Tauray: A Scalable Real-Time Open-Source Path Tracer for Stereo and Light Field Displays

Julius Ikkala
julius.ikkala@tuni.fi
Tampere University
Tampere, Finland

Markku Mäkitalo
markku.makitalo@tuni.fi
Tampere University
Tampere, Finland

Tuomas Lauttia
tuomas.lauttia@tuni.fi
Tampere University
Tampere, Finland

Erwan Leria
erwan.leria@tuni.fi
Tampere University
Tampere, Finland

Pekka Jääskeläinen
pekka.jaaskelainen@tuni.fi
Tampere University
Tampere, Finland

Figure 1: A simple scene including the "Stanford Bunny", rendered with Tauray in real-time on an RTX 3090, shown from two different angles on the Looking Glass Portrait light field display. Left pair: denoised path tracing at ~50 ms per frame. Right pair: DDISH-GI [Ikkala et al. 2021] at ~15 ms per frame.

ABSTRACT

Light field displays represent yet another step in continually increasing pixel counts. Rendering realistic real-time 3D content for them with ray tracing-based methods is a major challenge even accounting for recent hardware acceleration features, as renderers have to scale to tens to hundreds of distinct viewpoints.

To this end, we contribute an open-source, cross-platform real-time 3D renderer called Tauray. The primary focus of Tauray is in using photorealistic path tracing techniques to generate real-time content for multi-view displays, such as VR headsets and light field displays; this aspect is generally overlooked in existing renderers. Ray tracing hardware acceleration as well as multi-GPU rendering is supported. We compare Tauray to other open source real-time path tracers, like Lighthouse 2, and show that it can meet or significantly exceed their performance.

CCS CONCEPTS

• **Computing methodologies** → **Real-time simulation**; **Ray tracing**; **Image-based rendering**; **Mixed / augmented reality**; **Virtual reality**.

KEYWORDS

Computer graphics, Real-time, Path tracing, Multi-view, Light field

1 INTRODUCTION

We introduce an open-source, cross-platform real-time 3D rendering tool called **Tauray**. Tauray focuses on using ray tracing techniques for generating real-time content for multi-view displays, such as VR headsets and light field displays. It includes multiple different rendering modes, such as forward path tracing and DDISH-GI [Ikkala et al. 2021], and supports using multiple GPUs.

Even though some existing renderers do support rendering on multiple GPUs, their focus tends to be on increasing the total throughput, leveraging alternate frame rendering or focusing on high-spp dynamically scheduled tiled rendering. Tauray's multi-GPU support instead aims to minimize latency and only splits the workload in ways that also benefit low-spp rendering and do not introduce additional latency beyond necessary memory transfers.

Figure 1 shows Tauray rendering a scene in real-time on a light field display (Looking Glass portrait). The left photograph pair shows 1 spp path tracing with 5 light bounces denoised with SVGF, running at around ~50 ms per frame. The right-side pair shows DDISH-GI with an $8 \times 8 \times 8$ probe volume, with 256 8-bounce rays per probe, running at ~15 ms per frame. Both sides are rendering 64 different views at 512×683.

This work was supported by European Union's Horizon 2020 research and innovation programme under Grant Agreement No 871738 (CPSoSaware) and in part by the Academy of Finland under Grant 325530.
https://doi.org/10.1145/3550340.3564225

While there are already several open-source rendering tools available (a comparison table is included in supplementary material), the novelty of Tauray is in its unique combination of the following features:

- Open-source code. (https://github.com/vga-group/tauray)
- Hardware-accelerated path tracing.
- Multi-GPU support for real-time rendering.
- Real-time light field & virtual reality (VR) display output.

2 IMPLEMENTATION

Tauray uses the Vulkan API for rendering. Many Vulkan extensions are used in Tauray, but none of them are vendor-specific. These choices keep Tauray from being locked to just one GPU vendor or operating system. Tauray runs on both Linux and Windows operating systems, though multi-GPU support is limited to Linux.

Tauray records command buffers and assigns descriptor sets only at the start of the program or when models are dynamically added to or removed from the scene; this reduces CPU overhead and provides the GPU drivers more opportunity for optimization. This approach is not optimal for rasterization-based rendering, because it prevents per-frame culling of drawcalls. Since ray tracing-based methods do not issue draw calls for individual scene objects but rather one command to start tracing rays, there is no need to modify the contents of command buffers for each frame.

Scenes are specified using Khronos' glTF 2.0 format. Keeping shading complexity low is crucial for ray tracing performance [Dunn 2019], so we keep our material model simple by limiting it to the core metallic-roughness workflow in glTF, plus transmission for transparent objects. Specifically, Tauray currently uses the isotropic GGX/Trowbridge-Reitz [Trowbridge and Reitz 1975; Walter et al. 2007] BSDF and Lambertian diffuse BRDF. Additionally, spherical lights with a radius and directional lights with non-zero angular diameter are supported through a custom plugin that extends Blender's glTF exporter.

2.1 Rendering modes

While Tauray supports many rendering modes for debugging, dataset generation and comparison purposes, its primary focus is on methods aiming for photorealism: a forward path tracer and DDISH-GI [Ikkala et al. 2021] are available. Both methods use the cross-vendor Vulkan extensions `VK_KHR_ray_tracing_pipeline` and `VK_KHR_acceleration_structure` for ray tracing.

The forward path tracer supports Next Event Estimation, Hash-based Owen scrambling [Burley 2020] and Russian Roulette sampling [Arvo and Kirk 1990; Kahn 1955]. SVGF [Schied et al. 2017] and BMFR [Koskela et al. 2019] are available for real-time denoising. Box and Blackman–Harris filters are available for primary ray sampling, in order to achieve anti-aliasing. Temporal Anti-Aliasing [Karis 2014] is also supported.

The DDISH-GI renderer supports locally rendered or streamed spherical harmonics probes. Both the client-side renderer and the probe server are available in Tauray. The streaming mode uses ZeroMQ [Hintjens 2013] and is resilient to poor bandwidth, high network latency and unstable connections. This method is well suited for light field rendering, because the probes can be reused for all views. The ray tracing workload is independent of the number of views and pixels, and multi-view rendering scales practically as well as plain rasterization does.

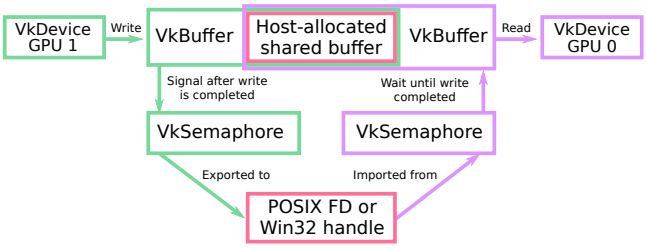

Figure 2: Diagram of the cross-GPU memory transfer as implemented in Tauray. Resources tied to the secondary GPU are marked with the green color, while the resources of the primary GPU are in purple. Host process resources are in red. When more than two GPUs are used, all non-primary GPUs form a similar pair with the primary GPU.

2.2 Multi-GPU rendering

Multi-GPU rendering is implemented so that devices from different Vulkan device groups can co-operate. Since there is currently no DMA extension that enjoys cross-vendor support, we pass the GPU-to-GPU memory transfers through host memory. This memory transfer is done in a way that avoids synchronizing the host process with the GPUs, as shown in Figure 2. We exploit two Vulkan extensions that are typically used for cross-API interoperation: `VK_KHR_external_memory_host` and `VK_KHR_external_semaphore`.

We use the external memory extension to create buffers with the same host-provided memory access for both GPUs taking part in the memory transfer. Then, we issue commands to write the transferred data from the sender GPU to this host buffer. Once that transfer is finished, we can issue a read command on the corresponding buffer on the other GPU. Because regular Vulkan semaphores do not work across devices, external semaphores are used to synchronize the read after writing on another GPU. During this process, while the OS and GPU driver on the host CPU most likely are involved, the host process (Tauray) itself does not need to synchronize with the GPUs for this memory transfer.

The ray tracing workload of each view is split between every GPU taking part in rendering. Tauray provides a way to fairly easily program new workload splitting methods; a scanline-based approach is included as an example, which is adequate for when the GPUs have matching performance. Due to our low-latency real-time aim, splitting the workload by alternate frame rendering (AFR) is not considered, as it does not reduce latency beyond a single GPU [Monfort and Grossman 2009]. Certain short tasks, such as scene data refreshes (typically in the order of 0.1 ms in total) and acceleration structure updates are duplicated on each GPU. This is done when transferring their results would incur greater latency or there is no guarantee of data compatibility between the GPUs.

2.3 Stereo & light field rendering

Both stereo and light field rendering use the same rendering architecture for multi-view rendering. Views are stored in an image array instead of separate images. This lets all viewport-related

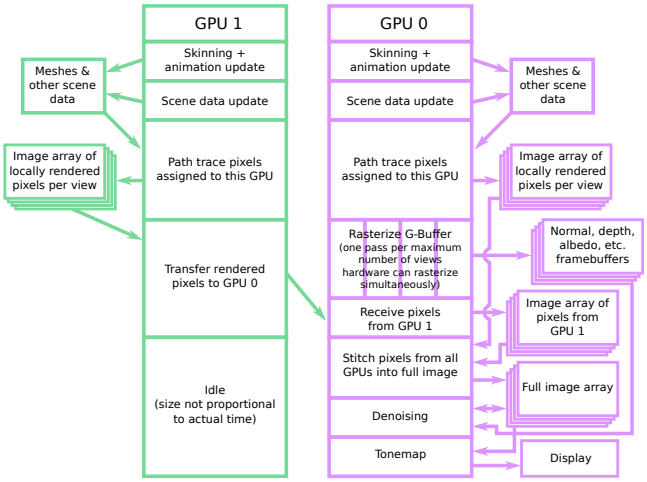

Figure 3: Typical multi-view, multi-GPU rendering pipeline diagram of Tauray; the details can change depending on parameters. The "G-Buffer" needed by many post-processing steps (like denoising) is rasterized on the primary output GPU in Tauray whenever possible. In our case, rendering the G-Buffer separately on the primary GPU is generally marginally faster than distributing it to multiple GPUs.

rendering stages of Tauray operate on multiple views in a single pass, which minimizes the overhead involved with launching and synchronizing shaders. Rasterization-based render passes are accelerated for multi-view rendering using the `VK_KHR_multiview` extension. Compute and ray tracing render passes operate on all views in one pass. In an example 128-view case, doing this allows the path tracing renderer to roughly halve the total frametime and go from about 90% GPU utilization to 99–100% utilization. Figure 3 shows an overview of the multi-view rendering pipeline in Tauray.

As an alternative for brute-force rendering of all viewports, Tauray also has a simple, real-time capable spatial reprojection implementation for quickly generating more viewports from only a few rendered viewports, though it does not fill in the disocclusions intelligently yet.

VR is supported with the OpenXR API. As an example platform for real-time light field rendering, Tauray also supports rendering to the Looking Glass light field displays [Looking Glass Factory, Inc. 2022]. Content for arbitrary multi-view displays can be generated with offline rendering by setting up a grid of cameras. Spatial and temporal reprojection modes are also available, enabling the reuse of samples across different views and frames.

3 COMPARISON TO RELATED WORK

We compare Tauray to three other renderers: Falcor, Lighthouse 2, and Blender (Cycles). The first two renderers were chosen, because they are also open-source renderers with similar real-time path tracing capabilities. Blender is used for offline rendering comparison due to its availability and high-quality Cycles renderer.

Performance is measured in three scenes: Sponza [The Khronos Group 2018] (~260k triangles), Emerald Square [Hull et al. 2017] (~2.7 million triangles) and Breakfast room [McGuire 2017] (~270k triangles). All scenes are lit by one punctual directional light. This choice was limited by each renderer having somewhat different feature sets, and this was the lowest common denominator for an identical lighting setup.

Table 1: Online (real-time) rendering performance of the renderers.

Sponza	1 spp	4 spp	8 spp
Tauray	8.66 ms	31.8 ms	62.9 ms
Tauray (dual-GPU)	**7.58 ms**	**19.1 ms**	**35.3 ms**
Falcor	12.9 ms	67.8 ms	131 ms
Lighthouse 2	11.5 ms	38.7 ms	73.7 ms
Lighthouse 2 (dual-GPU)	23.7 ms	61.3 ms	106 ms

Emerald Square	1 spp	4 spp	8 spp
Tauray	37.0 ms	145 ms	292 ms
Tauray (dual-GPU)	**21.9 ms**	**77.6 ms**	**153 ms**
Falcor	72.1 ms	357 ms	718 ms
Lighthouse 2	DNF	DNF	DNF
Lighthouse 2 (dual-GPU)	DNF	DNF	DNF

Breakfast room	1 spp	4 spp	8 spp
Tauray	**4.86 ms**	15.6 ms	30.3 ms
Tauray (dual-GPU)	5.33 ms	**10.6 ms**	**17.7 ms**
Falcor	8.05 ms	47.7 ms	95.4 ms
Lighthouse 2	5.29 ms	16.9 ms	32.0 ms
Lighthouse 2 (dual-GPU)	16.2 ms	32.9 ms	52.2 ms

All benchmarks in Tables 1 and 2 are measured when path tracing at 1920×1080 with 2 ray bounces (effectively 3, as all compared renderers implement next event estimation). Because the renderers do not provide identical denoising schemes, denoising is disabled. RTX 2080 Ti GPUs are used for the measurements. For all renderers, the timing measurements were full frame times as measured on the CPU. For Table 1, performance is averaged over 5 separate runs of 50 frames each. For Table 2, performance is averaged over 10 runs.

Lighthouse 2 was modified to do two light bounces instead of just one. Furthermore, the offline rendering benchmarks are done by using accumulation of 8 spp frames due to higher per-frame spp counts causing Lighthouse 2 to run out of memory on our setup.

In both online and offline cases, Tauray is consistently as fast or faster than the compared renderers. The dual-GPU setup in Lighthouse 2 seemed to be poorly supported: GPU utilization was low, 30–40% on both GPUs and its self-reported "frametime overhead" is in the order of ten milliseconds. Unfortunately, CUDA runs out of memory with Lighthouse 2 while loading the Emerald Square scene.

Figure 4 shows how Tauray scales lineary to path tracing multiple views simultaneously for real-time light field rendering. These measurements are done on a single RTX 2080 Ti. The Looking Glass Portrait is used as the output light field display, so timings include compositing the views into the format the display expects. Other than resolution and view count, settings are the same as earlier.

The multi-view rendering overhead depends greatly on the scene: at 128 views, Emerald Square and Sponza were about 26–29% slower

Table 2: Offline rendering performance of the renderers.

Sponza	32 spp	256 spp	1024 spp	4096 spp
Tauray	0.239 s	1.94 s	7.95 s	32.5 s
Tauray (dual-GPU)	**0.140 s**	**1.06 s**	**4.36 s**	**17.7 s**
Falcor	0.514 s	4.28 s	17.8 s	69.7 s
Lighthouse 2	0.331 s	2.44 s	9.61 s	38.9 s
Lighthouse 2 (dual-GPU)	0.413 s	3.32 s	13.4 s	53.5 s
Blender 3.2.2	3.42 s	15.2 s	55.6 s	217s
Blender 3.2.2 (dual-GPU)	2.87 s	8.67 s	29.0 s	111 s

Emerald Square	32 spp	256 spp	1024 spp	4096 spp
Tauray	1.13 s	9.19 s	37.5 s	152 s
Tauray (dual-GPU)	**0.607s**	**4.91 s**	**20.2 s**	**83.5 s**
Falcor	3.10 s	22.7 s	90.3 s	347 s
Lighthouse 2	DNF	DNF	DNF	DNF
Lighthouse 2 (dual-GPU)	DNF	DNF	DNF	DNF
Blender 3.2.2	10.5 s	43.2 s	155 s	601 s
Blender 3.2.2 (dual-GPU)	9.34 s	26.2 s	84.6 s	319 s

Breakfast room	32 spp	256 spp	1024 spp	4096 spp
Tauray	0.126 s	0.923 s	3.78 s	15.6 s
Tauray (dual-GPU)	**0.077 s**	**0.498 s**	**2.05 s**	**8.64 s**
Falcor	0.368 s	2.94 s	11.9 s	47.7 s
Lighthouse 2	0.125 s	0.961 s	3.92 s	15.6 s
Lighthouse 2 (dual-GPU)	0.192 s	1.53 s	6.36 s	30.2 s
Blender 3.2.2	2.76 s	14.9 s	56.6 s	223 s
Blender 3.2.2 (dual-GPU)	2.14 s	8.84 s	31.8 s	123 s

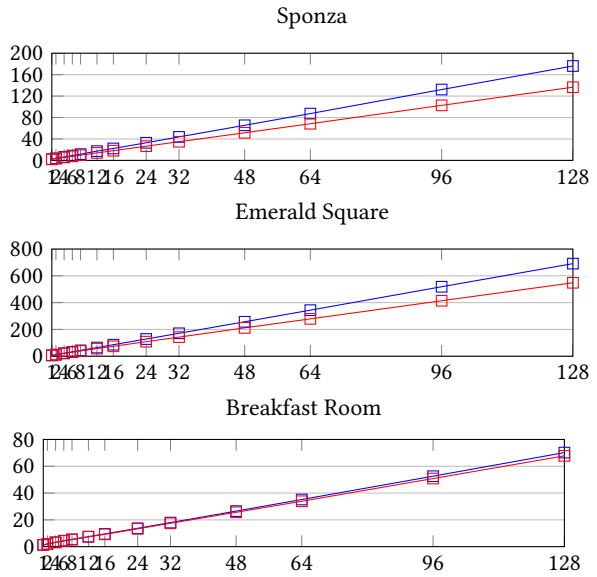

Figure 4: Path tracing performance (vertical axis, in ms) as a function of viewport count (horizontal axis) in each scene. The blue line represents performance with multiple views (512×512 each), while the red line represents single-view performance with an equivalent total number of pixels.

to render than the single-view equivalent with the same total number of pixels, while this same metric for the Breakfast room scene is only around 4%.

4 CONCLUSIONS

We introduced a scalable cross-platform real-time 3D rendering tool called Tauray. To our knowledge, it is the first open-source hardware-accelerated path tracer optimized for real-time rendering on light field and stereo displays. We demonstrated the optimized and scalable performance of Tauray: In both online and offline cases, Tauray's speed consistently matches or exceeds all compared renderers (Blender, Lighthouse 2, Falcor) and GPU setups. Tauray also scales efficiently for rendering multi-view content for VR and light field displays, roughly linearly with the number of views.

REFERENCES

James Arvo and David Kirk. 1990. Particle transport and image synthesis. In *Proc. 17th Annual Conference on Computer Graphics and Interactive Techniques*.

Brent Burley. 2020. Practical hash-based Owen scrambling. *Journal of Computer Graphics Techniques (JCGT)* 10, 4 (2020).

Alex Dunn. 2019. Tips and Tricks: Ray Tracing Best Practices. https://developer.nvidia.com/blog/rtx-best-practices/. Accessed: 2022-03-29.

Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc.

Nicholas Hull, Kate Anderson, and Nir Benty. 2017. NVIDIA Emerald Square, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/nvidia-emerald-square

Julius Ikkala, Petrus Kivi, Joel Alanko, Markku Mäkitalo, and Pekka Jääskeläinen. 2021. DDISH-GI: Dynamic Distributed Spherical Harmonics Global Illumination. In *Computer Graphics International Conference*. Springer.

Herman Kahn. 1955. *Use of different Monte Carlo sampling techniques*. Rand Corporation.

Brian Karis. 2014. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses* 1, 10.1145 (2014).

Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Trans. Graph.* 38, 5 (2019).

Looking Glass Factory, Inc. 2022. Product Overview. https://lookingglassfactory.com/product/overview. Accessed: 2022-04-04.

Morgan McGuire. 2017. Computer Graphics Archive. https://casual-effects.com/data

Jordi Roca Monfort and Mark Grossman. 2009. Scaling of 3D game engine workloads on modern multi-GPU systems. *Proceedings of the Conference on High Performance Graphics 2009* (2009).

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*.

The Khronos Group. 2018. glTF Sample Models. https://github.com/KhronosGroup/glTF-Sample-Models/tree/master/2.0/Sponza Accessed: 2022-06-07.

TS Trowbridge and Karl P Reitz. 1975. Average irregularity representation of a rough surface for ray reflection. *JOSA* 65, 5 (1975).

Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. 2007. Microfacet Models for Refraction through Rough Surfaces. *Rendering techniques* 2007 (2007).