GPU-Accelerated Interactive Virtual View Synthesis from Light Field Images

Hyeonjin Jung[1], Joungil Yun[2], Won-Sik Cheong[2], Youngmin Yi[1]

[1]University of Seoul {hyeonjin507, ymyi}@uos.ac.kr
[2]Electronics and Telecommunications Research Institute {sigipus, wscheong}@etri.re.kr
Keywords: Light Field, View Synthesis, GPU, CUDA

ABSTRACT

We present a GPU based acceleration of a virtual view synthesis from multiple sparse Light Field (LF) images. For the synthesis of a 2K x 2K virtual view from 24 images of the same resolution, we achieved 21.31 FPS using four Titan V GPUs with algorithmic optimizations, which corresponds 923 times of speedup.

1 INTRODUCTION

As Virtual Reality (VR) and Augmented Reality (AR) are becoming more popular, virtual view synthesis techniques are gaining attention. It synthesizes images from the different view of interest generated by Head Mounted Device (HMD) and thus it allows immersive experience to the users, providing a wide range of application domains including education, sports, and so forth.

Recently, the virtual view synthesis technique for Light Field (LF) image data has emerged[1]. Since a single LF image frame typically consists of dozens of high-resolution images acquired from a set of cameras, it requires huge computations to synthesis a virtual view. In particular, the computations have to be processed in a short time in order to provide interactivity based on the movement of the users wearing the HMD. However, the current view synthesis technique takes dozens of seconds to synthesize a single virtual view.

On the other hand, it has been more than a decade since General Purpose Graphics Processing Units (GPGPUs) have emerged and can accelerate various applications other than graphics. LF view synthesis algorithms are well suited for GPU acceleration since it has abundant data parallelism. In this paper, we propose a GPU accelerated virtual view synthesis technique that efficiently exploits image level and pixel level data parallelism of LF images on a multi-GPU system, which achieves interactive throughput of 21.31 FPS for 24 sparse LF images of 2K x 2K resolution with depth information. To our best knowledge, this is the first work that accelerated virtual view synthesis from sparse LF images on GPUs.

2 BACKGROUND

2.1 Light Field Virtual View Synthesis

Light filed camera can capture the light field information of a scene such as light ray directions as well as light intensity[2]. These days, the acquisition of LF images has been actively developed and the LF cameras that can capture the LF images in 360 degree have been released by GoPro, Facebook, and Lytro[3].

Given a view information, a virtual view can be synthesized from the already acquired multiple LF images. First, the LF 2D images are unprojected to a 3D space, followed by affine transformation with the view information. Then, it is projected back to the 2D space. Interpolation and blending are performed to add color to the virtual view. The still remaining holes or cracks can be filled with nearby pixels. The entire process involves a dozens of images with a number of pixel operations, making it too slow to be interactive.

2.2 GPU Architecture and CUDA

Graphics Processing Unit (GPU) has been rapidly evolved and one can effectively accelerate the algorithms with fine-grain data-parallelism, exploiting thousands of cores in a GPU. A GPU that supports CUDA[4] consists of dozens of SMs, each of which in turn consists of dozens of CUDA cores. In CUDA programming model, a kernel is a routine that is executed on a GPU, typically by a massive number of CUDA threads. A kernel is comprised of CUDA blocks, each of which is comprised of CUDA threads. When a kernel is launched, CUDA blocks are assigned to SMs and the CUDA threads in the block is scheduled to the CUDA cores in the SM. Such architectures are well suited for the acceleration of applications with fine-grain data-parallelism such as LF virtual view synthesis.

2.3 Reference View Synthesizer

MPEG provides Reference View Synthesizer (RVS) as the reference code for the virtual view synthesis from multi views[5]. It implements a series of steps to synthesis a view with the given viewpoint from the LF images: Unprojection, Affine Transformation, Projection, Triangulation, Blending, and Inpainting. In Unprojection, the input reference view is transformed from 2D to 3D using the depth information. In Affine Transformation, it is transformed again in the 3D space according to the given viewpoint, generating a virtual view. In Projection, the virtual view is transformed back to the 2D space from the 3D space. Now, it knows the mapping information of the pixel in a reference view onto that in a virtual view. In Triangulation, it adds color to the virtual view using tri-linear interpolation method; it first forms a triangle with the neighboring pixels for every pixel in the reference view. Then, it can add color of the pixels in the virtual view since it knows the mapping information of the pixels as well as the color information of pixels in the reference view. In Blending, it aggregates the results of Triangulation into a

single image; the pixel in each virtual view colorized by Triangulation can have different colors, and it is reduced a single value, which is computed as a weighted average. Finally, the remaining holes after Blending are filled by Inpainting, which uses the color of the neighboring pixels in the final virtual view. The most time consuming parts are Triangulation and Blending, as will be explained in Sec. 5.

3 ACCELERATION USING GPUS

We parallelized the entire process of RVS on GPUs. Each step is implemented as an efficient GPU kernel after carefully examining the parallelism in the step. Also, multiple GPUs are utilized and they are synchronized efficiently, which are required for the reduction in Blending, are studied.

3.1 Parallelization

The number of LF images used in this paper is 24 and the resolution of each image is 2K x 2K. Thus both image level parallelism and pixel level parallelism are abundant. Most steps except for Triangulation process the same computations for each pixel, which makes them well suited for the GPU computing.

We exploit image-level parallelism with multiple GPUs and pixel-level parallelism with the thousands of CUDA core in a GPU. Most steps do not require any reduction but Blending does and also needs memory copy of the partial results among multiple GPUs, which will be explained in detail in Sec 3.2.

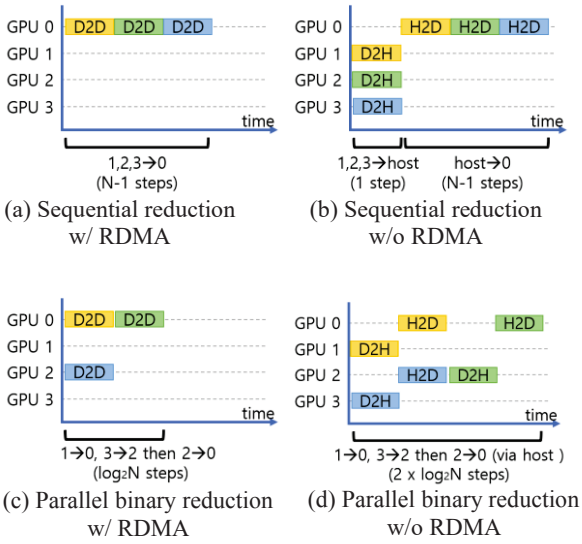
The GPU execution can be divided largely into the host-to-device data transfer, the execution of the kernels, and the device-to-host data transfer. Since each of these operations utilizes the different parts in a GPU, they can be processed asynchronously; for example, while the current image is processed in the kernel by CUDA cores, the host-to-device data transfer can be processed at the same time by the DMA controller, or the copy engine, overlapping the computation and data transfer.

3.2 Efficient Inter-GPU Reduction in Blending

Before Blending, every step can be executed in parallel in different GPUs, processing dozens of input images distributed over the multiple GPUs. In Blending, the final result image is obtained by performing weighted sum reduction. For example, if four GPUs are used for 24 images, each GPU is assigned 6 images, which is reduced locally by simply performing accumulation. The partial result image in each of the four GPUs should be reduced into the final image by copying it to the other GPUs. Since the partial result image size is 80MB when its resolution is 2K x 2K and the format is YUV, the data transfer through a PCI bus would take more than 10 ms if it is copied via the host. This makes the Blending, as well as Triangulation, the performance bottleneck and efficient reduction is required.

A naïve way for the reduction is to copy all the partial images to the same GPU and it accumulates them sequentially. When there are N GPUs, N-1 copies are required and N-1 accumulations are done sequentially.

To mitigate this problem, one can employ Parallel Binary Reduction, where each pair of GPUs can perform the reduction in parallel. For example, when four GPUs are used, GPU1 copies its partial result to GPU0 at the same time GPU3 does to GPU2. Then, GPU0 and GPU2 can perform the reduction in parallel. Finally, GPU2 copies its reduced partial result to GPU0 for the final reduction. If supported by the system, the copies can be done in parallel. The sum reduction can be done in parallel in pairs, which would take $\log_2 N$ steps. Note that the synchronization is required at every step to prevent any data race.



(a) Sequential reduction w/ RDMA (b) Sequential reduction w/o RDMA

(c) Parallel binary reduction w/ RDMA (d) Parallel binary reduction w/o RDMA

Fig. 1 Copies needed for reduction in Blending

If a GPU supports Remote Direct Memory Access (RDMA)[6], the peer-to-peer data transfer between GPUs is faster than the case without RDMA. If RDMA is not supported, a GPU first needs to copy the data to the host and back to the target GPU; the peer-to-peer data transfer rate would be halved in this case.

Fig. 1 illustrates the aforementioned reduction schemes when four GPUs are utilized. With RDMA supported, *Sequential Reduction* (Fig.1a) requires 3 steps for the copies while *Parallel Binary Reduction* (Fig. 1c) requires 2 steps. However, if RDMA is not supported, each method would take two times more steps as it implicitly copies the data to the host first; *Sequential Reduction* would require 6 steps of copies and *Parallel Binary Reduction* 4 steps. Thus, instead of the device-to-device copy API, we explicitly call the host-to-device API followed by the device-to-host API. Then, *Sequential Reduction* requires 4 steps of copies (Fig. 1b) and *Parallel Binary Reduction* 4 steps (Fig. 1d). Depending on the GPU's support for RDMA and the number of GPU utilized, one can choose the most efficient scheme for peer-to-peer data transfer in Blending.

4 ALGORITHMIC OPTIMIZATION

As mentioned earlier, Triangulation is one of the major performance bottleneck since it computes tri-linear interpolation for all the triangles formed by each pixel and its neighboring pixels in the reference views. The maximum

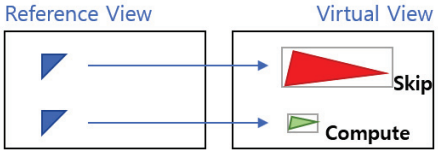


Fig. 2 Skipping triangles in Triangulation

number of triangles in a single 2K x 2K reference view can be around eight million.

We propose to skip unnecessary or insignificant triangles to reduce the computation time. First, we skip the meaningless pixels in the 2D space when Equirectangular Projection (ERP) input reference views are used; if the input format is ERP, there are blank areas with the default pixel values when it has been projected to 2D space in the Projection step. Those pixels can be easily and safely skipped. Second, we can skip a triangle if the depth difference of the three pixels in the triangle is large as illustrated in Fig. 2. The pixels in such triangles would have very small weight in Blending, contributing little to the final virtual view, while the computation time for tri-linear interpolation for such a triangle is large as the area of the triangle is large. We can skip those triangles, reducing much of the computation time, with no or negligible loss in picture quality. In fact, the interpolation results of such a triangle is not likely to be good, and could degrade the overall quality.

.

5 EXPERIMENTAL RESULTS

The server used in the experiments has four Titan V GPUs and Intel Xeon Gold 6130 CPU running at 2.10GHz. The OS used is Ubuntu 18.04.

The input data used is TechnicolorMuseum[7] where each frame consists of 24 sparse LF images of 2K x 2K resolution in ERP format with depth information. In this experiment, the time for reading the input file and rendering through HMD are excluded and the time only for the virtual view synthesis is included.

5.1 The Performance Results

Table 1 summarizes the execution time of each module in RVS on a single GPU. To correctly measure the execution time

Table 1 CPU and GPU execution time (ms)

	CPU	GPU (sync.)
Memcpy host-to-device	-	41.09
Preprocessing	1,851	4.66
Unprojection	2,461	4.22
Affine Transformation	625	4.18
Projection	989	4.08
Triangulation	30,214	34.92
Blending	20,688	2.55
Inpainting	170	0.29
Memcpy device-to-host	-	3.83
Synthesis time	56,943	101.56
	0.02 FPS	9.85 FPS

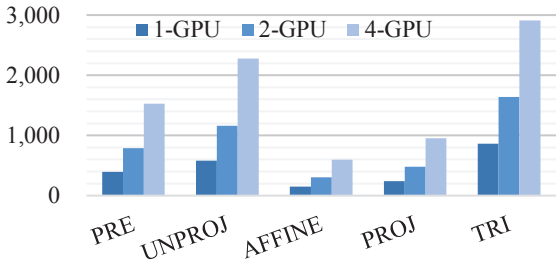


Fig. 3 The speedup of multi-GPU execution against CPU execution

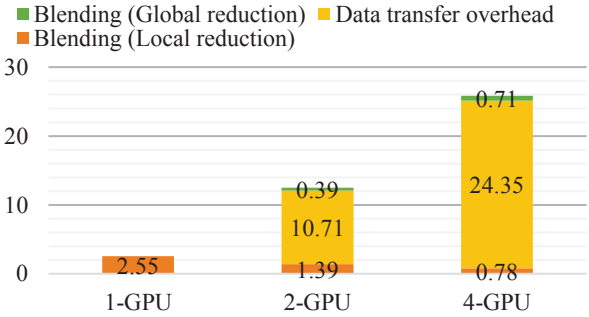


Fig. 4 Blending execution time (ms)

of each module, asynchronous execution is disabled and each module is synchronized. Most of the modules are significantly boosted by GPU execution as they have abundant of data-parallelism without any reduction. The end-to-end execution time on a GPU including the data transfer overhead between the host and the device is about 101 ms. Note that this is the time obtained with the synchronous execution. Out of 101 ms, it takes 41 ms for the data transfer. It is alleviated by the asynchronous execution as will be presented later.

Fig. 3 shows the speedup of each module except for the Blending module, compared to the CPU execution when varying the number of GPU from one to four: Preprocessing (PRE), Unprojection (UNPROJ), Affine Transformation (AFFINE), Projection (PROJ), and Triangulation (TRI). Hundreds of speedup is achieved with one GPU and up to thousands of speedups are achieved with two and four GPUs, scaling linearly to the number of GPUs.

In the CPU execution of RVS, Triangulation and Blending are the major performance bottlenecks accounting for 53% and 36% of the total time. The GPU execution of Triangulation without triangle skipping takes 164.9 ms, achieving 183x of speedup. With triangle skipping, the average number of triangles is reduced to about one million with 2.9x of reduction, and it takes 34.9 ms achieving 4.7x of further speedup, resulting in 865x of speedup for Triangulation. In Blending, the inefficient loop structure in RVS has been refactored first, which results in 3.1x of speedup from 20,688 ms to 6,590 ms. With GPU acceleration, it becomes 2.55 ms with 2,584x of further speedup.

Fig. 4 shows the execution time of Blending when the number of GPUs used varies. The processing time taken in

Table 2 End-to-end synthesis time (ms (FPS))

Method	(1) Sync.	(2) Async.
1-GPU	101.56 (9.85)	71.25 (14.04)
2-GPU	65.16 (15.35)	52.39 (19.09)
4-GPU	55.58 (17.99)	46.92 (21.31)

each GPU, denoted as *Local Reduction*, is reduced linearly to the number of GPUs since the 24 partial virtual views are equally distributed across all the GPUs. However, the data transfer overhead among GPUs become a performance bottleneck since Titan V does not support RDMA nor NVLink[8]. As explained in Sec. 3.2, if RDMA is not supported, the number of data transfer in *Parallel Binary Reduction* would be the same as *Sequential Reduction*, but would require synchronization. Thus we employ *Sequential Reduction* scheme. With RDMA support, *Parallel Binary Reduction* could be advantageous. With NVLink, the data transfer bandwidth is over 3x faster than that of a PCI bus, which could mitigate this overhead. The time taken in GPU0 after it has received all the partial outputs are denoted as *Global Reduction* and is negligible since the number of partial output views are only two or four, which is the number of GPUs used.

Table 2 shows the end-to-end time of both synchronous and asynchronous execution using 1, 2, and 4 GPUs. When the number of GPUs is doubled from one to two in synchronous execution, it becomes 1.56x faster. However, when it is doubled from two to four, it becomes only 1.17x faster. This is because the data transfer overheads (host-to-device and device-to-device) become the performance bottleneck.

By asynchronous execution, the data transfer overhead can be partially hidden with the kernel executions. When one GPU is used, asynchronous execution is 1.42x faster against synchronous execution. The speedup with asynchronous execution decreases as the number of GPUs increases due to the same reason.

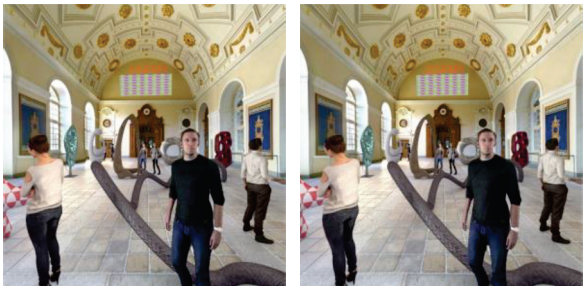
As a result, we achieved 46.92 ms per frame using four GPUs, which is 923x faster than the CPU execution time.

5.2 The Quality Evaluation

We have verified that the quality of GPU accelerated output image is almost the same as that of the reference CPU implementation. Root Mean Squared Error (RMSE) was used as a metric. Compared to a CPU output, RMSE of the GPU output without triangle skipping is 4.72 and RMSE of the GPU output with triangle skipping is 6.77. Fig. 5 shows the output images from both CPU and GPU implementations. They are indistinguishable with human eyes.

6 CONCLUSIONS

In this paper, we presented an efficient acceleration of virtual view synthesis from high-resolution sparse LF images with the depth information. With efficient parallelization and reduction using four GPUs, as well as algorithmic optimizations including triangle skipping, we could achieve 923x of speedup, which corresponds to 21.31 FPS.



(a) CPU Image (b) GPU Image

Fig. 5 Comparison of Image Quality

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00072, Development of Audio/Video Coding and Light Field Media Fundamental Technologies for Ultra Realistic Tera-media)

REFERENCES

[1] L. Yao, Y. Liu, W. Xu, "Real-time virtual view synthesis using light field," EURASIP Journal on Image and Video Processing, vol. 2016, pp. 1-10 (2016).

[2] Lee G., Lee E., Cheong W., Hur N., "Trend of Light-Field Image Acquisition and Representation Technology," Electronics and Telecommunications Trends. Vol. 31, No. 3, pp. 50-59 (2016).

[3] Jung J., Park Y., Yun K., Yun J., Cheong W., Seo J., "Trends in Acquisition and Service Technology for VR Media," Electronics and Telecommunications Trends. Vol. 33, No. 2, pp. 48-55 (2018).

[4] "CUDA Toolkit Documentation," https://docs.nvidia.com/cuda/.

[5] ISO/IEC JTC1/SC29/WG11, *Reference View Synthesizer (RVS) Manual*, W18068 (2018).

[6] "GPUDirect," https://developer.nvidia.com/gpudirect.

[7] ISO/IEC JTC1/SC29/WG11, *Technicolor 3DoF+ test materials*, M42349 (2018).

[8] "NVLink Fabric Multi-GPU Processing," https://www.nvidia.com/en-us/data-center/nvlink/.