High-Performance Graphics (2023)
J. Bikker and C. Gribble (Editors)

Surface Light Cones: Sharing Direct Illumination for Efficient Multi-viewer Rendering

Pascal Stadlbauer[1], Alexander Weinrauch[1], Wolfgang Tatzgern[1], Markus Steinberger[1,2]

[1]Graz University of Technology, Institute of Computer Graphics and Vision, Austria
[2] Huawei Technologies

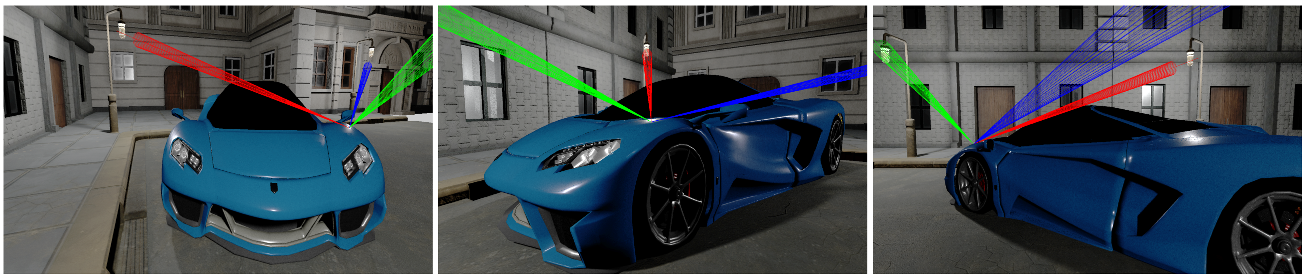


Figure 1: *Our proposed cone-based direct illumination information sharing abstraction enables efficient direct illumination rendering not only for single viewer setups but also for multi-user setups as can be seen here for the sports car model ([Yas], [DJM], [Zam]). Different views can be seen with the cone representation for a specific surface point.*

Abstract

Even though stochastic methods and hardware supported ray tracing are increasingly used for computing direct illumination, the efficient real-time rendering of dynamic area light sources still forms a challenge. In this paper, we propose a method for representing and caching direct illumination information using a compact multi-cone representation that is stored on the surface of objects. While shading due to direct illumination is typically heavily view-dependent, the incoming radiance for surface points is view-independent. Relying on cones, to represent the projection of the dominant visible light sources, allows to reuse the incoming radiance information across frames and even among multiple cameras or viewers within the same scene. Progressively refining and updating the cone structures not only allows to adapt to dynamic scenes, but also leads to reduced noise levels in the output images compared to sampling based methods. Relying on surface light cones allows to render single viewer setups 2-3x faster than random sampling, and 1.5-2x faster than reservoir-based sampling with the same quality. The main selling point for surface light cones is multi-camera rendering, For stereo rendering, our approach essentially halves the time required for determining direct light visibility. For rendering in the cloud, where multiple viewers are positioned close to another, such as in virtual meetings, gathering locations in games, or online events such as virtual concerts, our approach can reduce overall rendering times by a factor of 20x for as few as 16 viewers in a scene compared to traditional light sampling. Finally, under heavily constraint ray budgets where noise levels typically overshadow bias, surface light cones can dramatically reduce noise.

1. Introduction

Rendering systems in the cloud are on the way of revolutionizing the way we experience video games and virtual environments. However, these systems typically treat users as individual instances without considering that the respective image generation tasks are all executed within the same infrastructure. As a result, a large number of computations is potentially redundantly computed and thus computation power wasted. This may particularly be true for scenarios where multiple users are in the same area or look at the same object, such as showrooms or team-based games. In such cases, the rendering results are often similar, indicating a potential for merging computations. However, even in cases where potential sharing of computation seems straightforward, like stereo rendering for virtual reality (VR), computation sharing is hardly considered.

In general, real-time rendering realistic scenes involves computing various effects that are all contributing to the final output color.

DOI: 10.2312/hpg.20231137

While it seems natural to share computations for view-independent effects, which may be present in a shareable data structure, sharing view-dependent effects is significantly more difficult.

One essential effect is the shading of surfaces, or rather calculating the light emitted from surfaces towards the viewer, as expressed in the rendering equation [Kaj86],

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_\Omega f_r(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i)(\omega_i \cdot \mathbf{n}) d\omega_i,$$

where L_o is the outgoing radiance, i.e., the final color, L_e the radiance emitted from the surface, f_r is the Bidirectional Reflectance Distribution Function BRDF, L_i is the incoming radiance and ω_o the direction of the outgoing radiance (i.e. direction to the viewer).

The occurrence of ω_o clearly indicates that the result is view dependent and therefore sharing the final shading L_o between different views is not possible. However also a partial reuse of view-independent computations would be beneficial to reduce rendering costs. Looking at the rendering equation one can see the only view-independent parts are in the integral, which is also the most computational expensive part, due to it accumulating all incoming radiance. For specific surface setups where the BRDF is view-independent, for example for diffuse surfaces in combination with older reflectance models, the whole integral becomes view-independent and could be reused. Due to this only being an edge-case, we focus on reusing L_i and look at its individual parts splitting it into a direct and indirect part. The integral than can be expressed as,

$$\int_\Omega f_r(\mathbf{x},\omega_i,\omega_o)\left(L_{i,DI}(\mathbf{x},\omega_i) + L_{i,IND}(\mathbf{x})\right)(\omega_i \cdot \mathbf{n}) d\omega_i,$$

Direct illumination (DI) $L_{i,DI}$ is the radiance coming directly from a light source and indirect illumination $L_{i,IND}$ is the radiance arriving at a surface point by bouncing off other surfaces.

While indirect illumination is often smooth and thus exhibits little variance when changing viewing directions, DI, typically comes from well localized strong light sources. Due to smoothness multiple directional-dependent data structures have been proposed to share $L_{i,IND}$. However, for the highly directional-dependent DI, it can be more challenging to find an accurate representation. But finding a shareable representation and sharing results would reduce costly DI evaluations. If $L_{i,DI}$ comes from few point light sources, the evaluation of $L_{i,DI}$ comes down to evaluating whether the respective light sources are visible from $\mathbf{x}$. This information could even be shared using shadow maps or storing visibility information for each surface point in an on-surface cache [**?**]. The problem becomes more demanding and increasingly complex, when strong area light sources are involved. Computing their partial visibility from $\mathbf{x}$ is more time consuming and is often solved by sampling visibility through ray tracing. If multiple large light sources are found in a scene, tracing multiple shadow rays for each light sources for all visible points $\mathbf{x}$ in a view, quickly becomes a bottleneck.

In this paper, we propose the use of a cone-based representation to store incoming DI information. This directional based representation enables the sharing of rendering computations between different viewers for heavily view-dependent materials in scenes with large complex light source setups reducing overall computation costs significantly.

We make the following contributions:

- We present a cache-able data structure specifically designed for incoming radiance that can handle large area light sources. It is view-independent and can be stored on the surface of objects.
- We provide a progressive real-time generation and update method for our cone caches, which enables temporal reuse, adapts to fast moving light sources and allows the reuse between different viewers.
- We evaluate our caches in detail for single and multi-viewer scenarios, comparing them against simple and traditional data structures for caching radiance as well as optimized sampling strategies that keep information in screen space.

2. Related Work

In recent years, ray tracing has been increasingly used in real-time graphics. As such, sample reuse has been an important topic which is directly linked to caching radiance. A technique that recovers sampling distributions by on-line learning was proposed by [VKŠ*14]. Instead of reusing distributions samples can directly be stored and reused using reservoir resampling (ReSTIR) [BWP*20, LKB*22], which also works for global illumination [OLK*21]. As ReSTIR uses the BRDF for sample generation, such a sample cache is view-dependent and as such is only partially usable for multi-viewer DI, where different directions are responsible for view-dependent highlights.

Caching radiance likely goes back to Cabral et al. [CMS87] and has been widely explored for precomputed radiance transfer (PRT) typically in combination with spherical harmonics (SH) [SKS02].

Ng et al. [NRH04] proposed an accurate all-frequency PRT relighting approach, which requires high storage and precomputation costs. Analytic methods for area lights, such as Heitz et al. [HDHN16], do not consider shadows. There have been many subsequent developments in PRT, with Annen et al. [AKDS04] proposing spherical harmonic gradients for mid-range illumination, and Zhou et al. [ZHL*05] introducing dynamic scenes and near-field lights for all-frequency relighting. Ren et al. [RWS*06] used spherical harmonics for soft shadows in dynamic scenes, but only for sphere lights. Details on these developments can also be found in the (relatively dated) surveys on that topic [Leh07, R*09] Recent work in the direction of PRT uses analytic spherical harmonic gradients to represent complex light sources [WCZR20]. Recently, neural approaches have also been proposed for PRT [RBRD22]. However, while we also want to cache radiance, our focus is on real-time cache generation and rending dynamically lit scenes.

Keller et al. [Kel97] introduced Virtual Point Lights (VPLs) that are placed through a quasi-random walk. Shadows are determined by the use of a shadow map for each VPL. This and also more improved methods like [LSK*07] mostly generate diffuse shading and need a lot of VPLs to accurately represent realistic light setups. Derived methods like Progressive Lightcuts ([DGS12], [DKH*14]) deal with the memory issue, but still have problems with high directional lighting.

General approaches for irradiance and radiance caching have been used for global illumination [WRC88, KGPB05, GSHG98] and are to some degree related to our work although they typically do not capture high frequency. Those caches can actually be sampled sparsely and interpolated using sparse gradients [WH92, KBPZ06, JDZJ08]. For irradiance and smooth global illumnation, explicit caches can be built in real-time [KVS*14, MGNM19]. Recent approaches for radiance caching focus on neural structures, such as neural radiance caching [MRNK21], which can potentially be made faster with recent developments like instant neural graphics primitives [MESK22]. Similarly to our finding, neural radiosity [HCZ21] found that it is simpler to cache L_i than L_o. Still these approaches typically either do not run in real-time or only capture low frequencies.

Multiple non-screen space approaches have been proposed. Object space shading introduced by Cook et.al. [CCC87] was one of the earliest, which was later improved on by Burns et.al. [BFM10]. Another approach is to shade in texture space. Hillesland [HY16] showed a technique to record access to texels in screen space, but shade in texture space. Such shaded texels can also be used for streaming and view extrapolation [MVD*18]. As discussed before, reuse of shading can be difficult especially in the setups, but some (diffuse) shading can be temporarily reused [MNV*21]. However, this does typically not extend to different viewers if the material is not completely diffuse. A different split approach that shwos how ambient occlusion can be streamed was proposed by Neff et al. [NBD*23]. They transmit points from server to client and then blend the final image by using an efficient hash grid. Sharing computations for participating media was proposed by Stojanovic et al. [SWT*23]. They investigate a combination between world- and view-aligned caching methods for multi-viewer setups.

Weinrauch et al. [WTS*23] showed a proof-of-concept of a cloud-native rendering approach. Various effects were implemented using with what they call surface and world space caches. Additional effect were implemented by [WLT*23]. They [WTS*23] also showed a direct illumination effect for analytic light sources, based on hard shadows. We propose an approach that makes a direct illumination effect for area light sources possible.

2.1. On-Surface Caches

The surface caching system, that is the base for our effect, is an implementation of the On-Surface Cache (OSC) system [WTS*23]. OSC splits meshes into smaller connected components called islands to provide a unique UV-mapping with little distortion. Each island has its own texture space, which simplifies the pipeline and enables efficient triangle lookup textures. OSC also uses a software approach to virtualize the texture spaces by splitting textures into 8 by 8 texel blocks and using a combined single hash map of all virtualized textures. The OSC pipeline is split into multiple stages, as can be seen in figure 2. The first stage (Visibility) looks at which entries are visible and needed, generates non-duplicated requests and ensures that cache blocks are mapped. In the second stage (Cache Update Queueing) requests for shaders for the next stages are created. After that, the third stage (Triangle Lookup Rendering) generates a triangle lookup for cache blocks to create a mapping from

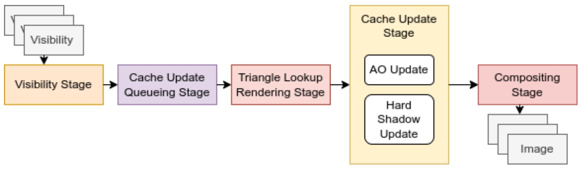


Figure 2: *Basic overview over the OSC system proposed by [WTS*23]. After determining which entries are visible and queuing updates for these entries, all updates, for example for the AO and HardShadows effect, are executed. After that the final images are composited.*

cache to surface space. In the fourth stage (Cache Update) various effects are updated, based on the shader requests made in the second stage. The last stage (Compositing) then combines these effects into an final image. Like other effects implemented in [?], we will implement our effect in the Cache Update and Compositing stages and do not interfere with others.

3. Cone-based Radiance Caching

Shading generated with direct illumination can not be fully reused, but reusing some parts of the information gathered in the DI computation process is possible, as was shown before. Efficient direct illumination information reuse needs to fulfill several key requirements. Firstly, a large amount of data must be abstracted to ensure fast shading performance. Secondly, an accurate abstraction must be chosen to generate high-quality results, which may involve incorporating a directional component in the caching strategy. This is crucial to accurately represent specular highlights and adhere to the principles of the rendering equation. Furthermore, the caching strategy should be memory-friendly to ensure efficient memory usage during real-time rendering.

We propose a cone-based DI information abstraction that fulfills all requirements above. More specifically we use multiple cones (typically three) to describe all incoming radiance at a surface point. For the cone abstraction, we use a direction D, an opening angle α and the radiance gathered, as shown in Figure 3. In addition to the cones we also store a cone count and an accumulation count for radiance noise reduction. Naturally increasing the maximum number of cones results in a finer division of space and therefore can increase the precision of our abstraction.

Our method is split in an information gathering part, which is added to the Cache Update Stage and a shading part, which is added to the Compositing Stage. Figure 4 shows the proposed process, where a single unified update for each surface point is executed to construct the cones. For the image generation process, cones are fetched, sampled, shaded and composited with the color of other effects.

3.1. DI Gathering and Cone Construction

The most important part of our system is the DI information gathering and cone-based representation construction process, that we add to the Cache Update Stage. The cache update is performed on a periodic basis for every visible surface point. When a surface point becomes visible the cone based representation is empty, otherwise the cycle starts by unpacking the stored cone representations. This includes format conversion and bit operations. After that we first determine whether an entry exists on a higher level-of-detail (LOD). If such an entry exists, we reuse this information.

To simplify the main processing we split it in two steps. The first step 3.1.1 only adds radiance information to our cone representation. For light sources that move relative to a surface point (either due to movement of the light source or the object), simply adding radiance to the cache would not suffice. Therefore, the second step 3.1.2 can shrink cones, when there is no longer a light source visible.

3.1.1. Incoming Radiance Updates

The first step adds new radiance to the cone representation. To this end, we randomly sample all scene light sources, including emissive surfaces, analytic area lights and environment maps. The number of samples drawn are chosen according to the desired quality and speed requirements. In practice we mostly use 8 samples. This achieves high frame rates, but due to the temporal accumulation, DI information stays accurate.

For each sample, we determine if there is a direct unobstructed path from the surface point to the light sample location. If ray tracing indicates that this radiance reaches the surface point we add this sample to the cone structures. As surface points can be visible for a long time and light sources may stay relatively constant, adding samples to the cones may not yield improvements anymore. To reduce these unnecessary computations, we reduce the number of traced rays over time if no dynamics are detected.

In Figure 5, we show the four different steps of how a single incoming radiance sample may be added to the cone structures: The left column shows the existing cone structures and the new sample illustrated by an arrow. Cone structures after adding the sample can be seen on the right. (A) shows a sample inside an existing cone which results in the radiance being added to the cone. (B) if the sample is outside existing cones and the maximum number of cones has not been reached a new cone is added. (C) if the maximum number of cones has been reached and the new sample's angle to an existing cone is less than the cones angles to each other, the cone is extended to include the sample's direction and radiance. (D) otherwise the closest existing cones are merged and a new cone is added.

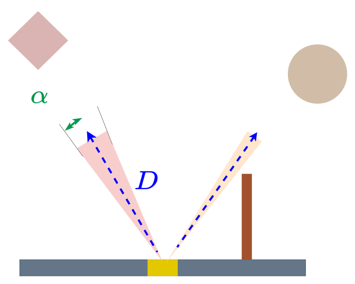


Figure 3: *Cones representing incoming radiance information (direction D, angle α, radiance) on a surface point.*

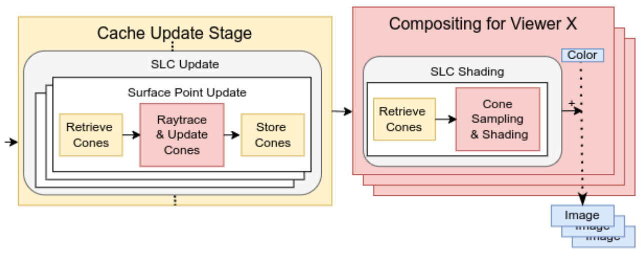


Figure 4: *Abstraction of our added DI effect update and shading generation.*

While the cone structure update describes plausible geometric setups before and after update, correct handling of the accumulated radiance is important. To keep track of the accumulated radiance and allow for adaptive updates over time, we use the accumulation count C: To produce high quality DI information we chose the second option, using the count C:

$$L_{i,new} = L_{i,old} \cdot (1 - 1/C) + L_{i,sampled} * (1/C).$$

Increasing C with the exact samples in every cycle, would only work for static scenes. Relying on a full reset C, would capture dynamics, but of course would lead to information loss. However, in our use case, we can detect changes through changes in the cone structure. Whenever a cone changes its direction it is an indication that the lighting situation that effects the surface point has changed. In this case we can reduce C such that the new information is incorporated faster. We use the angle α, which is the angle between the previous cone direction and its new direction, to help to reduce C. If α and the C surpass a small threshold, we use the following operation to adapt $C = C/min(max(1 + \alpha \cdot 0.05, 1.01), 3.0)$.

3.1.2. Dynamic Scene Updates

The previous steps all extend the cone structures. For dynamic changes, it is important to have a way to further adapt the cones. When for example a light moves, as can be seen in Figure 6, the first step discussed above yields an extension of the cone to enclose the light at the new position. This results in a cone that is too large, because it also includes the lights radiance at the previous positions. Therefore, we need a method to detect where cones are no longer needed.

In order to achieve this, we regularly sample within existing cones to determine if there is a light source still present. More specifically, for each cone we generate a directional sample inside the cone at some angle that is ε away from the opening angle. ε is chosen based on a percentage of the opening angle. If such a sample hits a light source, the sample is discarded, due to the cone being accurate in this direction.

If no light source is hit, small cones (with small opening angles) are simply discarded. Larger cones, are shrunk, as can be seen in Figure 7. First, we determine by how much the cone should be shrunk, $\delta = \alpha - acos(dot(\mathbf{D}, \mathbf{s}))$, where α is the opening angle, $\mathbf{D}$ the cone direction and $\mathbf{s}$ the sample direction. α is reduced by δ

and the cone is rotated away from **s** by $\delta/2$. Finally, we make sure that the adjustment for C only happens once in one update cycle, as cone shrinking often goes hand in hand with previous updates to the cone direction.

3.2. Shading

For every viewer, we sample the caches to compute the surface shading. For every pixel, we rely on the material information from the G-buffers and query the cache data. We still need to compute the integral over ω to evaluate the view-dependent BRDF f_r. An analytic computation is typically too costly, so we simply rely on sampling and iterating over all cones, as outlined in Figure 8

The final color evaluation then comes down to:

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \sum_c^{nC}\sum_s^{nS} f_r(\mathbf{x},\omega_{i,c,s},\omega_o)\frac{L_{i,DI,SLC}(c)}{nS}(\omega_{i,c,s}\cdot\mathbf{n})$$

where nC and nS are the number of cones and samples, $L_{i,DI,SLC}$ is the radiance stored in the surface light cone and $\omega_{i,c,s}$ is the sample direction.

In practice, we use 16 samples for high quality shading. Note that this sampling does not involve ray tracing as the caches already capture visibility. Furthermore, these samples are highly efficient to compute on the GPU, as all information can be fetched once and the computation itself is typically quite efficient. For simple less view-dependent materials, fewer samples (close to the cone center) often suffice according to our experiments. However, we used 16 samples for all our reported tests. The resulting shading color for DI then can be combined with other effects like indirect global illumination, hard shadows or ambient occlusion.

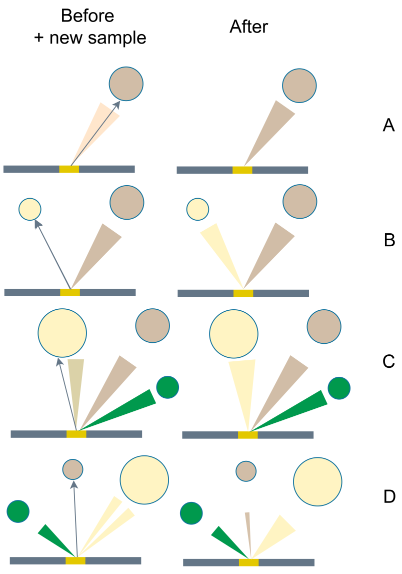


Figure 5: *Different steps in the cone construction process. On the left one can see cone structures with a new sample. The resulting structures with the integrated sample can be seen on the right.*

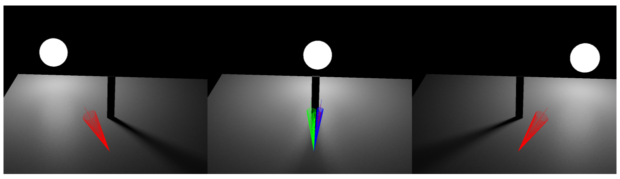


Figure 6: *Example visualization for a moving light source and the cone from the same surface location, which tracks the light source over time. When the light moves partially behind the pillar the radiance is represented by two cones that capture the radiance that passes left and right of the pillar.*

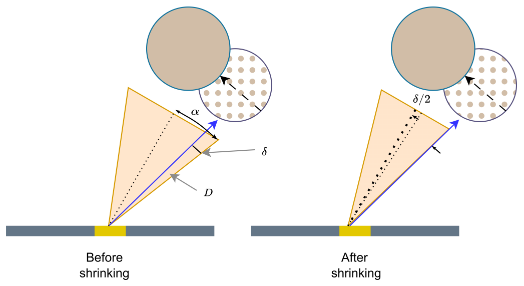


Figure 7: *On the left, one can see a moving light source and a cone with radiance added for the new position, resulting in a cone that still incorporates old radiance directions. The cone is sampled and light presence determined. On the right, one can see the shrinking of the cone to exclude the sample.*

3.3. Cache Compression

We do not need to store cache entries as full floating point numbers. We convert the cone direction from world space to a local on-surface space (considering the surface normal and tangent) and then further convert to spherical coordinates, which we store as 16-bit floats. For the opening angle, we use a 6-bit fixed point representation. We capture radiance in RGB space using 18 bits per channel. To capture C we use 8 bits, and finally 2 bits for the number of active cones. This overall results in 286 bits, which fits into three RGB32 textures with 2 bits to spare.

4. Results

For the evaluating of our proposed caching strategy, we focus on performance and quality. First we focus on single viewer performance, comparing against state-of-the-art rendering solutions. After that we investigate the usefulness of our approach for its indented use case: multi-viewer sharing. Additional evaluation results can be found in the supplementary material, where we carry out a small ablation study, discussing potential alternative cache entry structures, then we look how different cache biases effect performance and finally we show results for different light sizes and movement speeds.

Our implementation runs in NVIDIA Falcor, which we also use for all comparisons. If not otherwise specified, we use three cones as our radiance representation, 8 primary samples for cache update and a cache bias of 1.0. After 8 initial cache updates, we reduce

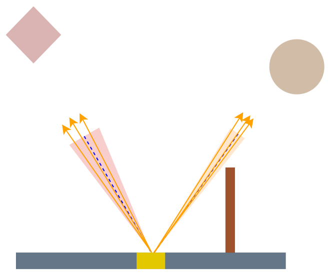


Figure 8: *All cones are sampled to generate samples for shading.*

the update rate to 1 ray per cache entry (unless there is motion detected). For comparison, we use the Falcor ray tracer implementation with tunable sample counts to randomly sample light sources. Due to the lighting setup, 16 samples represent a fast, low quality baseline, and 48 samples can be considered a good quality baseline. For the ground truth we used 2048 samples. For a state-of-the-art comparison, we use ReSTIR [BWP*20] in its latest optimized version as given in the RTXDI framework [NVI]. For timing measurements, we ignore the common steps (G-buffer generation and tone mapping). All evaluations were performed on a RTX 3090 GPU (24 GB) with an AMD Ryzen 7 3700X CPU at 1920×1080.

For our main evaluation, we use four well-known scenes, which we modified to offer challenging lighting setups, as shown in Figures 9a, 9c, 9b and 9d. Viking Village [Tec] is an outdoor scene, where we placed a large moon-like light source in the sky and used emissive torches to light the scene. Intel Sponza [Int] is an updated version of the Sponza test scene, which we extended with four emissive lights, three smaller round lights placed in the lower hallways and one large light in the top center of the scene, which we will refer to as Sponza. San Miguel [Lla] forms a test case with high geometric complexity, where multiple emissive lamps are present in the scene. The large amount of discontinuous foliage geometry makes it a difficult usecase for on-surface caches. Additionally, an emissive environment map [HDR] is present in the scene. Finally, we modified Bistro Exterior [Lum17] to form a challenging dynamic use case, where we placed 2 spherical emissive lights and a rotating emissive cube that move through the scene. Additionally, we placed rotating wheels, creating a high-frequency movement. For all tests, we recorded multiple uncorrelated camera paths of multiple seconds through the scene and our results are the averages over all frames.

4.1. Single Viewer

Our single viewer experiments are summarized in Table 1. While RT 16 was designed as our fast, low-quality base line, its quality is in most cases comparable to RTXDI.

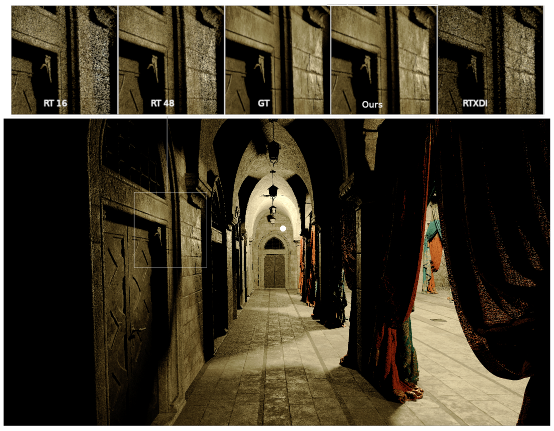
We attribute the unexpectedly low quality of RTXDI to the fact that larger area light sources require multiple distributed samples, while RTXDI focuses on reusing samples spatially and temporally and thus works best for many point light sources rather than few area light sources. Thus, both methods form the quality floor in our tests. In all tests, our approach achieves the highest SSIM value, and is in some cases only marginally beaten in Flip and PSNR by RT 48 which has a significantly higher runtime. In scenes with low geometric complexity (Viking Village) ray tracing itself is relatively efficient and thus RT 16 is quite fast. In more complex scenes, RTXDI is faster as it can reduce the number of traced rays and the constant overhead of reservoir resampling stays constant. For scenes with little motion, our approach can operate with few rays, leading to the fastest performance in Viking Village and very similar runtime to RXTDI in Sponza and San Miguel. If all light sources exhibit significant motion (Bistro Exterior), we increase the number of traced rays for higher quality. Thus, even in a single viewer setup—for which our method has not been designed for—our approach is among the fastest approaches and in most cases achieves the highest quality. Note that due to our design, our renderings are relatively noise free, but obviously biased. Thus, we do not require an additional per-viewer denoising step.

4.1.1. Multiple Viewers

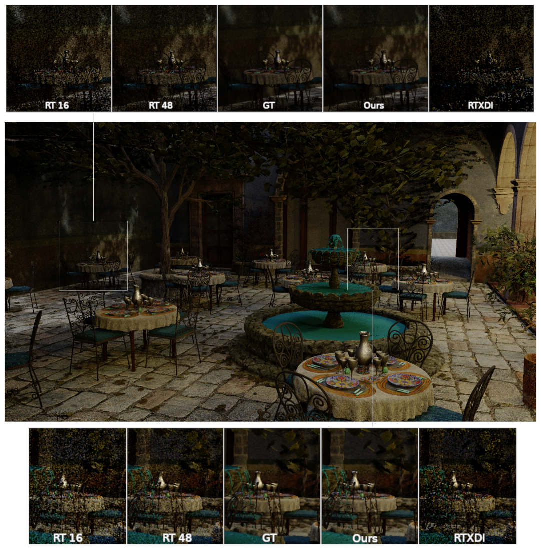
If there are multiple viewers in a scene, there is a high likelihood that there is some overlap among them, especially for constraint scenarios like virtual meetings or gathering locations in games. Due to the design of our approach, areas visible from multiple view points only need their caches to be generated once. An example rendering for Sponza for 16 viewers is shown in Figure 10.

To evaluate the possible scaling of our approach, we start with a controlled experiment, where we force specific amounts of overlap. We position cameras all at the same location as they move along one recorded camera trajectory, but rotate them so that there is a specific controlled overlap in their views. The result of this experiment can be seen in Figure 11. For 100 percent overlap, caches are only generated once, and only the per-viewer shading cost, i.e., going from L_i to L_o, increases the timing when adding another viewer to the scene. With lower overlap, more cache entries need to be generated, and thus less time is being saved. The actual achievable overlap highly varies between scenarios—an ideal scenario is formed by virtual meetings where all participants focus on the person currently talking, thus an overlap of 60%-80% can easily be achieved. Note that even for low overlaps our timing is far from doubling when doubling the viewer count, which shows that there is a constant overhead to multiple steps in our approach: Cache allocation and cache update queuing is only happening once for all viewers and hardly shows additional performance costs. Furthermore, larger cache update requests may only show little additional ray tracing costs, as rays are highly correlated (compared to more scattered tracing if surfaces are only partially visible).

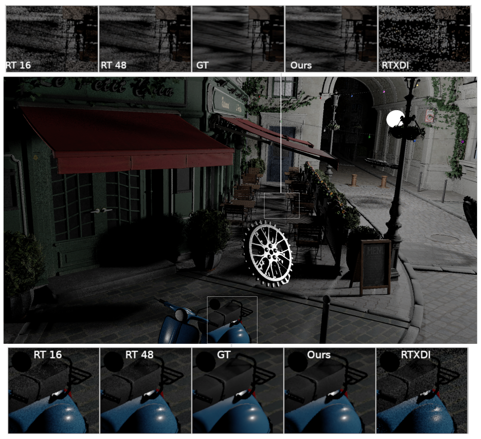
To test a more random scenario, we use camera trajectories that were recorded from different independent movements, see Figure 12. In this 16 viewers setup, all viewers start from different locations, looking somewhat towards the middle of the scene. At scene start every viewer looks to the center of the scene creating an overlap of about 50 percent. Afterwards they randomly move, leading to an average overlap of about 20%. Even relying on such random movement, our approach scales significantly better than single-viewer methods, as shown in Figure 13. Even for cases where our approach is slower for the single viewer setup, we quickly outperform the competing methods as additional viewers are added to the scene. Note that we still use the same quality settings as in the



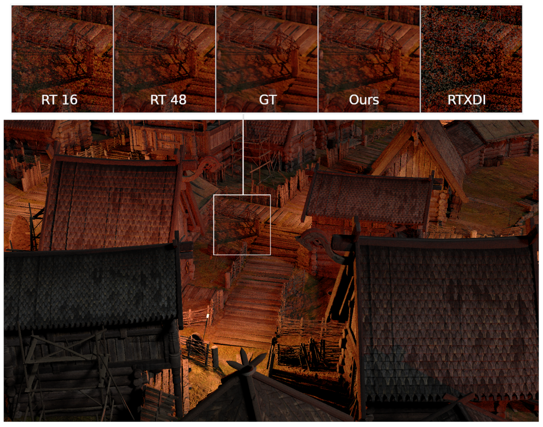
(a) *Example view from our modified Sponza scene with with lamps placed in the corridors. Although our approach is also based on stochastic sampling, restricting the representation to cones, leads to capturing highly plausible specular highlights and view-dependent details with little noise. Competing approaches exhibit less bias but significantly lower quality.*



(b) *Our modified San Miguel scene features high amounts of geometric detail under a difficult light setup. Our approach clearly outperform the competing approaches in quality.*



(c) *Our extended Bistro Exterior scene features fast moving light sources and dynamic objects, making it the most difficult test scene for our approach. Especially in areas with thin shadow features our approach significantly outperforms the competing approaches.*



(d) *Our modified Viking Village scene. In addition to the existing torch mesh lights we added a large moon-like light to the scene to form a challenging interesting use case.*

Figure 9: *Our test scenes. Sponza (a), San Miguel (b), Bistro Exterior (c), Viking Villange (d)*

	Viking Village				Intel Sponza				Bistro Exterior				San Miguel			
	t [ms]	Flip↓	PSNR↑	SSIM↑	t [ms]	Flip↓	PSNR↑	SSIM↑	t [ms]	Flip↓	PSNR↑	SSIM↑	t [ms]	Flip↓	PSNR↑	SSIM↑
RT 16	12.65	0.048	28.84	0.839	8.97	.047	25.08	.827	**10.07**	.039	29.54	.863	21.50	.076	23.53	.645
RT 48	36.49	**0.032**	33.38	0.933	25.84	**.032**	29.06	.898	28.72	**.026**	**34.42**	.942	62.31	.052	27.39	.788
RTXDI	13.73	0.072	24.31	0.656	**5.26**	.072	24.61	.806	10.60	.062	24.29	.723	**11.66**	.090	24.15	.628
Ours	**8.82**	0.037	**34.66**	**0.950**	7.09	.033	**31.72**	**.936**	19.65	.031	32.45	**.956**	12.60	**0.037**	**33.01**	**.933**

Table 1: *Single viewer results for our four main test scenes. Even though our approach is designed for multi-viewer sharing, it also is advantageous for single viewers, as it can progressively update its representation for L_i for each surface point. Bistro Exterior forms the most difficult test case as all light sources are moving fast, and we require additional cone updates. In all other test cases, our approach is among the fastest while achieving the best overall quality.*

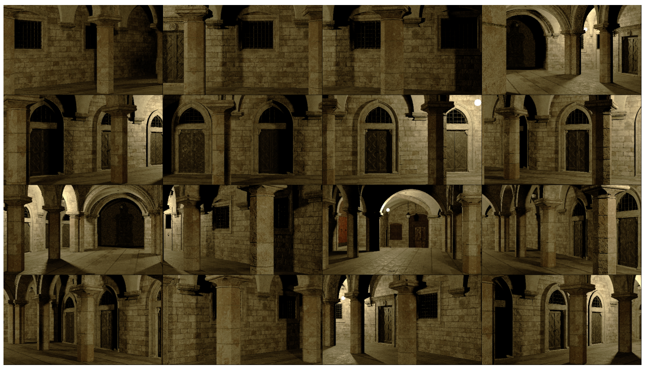


Figure 10: *Example multi-view rendering from our modified Sponza scene with 16 randomly moving viewers. We see an average overlap of* 20%.

single viewer setup and thus achieve significantly higher quality than RT 48 and RTXDI while outperforming them with a factor of $2-3\times$ for 16 viewers—even in the random overlap case featuring only 20% overlap.

In Figure 14 we show how different number of viewers impact used memory (excluding memory used by all approaches, like scene data, G-Buffers and output buffers). For our approach, the allocated cache blocks dominate the memory requirements (ours active), while additional data structures for triangle id rendering and gathering update requests contribute little. Both RT 16 and RT 48 do not require additional memory as their output can directly be accumulated to the output buffers. Finally, RTXDI requires reservoir buffers of the screen resolution for spatio- and temporal reservoir resampling. Even though these buffer do not require a lot of memory for a single viewer, duplicating screen-sized buffers for each viewer can quickly lead to significant amounts of memory. Our approach on the other hand scales with the overlap. Furthermore, we report actual memory requirements in term of allocated blocks, which have a size of 8×8 texels. A more fine granular memory management could significantly lower our memory requirements—typically our blocks are about 50% filled only.

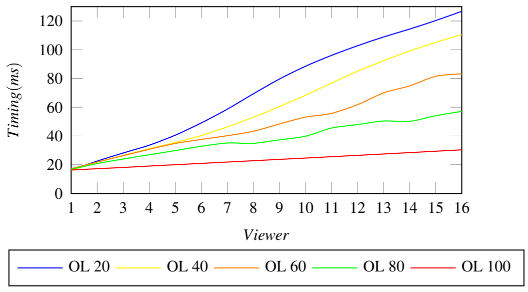


Figure 11: *Multi-viewer overlap comparison for the Bistro Exterior scene with rendering time in ms. Even for low overlap percentages of* 20% *we achieve very good scaling, as we can also amortize constant overhead for cache allocation and update queuing.*

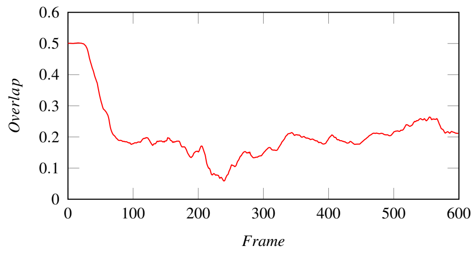


Figure 12: *To demonstrate the achievable overlap in difficult rendering setups, we use uncorrelated camera paths through the scene. In this example for Sponza 16 viewer are randomly positioned and start by looking into the center of the scene, before moving through the scene. On average, we still achieve 20% overlap in this setup. Other scenarios, like virtual meetings or gathering situations in games, can easily achieve 60-80% of average overlap.*

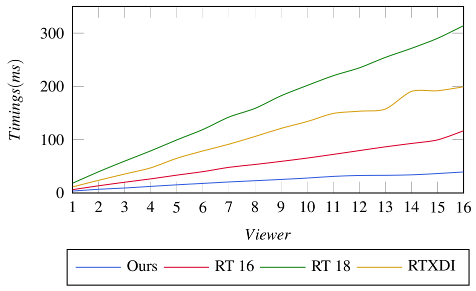


Figure 13: *Realistic scaling for 1 to 16 viewers in Sponza demonstrating our superior scaling in comparison to screen-space approaches. Note that our approach also achieves superior quality.*

4.2. Stereo Rendering

An ideal use case for our technique is stereo rendering. The overlap in this kind of setup is very high due to similar camera placement and orientation. In Figure 15 we show 16 stereo views, resulting in 32 rendered images, with timings in Figure 16. Additionally, we report timings for 8 stereo viewers. The other approaches behave roughly the same for 16 single and 8 stereo views (there may be a small gain due to higher coherence between rays in the stereo case as well). However, our approach reduces run time to roughly 74% due to the higher overlap. Similarly, going from 16 single viewers to 16 stereo viewers only increased runtime by $1.36\times$ in our case. RTXDI was not able to render 16 stereo views in this test as it ran out of memory.

4.3. Limitations

As mentioned above we always used three cones in our experiments. That is possible in our scenes as there are typically no more than three dominant light sources at any one location in the scene. However, if the number of light sources is significantly higher, our image quality reduces, as can be seen in figure 17. As our process of determining incoming radiance follows a random sampling our image quality degradation shows up as noise, similar to other random sampling approaches like RTXDI. In any case, it is advantageous

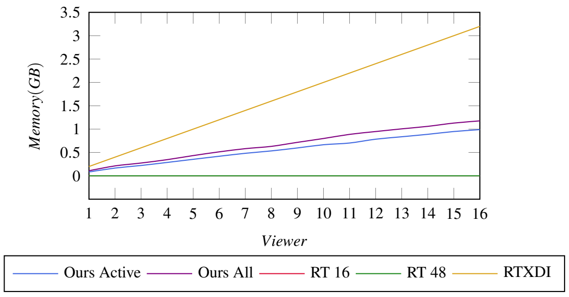


Figure 14: *Memory consumption of different techniques for Sponza 1-16 viewers on uncorrelated paths. Memory consumption increases steadily, while our method only consumes additional memory for newly visible regions. Ray traced methods do not use additional memory.*

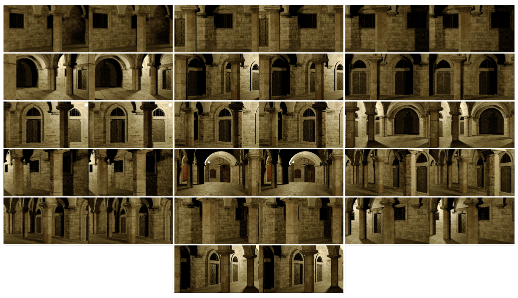


Figure 15: *Example views for Sponza with 16 stereo viewers.*

to configure the number of cones for a specific scene. If the number of cones significantly changes throughout one scene, using a high number of cones everywhere increases the memory requirements to the worst case.

While our approach also works well outside of the multi viewer scenario and even outperforms highly tuned algorithms like RTXDI in our single viewer tests, it is not universally applicable. In case a scene only consists of point lights, a cone approximation is less optimal, as we will simply merge close-by points. As such, relying on a simple visibility/occlusion caching approach may work significantly better. Similarly, if smooth DI, like from an environment map, should be represented, a equally smooth cache representation like e.g. spherical harmonics would likely work better. An example of a environment map setup that is difficult for our method can be seen in figure 18.

Finally, while our approach also works for fast moving light sources, our runtime nearly doubles for such light sources. While our cone cache is still competitive in this case (even in the single viewer scenario), our advantage is clearly diminished. However, large, fast-moving area light sources are in general more of an exception in widely used virtual scenes.

5. Conclusion and Future Work

We proposed a DI cache that abstracts incoming radiance on surface points with cone structures. These cache structures are view independent and as such can be reused by different viewers in the same scene. Our results show that our caches are even advantageous in a single viewer setup, leading to high quality noise-free images, especially if the light sources exhibit only little movement. As additional viewers are added to a scene, our approach shows superior scaling behavior, even under low view overlaps of around 20%. Our performance increases are not only due to reusing the cached data among viewers, but also the overhead of our cache

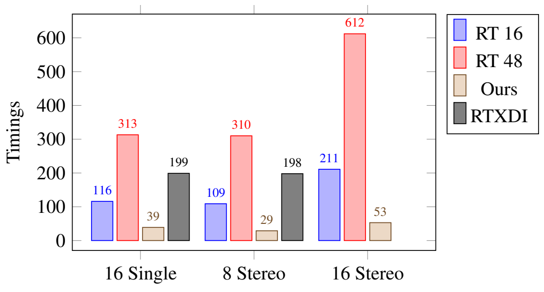


Figure 16: *Timing in ms for Sponza for stereo rendering compared to single viewer rendering. Stereo rendering leads to large overlaps between views, resulting in significant performance gains compared to equal view count non-stereo rendering.*

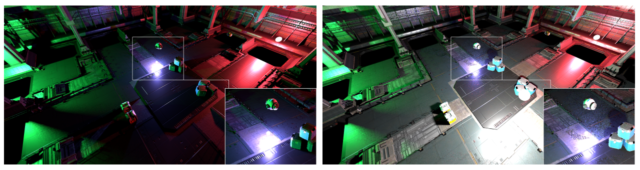


Figure 17: *Hangar scene(Composition of modified models [ZLM] [jQu] [Mat]) with multiple moving lights. One can see that when more dominant lights are added (right) noise starts appearing in certain areas.*

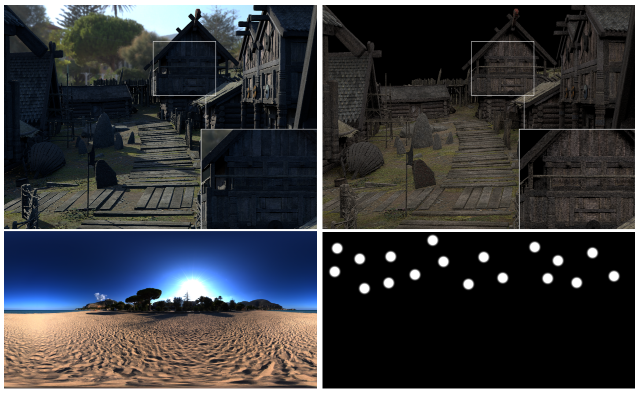


Figure 18: *We show two environment map setups for viking village. The left picture shows little noise for a usual environment map [And]. We also constructed an environment map(lower right) to show where our method develops noise, as can be seen in the top right image.*

management amortizes across the viewers. Obviously, in scenarios with even more overlap between views, like stereo rendering, we show even more significant performance gains.

Our current approach uses a single data structure to capture all incoming direct light. However, depending on the light source, e.g., point and directional lights, a cone structure is an overkill. Furthermore, smooth lighting, like from a smooth environment map or even global illumination may be better captured with spherical harmonics. Thus, in the future, we believe adapting the chosen cache structure dynamically for the actual experienced light situation is key.

Furthermore, we currently focus on capturing and reusing lighting information in a caching structure. For this technique to truly scale to the cloud, a fully distributed real-time caching and rendering system is needed. Depending on the use case, this may simply be a multi-GPU server or actually a fully distributed cloud service. This comes with various additional challenges, like distributed state sharing, integration of distributed rendering into an actual engine, and considering latencies. In any case, we believe that sharing rendering computation may be used to save computations and energy and truely support scaling rendering.

References

[AKDS04] Annen T., Kautz J., Durand F., Seidel H.-P.: Spherical harmonic gradients for mid-range illumination. In *Rendering Techniques* (2004), pp. 331–336. 2

[And] Andreas Mischok: Spiaggia di mondello. https://polyhaven.com/a/spiaggia_di_mondello. Accessed: 2023-05-17. 9

[BFM10] Burns C. A., Fatahalian K., Mark W. R.: A lazy object-space shading architecture with decoupled sampling. In *Proceedings of the Conference on High Performance Graphics* (2010), pp. 19–28. 3

[BWP*20] Bitterli B., Wyman C., Pharr M., Shirley P., Lefohn A., Jarosz W.: Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 39*, 4 (July 2020). doi:10/gg8xc7. 2, 6

[CCC87] Cook R. L., Carpenter L., Catmull E.: The reyes image rendering architecture. *ACM SIGGRAPH Computer Graphics 21*, 4 (1987), 95–102. 3

[CMS87] Cabral B., Max N., Springmeyer R.: Bidirectional reflection functions from surface bump maps. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), pp. 273–281. 2

[DGS12] Davidovič T., Georgiev I., Slusallek P.: Progressive lightcuts for gpu. In *ACM SIGGRAPH 2012 Talks*. 2012, pp. 1–1. 2

[DJM] DJMaesen: Street. https://skfb.ly/6wtQu. Accessed: 2023-05-26. 1

[DKH*14] Dachsbacher C., Křivánek J., Hašan M., Arbree A., Walter B., Novák J.: Scalable realistic rendering with many-light methods. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 88–104. 2

[GSHG98] Greger G., Shirley P., Hubbard P. M., Greenberg D. P.: The irradiance volume. *IEEE Computer Graphics and Applications 18*, 2 (1998), 32–43. 3

[HCZ21] Hadadan S., Chen S., Zwicker M.: Neural radiosity. *ACM Transactions on Graphics (TOG) 40*, 6 (2021), 1–11. 3

[HDHN16] Heitz E., Dupuy J., Hill S., Neubelt D.: Real-time polygonal-light shading with linearly transformed cosines. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–8. 2

[HDR] HDRMAPS: Mountain view environment map. https://hdrmaps.com/mountain-view/. Accessed: 2023-03-16. 6

[HY16] Hillesland K. E., Yang J.: Texel shading. In *Eurographics (Short Papers)* (2016), pp. 73–76. 3

[Int] Intel: Intel's sponza. https://www.intel.com/content/www/us/en/developer/topic-technology/graphics-research/samples.html. Accessed: 2023-03-16. 6

[JDZJ08] Jarosz W., Donner C., Zwicker M., Jensen H. W.: Radiance caching for participating media. *ACM Transactions on Graphics (TOG) 27*, 1 (2008), 1–11. 3

[jQu] jQueary: Game ready sci-fi crate pile. https://skfb.ly/6WNtx. Accessed: 2023-05-17. 9

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), pp. 143–150. 2

[KBPZ06] Křivánek J., Bouatouch K., Pattanaik S. N., Zara J.: Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. *Rendering Techniques 2006* (2006), 127–138. 3

[Kel97] Keller A.: Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 49–56. 2

[KGPB05] Křivánek J., Gautron P., Pattanaik S., Bouatouch K.: Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005), 550–561. 3

[KVS*14] Khlebnikov R., Voglreiter P., Steinberger M., Kainz B., Schmalstieg D.: Parallel irradiance caching for interactive monte-carlo direct volume rendering. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 61–70. 3

[Leh07] Lehtinen J.: A framework for precomputed and captured light transport. *ACM Transactions on Graphics (TOG) 26*, 4 (2007), 13–es. 2

[LKB*22] Lin D., Kettunen M., Bitterli B., Pantaleoni J., Yuksel C., Wyman C.: Generalized resampled importance sampling: foundations of restir. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–23. 2

[Lla] Llaguno G.: San miguel. https://casual-effects.com/data/. Accessed: 2023-03-16. 6

[LSK*07] Laine S., Saransaari H., Kontkanen J., Lehtinen J., Aila T.: Incremental instant radiosity for real-time indirect illumination. In *Rendering Techniques* (2007), pp. 277–286. 2

[Lum17] Lumberyard A.: Amazon lumberyard bistro, open research content archive (orca), July 2017. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 6

[Mat] Matthew-David: Sci-fi recon drone. https://skfb.ly/6VqqN. Accessed: 2023-05-17. 9

[MESK22] Müller T., Evans A., Schied C., Keller A.: Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG) 41*, 4 (2022), 1–15. 3

[MGNM19] Majercik Z., Guertin J.-P., Nowrouzezahrai D., McGuire M.: Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques 8*, 2 (2019). 3

[MNV*21] Mueller J. H., Neff T., Voglreiter P., Steinberger M., Schmalstieg D.: Temporally adaptive shading reuse for real-time rendering and virtual reality. *ACM Transactions on Graphics (TOG) 40*, 2 (2021), 1–14. 3

[MRNK21] Müller T., Rousselle F., Novák J., Keller A.: Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372* (2021). 3

[MVD*18] Mueller J. H., Voglreiter P., Dokter M., Neff T., Makar M., Steinberger M., Schmalstieg D.: Shading atlas streaming. *ACM Transactions on Graphics (TOG) 37*, 6 (2018), 1–16. 3

[NBD*23] NEFF T., BUDGE B., DONG Z., SCHMALSTIEG D., STEINBERGER M.: PSAO: Point-based split rendering for ambient occlusion. *Computer Graphics Forum 42*, 8 (2023). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14864, doi:10.1111/cgf.14864. 3

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. In *ACM SIGGRAPH 2004 Papers*. 2004, pp. 477–487. 2

[NVI] NVIDIA: Rtxdi framework. https://github.com/NVIDIAGameWorks/RTXDI. Accessed: 2023-03-16. 6

[OLK*21] OUYANG Y., LIU S., KETTUNEN M., PHARR M., PANTALEONI J.: Restir gi: Path resampling for real-time path tracing. In *Computer Graphics Forum* (2021), vol. 40, Wiley Online Library, pp. 17–29. 2

[R*09] RAMAMOORTHI R., ET AL.: Precomputation-based rendering. *Foundations and Trends® in Computer Graphics and Vision 3*, 4 (2009), 281–369. 2

[RBRD22] RAINER G., BOUSSEAU A., RITSCHEL T., DRETTAKIS G.: Neural precomputed radiance transfer. In *Computer Graphics Forum* (2022), vol. 41, Wiley Online Library, pp. 365–378. 2

[RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., SLOAN P.-P., BAO H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *Acm siggraph 2006 papers*. 2006, pp. 977–986. 2

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 527–536. 2

[SWT*23] STOJANOVIC R., WEINRAUCH A., TATZGERN W., KURZ A., STEINBERGER M.: Efficient rendering of participating media for multiple viewpoints. *Computer Graphics Forum 42*, 8 (2023). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14874, doi:10.1111/cgf.14874. 3

[Tec] TECHNOLOGIES U.: Viking village. https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140. Accessed: 2023-03-16. 6

[VKŠ*14] VORBA J., KARLÍK O., ŠIK M., RITSCHEL T., KŘIVÁNEK J.: On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 1–11. 2

[WCZR20] WU L., CAI G., ZHAO S., RAMAMOORTHI R.: Analytic spherical harmonic gradients for real-time rendering with many polygonal area lights. *ACM Transactions on Graphics (TOG) 39*, 4 (2020), 134–1. 2

[WH92] WARD G. J., HECKBERT P. S.: *Irradiance gradients*. Tech. rep., Lawrence Berkeley Lab., CA (United States); Ecole Polytechnique Federale ..., 1992. 3

[WLT*23] WEINRAUCH A., LORBEK S., TATZGERN W., STADLBAUER P., STEINBERGER M.: Clouds in the cloud: Efficient cloud-based rendering of real-time volumetric clouds. *Computer Graphics Forum 42*, 8 (2023). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14876, doi:10.1111/cgf.14876. 3

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), pp. 85–92. 3

[WTS*23] WEINRAUCH A., TATZGERN W., STADLBAUER P., CRICKX A., HLADKY J., COOMANS A., WINTER M., MUELLER J. H., STEINBERGER M.: Effect-based multi-viewer caching for cloud-native rendering. *ACM Trans. Graph. 42*, 4 (jan 2023). 3

[Yas] YASUTOSHI MORI: Sports car. https://casual-effects.com/g3d/data10/research/model/sportsCar/sportsCar.zip. Accessed: 2023-05-24. 1

[Zam] ZAMBUR: Buildings front. https://skfb.ly/owOJ8. Accessed: 2023-05-26. 1

[ZHL*05] ZHOU K., HU Y., LIN S., GUO B., SHUM H.-Y.: Precomputed shadow fields for dynamic scenes. In *ACM SIGGRAPH 2005 Papers*. 2005, pp. 1196–1201. 2

[ZLM] ZLM CRAFTER: Bot hangar. https://skfb.ly/6AqMY. Accessed: 2023-05-17. 9