rendering tasks of the tiles cannot be known in advance, at the beginning of path-tracing, each GPU is responsible for the tiles of equal size for simplicity. The workload division mode is shown in Figure 1. For a cluster composed of 2 GPUs, the picture is divided into two equal upper and lower tiles, and the same for 3 and 4 GPUs.

However, the number of rays emitted for tracing (numbers of ray bounce) and the amount of computation for each pixel

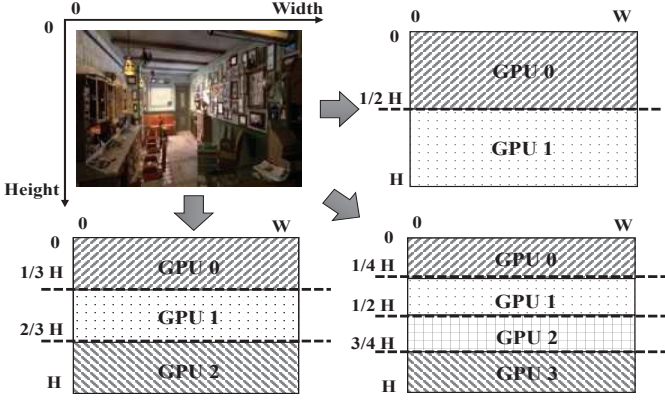


Figure 1. Task distribution for 2, 3 and 4 GPUs respectively.

in each tile are different, resulting in different rendering times for each GPU. Taking the cluster of 4 GPUs as an example, if GPU0 finishes rendering, it is possible that GPU3 only completes two-thirds of the rendering task, leading to a large amount of idle time for GPU0, so we have to use a workload balancing algorithm to reduce this situation.

B. Dynamic Workload Adjustment Algorithm

To solve the load imbalance problem, we propose Dynamic Workload Adjustment Algorithm (DWAA). Its basic principle is to compare the rendering time of the last path-tracing sample of each GPU, and dynamically adjust the workload distribution of the next path-tracing sample. The longer the rendering time of the GPU is, the smaller work tile for it to render, and vice versa for a short one.

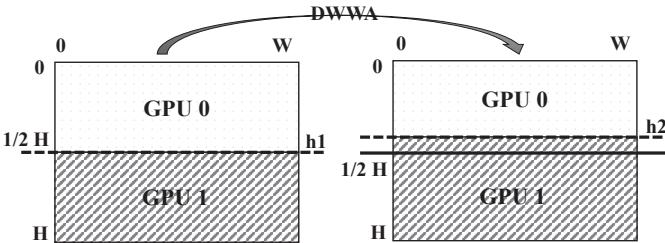


Figure 2. The size of the working tiles rendered by each GPU is resized by DWAA. The graph on the left shows the work distribution for the N path-tracing sample, and the graph on the right is the $N + 1$ sample after adjustment by DWAA.

For two GPUs cluster, GPU0 renders the pixel tile from $0 - h1$ and GPU1 renders the pixel tile from $h1 - height$. At the N th sample, if the rendering time of GPU0 is greater than GPU1, then the size of the tile to be rendered by GPU0 at the $N + 1$ th sample is reduced, that is GPU0 is responsible for rendering the pixel tile from $0 - h2$ and GPU1 renders the remaining area. At the next path-tracing sample, the rendering times of the two GPUs are compared again to adjust the workload at each path-tracing sample, so that the rendering times of the two GPUs are always very close to each other.

IV. EXPERIMENTAL RESULTS

Experiments on rendering performance for both scenes and on scalability for multi-GPU performance acceleration conduct on a single-node server. This server contains 2 Intel Xeon Gold 6230R CPUs together equipped with 104 cores and 4 identical NVIDIA A100 Tensor Core GPUs.

The correctness of our approach is proved by comparing the rendering times of the multi-GPU method and DWWA for two rendering scenes. The results are shown in Figure 5,6. In addition, the scalability of our approach is demonstrated by comparing the performance acceleration ratio of different GPU cluster consisting of 2 to 4 GPUs respectively compared to a single GPU rendering. Its results are shown in Figure 7,8.

A. Rendering Scenes

We do experiments in two different Scenes, as shown in Figure 3,4. The first one called Junkshop is a Benchmark modeled with Blender for testing GPU performance, similar to an anime movie. The other is a barbershop taken from Blender's animated movie Agent 327, containing various murals, furniture, self-illuminated objects and a lot of details, thus increasing the rendering time.

Junkshop. Resolution of the scene is 2000x1000 and 1024 samples.



Figure 3. Scene image: Junkshop by Blender Foundation.

Barbershop. Resolution of the scene is 2048x858 and 1024 samples.



Figure 4. Scene image: Barbershop from Agent 327 by Blender Foundation.

B. Results

1) *Rendering time comparison:* The rendering times of Junkshop and Barbershop for different GPU clusters and with or without DWWA are shown in Figures 5,6 respectively.

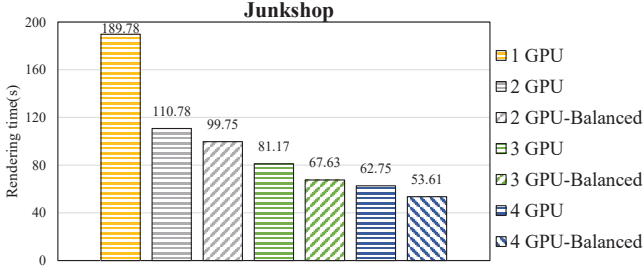


Figure 5. Rendering time for Junkshop scene and different GPU cluster.

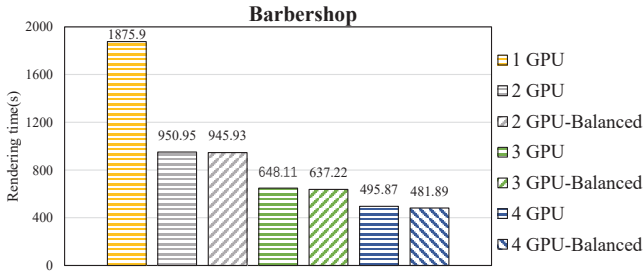


Figure 6. Rendering time for Barbershop scene and different GPU cluster. 4 GPU means a GPU cluster composed of 4 GPUs, the rest symbols are same, and the *Balanced* means that the load balancing algorithm DWWA is added.

As seen in Figure 5, the rendering time for 2 GPUs without the DWWA method is 111.8 seconds, about 1.71 times that of a single GPU. In contrast, adding DWWA results in a lower rendering time of 99.8 seconds and a speedup ratio of 1.9 times. Similarly, the speedup ratios for the 4 GPUs are 3.02x and 3.54x, respectively, indicating that using DWWA can effectively improves rendering performance and that the speedup ratio is closer to the number of GPUs. In Figure 6, we realize the acceleration ratio is higher for the Barbershop scene than Junkshop: up to 1.98x for 2 GPUs and up to 3.9x for 4 GPUs, which is basically proportional to the number of GPUs.

2) *Scalability of Multi-GPUs:* Figure 7,8 shows the speedup ratios for different numbers of GPUs rendering the Junkshop and Barbershop scenes respectively. It can be observed that the speedup of both scenes is linearly scalable, which indicates that the multi-GPU rendering approach we use is effective. Therefore, for a GPU cluster composed of more GPUs, the total rendering time will be further reduced, and the reduction ratio is proportional to the number of GPUs.

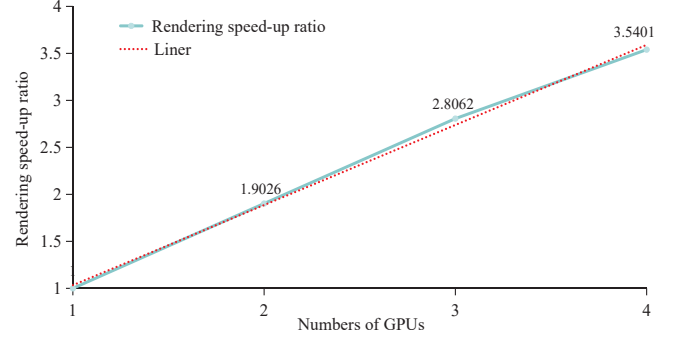


Figure 7. Speedup ratios for different numbers of GPUs rendering the Junkshop scene.

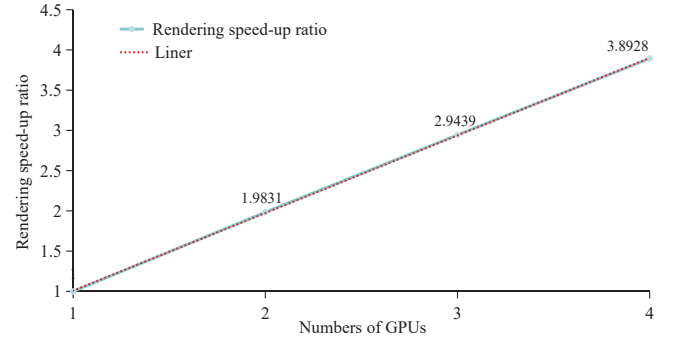


Figure 8. Speedup ratios for different numbers of GPUs rendering the Barbershop scene.

V. CONCLUSION

In this paper, we introduce a rendering method that uses multiple GPUs for the Blender Cycles renderer, and propose a method DWWA to solve the workload imbalance problem according to the characteristics that path tracing requires multiple samples. It has been shown that our proposed cluster rendering approach with multiple GPUs can reduce the rendering time nearly exponentially and the scalability is close liner. Therefore, if we continue to increase the number of GPUs (e.g., up to 64), the rendering time will be further reduced significantly to meet the rendering real-time requirements.

REFERENCES

- [1] Ren, Xiaowei, and Mieszko Lis. "CHOPIN: Scalable Graphics Rendering in Multi-GPU Systems via Parallel Image Composition." 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021.
- [2] Kuang, Yang, and Jie Jiang. "The research of render to textures during making virtual reality based on Unity 3D technology." 2015 International Industrial Informatics and Computer Engineering Conference. Atlantis Press, 2015.
- [3] Wu, Jiayi, Long Deng, and Anand Paul. "3D terrain real-time rendering method based on CUDA-OpenGL interoperability." IETE Technical Review 32.6 (2015): 471-478.
- [4] Domali, E., et al. "P14. 12: 3D parameters via Render Mode during transvaginal ultrasonography and endometrial malignancy." Ultrasound in Obstetrics Gynecology 50 (2017): 200-200.
- [5] B. O. Community, "Blender - a 3d modelling and rendering package," 2017. <http://www.blender.org>.

- [6] F. Steinmetz and G. Hofmann, The Cycles Encyclopedia. blender.org, 2014.
- [7] Xie, Chenhao, et al. "Processing-in-memory enabled graphics processors for 3D rendering." 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2017.
- [8] Xie, Chenhao, Xin Fu, and Shuaiwen Song. "Perception-oriented 3D rendering approximation for modern graphics processors." 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018.
- [9] Xie, Chenhao, et al. "OO-VR: NUMA friendly object-oriented VR rendering framework for future NUMA-based multi-GPU systems." 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2019.
- [10] James, T. Kajiya. "The rendering equation." ACM SIGGRAPH Computer Graphics 20.4 (1986): 143-150.
- [11] Lafortune, Eric P., and Yves D. Willems. "Bi-directional path tracing." (1993): 145-153.
- [12] Veach, Eric, and Leonidas J. Guibas. "Metropolis light transport." Proceedings of the 24th annual conference on Computer graphics and interactive techniques. 1997.
- [13] Lafortune, Eric. "Mathematical models and Monte Carlo algorithms for physically based rendering." Department of Computer Science, Faculty of Engineering, Katholieke Universiteit Leuven 20 (1996): 74-79.
- [14] Morokoff, William J. "Generating quasi-random paths for stochastic processes." Siam Review 40.4 (1998): 765-788.
- [15] Niederreiter, Harald. Random number generation and quasi-Monte Carlo methods. Society for Industrial and Applied Mathematics, 1992.
- [16] Eilemann, Stefan, Maxim Makhinya, and Renato Pajarola. "Equalizer: A scalable parallel rendering framework." IEEE transactions on visualization and computer graphics 15.3 (2009): 436-452.
- [17] Eilemann, Stefan, David Steiner, and Renato Pajarola. "Equalizer 2.0—convergence of a parallel rendering framework." IEEE Transactions on Visualization and Computer Graphics 26.2 (2018): 1292-1307.
- [18] Humphreys, Greg, et al. "Chromium: a stream-processing framework for interactive rendering on clusters." ACM transactions on graphics (TOG) 21.3 (2002): 693-702.