Real-time Neural Radiance Caching for Path Tracing

THOMAS MÜLLER, NVIDIA
FABRICE ROUSSELLE, NVIDIA
JAN NOVÁK, NVIDIA
ALEXANDER KELLER, NVIDIA

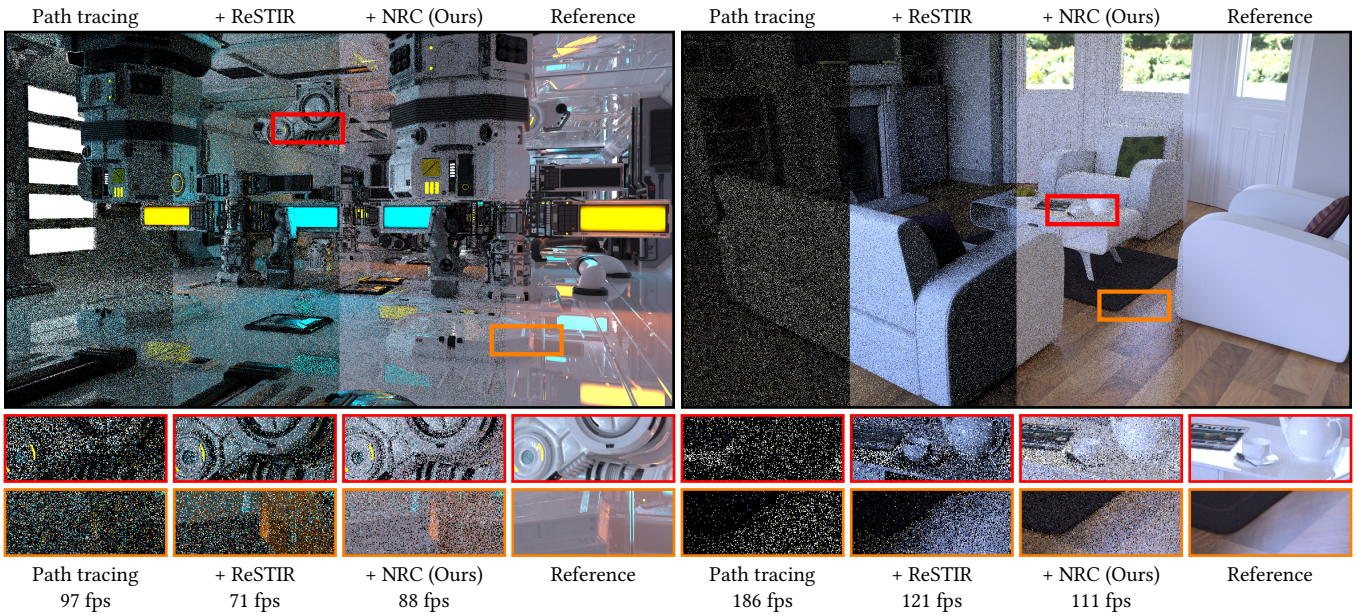


Fig. 1. A path-traced frame from the Zero Day animation (left) and the Living Room scene (right), rendered with 1 sample per pixel each. Direct-illumination sampling such as ReSTIR [Bitterli et al. 2020] reduces noise at the first path vertex, but does not address noise from indirect illumination. We propose to reduce the remaining noise by (online) training a neural network to approximate the radiance field—a new take on classical radiance caching. Terminating the paths into the neural cache not only shortens paths—leading to an overall cost *reduction* in the Zero Day scene—but also removes most of the remaining noise while introducing little bias. The images were rendered at a resolution of 1920 × 1080 on a high-end desktop machine (i9 9900k and RTX 3090). Zero Day ©beeple

We present a real-time neural radiance caching method for path-traced global illumination. Our system is designed to handle fully dynamic scenes, and makes no assumptions about the lighting, geometry, and materials. The data-driven nature of our approach sidesteps many difficulties of caching algorithms, such as locating, interpolating, and updating cache points. Since pretraining neural networks to handle novel, dynamic scenes is a formidable generalization challenge, we do away with pretraining and instead achieve *generalization via adaptation*, i.e. we opt for training the radiance cache while rendering. We employ self-training to provide low-noise training targets and simulate infinite-bounce transport by merely iterating few-bounce training updates. The updates and cache queries incur a mild overhead—about 2.6ms on full HD resolution—thanks to a streaming implementation of the neural network that fully exploits modern hardware. We demonstrate significant noise reduction at the cost of little induced bias, and report state-of-the-art, real-time performance on a number of challenging scenarios.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Ray tracing**; *Supervised learning by regression*; *Reinforcement learning*.

Additional Key Words and Phrases: real-time, rendering, deep learning, neural networks, path tracing, radiance caching

ACM Reference Format:
Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4, Article 36 (August 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

Authors' addresses: Thomas Müller, NVIDIA, tmueller@nvidia.com; Fabrice Rousselle, NVIDIA, frousselle@nvidia.com; Jan Novák, NVIDIA, jnovak@nvidia.com; Alexander Keller, NVIDIA, akeller@nvidia.com.

1 INTRODUCTION

Path-traced global illumination is a long standing challenge in real-time rendering [Keller et al. 2019]. The problem remains onerous even in offline rendering, especially when considering high-order indirect illumination. Fortunately, radiative quantities feature significant spatial, directional, and temporal correlations, which can be exploited in various ways to accelerate rendering.

One particularly appealing approach is to cache radiance samples for later reuse. This can be done in a precomputation step [Seyb et al. 2020], or while rendering [Majercik et al. 2019]. Such systems, however, can be difficult to harness, as they often rely on human intervention or involved heuristics to minimize rendering artifacts [Hooker 2016]. We propose to alleviate these difficulties through the use of a *neural* radiance cache, as neural networks are known to be particularly apt at replacing complex heuristics. Our system is designed according to the following principles:

- **Dynamic content.** To handle fully interactive content, the system must support arbitrary dynamics of the camera, lighting, geometry, and materials. We strive for a solution that *does not require* precomputation.
- **Robustness.** Case-specific handling eventually leads to complex, brittle systems. Hence, the cache should be agnostic of materials and scene geometry. This is particularly important for user-generated content, where design assumptions cannot be easily enforced. Yet, additional attributes may be provided to the system to improve its rendering quality.
- **Predictable performance and resource consumption.** Fluctuations in work load and memory usage lead to unstable framerates. We seek a solution with stable runtime overhead and memory footprint, both of which should be independent of scene complexity. The rendering cost must scale at worst linearly with the number of pixels.

The first two principles—dynamic content and robustness—present a major challenge for *pre-trained* networks: the trained model must generalize to novel configurations and, worse, content possibly never observed before. This form of generalization has not been demonstrated with previous approaches in the context of learning radiative quantities [Hermosilla et al. 2019; Mildenhall et al. 2020; Ren et al. 2013], and it is unclear if it can ever be achieved.

Instead, we build on the simple but powerful realization that the generalization challenge can be completely sidestepped by fast adaptation. We rely solely on optimizing the model *online*, during rendering. Online learning of neural networks has so-far only been used in offline and interactive rendering [Lehtinen et al. 2018; Müller et al. 2019; Müller et al. 2020]. Fitting the optimization and inference inside the tight rendering loop of real-time applications is a non-trivial task that remains to be tackled.

We present two key contributions that enable *generalization via adaptation* in real time. First, we describe an efficient mechanism for optimizing the network using (relatively) inexpensive radiance estimates. The core of this mechanism is self-training of the neural network from its own prediction at a later vertex of the path, providing multi-bounce illumination at the cost of tracing single rays or very short paths.

Second, we propose a streamlined network architecture designed to maximize the quality-cost tradeoff when rendering fully dynamic scenes. This architecture is key to our system, as its simplicity not only leads to extremely fast convergence but also enables extensive optimizations. To fully exploit these opportunities, we propose a fully fused implementation tailored to modern GPUs; the underlying principles however are general and could be adapted to a range of platforms. We also show how recently proposed input encodings [Mildenhall et al. 2020; Müller et al. 2019] can be combined to greatly enhance fidelity even with our severely constrained budget.

As shown in Figure 1, our system achieves real-time framerates on current hardware and handles a wide range of material and lighting configurations. Our accompanying video demonstrates the temporal adaptation to dynamic geometry and lighting. Lastly, we report preliminary results with an off-the-shelf denoiser demonstrating significantly improved temporal coherence when using our cache.

2 RELATED WORK

Reviewing techniques for accelerating global illumination by radiance caching, we focus on precomputation-based techniques, fully dynamic algorithms, and approaches based on artificial neural networks that are most related to our work. For an extensive survey we refer to Ritschel et al. [2012].

Radiance caching. Most real-time global illumination techniques can be traced back to the seminal work of Ward et al. [1988] on irradiance caching. Modern techniques for modeling diffuse interreflections follow the same assumption that irradiance tends to vary smoothly across the scene, and texture detail can be recovered using albedo modulation. Later, Greger et al. [1998] introduced the irradiance probe volume, which became ubiquitous in modern game engines. The interpolation and location of the various cache records is a key challenge in these techniques, especially when the aforementioned assumptions on smoothness do not hold. While robust, principled solutions exist [Jarosz et al. 2008; Křivánek and Gautron 2009], real-time applications often have to resort to clever heuristics and impose restrictions on scene design to fit their harsh constraints. In order to handle glossy surfaces, which invalidate the Lambertian assumption at the core of irradiance caching algorithms, Křivánek et al. [2005] proposed the use of a radiance cache, representing the directional domain with spherical harmonics. A wealth of recent works further explored the use of radiance caching in offline [Dubouchet et al. 2017; Marco et al. 2018; Zhao et al. 2019] and real-time rendering, where advances in real-time rendering were enabled for example by compression [Vardis et al. 2014], sparse interpolation [Silvennoinen and Lehtinen 2017], pre-convolved environment maps [Rehfeld et al. 2014; Scherzer et al. 2012], and spatial hashing [Binder et al. 2018; Pantaleoni 2020]. In contrast, our own work achieves robustness through online deep learning.

Precomputation-based techniques. The high computational cost of simulating global illumination spurred the development of precomputation techniques [Arvo 1986; Heckbert 1990], which have been further developed to address the stringent constraints of real-time applications. Assuming both the scene lighting and geometry are fixed, irradiance can be computed and then stored in texture

space using lightmaps [Abrash 1997] and in world space using light probes [Oat 2005]. Offering both approaches in one system, Martin and Einarsson [2010] introduced iterated dynamic lighting updates. These techniques are widespread in modern game engines [Barré-Brisebois 2017]. Light probes can be combined with precomputed radiance transfer [Sloan et al. 2002] or visibility [Iwanicki and Sloan 2017; McGuire et al. 2017], to account for (self) occlusion when shading scene objects. While precomputation-based solutions offer a number of indisputable advantages [Seyb et al. 2020], we embrace online caching mechanisms that facilitate dynamically changing scenes without assuming (parts of) the scene to be static or known in advance.

Fully dynamic techniques. Dynamic real-time global illumination methods build upon efficient rendering algorithms that reuse shading and visibility computation across pixels, such as photon mapping [Jensen 1996], many-light rendering [Keller 1997] and radiosity maps [Tabellion and Lamorlette 2004], extracting further efficiency through various approximations. Some approximate the scene geometry using a (hierarchical) point cloud, which is then efficiently rasterized into shadow maps [Ritschel et al. 2008] or micro-rendering buffers [Ritschel et al. 2009a]. Volumetric approximations of the scene lighting and geometry [Crassin et al. 2011; Kaplanyan and Dachsbacher 2010], bootstrapped with large numbers of virtual point lights from reflective shadow maps [Dachsbacher and Stamminger 2005], allowed to scale to larger scenes. These approaches are sometimes combined with very efficient screen-space approximations of ambient occlusion [Mittring 2007], directional occlusion [Ritschel et al. 2009b], or reflections [Sousa et al. 2011]. Recently, ray-tracing hardware has been used to compute specific components of light transport online, such as diffuse interreflections [Majercik et al. 2019] or glossy reflections [Deligiannis and Schmid 2019]. Aside of accuracy limitations inherent to the approximations, such as blurring, missing interactions, or assumptions about the material model, a key limitation of many of these techniques is the reliance on a dual representation of the scene which must be continuously refreshed. Our neural radiance cache sidesteps the need for an approximate scene representation by operating on a set of sampled path contributions, which effectively decouples the algorithm from the scene lighting and geometric complexity.

Path guiding. The family of path guiding techniques, originating from Lafortune and Willems [1995] and Jensen [1995], is closely related to that of radiance caching in that they often learn an approximation of incident radiance that is amenable to importance sampling. Recent incident-radiance models tend to be either parametric mixtures [Vorba et al. 2014] or probability trees [Müller et al. 2017], but (neural) normalizing flows are also possible [Müller et al. 2019]. While these techniques are highly successful in offline rendering of mostly static scenes [Vorba et al. 2019], adapting them to the constraints of animated real-time rendering is non-trivial ongoing work [Dittebrandt et al. 2020]. Methods that use an explicit model of the BRDF [Herholz et al. 2018, 2016] or the product integrand [Müller et al. 2019; Müller et al. 2020] are likely the most promising to repurpose as radiance caches, as they yield a more accurate scattered-radiance estimate than methods approximating the incident radiance at the previous camera-path vertex.

Neural techniques. Neural networks are capable of approximating various visual phenomena remarkably well, whether they operate in screen space [Nalbach et al. 2017] or in world space, whether they are pre-trained over multiple scenes [Hermosilla et al. 2019; Jiang and Kainz 2021; Kallweit et al. 2017; Nalbach et al. 2017] or fit to single scene [Keller and Dahm 2019; Mildenhall et al. 2020; Müller et al. 2019; Müller et al. 2020; Ren et al. 2013]. The latter approaches are most closely related to ours. Ren et al. [2013] propose to train a set of local neural radiance caches, conditioned on the position of a single point light source. While lighting can be changed dynamically and area lighting can be approximated using a set of point lights at the cost of multiple cache queries, geometry and materials have to remain static as a consequence of the cost of the training procedure. Our technique differs on two important aspects: (i) we use a single neural radiance cache leveraging recently proposed encodings [Müller et al. 2019; Vaswani et al. 2017] to adapt to local scene variations, and (ii) we train our model online which allows for fully dynamic scenes and readily accounts for all lighting in the scene in a single query. Neural control variates [Müller et al. 2020] and NeRF [Mildenhall et al. 2020], developed in the context of offline rendering, encompass a radiance cache that is parameterized similarly. The key differences of our work are: (i) a network architecture and implementation designed for a rendering budget on the order of milliseconds instead of minutes, and (ii) integration in a renderer using self-training, which has been connected with Q-learning [Dahm and Keller 2018], to account for infinite bounces of indirect illumination despite tracing paths of finite length.

3 NEURAL RADIANCE CACHING

Our goal is to cache radiance using *one single* neural network that maps spatio-directional coordinates to radiance values and is trained in real-time to support dynamic scenes. We opt for approximating the *scattered radiance* as it is the most computationally expensive part of the rendering equation [Kajiya 1986]. The *scattered radiance*

$$L_\mathrm{s}(\mathbf{x}, \omega) := \int_{S^2} f_\mathrm{s}(\mathbf{x}, \omega, \omega_\mathrm{i})\, L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, |\cos\theta_\mathrm{i}|\, \mathrm{d}\omega_\mathrm{i} \tag{1}$$

represents the radiative energy leaving point $\mathbf{x}$ in direction ω after being scattered at $\mathbf{x}$. For a given direction of incidence ω_i, the integrand is the product of the bidirectional scattering distribution function (BSDF) $f_\mathrm{s}(\mathbf{x}, \omega, \omega_\mathrm{i})$, the incident radiance $L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})$, and the foreshortening term $|\cos\theta_\mathrm{i}|$, where θ_i is the angle between ω_i and the surface normal at $\mathbf{x}$. Our neural network approximates L_s by the cached radiance $\widehat{L}_\mathrm{s}$.

In this section, we discuss the algorithmic choices for building a neural radiance cache that are key to satisfy the design principles outlined in Section 1. Real-time performance is enabled by an optimized fully fused network, which is discussed in Section 4.

3.1 Algorithm Overview

Rendering a single frame consists of computing pixel colors and updating the neural radiance cache; see Figure 2 for an illustration.

First, we trace short rendering paths, one for each pixel, and terminate them as soon as the approximation provided by the radiance cache is deemed sufficiently accurate. We use the heuristic by Bekaert et al. [2003], that was originally developed in the context of

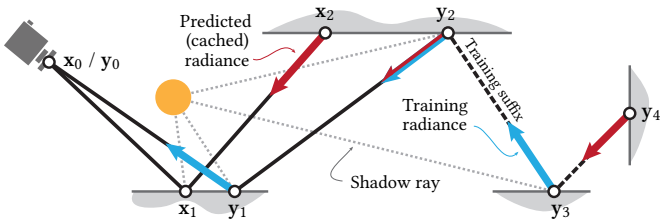


Fig. 2. For rendering, we trace short "rendering" paths (e.g. $\mathbf{x}_0 \cdots \mathbf{x}_2$) and terminate them into the neural radiance cache; queries of cached radiance $\widehat{L}_s$ are highlighted by red arrows. To optimize the cache, we extend a small subset of the rendering paths by a few vertices (called "training suffix", e.g. $\mathbf{y}_2 \cdots \mathbf{y}_4$). We collect radiance estimates (blue arrows) to update the neural radiance cache along the vertices of the longer training path, reusing the initial path segment that was already traced for rendering. Furthermore, the online training together with the termination of training paths into the cache progressively increases the number of simulated light bounces.

photon density estimation, to only query the cache once the spread of the path is sufficiently large to blur small inaccuracies of the cache (more detail in Section 3.4). At each intermediate vertex, we use next-event estimation to integrate light from emitters. To this end, we use screen-space ReSTIR [Bitterli et al. 2020] at the primary vertex and a LightBVH [Moreau et al. 2019], combined with the BSDF via multiple importance sampling [Veach and Guibas 1995], at the subsequent vertices. Truncating the path at the terminal vertex $\mathbf{x}_k$, we evaluate the neural radiance cache to approximate $L_s(\mathbf{x}_k, \omega_k)$.

Second, to train the radiance cache, we extend a fraction (typically under 3%) of the short rendering paths by a few vertices—a *training suffix*. As before, we terminate these longer training paths once the area spread of their suffix is sufficiently large; for that purpose we consider the query vertex $\mathbf{x}_k$ as a primary vertex (see Figure 5). In the majority of cases, the suffix consists of one vertex. The radiance estimates collected along *all* vertices of the longer training paths are used as reference values for training the radiance cache.

Discussion. Terminating the paths into the radiance cache saves computation and, importantly, replaces a one-sample estimate with an approximation that aggregates samples from spatially and temporally nearby locations. The variance is thus reduced, however, the viability of caching for real-time applications is still conditioned on how efficiently and quickly we update and query the cache.

3.2 Fast Adaptation of the Cache by Self-training

As in any data-driven approach, the quality of the approximation depends on the accuracy of the target values L_s that the network is trained on. The distinct challenge of rendering dynamic scenes in real time requires to continuously adapt the neural radiance cache according to the changing radiance field, for example, due to moving lights or geometry. This means we neither have the luxury of precomputing precise target values L_s, nor can we tolerate noisy estimates that would slow down convergence.

Instead of estimating target values via Monte Carlo path tracing [Müller et al. 2019; Müller et al. 2020], we leverage the neural radiance cache itself by evaluating it at the terminal vertices of the longer training paths. The collected radiance is transported to the preceeding vertices, at each one generating a target value for training the neural network. Updating the neural radiance cache using its own values resembles the concept of Q-learning [Dahm and Keller 2018; Keller and Dahm 2019].

The self-training approach has two distinct advantages over fully path-traced estimates: it trades a large fraction of the undesired noise for (potential) bias when estimating L_s. It also allows for capturing global illumination as long as the training procedure is iterated: the radiance learned by one training path is transported using multiple other training paths in the next iteration. Hence, each iteration increases the number of simulated light bounces. This is reminiscent of progressive radiosity algorithms [Martin and Einarsson 2010] that simulate multi-bounce diffuse transport by iterating single-bounce radiative transfer.

However, self-training the neural radiance cache also has two caveats: first, the last vertex of the training path may reach scene locations that the radiance cache has not been trained for, which may incur a larger approximation error. The second drawback is that the iterated optimization may simulate only a subset of multi-bounce illumination rather than all light transport. Specifically, only the transport from emitters that can be reached by training paths will be further bounced around, and only so, if tails of training paths in subsequent frames land near the current optimization points (i.e. $\mathbf{y}_4$ needs to be close to $\mathbf{y}_2$ or $\mathbf{y}_3$ in Figure 2). Both caveats can be alleviated almost for free by making a small fraction u of the training paths truly unbiased, thereby injecting correct source values to be propagated by the self-training mechanism. We use $u = 1/16$, i.e. every 16^{th} training suffix is only terminated by Russian roulette.

3.3 Temporally Stable Cache Queries

When rendering dynamic content, for example changing camera position or animated geometry, the neural radiance cache continuously needs to adapt, forcing us to use a high learning-rate when optimizing the network by gradient descent. In addition, we also perform *multiple* (in our case 4) gradient descent steps per frame[1], which leads to even faster adaptation.

However, a side effect of such an aggressive optimization schedule are temporal artifacts like flickering and oscillations across the rendered frames—even when the scene and camera are static, because there is noise in the estimated radiance targets.

We therefore propose to dampen such oscillations by averaging the network weights produced by the optimization. More specifically, we compute an exponential moving average (EMA) of the network weights W_t produced by the t^{th} gradient descent step, which creates a *second* set of weights $\overline{W}_t$ that we use when evaluating the cache for rendering. The exponential moving average reads

$$\overline{W}_t := \frac{1-\alpha}{\eta_t} \cdot W_t + \alpha \cdot \eta_{t-1} \cdot \overline{W}_{t-1}, \text{ where } \eta_t := 1 - \alpha^t \qquad (2)$$

corrects the bias of the average for small t and $\alpha \in [0, 1]$ controls the strength of exponential averaging. We use $\alpha = 0.99$ for a good trade-off between fast adaptation yet temporally stable evolution of the weights $\overline{W}_t$; we illustrate the temporal stability in Figure 3.

[1]Each step uses a disjoint random subset of the training data that was gathered while rendering the frame to prevent the same data from being seen twice.

Note that the averaging process does not feed back into the training loop; $\overline{W}_t$ depends on W_t, but not the other way around. Still, recent work suggests that the EMA filtered weights $\overline{W}_t$ may be closer to the optimum than any of the raw weights W_t produced by the optimizer [Izmailov et al. 2018].

Indeed, Figure 4 shows that when the radiance cache is trained from scratch, its evolution is gradual and quick at the same time. Using the cache at the end of paths instead of visualizing it directly filters its approximation error, converging to satisfactory quality in as few as 8 frames (~ 70 ms); see the supplementary video for more results on animated content.

3.4 Path Termination

All paths are terminated according to a simple heuristic based on the area-spread of path vertices, illustrated as cones in Figure 5; we index the camera vertex as $\mathbf{x}_0$ and the primary vertex as $\mathbf{x}_1$. Once the spread becomes large enough to blur away the small-scale inaccuracies of our cache (c.f. Figure 12), we terminate the path.

Following Bekaert et al. [2003], the area spread along the subpath $\mathbf{x}_1 \cdots \mathbf{x}_n$ can be cheaply approximated as the sum

$$a(\mathbf{x}_1 \cdots \mathbf{x}_n) = \left(\sum_{i=2}^{n} \sqrt{\frac{\|\mathbf{x}_{i-1} - \mathbf{x}_i\|^2}{p(\omega_\mathrm{i} \mid \mathbf{x}_{i-1}, \omega) \, |\cos \theta_\mathrm{i}|}} \right)^2, \tag{3}$$

where p is the BSDF sampling PDF and θ_i is the angle between ω_i and the surface normal at $\mathbf{x}_i$.

To terminate a path, we compare the subpath spread $a(\mathbf{x}_1 \cdots \mathbf{x}_n)$ to the spread at the primary vertex as viewed from the camera, which can be approximated[2] as

$$a_0 := \frac{\|\mathbf{x}_0 - \mathbf{x}_1\|^2}{4\pi \cos \theta_1}. \tag{4}$$

That is, we will terminate a path if $a(\mathbf{x}_1 \cdots \mathbf{x}_n) > c \cdot a_0$, where c is a hyperparameter that trades variance (longer paths) for bias and speed (shorter paths). We found $c = 0.01$ to yield satisfactory results.

Lastly, if the path is selected to become a training path, the heuristic will be used once again, this time to terminate the training suffix when $a(\mathbf{x}_n \cdots \mathbf{x}_m) > c \cdot a_0$ is satisfied. The heuristic is illustrated in Figure 5 for a training path, where the short rendering part of the path ends at vertex $n = 2$ and the training suffix at $n = 4$.

3.5 Amortization in a Real-time Path Tracer

We target real-time applications, setting ourselves a 16.6 milliseconds rendering budget in order to achieve a framerate of 60 frames per second. That budget includes the tracing of paths, shading at every vertex, as well as querying and updating the cache. In practice, this leaves just a few milliseconds to handle the cache overhead. We not only tackle this using our proposed fully fused network, described in Section 4, but also in the path tracer integration itself.

We interleave short rendering paths and long training paths by tiling the viewport. Using a single random offset, we promote one path per tile to be a long training path (see Figure 5), resulting in a uniform sparse set of training paths in screen space. This approach of merely prolonging a rendering path to obtain a training path

[2]By assuming a spherical image plane and ignoring constant factors.

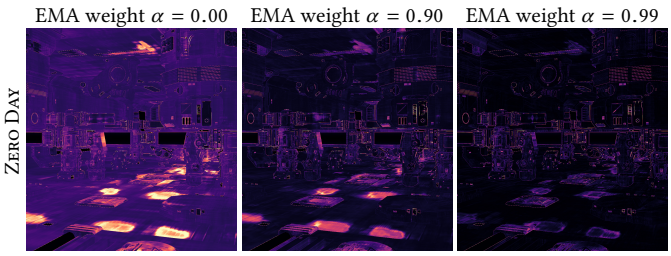


Fig. 3. Temporal stability of online learning with three weight-averaging strategies: no averaging (left) and an exponential moving average (EMA) with weights 0.90 and 0.99. The false color images depict temporal stability across 100 frames, measured as the symmetric mean absolute percentage error (SMAPE) averaged over all consecutive pairs of frames; i.e. darker means less variation across frames. To be more meaningful in print, the scene and the camera have been fixed. Please see the accompanying video to best assess the temporal stability.

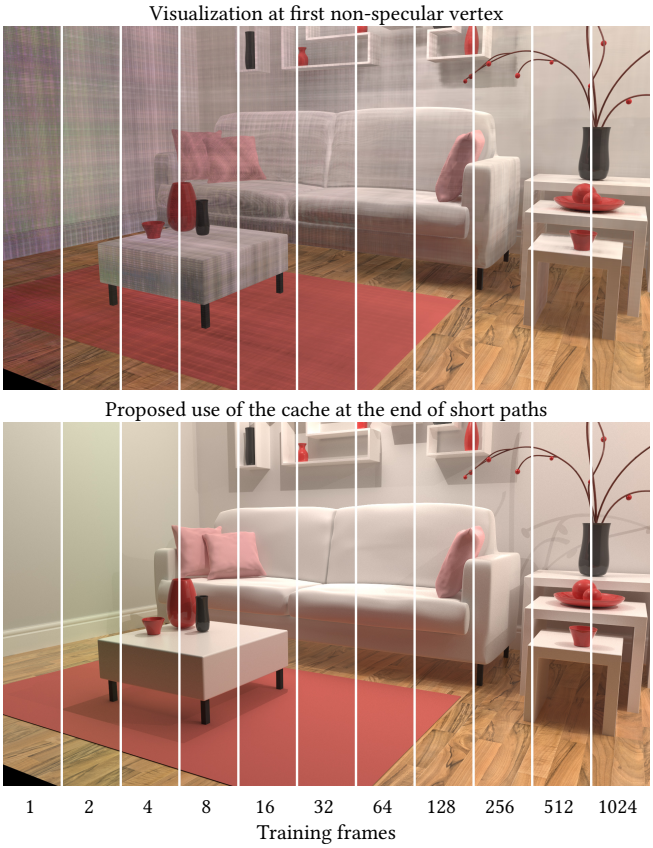


Fig. 4. From-scratch training of the neural radiance cache. We visualize the cache after 1, 2, 4, ..., 1024 frames. Top: to illustrate its training behavior, the radiance cache is visualized directly at the first non-specular vertex. Already after the first 64 frames (~ 0.5s) the overall colors are correct and only subtle high-frequency artifacts remain. Bottom: using the cache as proposed, the high-frequency artifacts are hidden behind path indirections and the cache is usable starting from the ~ 8th frame (i.e. after ~ 70ms). This confirms that the cache trains sufficiently fast for the online adaptation to animated content; see the supplementary video for more results.

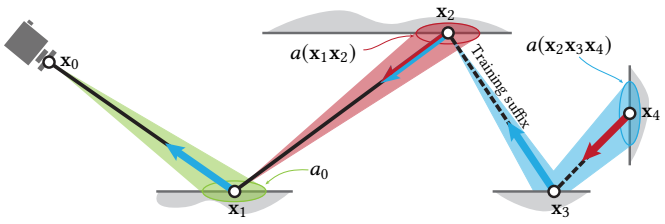


Fig. 5. We terminate our short rendering paths into the neural radiance cache once their scattering interactions blur the signal sufficiently well. To this end, we compare the size of the footprint of the path $a(\mathbf{x}_1\mathbf{x}_2)$ to the size of the directly visible surface in the image plane a_0. The longer training paths are terminated by the same heuristic applied to the vertices of the suffix, i.e. we compare $a(\mathbf{x}_2\mathbf{x}_3\mathbf{x}_4)$ to a_0.

greatly reduces the overhead, as computation is shared between the two. This contrasts with caching techniques based on probe volumes, which use a separate set of rays to update the cache that do not contribute to the image itself.

Once the tracing is complete, we reconstruct the image by back-propagating the cached radiance from the terminal vertex of each short rendering path. For training paths, we track two values: the aforementioned rendering radiance and the training radiance. For every vertex along a training path, we store the training radiance (along with the vertex information) in an array; it will be used as the target to optimize the cache.

We shuffle all training records using a linear congruential generator and distribute them over s training batches of l training records each. The shuffling ensures that the training batches are not correlated with image regions. Each training batch is used to perform a single optimization step of the cache. As we want to ensure a stable work load, we use an adaptive tiling mechanism to match a target training budget; we select $s = 4$ training batches and $l = 16384$ records per batch in practice, for a total of 65536 training records per frame. We dynamically adjust the tile size at each frame based on the number of training records generated during the image reconstruction. Similar to other caching techniques, the cost of training is decoupled from the image resolution, since we use a bounded number of training records. Lastly, we observe that training our neural radiance cache amounts to a regression over many samples from spatially and temporally nearby locations, i.e. a form of path-space denoising. The variance is thus significantly reduced by replacing one-sample radiance estimates with the cache approximation.

3.6 Input Encoding

Ren et al. [2013] showed that solely using the spatio-directional coordinates $(\mathbf{x}, \omega)$ of the scattered radiance as input to a neural network does not allow it to represent radiance well. Therefore, the input is augmented by additional parameters that correlate with the scattered radiance: the surface normal $\mathbf{n}$, the surface roughness r, the diffuse reflectance α, and the specular reflectance β. Being able to exploit such correlations, the neural approximation becomes much more accurate.

It is easier for the network to identify these correlations when they are (nearly) linear. This is already the case for the diffuse and specular reflectances; we thus input them to the network as-is. However, the quantities $\mathbf{x}$, ω, $\mathbf{n}$, and r have a highly non-linear relation to the scattered radiance. For these quantities, a well-chosen encoding to a higher-dimensional space can make the relation more linear and thereby make the neural approximation more accurate.[3]

Table 1. Parameters and their encoding, amounting to 62 dimensions: freq denotes frequency encoding [Mildenhall et al. 2020], ob denotes one-blob encoding [Müller et al. 2019], sph denotes a conversion to spherical coordinates, normalized to the interval $[0, 1]^2$, and id is the identity.

Parameter	Symbol	with Encoding
Position	$\mathbf{x} \in \mathbb{R}^3$	$\text{freq}(\mathbf{x}) \in \mathbb{R}^{3\times12}$
Scattered dir.	$\omega \in S^2$	$\text{ob}(\text{sph}(\omega)) \in \mathbb{R}^{2\times4}$
Surface normal	$\mathbf{n}(\mathbf{x}) \in S^2$	$\text{ob}(\text{sph}(\mathbf{n}(\mathbf{x}))) \in \mathbb{R}^{2\times4}$
Surface roughness	$r(\mathbf{x}, \omega) \in \mathbb{R}$	$\text{ob}\left(1 - e^{-r(\mathbf{x},\omega)}\right) \in \mathbb{R}^4$
Diffuse reflectance	$\alpha(\mathbf{x}, \omega) \in \mathbb{R}^3$	$\text{id}(\alpha(\mathbf{x}, \omega)) \in \mathbb{R}^3$
Specular reflectance	$\beta(\mathbf{x}, \omega) \in \mathbb{R}^3$	$\text{id}(\beta(\mathbf{x}, \omega)) \in \mathbb{R}^3$

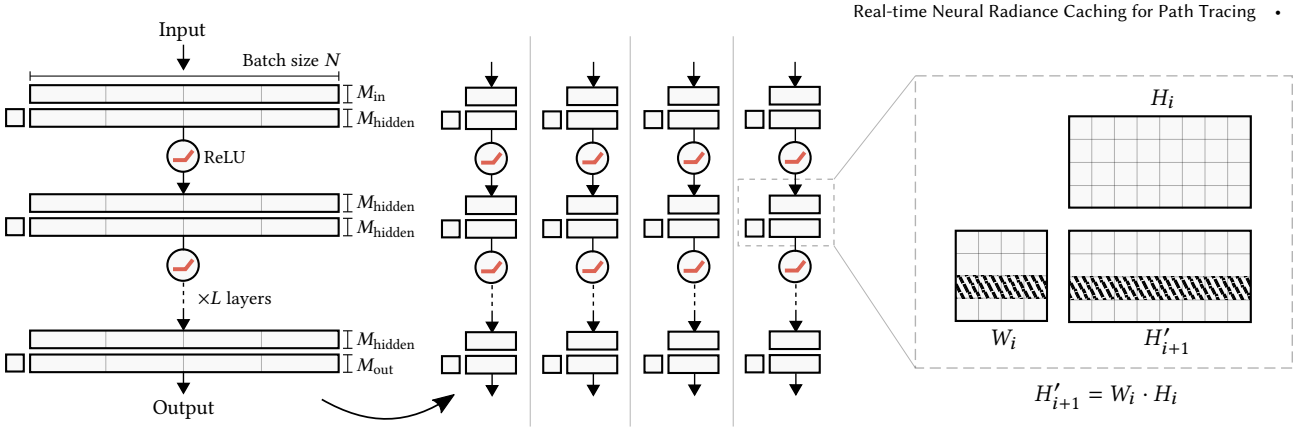
The extra dimensions do not come for free, as they increase the required memory traffic as well as the cost of the first layer of the neural network. We thus aim at encoding the quantities $\mathbf{x}$, ω, $\mathbf{n}$, and r using *as few as possible* extra dimensions while still profiting from the linearization.

To this end, the one-blob encoding [Müller et al. 2019] works well when the scale of the nonlinearities is about the same order of magnitude as the size of the blobs. This is a good fit for ω, $\mathbf{n}$, and r as tiny variations in these parameters typically do not change the scattered radiance much. We thus encode them using a very small number (e.g. $k = 4$) of evenly spaced blobs.

However, tiny changes in the position $\mathbf{x}$ can cause large variation in the scattered radiance, e.g. along shadow and geometric boundaries or in outdoor environments that are much larger than the view frustum. One-blob encoding is therefore unsuitable for robustly encoding the position within just a few extra dimensions. Instead, we adopt the frequency encoding from transformer networks [Vaswani et al. 2017], introduced to radiance learning by Mildenhall et al. [2020], that leverages a geometric hierarchy of periodic functions to represent a high dynamic range of values in few encoded dimensions. We use 12 sine functions, each with frequency $2^d, d \in \{0, \ldots, 11\}$. To save on dimensions, we found that omitting the cosine terms of the original method does not compromise approximation quality.

In summary, the input of our neural network is a concatenation of the following: the frequency-encoded position $\mathbf{x}$, the one-blob encoded parameters ω, $\mathbf{n}$, and r, and the raw diffuse albedo α and specular reflectance β; see Table 1 for a detailed breakdown. This results in a total of 62 input dimensions to the neural network, which we pad to 64 for compatibility with the hardware matrix-multiplication accelerator. We pad with a value of 1, which allows the network to implicitly learn a bias term (the corresponding columns of the first weight matrix) even though our architecture lacks explicit biases.

[3]This is analogous to the "kernel trick" that is often employed in machine learning to make the data linearly separable [Theodoridis 2008].



(a) Batched neural network evaluation **(b)** Distribution of a batch over thread blocks **(c)** Per-thread-block matrix multiplication

Fig. 6. **(a)** Evaluating a multi-layer perceptron (MLP) for a large batch of inputs (e.g. $N \approx 2^{21}$ for a 1920×1080 frame) amounts to alternating weight-matrix multiplication and element-wise application of the activation function. **(b)** In our fully fused MLP, we parallelize this workload by partitioning the batch into 128 element wide chunks that are each processed by their own thread block. Since our MLP is narrow ($M_{\text{hidden}} = M_{\text{in}} = 64$ neurons wide), its weight matrices fit into registers and the intermediate 64×128 neuron activations fit into shared memory. This is key to the superior performance of the fully fused approach. **(c)** The matrix multiplication performed by each thread block transforms the i-th layer H_i into the pre-activated next layer H'_{i+1}. It is diced into blocks of 16×16 elements to match the size of our hardware-accelerated half-precision matrix multiplier (TensorCore). Each warp of the thread block computes one 16×128 block-row of H'_{i+1} (e.g. the striped area) by first loading the corresponding 16×64 striped weights from W_i into registers and subsequently multiplying them by all 64×16 block-columns of H_i. Thus, each thread block loads the weight matrix from global memory exactly once (the least possible amount), making multiple passes only over H_i which, however, is located in fast shared memory.

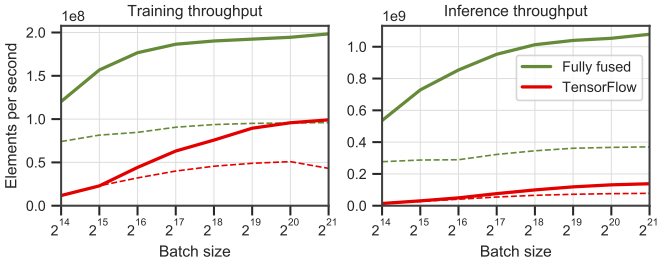


Fig. 7. Our fully fused neural network outperforms an equivalent XLA-enabled TensorFlow (v2.5.0) implementation. Both implementations utilize half precision floating point numbers and TensorCore hardware for matrix multiplication. We compare the throughput of training (left) and inference (right) for a 64 (solid line) and a 128 (dashed line) neurons wide multi-layer perceptron. The relevant batch sizes for our goal of neural radiance caching are *small* training batches (e.g. 2^{14} elements) and *large* inference batches (e.g. 2^{21} elements for evaluating a 1920×1080 frame). For these batch sizes, the speed-up over TensorFlow ranges from 5× to 10×.

4 FULLY FUSED NEURAL NETWORKS

We implemented our neural network from scratch in a GPU programming language in order to take full advantage of the GPU memory hierarchy. In Figure 7, we compare the performance of this implementation to TensorFlow (v2.5.0) [Abadi et al. 2015], which we outperform by almost an order of magnitude.

To understand where this dramatic speedup comes from, we examine the bottleneck of evaluating a fully connected neural network like ours. The *computational* cost of such a neural network scales quadratically with its width, whereas its *memory traffic* scales linearly. Modern GPUs have vastly larger computational throughput than they have memory bandwidth, though, meaning that for *narrow* neural networks like ours, the linear memory traffic is the bottleneck. The key to improving performance is thus to minimize traffic to slow "global" memory (VRAM and high-level caches) and to fully utilize fast on-chip memory (low-level caches, "shared" memory, and registers).

Our fully fused approach does precisely this: we implement the *entire* neural network as a single GPU kernel that is designed such that the only slow global memory accesses are reading and writing the network inputs and outputs. Furthermore, implementing the kernel from scratch as opposed to building it out of existing frameworks allows us to specifically tailor the implementation to the network architecture and the GPU that we use.

Figure 6 illustrates how the fully fused approach is mapped to the memory hierarchy. Using CUDA terminology: a given batch of input vectors is partitioned into block-column segments that are processed by a single thread block each (Figure 6(b)). The thread blocks independently evaluate the network by alternating between weight-matrix multiplication and element-wise application of the activation function. By making the thread blocks small enough such that all intermediate neuron activations fit into on-chip shared memory, traffic to slow global memory is minimized. This is the key advantage of the fully fused approach and stands in contrast to typical implementations of general matrix multiplication.

Within a matrix multiplication (Figure 6(c)), each warp of the thread block computes the matrix product of a single block-row (striped area). In our case, the striped weights in W_i are few enough to fit into the registers of the warp and can thus be re-used for every block of H'_{i+1} that the warp computes, yielding an additional performance gain. Furthermore, since each warp loads a distinct block-row of the weight matrix, the entire thread block loads the weight matrix from global memory exactly once, which cannot be reduced further.

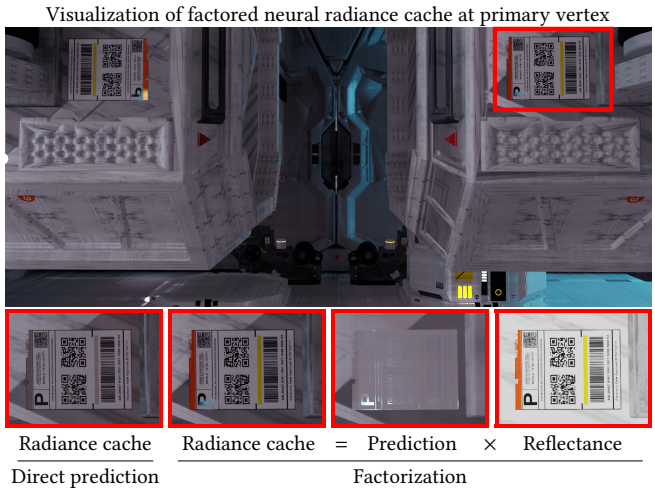


Fig. 8. Reflectance factorization leads to more accurate textured colors and lets the neural prediction focus on complementary detail such as glossy highlights. Note that sharp texture detail is also present when predicting the product, because the reflectance is input to the network in any case.

The only possible remaining reduction of global memory traffic is thus to minimize the *number* of thread blocks by making them as large as fits into shared memory. On our hardware (NVIDIA RTX 3090) and with our 64-neurons-wide network, this sweet-spot is met when each thread block processes 128 elements of the batch. Each thread block thus computes matrix products of a 64×64 weight matrix with a 64×128 chunk of the data.

Training the fully fused neural network. For training, the forward and backward passes admit the same matrix multiplication structure as the previously discussed inference pass. However, they require additional global-memory traffic, because intermediate activations and their gradients must be written out for backpropagation. Furthermore, additional matrix multiplications are necessary to turn the results of backpropagation into the gradients of the weight matrices. We compute these additional matrix multiplications using the general matrix multiplication (GEMM) routines of the CUTLASS template library (in split-k mode) as we were unable to produce a faster implementation ourselves.

All these additional operations make training slower than inference by a factor of roughly 4×–5×; see Figure 7.

5 PRACTICAL CONSIDERATIONS

Architecture. Our fully fused neural network architecture (see Figure 6) comprises of seven fully connected layers. The five hidden layers have 64 neurons each with ReLU activation functions. The output layer reduces the 64 dimensions to three RGB values. None of the layers has a bias vector, as biases did not result in any measurable quality benefit and omitting them makes the fully fused implementation simpler and more efficient. Note that the neural network is shallow enough for vanishing gradients not to be a problem. Hence there is no need for residual layers using skip links to help training, which we confirmed experimentally.

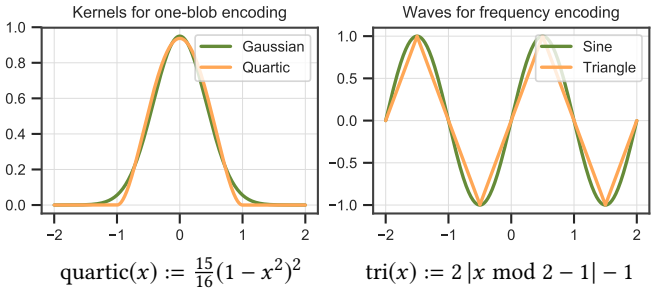


Fig. 9. To avoid expensive mathematical operations, we replace the Gaussian kernel of the one-blob encoding by a quartic kernel and the sine function in the frequency encoding by a triangle wave. With these replacements, the cost per frame is reduced by 0.25ms with no visible loss of quality.

Reflectance factorization. To improve textured colors reproduction, we multiply the network output by the sum of the diffuse albedo and specular reflectance $\alpha(\mathbf{x}, \omega) + \beta(\mathbf{x}, \omega)$. For Lambertian materials, this amounts to irradiance factorization [Ward et al. 1988], and the network is effectively tasked with learning irradiance as opposed to reflected radiance. However, even in our highly non-Lambertian scenes, the above factorization is helpful. The factorization is *not* necessary to recover sharp detail—$\alpha(\mathbf{x}, \omega)$ and $\beta(\mathbf{x}, \omega)$ are input to the network in any case—but it helps recover colors while letting the cache focus on complementary details; see Figure 8.

High-performance primitives for encoding. The one-blob and frequency encodings rely on primitives that are computationally expensive: Gaussian kernels and trigonometric functions. We thus replace the primitives with approximations that are far cheaper to evaluate. Specifically, we replace the Gaussian with a quartic kernel and the sine function with a triangle wave as illustrated in Figure 9, reducing the cost per frame by 0.25 ms with no visible loss of quality.

Relative loss. To facilitate effective training, we use the relative $\mathcal{L}^2$ loss that admits unbiased gradient estimates when the training signal—the reflected radiance $L_s(\mathbf{x}, \omega)$—is noisy [Lehtinen et al. 2018]. The loss is normalized by the neural prediction:

$$\mathcal{L}^2\left(L_s(\mathbf{x}, \omega), \widehat{L}_s(\mathbf{x}, \omega; W_t)\right) := \frac{\left(L_s(\mathbf{x}, \omega) - \widehat{L}_s(\mathbf{x}, \omega; W_t)\right)^2}{\mathrm{sg}\left(\widehat{L}_s(\mathbf{x}, \omega; W_t)\right)^2 + \epsilon}, \quad (5)$$

where $\epsilon = 0.01$ and $\mathrm{sg}(\cdot)$ denotes that its argument is treated as a constant in the optimization, i.e. no gradient is propagated back. Furthermore, for spectral values of $L_s(\mathbf{x}, \omega)$, we normalize the loss of each color channel by the squared *luminance* across the spectrum.

Optimizer. The choice of optimizer is crucial to effectively leverage the little training data we have per frame. To this end, we compared multiple first-order optimizers, i.e. stochastic gradient descent (SGD), Adam [Kingma and Ba 2014], and Novograd [Ginsburg et al. 2019]) and found that Adam converges in the fewest iterations while having an overhead of practically zero.

We also investigated a second order optimizer, Shampoo [Anil et al. 2020; Gupta et al. 2018], which converges with slightly fewer iterations than Adam. However, the 0.3 milliseconds per-frame overhead of our optimized implementation did not justify its benefit in our experiments. We thus use Adam in all results.

Fig. 10. We demonstrate the benefit of our neural radiance cache (NRC) in 1 spp real-time renderings with varied material and lighting complexity. From left to right: our baseline is unbiased path tracing with Russian roulette and next-event estimation driven by a light BVH [Moreau et al. 2019]. Then, we add spatiotemporal reservoir resampling (ReSTIR) [Bitterli et al. 2020] for low-variance direct and NRC for low-variance indirect illumination. Individually, these complementary techniques excel in their respective domains (e.g. ReSTIR in the directly lit Bistro and NRC in the glossy Zero Day scene), but their combination unlocks the biggest improvement. Together, the three techniques reduce the mean relative squared error (MRSE) of path tracing by 1–2 orders of magnitude while incurring a comparatively little performance loss thanks to the drastic path shortening of NRC. In all scenes, the combined technique exceeds 60 frames per second at a resolution of 1920 × 1080. We also report the perceptually based ꟻLIP metric that is more robust to outliers ("fireflies").

Table 2. Time to converge to equal MRSE.

Scene	Method	Frames	Time	MRSE	Speedup
Attic	PT+ReSTIR	7	92.5 ms	2.739	**6.6×**
	PT+ReSTIR+NRC	1	14.0 ms	2.727	
Bistro	PT+ReSTIR	8	110.7 ms	1.498	**7.6×**
	PT+ReSTIR+NRC	1	14.6 ms	1.407	
Classroom	PT+ReSTIR	145	2625 ms	5.882	**172.7×**
	PT+ReSTIR+NRC	1	15.2 ms	5.847	
Living Room	PT+ReSTIR	53	431.5 ms	1.379	**49.6×**
	PT+ReSTIR+NRC	1	8.7 ms	1.376	
Pink Room	PT+ReSTIR	10	66.9 ms	0.769	**8.4×**
	PT+ReSTIR+NRC	1	8.0 ms	0.765	
Zero Day	PT+ReSTIR	5	69.4 ms	3.799	**6.1×**
	PT+ReSTIR+NRC	1	11.3 ms	3.430	
Average	PT+ReSTIR	16.6	154.1 ms	2.037	**13.6×**
	PT+ReSTIR+NRC	1	11.3 ms	1.941	

6 RESULTS AND DISCUSSION

We implemented all components of our neural radiance cache in CUDA, i.e. input encoding, the fully fused network, and the optimizer, the source code of which we release publically [Müller 2021]. The radiance cache is integrated into a path tracer implemented in Direct3D 12 using the Falcor rendering framework [Benty et al. 2020] with which we generated all results in this paper.

All images were rendered at a resolution of 1920×1080 on a high-end desktop machine (i9 9900k and RTX 3090). For each image, we report the mean relative squared error (MRSE) [Rousselle et al. 2011] or its decomposition into relative bias (rBias) and variance (rVar) to aid the reader in gauging the improved Monte Carlo efficiency. When applicable, we also list the perceptually based ꟻLIP metric [Andersson et al. 2020] that is more robust to outliers ("fireflies"). Our reference images were created using *ReSTIR-enabled* path tracing to ensure that we only measure the bias caused by radiance caching and not that of ReSTIR.

Real-time rendering. In Figure 10, we utilize neural radiance caching (NRC) to reduce indirect illumination noise of a path tracer. By combining NRC with complementary direct-lighting techniques, we get global illumination with both low noise *and* little bias in real-time.

Our baseline is an unbiased path tracer with Russian roulette and next-event estimation driven by a light BVH [Moreau et al. 2019], to which we add screen-space spatiotemporal reservoir resampling (ReSTIR) [Bitterli et al. 2020]. While this algorithm has low variance in its direct lighting estimates (PT+ReSTIR column), it suffers from noisy estimates of *indirect* lighting, which can be remedied by terminating paths into our cache (PT+ReSTIR+NRC column). Shortening the paths in this way not only results in lower variance but sometimes also in *higher* framerates due to the low overhead of querying and training the fully fused network.

Compared with path tracing, the combination of ReSTIR and NRC reduces the MRSE by 1–2 orders of magnitude while having a comparatively small impact on performance. In all scenes, the combined technique exceeds 60 frames per second.

We measure our speedup in Table 2 by letting the baseline converge to equal MRSE as a single rendered frame of our method. The average speedup across the test scenes is 13.6×.

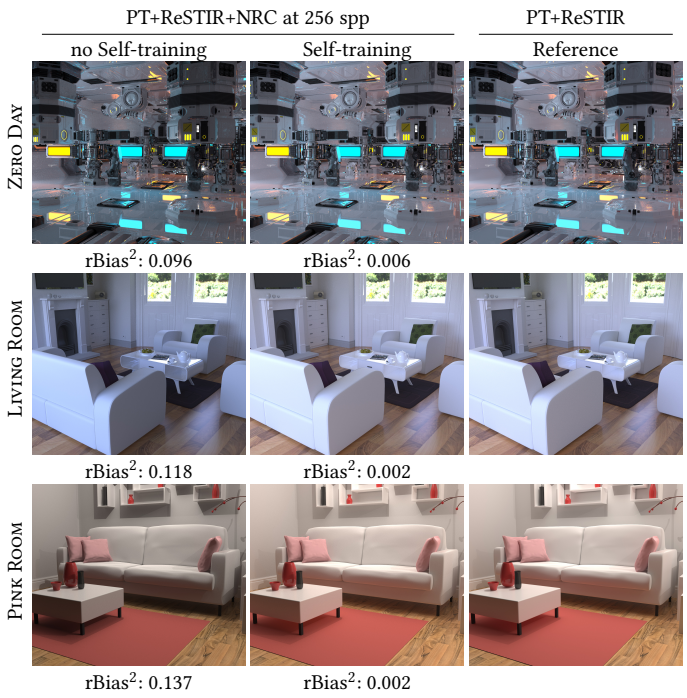


Fig. 11. Self-training enables efficient learning of indirect radiance that is not captured by short paths: compare the left and middle images, both of which trace paths of the same length (including the sparse set of unbiased training paths described in Section 5). On the right, we provide a reference image obtained by tracing paths that are truncated with Russian roulette.

Self-training. We compare the performance of self-training to training relying on pure path-tracing. Terminating training paths with our termination heuristic, we set the tail contribution either to black (path-traced training) or to the radiance prediction at the last vertex (self-training). As shown in Figure 11, the self-trained solution accurately captures multi-bounce light transport. The computational overhead of self-training amounts to the cost of querying the radiance cache one additional time for each of the few training paths, which amounts roughly to a 1% overall overhead—a small amount compared with the cost that is saved by not having to trace longer paths to learn global illumination.

Quality of the cache. In Figure 12, we study the quality of the neural radiance cache by visualizing it at the first non-specular vertex. By being agnostic to the materials and geometry of the underlying scenes, the cache handles a wide range of visual phenomena. For example, it handles complex glossy transport in the Zero Day scene, shadow detail at a distance in the Bistro scene, and thin geometry without light leakage, such as the tarp in the Attic scene. The cache also performs well at capturing the overall color of almost every region in each scene.

The limitations of the cache are twofold. First, the cache does not capture sharp detail very well if that detail is absent from the inputs to the network (e.g. a sharp contact shadow or caustic). And second, the cache exhibits subtle axis-aligned stripes that are a byproduct of the frequency encoding [Mildenhall et al. 2020]. Since we rely on

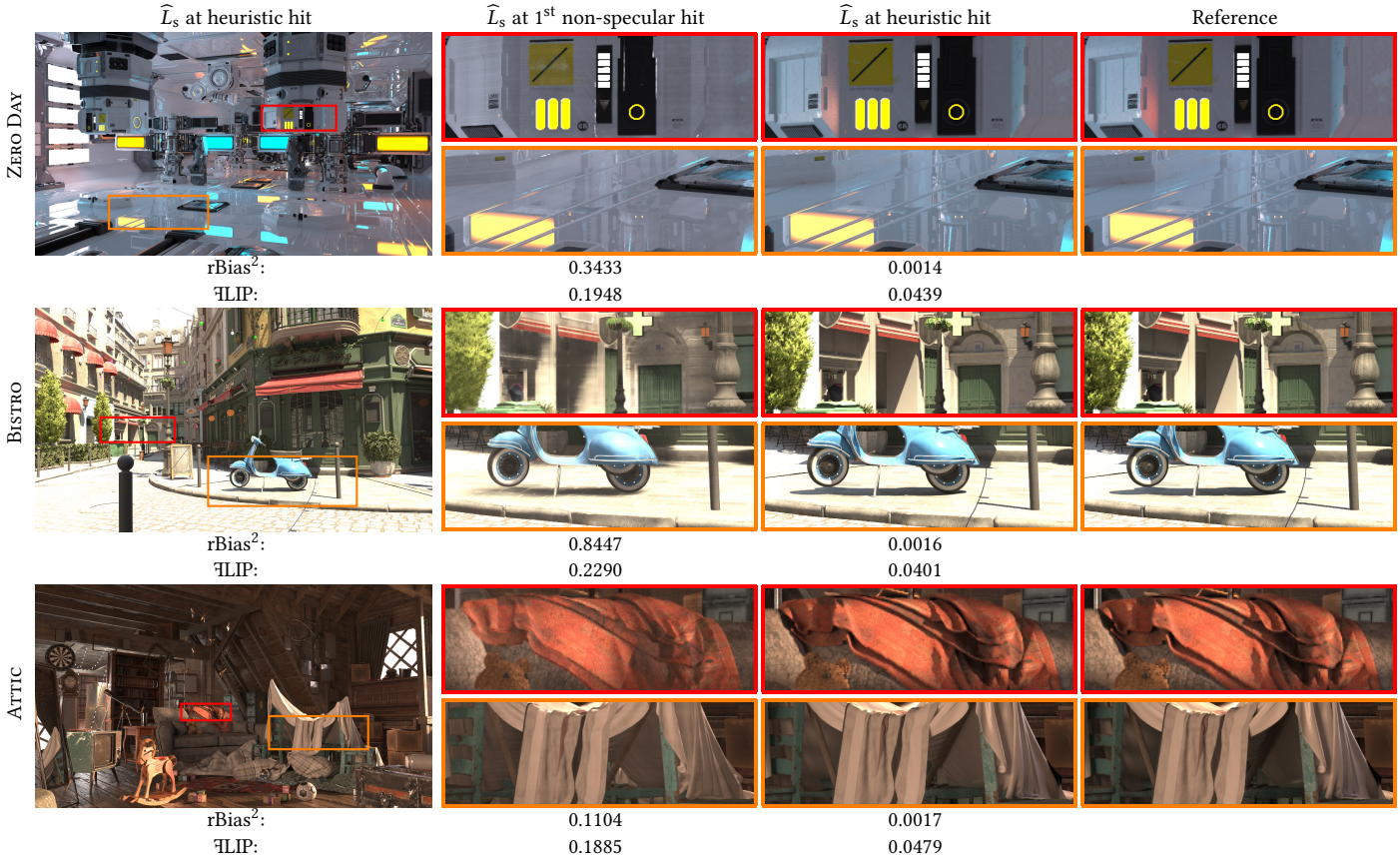


Fig. 12. Converged renderings when querying the cache at the first non-specular vertex, or according to the path termination heuristic. We measure accuracy using the objective relative square bias metric, and the perceptual ꟻLIP metric [Andersson et al. 2020]. The Zero Day scene features complex area lighting, glossy materials, and high-order indirect illumination due to the high albedo of surfaces. The Bistro scene features small geometry and shadow details in a relatively large environment, highlighting the local adaptation afforded by the encoding of the network inputs. The Attic scene is an interesting test case for light leakage as it features a fairly high geometric complexity, including thin elements such as the tarp. In all three scenes, employing the termination heuristic leads to a high quality result that closely resembles the reference image. In particular, the heuristic recovers contact shadows and local ambient occlusion, whereas other real-time caching techniques typically require an additional screen-space ambient occlusion (SSAO) pass to recover these details.

the frequency encoding to handle a spatial detail at scale (e.g. the accurate far-away shadows in the Bistro scene), we cannot simply switch to a different encoding. All other encodings that we tried either exhibited worse artifacts or did not scale well.

Thus, to mask away the remaining cache inaccuracies, we defer its evaluation according to the termination heuristic from Section 3.4. Since the heuristic is based on the path footprint, the cache artifacts are observed through rough reflections and therefore averaged out. The heuristic also leads to fewer cache queries in concavities, where contact-shadow inaccuracies could become an issue. The insets in Figure 12 confirm this observation: combined with the termination heuristic, the cache produces images that are difficult to distinguish from the ground truth, both visually and numerically.

Comparison with dynamic diffuse global illumination (DDGI). In Figure 13 we compare the neural cache to DDGI [Majercik et al. 2019]. DDGI is a modern extension of irradiance probes, relying on modulation by the surface normal and albedo to approximate the scattered radiance. As a consequence, DDGI works best on Lambertian materials, which is why we show results using both a Lambertian diffuse BSDF model as well as the more physically based Frostbite model [Lagarde and de Rousiers 2014] that our scenes were modeled with.

DDGI makes an aggressive trade-off for performance and low noise: paths are terminated into the irradiance probes at their *first* diffuse interaction (as opposed to glossy or specular), which is frequently the primary vertex. Consequently, DDGI is on one hand very performant (paths are short) and has little noise, but on the other hand lacks ambient occlusion and can expose visible bias (e.g. in the Pink Room) due to its limited spatial resolution.

With NRC, we make the opposite trade-off. We minimize bias at the cost of slightly reduced performance and (sometimes much) more noise. In contrast to DDGI, our neural representation makes no assumptions about the underlying material model, and our path termination criterion helps to avoid the remaining inaccuracies of NRC while also recovering ambient occlusion. The larger cost of our model could thus be offset by the cost of the separate ambient occlusion pass that DDGI requires.

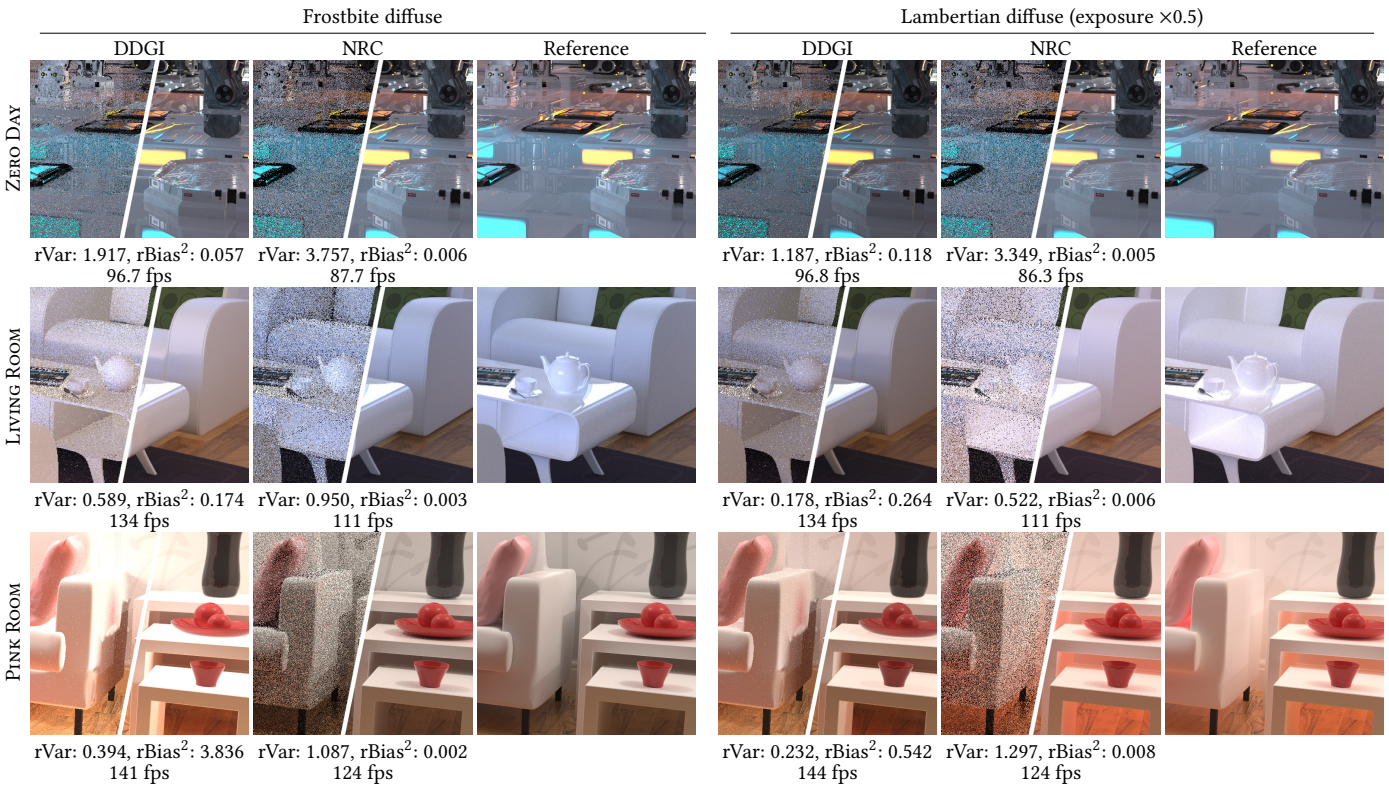


Fig. 13. Comparison of NRC with DDGI under challenging indirect illumination. Both methods use ReSTIR for direct illumination. Since DDGI caches radiance as an extension of irradiance probes, its implementation assumes a Lambertian diffuse BRDF model and recovers detail through surface normal and albedo modulation. As our scenes were modeled using the Frostbite model [Lagarde and de Rousiers 2014], we show results with both the Frostbite (left) and the Lambertian (right) model. DDGI performs at its best on the Lambertian material that it was designed for, whereas NRC—being agnostic to the BSDF—performs similarly in all situations. For each method, we show results at 1spp (left trapezoid) and 1024spp (converged, right trapezoid) to allow for the visual inspection of the bias-variance trade-off. We also report relative bias (rBias) and variance (rVar). DDGI achieves the least variance and runs fastest at the cost of sometimes incurring strong bias, e.g. around problematic geometry in the Pink Room. NRC has much lower bias, however incurring a moderately higher variance and cost. Note that contact shadows and ambient occlusion are implicitly built into NRC via the path termination heuristic, whereas DDGI lacks these visual features, necessitating an additional render pass in practice.

Table 3. Breakdown of rendering cost by component.

Scene	Method	Trace & shade	Query	Training
Attic	PT+ReSTIR	12.96 ms	—	—
	PT+ReSTIR+DDGI	11.56 ms	0.64 ms	1.78 ms
	PT+ReSTIR+NRC	10.88 ms	1.66 ms	1.12 ms
Bistro	PT+ReSTIR	13.75 ms	—	—
	PT+ReSTIR+DDGI	12.71 ms	0.65 ms	1.68 ms
	PT+ReSTIR+NRC	11.96 ms	1.38 ms	1.11 ms
Classroom	PT+ReSTIR	18.06 ms	—	—
	PT+ReSTIR+DDGI	12.93 ms	0.59 ms	1.65 ms
	PT+ReSTIR+NRC	12.28 ms	1.70 ms	1.11 ms
Living Room	PT+ReSTIR	8.32 ms	—	—
	PT+ReSTIR+DDGI	5.68 ms	0.52 ms	0.99 ms
	PT+ReSTIR+NRC	5.82 ms	1.85 ms	1.11 ms
Pink Room	PT+ReSTIR	6.73 ms	—	—
	PT+ReSTIR+DDGI	5.56 ms	0.52 ms	0.89 ms
	PT+ReSTIR+NRC	5.36 ms	1.54 ms	1.12 ms
Zero Day	PT+ReSTIR	13.89 ms	—	—
	PT+ReSTIR+DDGI	8.34 ms	0.54 ms	1.21 ms
	PT+ReSTIR+NRC	8.67 ms	1.41 ms	1.09 ms
Average	PT+ReSTIR	12.29 ms	—	—
	PT+ReSTIR+DDGI	9.46 ms	0.58 ms	1.37 ms
	PT+ReSTIR+NRC	9.16 ms	1.59 ms	1.11 ms

Performance breakdown. In Table 3, we analyze the time spent in DDGI and NRC in more detail. We break down the rendering cost into (i) path tracing & shading, (ii) querying the cache/DDGI, and (iii) training the cache/DDGI.

Compared to the PT+ReSTIR baseline, the low path-tracing cost of DDGI and NRC arises from tracing fewer rays: in addition to Russian roulette, DDGI terminates its paths when a diffuse lobe is sampled, and NRC terminates each path according to its footprint. On average, both methods reduce the cost of path tracing by a similar amount of roughly 2.8 ms per frame (25%). However, this improvement is partially offset by the overhead of querying and training or updating the respective caches.

As expected, querying DDGI is faster than querying our neural network. However, given the reputation of neural networks to be expensive, the difference is smaller than what might be expected: A full-frame DDGI query costs on average 0.58 ms, whereas the neural radiance cache costs 1.59 ms. Both methods are thus well within reasonable cost for real-time settings.

Most interestingly, training the neural radiance cache is *cheaper* than training the DDGI volume (1.11 ms vs. 1.37 ms on average). This has two reasons. First, NRC is very data efficient, using only $2^{16} = 65536$ training records per frame. On the other hand, DDGI in our $16 \times 16 \times 16$ probe-grid configuration traces $16^3 \cdot 256 = 1048576$ update rays per frame—more than 10 times as many as NRC. Second, the few training paths that are traced for NRC share their first couple of vertices with the paths that need to be traced for rendering anyway, further saving cost.

7 DISCUSSION AND FUTURE WORK

Precomputation. While we do not perform any precomputation, it could be incorporated by e.g. pre-training a good initial state of the neural network or by utilizing a low-overhead meta-learning technique [Hospedales et al. 2020]. In our design, we consider precomputation as strictly optional and merely to enhance the performance when possible. In fact, as the neural radiance cache rapidly learns the current situation (8 frames are sufficient, see Figure 4), precomputation can be only of limited benefit. Still, it is of interest to explore the utility of static sets of neural network weights and to determine the bounds of the domain of validity of a fixed set of neural network weights.

Cache artifacts. While we were able to suppress high-frequency temporal flickering using an exponential moving average over the optimized network weights, subtle low-frequency scintillation remains. Additionally, the frequency encoding causes distracting axis-aligned oscillations throughout space. These artifacts are imperceptible when using the neural radiance cache at non-primary path vertices, but in some use cases (e.g. when noise is not an option), using the cache at the primary vertex would be desirable. To this end future work is needed to stabilize the prediction in a visually pleasing manner.

Additional network inputs. Input encodings alone are not sufficient to learn a detailed, high-frequency representation of features in the scattered radiance that correlate poorly with all of the network inputs. Examples of such features are shadows and caustics, which are unrelated to the local surface attributes that we can easily pass to the network. Shadows and caustics are therefore learned at a much slower rate—or not at all, if the network is too small or the radiance estimates are too noisy. It is thus very interesting to think about additional, simple-to-compute network inputs that correlate well with such features.

Offline rendering. While our neural cache design focuses on real-time rendering, we believe its use of self-training would be beneficial in offline scenarios. Indeed, it allows to capture high-order indirect illumination without the costly tracing and shading of long paths; this could be particularly useful to tackle path-length limitations of batch rendering [Burley et al. 2018].

Volumes. We note that our neural cache parameterization is not tied to a surface representation and can thus also be used in volumetric rendering. A straightforward implementation yields promising results (see Figure 14), but an in-depth investigation needs to be carried out.

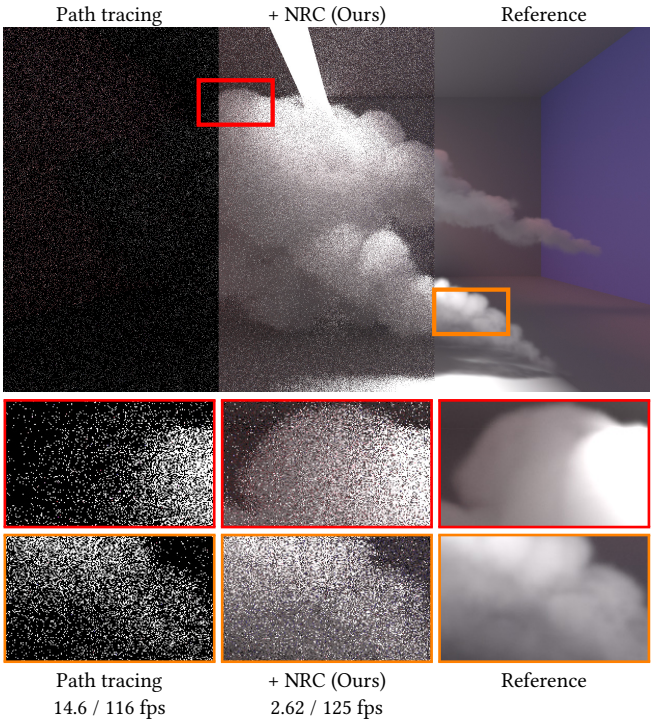


Fig. 14. Neural radiance caching also works in volumetric rendering. We render at 1 spp and report MRSE (left number) and frames per second. Illumination enters the room through a narrow slit in the ceiling and the volume has an isotropic phase function. There is one single neural radiance cache for the scene. For volume queries, undefined parameters (surface roughness, normal, albedo, and specular coefficients) are simply set to default constants.

Path guiding. The main source of noise in our results is the indirect use of the cache. While the indirection results in decreased bias, it also leads to increased variance due to the (hemi-)spherical sampling, even though the cache approximation itself is noise-free. One could consider bringing neural importance sampling [Müller et al. 2019] to real-time applications; as with our neural radiance cache, the cost appears prohibitive at first, yet applying similar principles may prove successful.

Improved path termination. More accurate approximations of the anisotropic area spreads, such as covariance tracing [Belcour et al. 2013] or bundle coherence [Meng et al. 2015], could be used. While we did not encounter specific failure modes of the isotropic approximation of Bekaert et al., scenes with strong anisotropic lighting effects would likely benefit from the aforementioned methods.

An additional challenge that goes beyond the choice of area-spread approximation is the lack of path termination in long, branching, specular chains of interactions. Consequently, our cache currently provides little benefit when transport is dominated by dielectric materials such as glass, as seen in Figure 15. An improvement of the termination heuristic in such cases, as well as a more accurate cache to resolve sharp specular details, is of high interest.

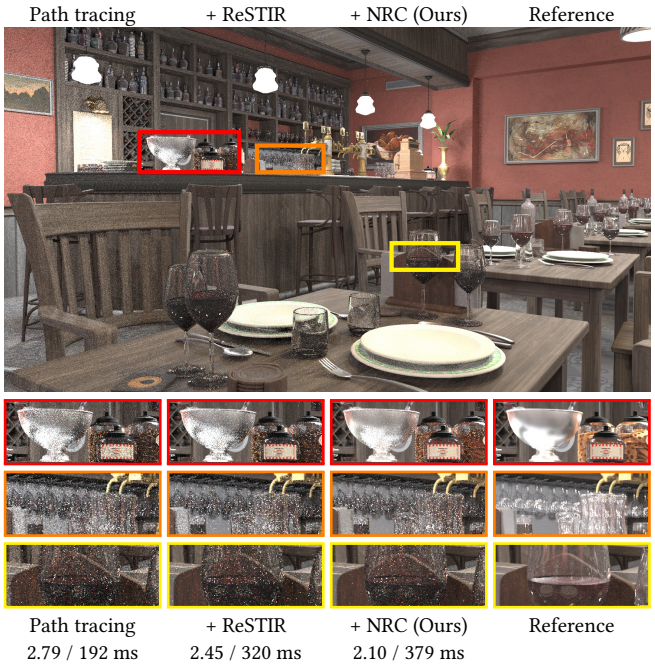


Fig. 15. 32 spp rendering of the Bistro scene, of which we report MRSE (left number) and render time. The improvement by neural radiance caching is marginal, because it does not address complex, branching specular transport. To alleviate this, we believe that a combination with path guiding as well as an improved path termination criterion should be investigated in the future.

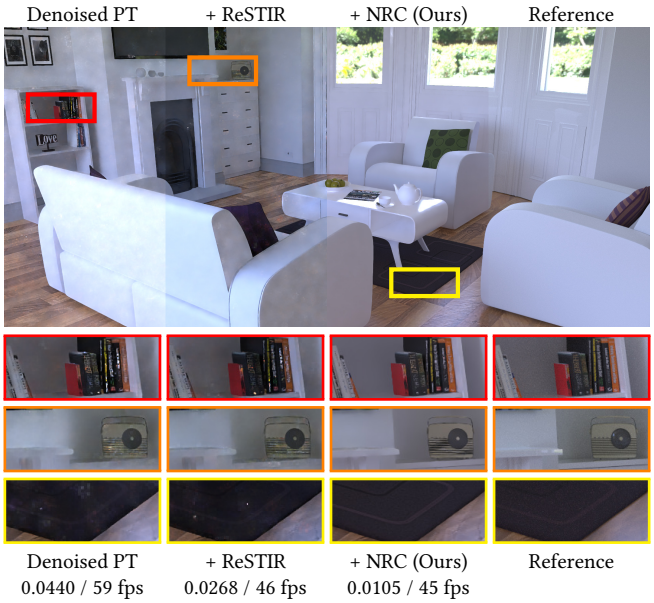


Fig. 16. The Living Room scene from the teaser image at 1 spp, passed through a deep-learning based real-time denoiser [Hasselgren et al. 2020]. Despite being trained on datasets with noise characteristics much different from our algorithm, the denoiser produces the cleanest results on it, achieving the lowest relative squared bias (left number). Even with denoising enabled, the framerate of our method stays well above 30.

Denoising. Our neural radiance cache could be considered a path-space denoiser, as it effectively performs a regression over spatio-temporal samples to produce noise-free approximations. In Figure 16, we demonstrate that this path-space denoising *complements* existing screen-space techniques. This preliminary experiment employed an off-the-shelf denoiser that was trained on data not representative of our method, so we expect a tighter coupling of our cache and the denoiser to offer potential for further improvements.

8 CONCLUSION

We have introduced a real-time neural radiance caching technique for path-traced global illumination. It can handle dynamic content while providing predictable performance and resource consumption, which is enabled by our fully fused neural networks that achieve *generalization via online adaptation*. While the necessary performance requires a lot of engineering, robustness comes as a collateral. The resulting high rendering quality makes up for the cost of the neural radiance cache, and could be further improved through orthogonal variance reduction techniques such as by path guiding.

The neural radiance cache is a rather different approach to real-time rendering than previous techniques. It could be characterized as wasteful in terms of *compute*—some neurons have little impact on the output, yet their contribution is still evaluated. Competing techniques with sophisticated data structures could be characterized as wasteful in terms of *memory*—the memory is never used in its entirety as queries access only small (random) neighborhoods.

The neural radiance cache employs fixed function hardware (the GPU tensor cores), and heavily relies on cheap computation, instead of costly memory accesses. This efficiency is reflected in the timings of Table 3, where our neural approach is only twice as expensive as irradiance probes, despite requiring a comparatively massive amount of compute (both implementations are reasonably well optimized). We posit this compute efficiency is the key ingredient of the neural cache robustness. This paradigm—cheap compute—appears to be worth investigating [Dally et al. 2020], and we hope it inspires further experiments in applications where compute is typically considered a precious commodity.

ACKNOWLEDGMENTS

The article is dedicated to our dear friend and colleague Jaroslav Křivánek. The authors thank Nikolaus Binder for his early contributions to the high performance neural networks. Zander Majercik assisted us for the DDGI comparison, and Simon Kallweit contributed to the Falcor integration.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/

Michael Abrash. 1997. Quake's lighting model: Surface caching. In *Graphics Programming Black Book*. Coriolis Group, Chapter 68, 1245–1256.

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2, Article 15 (Aug. 2020), 23 pages. https://doi.org/10.1145/3406183

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. 2020. Second Order Optimization Made Practical. *arXiv:2002.09018* (Feb. 2020).

James Arvo. 1986. Backward Ray Tracing. In *In ACM SIGGRAPH '86 Course Notes - Developments in Ray Tracing*. 259–263.

Colin Barré-Brisebois. 2017. A Certain Slant of Light: Past, Present and Future Challenges of Global Illumination in Games. In *Open problems in real-time rendering*. ACM SIGGRAPH 2017 Courses.

Philippe Bekaert, Philipp Slusallek, Ronald Cools, Vlastimil Havran, and Hans-Peter Seidel. 2003. *A custom designed Density Estimation Method for Light Transport*. Technical Report. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Laurent Belcour, Cyril Soler, Kartic Subr, Nicolas Holzschuch, and Fredo Durand. 2013. 5D Covariance Tracing for Efficient Defocus and Motion Blur. *ACM Trans. Graph.* 32, 3, Article Article 31 (July 2013), 18 pages. https://doi.org/10.1145/2487228.2487239

Nir Benty, Kai-Hwa Yao, Petrik Clarberg, Lucy Chen, Simon Kallweit, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. 2020. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.

Nikolaus Binder, Sascha Fricke, and Alexander Keller. 2018. Fast Path Space Filtering by Jittered Spatial Hashing. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 71, 2 pages. https://doi.org/10.1145/3214745.3214806

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (July 2020). https://doi.org/10/gg8xc7

Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Daniel Teece. 2018. The Design and Evolution of Disney's Hyperion Renderer. *ACM Trans. Graph.* 37, 3, Article 33 (July 2018), 22 pages. https://doi.org/10.1145/3182159

Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum* 30, 7 (2011), 1921–1930. https://doi.org/10.1111/j.1467-8659.2011.02063.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02063.x

Carsten Dachsbacher and Marc Stamminger. 2005. Reflective Shadow Maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games (I3D '05)*. Association for Computing Machinery, New York, NY, USA, 203–231. https://doi.org/10.1145/1053427.1053460

Ken Dahm and Alexander Keller. 2018. Learning Light Transport the Reinforced Way. In *Monte Carlo and Quasi-Monte Carlo Methods*, Art B. Owen and Peter W. Glynn (Eds.). Springer International Publishing, 181–195.

William J. Dally, Yatish Turakhia, and Song Han. 2020. Domain-Specific Hardware Accelerators. *Commun. ACM* 63, 7 (June 2020), 48–57. https://doi.org/10.1145/3361682

Johannes Deligiannis and Jan Schmid. 2019. "It Just Works" Ray-Traced Reflections in 'Battlefield V'. In *Game Developers Conference*.

Addis Dittebrandt, Johannes Hanika, and Carsten Dachsbacher. 2020. Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing. In *Eurographics Symposium on Rendering - DL-only Track*, Carsten Dachsbacher and Matt Pharr (Eds.). The Eurographics Association. https://doi.org/10.2312/sr.20201135

Renaud Adrien Dubouchet, Laurent Belcour, and Derek Nowrouzezahrai. 2017. Frequency Based Radiance Cache for Rendering Animations. In *Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations (EGSR '17)*. Eurographics Association, Goslar, DEU, 41–53. https://doi.org/10.2312/sre.20171193

Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, and Jonathan M. Cohen. 2019. Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. *arXiv:1905.11286* (June 2019).

Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. 1998. The irradiance volume. *IEEE Computer Graphics and Applications* 18, 2 (1998), 32–43.

Vineet Gupta, Tomer Koren, and Yoram Singer. 2018. Shampoo: Preconditioned Stochastic Tensor Optimization. *arXiv:1802.09568* (Aug. 2018).

Jon Hasselgren, Jacob Munkberg, Marco Salvi, Anjul Patney, and Aaron Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. *Computer Graphics Forum* 39, 2 (2020), 147–155. https://doi.org/10.1111/cgf.13919 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13919

Paul S. Heckbert. 1990. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 145–154. https://doi.org/10.1145/97880.97895

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik P. A. Lensch. 2018. A Unified Manifold Framework for Efficient BRDF Sampling based on Parametric Mixture Models. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, Wenzel Jakob and Toshiya Hachisuka (Eds.). The Eurographics Association.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* (2016). https://doi.org/10.1111/cgf.12950

Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. 2019. Deep-learning the Latent Space of Light Transport. *Computer Graphics Forum* 38, 4 (2019).

John T. Hooker. 2016. Volumetric Global Illumination at Treyarch. In *Advances in real-time rendering, part I*. ACM SIGGRAPH 2016 Courses.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2020. Meta-Learning in Neural Networks: A Survey. *arXiv:2004.05439* (April 2020).

Michał Iwanicki and Peter-Pike Sloan. 2017. Precomputed Lighting in Call of Duty: Infinite Warfare. In *Advances in Real-Time Rendering, Part I (ACM SIGGRAPH 2017 Courses)*. Association for Computing Machinery, New York, NY, USA, Article 7a, 1 pages. https://doi.org/10.1145/3084873.3096476

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv:1803.05407* (March 2018).

Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. 2008. Radiance Caching for Participating Media. *ACM Trans. Graph.* 27, 1, Article 7 (March 2008), 11 pages. https://doi.org/10.1145/1330511.1330518

Henrik Wann Jensen. 1995. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques*. Springer Vienna, Vienna, 326–335. https://doi.org/10.1007/978-3-7091-9430-0_31

Henrik Wann Jensen. 1996. Global Illumination using Photon Maps. In *Rendering Techniques '96*, Xavier Pueyo and Peter Schröder (Eds.). Springer Vienna, Vienna, 21–30.

Giulio Jiang and Bernhard Kainz. 2021. Deep radiance caching: Convolutional autoencoders deeper in ray tracing. *Computers & Graphics* 94 (2021), 22 – 31. https://doi.org/10.1016/j.cag.2020.09.007

James T. Kajiya. 1986. The Rendering Equation. *Computer Graphics* 20 (1986), 143–150.

Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. 2017. Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks. *ACM Trans. Graph.* 36, 6, Article 231 (Nov. 2017), 11 pages. https://doi.org/10.1145/3130800.3130880

Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)*. Association for Computing Machinery, New York, NY, USA, 99–107. https://doi.org/10.1145/1730804.1730821

Alexander Keller. 1997. Instant Radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 49–56. https://doi.org/10.1145/258734.258769

Alexander Keller and Ken Dahm. 2019. Integral Equations and Machine Learning. *Mathematics and Computers in Simulation* 161 (2019), 2–12.

Alexander Keller, Timo Viitanen, Colin Barré-Brisebois, Christoph Schied, and Morgan McGuire. 2019. Are We Done with Ray Tracing?. In *ACM SIGGRAPH 2019 Courses (SIGGRAPH '19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 381 pages. https://doi.org/10.1145/3305366.3329896

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (June 2014).

Jaroslav Křivánek and Pascal Gautron. 2009. *Practical Global Illumination with Irradiance Caching*. Morgan & Claypool.

Jaroslav Křivánek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. 2005. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics* 11, 5 (2005), 550–561.

Eric P. Lafortune and Yves D. Willems. 1995. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In *Proc. EGWR*. 11–20.

Sébastien Lagarde and Charles de Rousiers. 2014. Moving Frostbite to Physically Based Rendering 3.0. In *ACM SIGGRAPH Courses: Physically Based Shading in Theory and Practice, Chapter 10 (SIGGRAPH '14)*. ACM, New York, NY, USA. https://doi.org/10.1145/2614028.2615431

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. 2018. Noise2Noise: Learning Image Restoration without Clean Data. *arXiv:1803.04189* (March 2018).

Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT)* 8, 2 (5 June 2019), 1–30. http://jcgt.org/published/0008/02/01/

Julio Marco, Adrian Jarabo, Wojciech Jarosz, and Diego Gutierrez. 2018. Second-Order Occlusion-Aware Volumetric Radiance Caching. *ACM Trans. Graph.* 37, 2, Article 20 (July 2018), 14 pages. https://doi.org/10.1145/3185225

Sam Martin and Per Einarsson. 2010. A Real-Time Radiosity Architecture for Video Games. SIGGRAPH 2010 Course: Advances in Real-Time Rendering in 3D Graphics and Games.

Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. 2017. Real-Time Global Illumination Using Precomputed Light Field Probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. https://doi.org/10.1145/3023368.3023378

Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. 2015. Multi-Scale Modeling and Rendering of Granular Materials. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 34, 4 (July 2015). https://doi.org/10/gfzndr

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Martin Mittring. 2007. Finding next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses (SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 97–121. https://doi.org/10.1145/1281500.1281671

Pierre Moreau, Matt Pharr, and Petrik Clarberg. 2019. Dynamic Many-Light Sampling for Real-Time Ray Tracing. In *High-Performance Graphics 2019 - Short Papers, Strasbourg, France, July 8-10, 2019*, Markus Steinberger and Tim Foley (Eds.). Eurographics Association, 21–26. https://doi.org/10.2312/hpg.20191191

Thomas Müller. 2021. Tiny CUDA Neural Network Framework. https://github.com/nvlabs/tiny-cuda-nn.

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum* 36, 4 (June 2017), 91–100. https://doi.org/10.1111/cgf.13227

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (Oct. 2019), 19 pages. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural Control Variates. *ACM Trans. Graph.* 39, 6, Article 243 (Nov. 2020), 19 pages. https://doi.org/10.1145/3414685.3417804

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen-Space Shading. 36, 4 (2017).

Christopher Oat. 2005. Irradiance Volumes for Games. In *Game Developers Conference*.

Jacopo Pantaleoni. 2020. Online path sampling control with progressive spatio-temporal filtering. *arXiv:2005.07547* (May 2020).

Hauke Rehfeld, Tobias Zirr, and Carsten Dachsbacher. 2014. Clustered Pre-Convolved Radiance Caching. In *Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization (PGV '14)*. Eurographics Association, Goslar, DEU, 25–32.

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global Illumination with Radiance Regression Functions. *ACM Trans. Graph.* 32, 4, Article 130 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2462009

Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. 2012. The State of the Art in Interactive Global Illumination. *Computer Graphics Forum* 31, 1 (2012), 160–188. https://doi.org/10.1111/j.1467-8659.2012.02093.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.02093.x

Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. 2009a. Micro-Rendering for Scalable, Parallel Final Gathering. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–8. https://doi.org/10.1145/1618452.1618478

Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph.* 27, 5, Article 129 (Dec. 2008), 8 pages. https://doi.org/10.1145/1409060.1409082

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009b. Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. Association for Computing Machinery, New York, NY, USA, 75–82. https://doi.org/10.1145/1507149.1507161

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive Sampling and Reconstruction Using Greedy Error Minimization. *ACM Trans. Graph.* 30, 6, Article 159 (Dec. 2011), 12 pages. https://doi.org/10.1145/2070781.2024193

Daniel Scherzer, Chuong H. Nguyen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Pre-Convolved Radiance Caching. *Comput. Graph. Forum* 31, 4 (June 2012), 1391–1397. https://doi.org/10.1111/j.1467-8659.2012.03134.x

Dario Seyb, Peter-Pike Sloan, Ari Silvennoinen, Michał Iwanicki, and Wojciech Jarosz. 2020. The design and evolution of the UberBake light baking system. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (July 2020).

Ari Silvennoinen and Jaakko Lehtinen. 2017. Real-Time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes. *ACM Trans. Graph.* 36, 6, Article 230 (Nov. 2017), 13 pages. https://doi.org/10.1145/3130800.3130852

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. 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 (SIGGRAPH '02)*. Association for Computing Machinery, New York, NY, USA, 527–536. https://doi.org/10.1145/566570.566612

Tiago Sousa, Nickolay Kasyan, and Nicolas Schulz. 2011. Secrets of CryENGINE 3 Graphics Technology. In *Advances in Real-Time Rendering in Games: Part I*. ACM SIGGRAPH 2011 Courses.

Eric Tabellion and Arnauld Lamorlette. 2004. An Approximate Global Illumination System for Computer Generated Films. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 469–476. https://doi.org/10.1145/1186562.1015748

Sergios Theodoridis. 2008. *Pattern Recognition*. Elsevier.

Kostas Vardis, Georgios Papaioannou, and Anastasios Gkaravelis. 2014. Real-time Radiance Caching using Chrominance Compression. *Journal of Computer Graphics Techniques (JCGT)* 3, 4 (16 December 2014), 111–131. http://jcgt.org/published/0003/04/06/

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv:1706.03762* (June 2017).

Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proc. SIGGRAPH*. 419–428. https://doi.org/10.1145/218380.218498

Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. 2019. Path Guiding in Production. In *ACM SIGGRAPH Courses*. ACM, New York, NY, USA, 18:1–18:77. https://doi.org/10.1145/3305366.3328091

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. Graph.* 33, 4 (Aug. 2014).

Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. 1988. A Ray Tracing Solution for Diffuse Interreflection. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. Association for Computing Machinery, New York, NY, USA, 85–92. https://doi.org/10.1145/54852.378490

Yangyang Zhao, Laurent Belcour, and Derek Nowrouzezahrai. 2019. View-Dependent Radiance Caching. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019 (GI'19)*. Canadian Human-Computer Communications Society, Waterloo, CAN, Article 22, 9 pages. https://doi.org/10.20380/GI2019.22