https://doi.org/10.2352/EI.2024.36.13.ERVR-181

Real-time Stereoscopic Image-parallel Path Tracing

Erwan Leria, Markku Mäkitalo, Pekka Jääskeläinen
Tampere University, Finland
erwan.leria[at]tuni.fi

Abstract

Spatial reprojection can be utilized to lower the computational complexity of stereoscopic path tracing and reach real-time requirements. However it adds dependencies in the pipeline. We perform the handling of data dependencies through a task scheduler framework that embeds workload and dependency information at each stage of the pipeline. We propose a novel image-parallel stereoscopic path tracing pipeline, parallelizing the spatial reprojection stage, the hole-filling stage and post-processing algorithms (denoising, tonemapping) to multiple GPUs. Distributing the workload of the spatial reprojection stage to each GPU allows to locally detect the holes in the images, which are caused by non-reprojected pixels. For the spatial reprojection, denoising and hole-filling stages, we have respectively a speedup of $\times 2.75$*,* $\times 4.2$ *and* $\times 2.89$ *per GPU on animated scenes. Overall, our pipeline shows a speedup of* $\times 2.25$ *compared to the open source state-of-the-art path tracer Tauray which only parallelizes path tracing.*

Introduction

Stereoscopic rendering is a technique used for providing depth cues as used for virtual reality (VR) applications. In short, a 3D scene is rendered from two different viewpoints, one for each eye, mimicking the natural binocular vision. However, this increases the rendering workload compared to monoscopic applications, thus presenting additional challenges for real-time applications.

The critical flicker frequency (CFF) describes the frequency at which a flickering light stimulus is perceived as continuous rather than flickering. For real-time graphics, this is the frequency at which an image or a set of images should be rendered and displayed. In VR applications, below this frequency, a head-mounted display (HMD) user can be subject to cybersickness symptoms. In this work, we base the real-time constraint on the generally reported range for the human CFF 50–90 Hz, thus fixing the time budget to render a stereo frame between 11 and 20 milliseconds.

Path tracing is easily parallelizable and allow to generate photorealistic synthetic images. It simulates automatically global illumination effects. On the contrary, for rasterization, complex illumination effects (i.e, caustics, refraction, self reflection, etc) are trickier to handle. Moreover in video games, static lighting is usually baked into textures which restrains using the same textures for different views. However, the workload of path tracing is significantly too high for a single GPU in order to remain within the range of the human CFF or above and sustaining high quality at native display resolution [11, 5].

For VR, there is a significant overlap in the field of view of both eyes, so the computational workload can be reduced by reprojecting redundant computed pixels from one view to another. This is a widely used technique in real-time stereoscopic and multi-view applications [3]. The major issue is the computational cost to achieve a good trade-off between quality and performance. Distributing the workload to multiple GPUs can reduce the rendering time. However, in terms of data locality, some neighbouring pixels may be stored in different GPUs, complicating post-processing algorithms that require spatial information such as spatial reprojection or denoising.

This involves cautious workload management, where the amount of pixels to be processed by a GPU (either for one or two views) differs depending on the stage that is executed: fewer pixels are processed before spatial reprojection.

In this paper, we propose a novel image-parallel stereoscopic path tracing pipeline, where the spatial reprojection of pixels is used to efficiently exploit inter-view correlations; the spatial reprojection stage, the hole-filling stage and post-processing algorithms (denoising, tonemapping) are parallelized across multiple GPUs within a single node of four GPUs. Pipeline dependencies are managed through a task scheduling framework in which are embedded dependencies and workload distribution information. This pipeline has a data-dependence awareness to schedule the pipeline stages to minimize data movements. In order to adapt to dynamic content, we also propose an adaptive control loop which tunes rendering parameters on-the-fly to adjust the quality with respect to real-time constraints.

Background

Spatial reprojection takes pixels in a source image and reprojects them into a target image [1]. Both of these images are associated with a different viewpoint (i.e., left and right eye for stereoscopic HMDs). However, in stereoscopic path tracing, due to occlusions and disparity between the two viewpoints, some elements in the scene are visible for one eye but not for the other. Moreover, from one view to another, reprojection may not be possible because of the angle between the views not matching the reflection angle of some visible surfaces, since the associated BRDFs in the target view would differ from the BRDFs in the source view. This results in missing pixels in the target image that are not visible in the source image (see Fig. 1, first image with missing pixels). When reprojection is not possible, discarded pixels are left without any value, ending up being holes in the image. These holes simply represent missing information, which must be reconstructed after the spatial reprojection stage. This reconstruction stage is termed *hole-filling*, also known as image *in-painting* in the realm of image processing.

Prior works related to spatial reprojection [3, 14, 15, 8] are focusing on single GPU utilization. In this paper, we consider multi-GPU rendering and the relative distribution of the workload, including the real-time constraint as well.

Image-parallel (or sort-first) rendering [7] groups the different rendering techniques related to image (or screen) space subdivision. The image is divided into smaller regions, where the size of a region ranges from a single fine-grained pixel to a coarse tile [2, 16, 17]. As set out by the sort-first approach, our pipeline considers that the geometry of the scene fits entirely in the memory of each GPU; this is called scene data replication, as opposed to scene data distribution of the data-parallel (or sort-last) approach.

Related work

OO-VR [16] considers rasterization-based VR rendering for multi-GPU systems. Although they decompose the image into disjoint regions, the constraint of rasterization is that the triangles (primitives of the scene) have to be distributed among the GPUs in order to balance the workload such that each region only contains the triangle visible in the view frustum of that region, which becomes data-parallel rendering. In their work, each GPU renders the same region of the viewport in the left and right images. To reduce the computational cost of VR rendering, they rasterize the texture of triangles visible from the left and the right eye in one pass, instead of rendering them twice. However, their rasterization-based work does not consider photorealistic rendering, where different types of surfaces can cause view-dependent effects due to, e.g., reflections.

The work on real-time cluster path tracing proposed by Xie et al. [17] is the closest work to ours, since they consider at least path tracing in a single node and over a cluster using an image-parallel pipeline. However, they do not consider the stereo or multiview case, thus not addressing the problem of dependencies with spatial reprojection and parallel denoising.

Path tracing is also used in Tauray, a state-of-the-art open-source path tracer optimized for stereo and multi-view rendering proposed by Ikkala et al. [4]. In Tauray, the distribution of path traced pixels increases the GPU utilization and the balance between GPUs, but the pipeline stages after path tracing, such as denoising, are done only on the primary (or master) GPU. Tauray achieves $\sim$ 1.65 Hz for a single view at 32 spp in a $\times$2 RTX 2080 Ti GPUs setup for a scene composed of roughly 2.7M triangles.

Stereo image-parallel pipeline

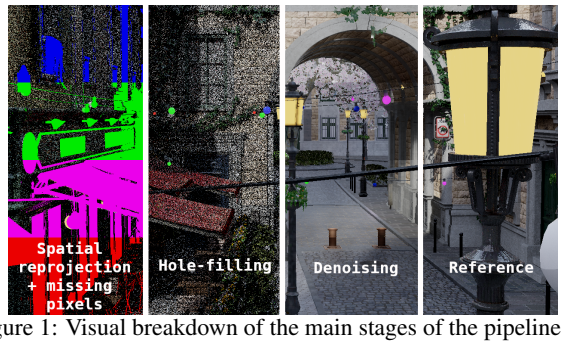

Figure 1: Visual breakdown of the main stages of the pipeline for the right eye image. Missing pixels from reprojection are encoded in blue, green, magenta, red, each color corresponding to the GPU in which they are residing.

In this section we present our proposed approach for stereoscopic image-parallel path tracing. We discuss the challenges and theoretical limitations.

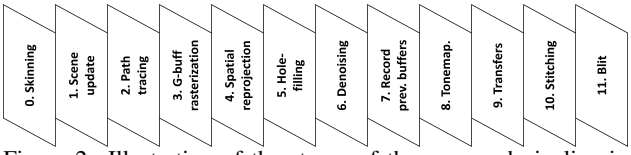

Figure 2: Illustration of the stages of the proposed pipeline in sequential order of command buffer submission

Pipeline overview

Our image-parallel pipeline is composed of 10 stages as depicted in Fig. 2. Basically, one view is path traced (stage: #2), and then the rendered pixels are reprojected into the second view (#3, #4). The pixels that could not be reprojected are then filled with path tracing (#5). After these rendering stages, post-processing algorithms are applied (#6, #8), and the image regions are sent to the primary GPU (#9) and pushed to the swapchain (#11).

For the sake of simplicity, in the rest of this paper we consider that spatial reprojection is done from the left-eye view to the right-eye view. First, our image-parallel renderer requires an image-space decomposition algorithm. Since eyes are separated horizontally by a constant distance (interpupillary distance), missing pixels are more likely to appear along horizontal directions. The shuffled strips pattern [2], used by default in Tauray limits the overall rendering speed when we consider dependencies between pixels for algorithms such as spatial reprojection, hole-filling and denoising. Indeed, using smaller non-contiguous path traced regions per image and per GPU increases the rate of discarded pixels. In other words, a pixel in the left view cannot reach its reprojected position in the right view if this position belongs to another GPU's memory. In order to cover the largest amount of future discarded pixels for each GPU, we choose to decompose the viewport of each view into N rectangular regions, each covering the entire width of the viewport. They are each assigned to a GPU, such as done in OO-VR [16], N being the number of GPUs. Having large pixel regions minimizes the amount of missing pixels and favors pixel operations that consider spatial coherence of pixels. However, this makes the balance of the path tracing workload uneven. The complexity of path tracing each pixel varies depending on the surfaces a ray encounters. In this work we do not consider load balancing between GPUs, since we want to improve data locality and to have a fixed amount of pixels processed by each GPU during the pipeline execution.

Task scheduling framework

The role of the task scheduling framework is to help the renderer to take decisions regarding the distribution of the workload and dependencies. In the same way as in the hierarchical structure of the distributed framebuffer (DFB) [12], we define a GPU which is the owner of the left and right views. In the DFB, screen regions are assigned in a round robin fashion, which we do as well in our pipeline.

For each GPU, the task scheduler keeps track of the workload ratio that the GPU has for each view. The workload ratio is a value between 0 and 1, which allows to parameterize the amount of pixels that must be processed by each GPU [6, 2]. For each stage, we construct a pair p_i of workload ratios per GPU i, the values of the pair correspond to the ratio of left and the right eye.

Once all workload ratios are set, the GPU commands are buffered at each stage.

A problem is that the two views are already allocated in the memory of the GPUs, however when the source must be path traced, the target view must be ignored at this stage. This means that for a GPU in a node of 4 GPUs, the workload ratio $w_{i,j}$ per GPU i and per view j during the path tracing stage (#2) is $p_i = \{0.25, 0.0\}$, where $w_{i,0} = 0.25$ is one fourth of the left view and $w_{i,1} = 0.0$ is used to ignore the right view. After spatial reprojection, only the right view has to be path traced to fill the holes (#5), with reprojected pixel being discarded in this pass, so $p_i = \{0.0, 0.25\}$. In the case of denoising and tonemapping (#6 and #8), both left and right image regions must be processed by their associated GPU, leading to have $p_i = \{0.25, 0.25\}$. This allows to handle the change in the amount of pixels to process from one view to another and from one stage to another.

Regarding dependencies between GPUs, to define which GPU has to send its image regions to which GPU, we build a non-binary tree, where each node of the tree represents a GPU embedding its workload ratio information. The position of the incoming image regions from the child nodes to the image of the parent (owner) node is retrieved from the assignment order of the image-space decomposition algorithm, which is a simple round robin assignment of rectangular regions in our pipeline. All transfers are executed from the deepest layer to the root node. The dependency tree of our pipeline has only two layers: the owner (root node) and its child nodes.

Control loop

To force the rendering time to be in the $11--20$ ms range, we use a control loop. The control loop adjusts the number of samples per pixel, the number of bounces and the number of iterations for the SVGF denoiser [9]. As soon as the rendering time gets outside of the interval $11--20$ ms, the control loop will adjust the parameters such that the rendering time will be predicted to be within the interval for a given parameter. As soon as the predicted rendering time $\hat{\Upsilon}$ reaches the interval I, the algorithm is stopped. We assume the rendering time to evolve linearly in proportion to the parameters, except for the rendering frequency, which follows the tendency of an inverse function.

For a parameter p with a value x_p, we get the execution time of all the stages s_p that depends on this parameter:

$$T(\cup_{s_p}) = \sum T(s_p)\,. \tag{1}$$

By injecting Eq. 1 into the following equation we predict how long it takes to execute one unit u_p of this parameter:

$$T(u_p) = \frac{T(\cup_{s_p})}{x_p}\,, \tag{2}$$

with $T()$ representing a function returning the execution time of its input. In practice, $T(s_p)$ is directly obtained from the renderer.

We predict the execution time of the pipeline by recalculating the execution time of the stages influenced by the parameter p. Note that the predicted time of path tracing parameters depends on the path tracing stage and also the hole-filling stage in our pipeline, hence the emphasis on the notation $\cup_{s_p}$ in that case. If the rendering time must be adjusted to improve the quality, the value x_p of the parameter p is increased, otherwise it is decreased. The new parameter $\hat{x}_p$ is used to determine if the newly calculated rendering time $\hat{\Upsilon}$ is above or below a fixed bound τ, τ being either 11 ms or 20 ms. Within the interval $[11, 20]$, the parameters are not modified. For each parameter we fix an upper limit l_u or a lower limit l_l to restrain the control loop from over-tuning a parameter that would not increase the quality up to some point; in practice, we set these limits empirically.

Implementation details

In this section we discuss the implementation details of our approach. We implemented our pipeline inside Tauray [4] (from the git commit hash `4cdf4c4765e97a983ca4fe804f9a80ff5df86c76`).

The implementation of spatial reprojection is based on rasterized G-buffers containing information such as world positions, material and normals of the visible surfaces. The time to generate G-buffers increases for large scenes containing a lot of triangles, eventually it becomes a limiting factor in our pipeline in terms of workload distribution. Since we employ an image-parallel approach, we replicate the scene into each GPU, which means each GPU has identical data-space (i.e, scene primitives). Therefore, the image-space of a GPU cannot force rasterization-based computation to process triangles only visible in its associated region view frustums, because triangles are tied to data-space which we do not partition according to the image-space. This leads to an identical G-buffer rasterization time for all the GPUs. Employing a hybrid approach between sort-first (image-parallel) and sort-last (data-parallel) for multi-GPU rasterization-based computation makes sense when rendering itself is based on rasterization, as done in [16]. However, in our case, since we use path tracing, mixing our image-parallel pipeline with a data-parallel approach would completely change the intrinsic design of our pipeline architecture. This would eventually make G-buffer rasterization quicker, but it would also involve additional inter-GPU communications to transition rays between different data domains [13, 18], which we want to avoid in order to keep a low latency. In spatial reprojection we discard pixels whose primary rays hit glossy/specular surfaces. Therefore, we choose to reproject only diffuse surfaces.

In the baseline, the Tauray path tracer, the stages are executed as follows: #0, #1, #2, #9, #10 #3, #4, #5, #6, #7, #8. The images are put into the swapchain directly during the tonemapping stage, and therefore blitting the images to the swapchain (#11) is not part of its pipeline. In Tauray, stages from #3 to #8 are only executed on the primary GPU. In contrast, in our proposed pipeline, only stages #10 and #11 could not be parallelized on multiple GPUs.

For denoising, we use the SVGF algorithm [9] in order to provide a fair comparison to the Tauray baseline, which already implements single-GPU denoising with SVGF. However, multi-GPU denoising requires special consideration due to seam artifacts at the GPU region boundaries, hence the optional bilateral seam filtering in our pipeline.

The image buffers of the swapchain are located on the primary GPU (#11), which involves a final transfer from the secondary GPUs, to the primary GPU, which is the main bottleneck.

For transfers between GPUs, we do not use peer-to-peer Direct Memory Access (DMA) nor NVLink technology; this is not utilized in Tauray either. This is due to Vulkan drivers limitations on Linux (at least at the time of the experiments). We perform transfers in the following order: images are first copied from a GPU to the host, then sent to the primary GPU from the host. Direct peer-to-peer access would theoretically halve the transfer timings, bypassing the copy to/from the host. In our pipeline the tonemap stage #8 is not tied to the Vulkan swapchain, unlike Tauray. Therefore, to reduce the time of data transfers from secondary GPUs to the primary GPU, we output the images from the tonemap stage into a buffer that has an `R8G8B8A8 UNORM` format. Image regions are then copied to their right place in the owner's image buffer which is also an `R8G8B8A8 UNORM` buffer.

Experimental setup

In our experiments, we use a server composed of 4× RTX A6000 GPUs 48 GB GDDR6, PCI-E 4.0 ×16. Our stereo images have each an HD resolution (1280×720 per eye). The interpupillary distance between the right and left eye is fixed to 62 millimeters. Unless specified otherwise, in each of our experiments we use 1 spp and 6 bounces.

For the purposes of this paper, we also implemented a hole-filling stage into Tauray, as it did not yet provide such a feature at the time of this writing.

Multi-GPU performance

	Tauray	**Proposed**
Pipeline stages	Execution time (ms)	
0. Skinning	0.003	0.08
1. Scene update	0.09	0.09
2. Path tracing	3.45	3.07
3. G-buffer rasterization	2.19*	2.61
4. Spatial reprojection	0.11*	**0.04**
5. Hole-filling	3.38*	**1.17**
6. Denoising	3.65*	**0.87**
7. Record previous buffer	0.45*	0.53
8. Tonemapping	0.07*	0.03
9. Transfer to primary GPU (S+R)	1.05 + 0.60*	0.26 + 0.29*
10. Stitching image regions	0.07*	0.03*
11. Blit images to the swapchain	n/a	0.12*
	GPU utilization (%)	
Averaged utilization per GPU	99* | 53	94* | 97

Table 1: **Per-stage** timings for the two rendering pipelines averaged over four RTX A6000 GPUs and averaged over 500 stereo-frames (resolution: $2 \times 1280 \times 720$) with San Miguel (no animation). The asterisk **(*)** indicates values measured on the primary GPU. The separator | separates values related to the primary GPU only and values related to the four GPUs. *S* stands for Send and *R* for Receive.

Scene	**San Miguel**	**Sponza**	**Bistro Ext.**
Pipeline	Execution time (s)		
Tauray	1.59	1.93	3.26
Proposed	**0.87**	**0.69**	**1.54**

Table 2: Per-scene global rendering makespan over a sequence of successive stereo-images. San Miguel has 104 frames with animated camera motion, and both Sponza and Bistro Exterior have 100 frames with a static camera.

The different execution times in Tab. 1 represent the time it takes for a GPU (on average) to execute a stage for the San Miguel scene. The sum of all of the stages is not necessarily the total rendering time since some stages can overlap, especially data transfers.

By executing spatial reprojection on multiple GPUs, we see that stages such as hole-filling and denoising are drastically faster, respectively by a factor ×2.89 and ×4.2 due to the workload being parallelized. A limitation of the hole-filling(#5) stage is warp divergence, which restrains full performance gain when multiple GPUs are used. Since discarded pixels are not spatially close to each other in image-space, this results in some threads of the same warp not being fed with enough work while others are working. This causes warp divergence to occur, implying incoherent memory access.

The rendering time of a full stereo image depends on the last GPU that completes its work. The difference in the rendering time between the GPUs is not significant; especially we observe in Tab. 1 that GPU utilization with the proposed pipeline is very high on average, showing that there is not so much load imbalance. We also see that the execution time of the primary GPU on average is the upper limit of the rendering time caused by additional stages to be computed on that GPU. We monitored the GPU utilization with nvidia-smi over 500 frames of a static viewer position and orientation. In Tab. 2 we report the makespan (including host time) for our three test scenes (shown in Fig. 3) for a given number of frames. The makespan is the total duration of time it takes to complete some tasks. With our pipeline, we reach a rendering frequency of ∼ 120 Hz for animated San Miguel. We define the rendering frequency as the frequency at which the renderer can provide *simultaneously* the left and the right eye images (stereo image). We observe, respectively for the San Miguel, Sponza and Bistro Exterior scenes, a speedup of ×1.83, ×2.80 and ×2.11 compared to Tauray, resulting in a ×2.25 speedup factor on average.

Xie et. al. [17] reported on a 10× Quadro RTX 8000 GPUs node, for a single view at 30 spp and 10 bounces and a resolution of 1280×720, for a scene containing 964k triangles, a rendering frequency of ∼ 3 Hz. With a scene of approximately 1M triangles (Bistro exterior) and two views, each at 1280×720 resolution, our pipeline runs at ∼ 1.85 Hz to render two views, leading to roughly ∼ 3.7 Hz to render a single view on our 4 GPUs setup with 30 spp and 10 bounces which shows that our pipeline is competitive against other image-parallel pipelines [4, 17], especially for stereo path tracing.

Control loop

The rendering time of our pipeline with the proposed control loop is shown in Fig. 4. We observe that after the control loop tunes the number of samples per pixel, the rendering time remains between 11 ms and 20 ms. It demonstrates the convergence of our control heuristic within the CFF range. The fluctuating rendering time, while within the target range, is caused by the animated camera motion. The average number of samples per pixel during the execution of our pipeline with the control loop is 3.94 spp and the makespan is 1.811 seconds, which corresponds to an average rendering frequency of ∼ 57 Hz. On the other hand, our pipeline without the control loop and Tauray both render the scene at 1 spp.

(a) (b) (c)

Figure 3: 4096 spp reference images of the test scenes (right eye): (a) San Miguel © Guillermo M. Leal Llaguno, (b) Sponza © Frank Meinl - Crytek, (c) Bistro Exterior © Amazon Lumberyard.

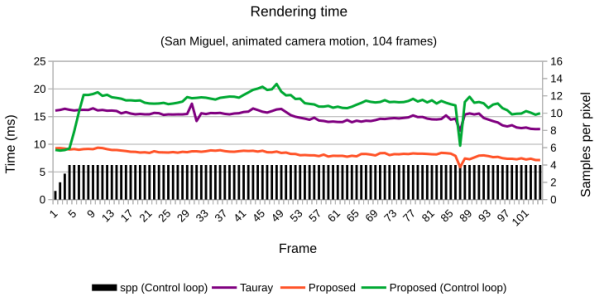

Figure 4: Overview of the rendering time and spp of stereo images of our pipeline with our proposed control loop (green) against the pipeline without the control loop (orange, constant 1 spp) and against Tauray (purple, constant 1 spp).

Scene	San Miguel	Sponza	Bistro Ext.
PSNR (no filter)	20.30	21.73	20.89
PSNR (bilateral filter)	19.90	21.71	20.46
PSNR (no seam)	20.83	21.97	21.09
CAMBI (no filter)	0.06	0.05	0.09
CAMBI (bilateral filter)	0.06	0.06	0.09
CAMBI (no seam)	0.07	0.06	0.09

Table 3: Quality metrics for different scenes. PSNR is measured around seam regions (1280×8 per seam), and CAMBI is measured for the whole image. San Miguel has dynamic camera movement, whereas Sponza and Bistro Exterior are static scenes.

Image quality measurements

To compare the quality of our images, we use the PSNR and Contrast-Aware Multiscale Banding Index (CAMBI) [10] metrics. CAMBI is a no-reference objective metric designed to quantify the annoyance of banding artifacts based on subjective assessments, unlike PSNR which is not correlated to subjective scores. Seam artifacts are due to a difference of contrast between different image regions. This is caused by pixels being denoised with a loss of spatial information, since their upper or lower neighbours are not residing in the same GPU. Therefore, we use CAMBI (through the `libvmaf` tool) to estimate the negative effects of the seams between different baselines; lower CAMBI scores indicate that the artifacts are less annoying.

In our work the quality is strongly dictated by the denoiser. For denoising we use SVGF, with the following parameters $\sigma_l = 32.0$, $\sigma_z = 128.0$, $\sigma_n = 4.0$. In particular, we are interested in the variation of quality that the denoising may introduce into our pipeline due to data locality. We use the PSNR to compare the quality of images containing seam artifacts against two other baselines: filtered seam artifacts with bilateral filter, and no seam artifacts (Tauray). The reference images for the PSNR measurements are a 4k spp reference without seam artifacts. Since we are using 4 GPUs, each rendering a rectangular region in the images, there are 3 seam artifacts between image regions rendered by different GPUs. These artifacts are denoted as seam 0, seam 1 and seam 2. Hence, we apply a bilateral filter around the seams, and evaluate its effect on the perception of these artifacts.

Fig. 5 shows the variation of PSNR around seam artifact regions for the San Miguel animated scene. As expected, the absence of seam artifacts yields the highest PSNR. On the other hand, we observe that the use of a bilateral filter does not increase the PSNR against unfiltered seam artifacts, but rather, the PSNR values of the unfiltered seam regions seems to slightly dominate the PSNR values of the bilateral filter. The bilateral filter is expensive to compute and is therefore applied only around seam regions. This means that for seam regions with a high contrast difference and high frequency details, the spatial borders of the bilateral filter may end up producing seam artifacts on their own between the filtered and unfiltered regions.

Tab. 3 summarizes the PSNR values around seam regions and CAMBI values for the entire image for the three test scenes. The common behaviour of PSNR and CAMBI is that they have stable values between the evaluated approaches. Even though Tauray produces images where the seam regions have about 0.2–0.9 dB higher PSNR values compared to our proposed pipeline, CAMBI estimates that the subjective visual quality is not significantly different with the seam artifacts and with the bilateral filter. This discrepancy suggests that both PSNR and CAMBI may fail to capture these type of artifacts; a more thorough evaluation would be required for conclusive results about the seam filtering approach.

Conclusions

We proposed an image-parallel stereoscopic path tracing pipeline where spatial reprojection, hole-filling and denoising are parallelized onto multiple GPUs. In contrast, previous research works on distributed path tracing that utilize reprojection and denoising do not parallelize those stages onto multiple GPUs. We showed that by parallelizing the workload of spatial reprojection, the information about the missing pixels can be directly exploited locally on each GPU, which allows to scale the workload of hole-filling, denoising and tonemapping to multiple GPUs rather than having the primary GPU assuming the whole workload while other GPUs are idle.

Acknowledgment

This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 956770, and in part by the Academy of Finland under Grant 325530 and Grant 351623.

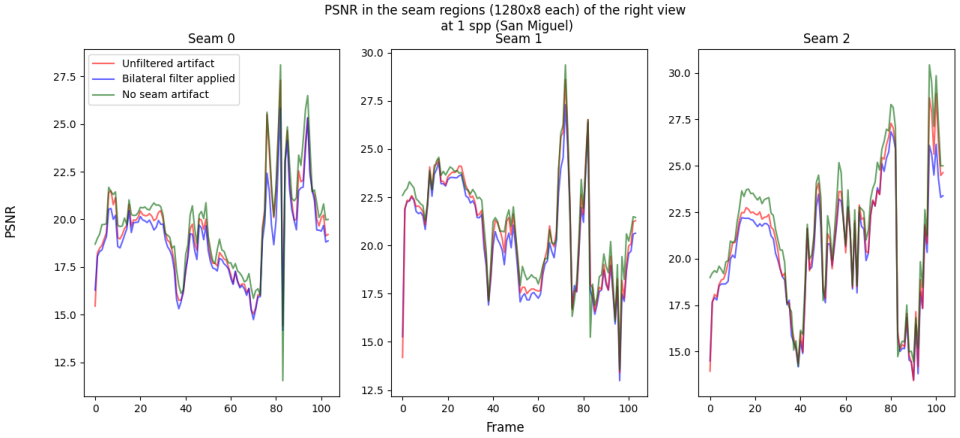

Figure 5: Graph of the PSNR around each seam artifact at each frame.

References

[1] S. J. Adelson and L. F. Hodges. Visible surface ray-tracing of stereoscopic images. In *Proceedings of the 30th Annual Southeast Regional Conference*, ACM-SE 30, New York, NY, USA, 1992. ACM.

[2] D. v. Antwerpen, D. Seibert, and A. Keller. A simple load-balancing scheme with high scaling efficiency. *Ray Tracing Gems*, 2019.

[3] L. F. Hodges, S. Adelson, S. J. Adelson, et al. Stereoscopic ray-tracing. In *The Visual Computer*. Citeseer, 1993.

[4] J. Ikkala, M. Mäkitalo, T. Lauttia, E. Leria, and P. Jääskeläinen. Tauray: A scalable real-time open-source path tracer for stereo and light field displays. In *SIGGRAPH Asia 2022 Technical Communications*, SA '22, New York, NY, USA, 2022. ACM.

[5] P. Kelly, Y. O'Donnell, K. Elst, J. Cañada, and E. Hart. *Ray Tracing in Fortnite*, pages 791–821. 08 2021.

[6] E. Leria, M. Mäkitalo, J. Ikkala, and P. Jääskeläinen. Dynamic load balancing for real-time multiview path tracing on multi-GPU architectures. *Virtual Reality & Intelligent Hardware*, 09 2022.

[7] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE computer graphics and applications*, 14(4), 1994.

[8] M. J. Mäkitalo, P. E. J. Kivi, and P. O. Jääskeläinen. Systematic evaluation of the quality benefits of spatiotemporal sample reprojection in real-time stereoscopic path tracing. *IEEE Access*, 8, 2020.

[9] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi. Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*. 2017.

[10] P. Tandon, M. Afonso, J. Sole, and L. Krasula. CAMBI: Contrast-aware multiscale banding index. In *2021 Picture Coding Symposium (PCS)*. IEEE, 2021.

[11] M. M. Thomas, G. Liktor, C. Peters, S. Kim, K. Vaidyanathan, and A. G. Forbes. Temporally stable real-time joint neural denoising and supersampling. *Proc. ACM Comput. Graph. Interact. Tech.*, 5(3), jul 2022.

[12] W. Usher, I. Wald, J. Amstutz, J. Günther, C. Brownlee, and V. Pascucci. Scalable ray tracing using the distributed framebuffer. In *Computer Graphics Forum*, volume 38. Wiley Online Library, 2019.

[13] I. Wald, M. Jaroš, and S. Zellmann. Data parallel multi-gpu path tracing using ray queue cycling. *Computer Graphics Forum*, 42(8):e14873, 2023.

[14] T. Willberger, C. Musterle, and S. Bergmann. Deferred hybrid path tracing. *Ray Tracing Gems*, 2019.

[15] N. Wißmann, M. Mišiak, A. Fuhrmann, and M. E. Latoschik. Accelerated stereo rendering with hybrid reprojection-based rasterization and adaptive ray-tracing. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2020.

[16] C. Xie, F. Xin, M. Chen, and S. L. Song. OO-VR: NUMA Friendly Object-Oriented VR Rendering Framework for Future NUMA-Based Multi-GPU Systems. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, New York, NY, USA, 2019. ACM.

[17] F. Xie, P. Mishchuk, and W. Hunt. Real time cluster path tracing. In *SIGGRAPH Asia 2021 Technical Communications*, SA '21 Technical Communications, New York, NY, USA, 2021. ACM.

[18] S. Zellmann, N. Morrical, I. Wald, and V. Pascucci. Finding Efficient Spatial Distributions for Massively Instanced 3-d Models. In S. Frey, J. Huang, and F. Sadlo, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2020.

Author Biography

Erwan Leria is an early stage researcher that conducts his joint doctoral studies between Tampere University and Mid-Sweden University as part of the Marie Sklodowska-Curie programme on Plenoptic Imaging (PLENOPTIMA). He works mainly as a doctoral researcher within the VGA research group at Tampere University. His research topic focuses on Distributed and Parallel Real-time Multiview Path Tracing for Light Field displays.

Markku Mäkitalo received the Ph.D. degree in signal processing from the Tampere University of Technology (TUT) in 2013. He is currently a Senior Research Fellow at Tampere University. His research interests include photorealistic real-time rendering, noise filtering and modeling, and immersive rendering technologies, such as light field rendering.

Pekka Jääskeläinen is an Associate Professor at Tampere University. He has researched heterogeneous platform customization and programming since the early 2000s. He is also an active contributor to various heterogeneous parallel platform related open source projects. Related to his work on implementing challenging real-time applications, he is interested in next generation distributed architectures for photorealistic real-time rendering.