

NRRS: Neural Russian Roulette and Splitting

Haojie Jin, Jierui Ren, Yisong Chen, Guoping Wang*, and Sheng Li*, *Member, IEEE*

Abstract—We propose a novel framework for Russian Roulette and Splitting (RRS) tailored to wavefront path tracing, a highly parallel rendering architecture that processes path states in batched, stage-wise execution for efficient GPU utilization. Traditional RRS methods, with unpredictable path counts, are fundamentally incompatible with wavefront’s preallocated memory and scheduling requirements. To resolve this, we introduce a normalized RRS formulation with a bounded path count, enabling stable and memory-efficient execution. Furthermore, we pioneer the use of neural networks to learn RRS factors, presenting two models: NRRS and AID-NRRS. At a high level, both feature a carefully designed RRSNet that explicitly incorporates RRS normalization, with only subtle differences in their implementation. To balance computational cost and inference accuracy, we introduce Mix-Depth, a path-depth-aware mechanism that adaptively regulates neural evaluation, further improving efficiency. Extensive experiments demonstrate that our method outperforms traditional heuristics and recent RRS techniques in both rendering quality and performance across a variety of complex scenes.

Index Terms—Global Illumination, Neural Networks, Russian Roulette and Splitting, Wavefront architecture

I. INTRODUCTION

PATH tracing is a widely adopted Monte Carlo method that approximates the rendering equation [1] by stochastically tracing light transport paths from the camera into the scene. While path tracing is physically accurate, its convergence is slow due to high variance, especially in scenes with complex indirect lighting situations. There are several approaches to improving the efficiency of unidirectional path tracing, including path guiding, multiple importance sampling, and Russian roulette and splitting (RRS) [2]. In this paper, we primarily focus on RRS. RRS consists of two components: Russian roulette (RR) and splitting. RR aims to increase efficiency by terminating paths with low contributions, thereby reducing computational cost. Splitting seeks to improve efficiency by generating more paths with high contributions, which helps to reduce variance.

The wavefront architecture [3] demonstrates higher efficiency than the megakernel architecture on GPUs, as it decomposes large kernels into smaller ones, thereby effectively reducing divergence during task execution and increasing parallelism. However, traditional RRS methods are not suitable for wavefront path tracing on GPUs. In the wavefront context, memory for work queues must be pre-allocated. If the path

tracing algorithm does not perform splitting, it is sufficient to pre-allocate the work queues based on the number of pixels. However, when applying RRS, the maximal paths to be generated are highly uncertain, making static pre-allocation challenging. To address this issue, we propose a new RRS framework for wavefront path tracing. In short, we normalize the RRS factors and control the number of emitted rays along with their associated paths within an upper bound. Under this framework, the probability of overflow is theoretically bounded, and the resulting overflow issue can be empirically negligible in practice, making the method practical for wavefront-based scheduling.

Under this new RRS framework, the assumptions of state-of-the-art RRS methods such as EARS [4] are no longer valid. EARS employs separate evaluation branches for RR and splitting, but normalization may cross the boundary between them. Moreover, the normalization factor is not constant, which cannot be easily incorporated into its theoretical analysis.

Recently, neural networks have demonstrated strong performance in various rendering tasks, such as neural radiance caching [5] and path guiding [6], [7]. Therefore, we leverage neural networks to learn the RRS factors. Unlike previous methods, which cannot be easily adapted to our new framework, our approach allows for careful loss function design to ensure compatibility. We propose two neural RRS methods, NRRS and AID-NRRS. To our knowledge, this is the first neural method for estimating RRS factors. Since neural networks introduce additional computational overhead, their benefits may not always justify the cost. Inspired by the common heuristic of disabling RR at early bounces, we propose Mix-Depth to integrate alternative RRS strategies into NRRS and AID-NRRS for improved efficiency.

To summarize, our key contributions include:

- We propose a RRS framework for wavefront path tracing, making RRS both efficient and practical in the wavefront context.
- Building on this framework, we introduce two neural RRS models, NRRS and AID-NRRS. To the best of our knowledge, this is the first use of neural networks for estimating RRS factors with high efficiency.
- We present Mix-Depth, a general method for combining multiple RRS approaches to further enhance the efficiency of NRRS and AID-NRRS, called NRRS+ and AID-NRRS+ respectively. Mix-Depth is broadly applicable to any RRS method that shares the same cache structure, incurs minimal overhead, and is not limited to our neural approaches.

Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li are with the School of Computer Science, Peking University, China. E-mail: {jhj, chenysisong, wgp, lisheng}@pku.edu.cn.

Guoping Wang and Sheng Li are also with the National Key Laboratory of Intelligent Parallel Technology.

Jierui Ren is with the College of Future Technology, Peking University, China. E-mail: jerry@stu.pku.edu.cn.

Guoping Wang and Sheng Li are the corresponding authors.

II. RELATED WORK

A. Unidirectional Path Tracing

Unidirectional path tracing (UPT) is a Monte Carlo-based method widely used for generating photorealistic images due to its simplicity [1]. Rays are cast from the camera into the scene and continue to bounce within the environment until they either intersect a light source, exit the scene, or exceed a predefined bounce limit.

Thereafter, numerous unbiased improvements have been proposed to enhance its efficiency, including path guiding (PG) [6]–[12], RRS [2], [4], [13], control variates (CV) [14], [15], and multiple importance sampling (MIS) [16]–[18], among others. They orthogonally improve the efficiency of UPT. PG seeks to learn a better sampling distribution for generating new rays, improving the efficiency of Monte Carlo integration. MIS aims to combine multiple sampling techniques to reduce variance optimally. CV constructs an auxiliary distribution such that subtracting it from the original yields a smoother, lower-variance residual that is easier to estimate. RRS adaptively allocates computational resources among paths to maximize overall efficiency. Our work focuses on RRS but can readily be combined with PG, CV, and MIS to further enhance efficiency.

B. Russian Roulette and Splitting

RRS consists of two techniques: Russian roulette (RR) and splitting. It was first introduced into path tracing in 1990 [19]. RR terminates a path with a certain probability $1 - p_{\text{rr}}$ ($0 < p_{\text{rr}} \leq 1$), allowing support for infinite path lengths. In contrast, splitting emits n_s ($n_s \geq 1$) rays to estimate the local radiance. Typically, these probabilities are set heuristically. Throughput-based RR sets p_{rr} equal to the current prefix path's throughput, which is the default setting in many renderers such as Mitsuba [20] and PBRT [21]. Szirmay-Kalos et al. [22] analyzed the variance under a constraint on the total number of rays, and set p_{rr} and n_s according to both the prefix path and global properties. However, their approximation causes the method to degenerate into relying solely on the prefix path.

ADRRS [2] combines RR and splitting into a unified framework. It determines the number of rays to generate based on a real-valued parameter q ($0 \leq q < \infty$), which we refer to as the RRS factor in this paper. ADRRS also provides a method for computing q , defining it as the ratio of the estimated contribution of the entire path starting from the current prefix path to the estimated contribution of the pixel that the current prefix path contributes to. EARS [4] employs a fixed-point iteration scheme to generate q with the goal of maximizing efficiency, based on the theoretical analysis of Bolin and Meyer [23]. In EARS, q does not distinguish between BSDF rays and NEE rays. MARS [13] extends EARS to address this limitation. However, these RRS methods are all implemented in Megakernel-style path tracers and do not take the Wavefront architecture into consideration.

RRS has also been applied in other rendering algorithms, such as bidirectional path tracing (BDPT). Veach [24] uses RR to reduce the number of visibility tests during vertex connection. Su et al. [25] employ RRS to maximize efficiency

in reciprocal estimation. Meyer et al. [13] also utilize RRS to determine the number of samples for path tracing, light path connections, and next event estimation (NEE).

C. Neural Methods

With rapid development of deep learning, neural methods have been introduced into the context of path tracing. These approaches can be divided into two categories: *Final Image Prediction* and *Intermediate Quantity Prediction*. The former directly outputs the final image, while the latter estimates intermediate quantities such as radiance, hit point, or visibility during the path tracing process. Our method falls into the latter category.

a) *Final Image Prediction*: One of the most popular directions is denoising [26]–[29], where a neural network produces a noise-free image from a noisy input and auxiliary buffers. As for Global illumination, Xin et al. [30] also use convolutional neural networks (CNNs) to predict single-bounce diffuse indirect illumination using direct illumination and auxiliary buffers as inputs.

b) *Intermediate Quantity Prediction*: NRC [5] and NIRC [31] employ a simple multi-layer perceptron (MLP) to predict local radiance. Under certain conditions, the network is queried for local radiance instead of continuing path tracing. The key distinction is that NRC predicts the outgoing radiance at the hit point, whereas NIRC predicts the incoming radiance along a specified direction. Similarly, Fujieda et al. [32] use an MLP to predict visibility for calculating shadows from direct illumination. Weier et al. [33] use an MLP to predict the intersection point and its normal; combined with scene-specific material parameters, this enables ray tracing to proceed using the network's predictions. In the context of path guiding, Müller et al. [9] employ normalizing flows to sample incoming directions; Dong et al. [6], Huang et al. [7] use a small MLP to predict the parameters of the sampling distribution; Figueiredo et al. [12] decompose the 2D sampling distribution into two 1D marginal distributions and use a small MLP to output an array representing each 1D distribution; and Rath et al. [11] combine the theoretical frameworks of EARS and resampling, leveraging a neural network to cache radiance information and thus enable an efficient path guiding algorithm.

III. PRELIMINARY

a) *Rendering Equation*: The core of solving the light transport problem is the rendering equation [1] as:

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\Omega} L_i(\mathbf{p}, \omega_i) f_r(\mathbf{p}, \omega_i, \omega_o) |\cos \theta_i| d\omega_i, \quad (1)$$

which describes the relationship between the radiance L_o leaving a point $\mathbf{p}$ in direction ω_o and the incoming radiance L_i arriving at $\mathbf{p}$ from direction ω_i . Here, f_r is the bidirectional scattering distribution function (BSDF) that describes how light is scattered at point $\mathbf{p}$, and θ_i is the angle between the incoming direction ω_i and the normal vector at $\mathbf{p}$. $L_e(\mathbf{p}, \omega_o)$ denotes the radiance emitted from the point $\mathbf{p}$ in the direction ω_o .

b) *Monte Carlo Integration*: To solve the rendering equation, Monte Carlo (MC) integration is used to approximate the integral in Eq. (1). The basic idea is to sample N incoming directions ω_i according to a probability density function (PDF) $p(\omega_i | \omega_o, \mathbf{p})$, and then estimate the integral by averaging the sampled values. Monte Carlo integration for the rendering equation is given by

$$\langle L_o(\mathbf{p}, \omega_o) \rangle = L_e(\mathbf{p}, \omega_o) + \frac{1}{N} \sum_{i=1}^N \frac{L_i(\mathbf{p}, \omega_i^i) f_r(\mathbf{p}, \omega_i^i, \omega_o) |\cos \theta_i^i|}{p(\omega_i^i | \omega_o, \mathbf{p})}. \quad (2)$$

c) *Unidirectional Path Tracing*: Unidirectional path tracing (UPT) solves the rendering equation by tracing paths from the camera to the light source. Each time a ray intersects a scene surface, Monte Carlo integration is performed to estimate the local radiance. This recursive process is repeated until the ray reaches a light source or a maximum number of bounces is reached. Before performing Monte Carlo integration, RRS is used to determine how many samples should be taken.

d) *Unified RRS Framework*: ADRRS provides a unified framework for RRS. Given an RRS factor q , the number of samples N_q is determined as shown in Eq. (3). During the UPT process, the RRS factor q specifies how many samples should be taken before applying Monte Carlo integration at each intersection point.

$$N_q = \begin{cases} \lfloor q \rfloor + 1 & \text{with the probability } q - \lfloor q \rfloor \\ \lfloor q \rfloor & \text{otherwise.} \end{cases} \quad (3)$$

e) *Prefix and Suffix Paths*: Following the analysis of EARS [4], we can divide the entire path $\mathbf{z}$ generated during the UPT process into two parts at the point $\mathbf{p}$ where Monte Carlo integration is performed. The first part is the prefix path $\mathbf{x}$, which extends from the camera to the point $\mathbf{p}$. The second part is the suffix path $\mathbf{y}$, which continues from $\mathbf{p}$ to the light source or infinity if the ray does not intersect with the scene (see Fig. 1). This decomposition allows the rendering equation to be reformulated as:

$$I_{\text{px}} = \int_{\mathcal{Z}} W_{\text{px}}(\mathbf{z}) f(\mathbf{z}) d\mu(\mathbf{z}) = \int_{\mathcal{X}} g(\mathbf{x}) H(\mathbf{x}) d\mathbf{x} \quad (4)$$

where $H(\mathbf{x}) = \int_{\mathcal{Y}} h(\mathbf{x}, \mathbf{y}) d\mathbf{y}$.

Here, I_{px} denotes the value of pixel px , and W_{px} represents the camera response function for pixel px . The term f denotes the cumulative product of throughput (including BSDF and cosine term as defined in Eq. (1)) along the path $\mathbf{z}$, while g and h denote the cumulative product of throughput terms along the prefix path $\mathbf{x}$ (excluding $\mathbf{p}$) and the suffix path $\mathbf{y}$ (including $\mathbf{p}$), respectively. For convenience, W_{px} is absorbed into g . $\mathcal{Z}$ is defined as the set of all possible paths, $\mathcal{X}$ is defined as the set of prefix paths ending at the same point $\mathbf{p}$, and $\mathcal{Y}$ as the set of suffix paths starting from $\mathbf{p}$. Notably, this is only an intuitive illustration like EARS [4]. For a more formal derivation, please refer to the appendix.

Under the UPT framework, the MC estimation of Eq. (4) is presented in Eq. (5). Here, $p(\mathbf{x})$ denotes the probability of sampling the prefix path $\mathbf{x}$, $\langle H(\mathbf{x}) \rangle$ denotes the MC estimator

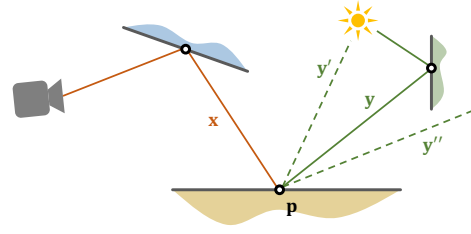


Fig. 1. **Prefix and Suffix Paths**. The entire path $\mathbf{z}$ is divided into two parts at point $\mathbf{p}$. The prefix path $\mathbf{x}$ extends from the camera to point $\mathbf{p}$. And the suffix path $\mathbf{y}$ continues from $\mathbf{p}$ to the light source or infinity.

of $H(\mathbf{x})$, and $p(\mathbf{y}_j | \mathbf{x})$ represents the probability of sampling the suffix path $\mathbf{y}_j$ given the prefix path $\mathbf{x}$. The term $q(\mathbf{x})$ corresponds to the RRS factor.

$$\langle I_{\text{px}} \rangle = \frac{g(\mathbf{x})}{p(\mathbf{x})} \langle H(\mathbf{x}) \rangle = \frac{g(\mathbf{x})}{p(\mathbf{x})} \sum_{j=1}^{N_{q(\mathbf{x})}} \frac{h(\mathbf{x}, \mathbf{y}_j)}{q(\mathbf{x}) p(\mathbf{y}_j | \mathbf{x})}. \quad (5)$$

f) *Variance Analysis*: The variance analysis of RRS is provided by Bolin and Meyer [23], and EARS [4] adopts the same methodology. However, there are subtle differences between RR and splitting in how they affect variance. Here, we present only the final results; for detailed derivations, please refer to EARS [4].

For splitting, the relationship between the variance of the suffix path $\mathbf{y}$ and the variance of the entire path $\mathbf{z}$ is given by Eq. (6). Here, $n_s(\mathbf{x})$ denotes the number of samples for suffix paths $\mathbf{y}$ (i.e., the splitting factor), and $p(\mathbf{x})$ is the PDF of the prefix path $\mathbf{x}$. The derivative of $\mathbb{V}[\langle I_{\text{px}} \rangle]$ with respect to $n_s(\mathbf{x})$ is shown in Eq. (7). These expressions treat $n_s(\mathbf{x})$ as a real number and ignore the variance introduced by stochastic rounding.

$$\mathbb{V}[\langle I_{\text{px}} \rangle] = \mathbb{V} \left[\frac{g(\mathbf{x})}{p(\mathbf{x})} H(\mathbf{x}) \right] + \mathbb{E} \left[\left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \frac{\mathbb{V}[\langle H(\mathbf{x}) \rangle]}{n_s(\mathbf{x})} \right]. \quad (6)$$

$$\frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dn_s(\mathbf{x})} = -p(\mathbf{x}) \left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \frac{\mathbb{V}[\langle H(\mathbf{x}) \rangle]}{n_s^2(\mathbf{x})}. \quad (7)$$

For RR, the relationship between the variance of the suffix path $\mathbf{y}$ and the variance of the whole path $\mathbf{z}$ is given by Eq. (8), where $p_{\text{rr}}(\mathbf{x})$ is the RR factor.

$$\mathbb{V}[\langle I_{\text{px}} \rangle] = \mathbb{V} \left[\frac{g(\mathbf{x})}{p(\mathbf{x})} H(\mathbf{x}) \right] + \mathbb{E} \left[\left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \left(\frac{\mathbb{E}[\langle H(\mathbf{x}) \rangle^2]}{p_{\text{rr}}(\mathbf{x})} - H^2(\mathbf{x}) \right) \right]. \quad (8)$$

$$\frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dp_{\text{rr}}(\mathbf{x})} = -p(\mathbf{x}) \left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \frac{\mathbb{E}[\langle H(\mathbf{x}) \rangle^2]}{p_{\text{rr}}^2(\mathbf{x})}. \quad (9)$$

g) *Wavefront Architecture*: Laine et al. [3] proposed the wavefront architecture to improve the efficiency of UPT on GPUs. In contrast to the traditional megakernel architecture, which performs all operations for a ray within a single kernel, the wavefront architecture separates the computation of a ray into several stages, each implemented by a different kernel. This separation reduces thread divergence within each kernel and thus improves performance. The wavefront architecture is illustrated in Fig. 2. In this framework, camera rays intersect the scene and are dispatched to four work queues. Rays hitting surfaces spawn new rays based on RRS factors, while those hitting lights or missing geometry terminate. The process repeats until the maximum path depth is reached. In this example, next event estimation (NEE) is disabled, and a maximum path depth is enforced.

Work queues must be pre-allocated under the wavefront architecture. In traditional UPT, the RRS module typically only performs RR, so the size of the work queues does not increase. However, once splitting is applied, the maximum number of paths to be generated becomes unknown, making pre-allocation infeasible. This unpredictability also leads to uncontrollable memory usage and scheduling challenges. To address this problem, we propose a new RRS framework, which will be discussed in Sec. IV-A. At a high level, we perform normalization at each depth to ensure that the number of rays remains within a specified limit.

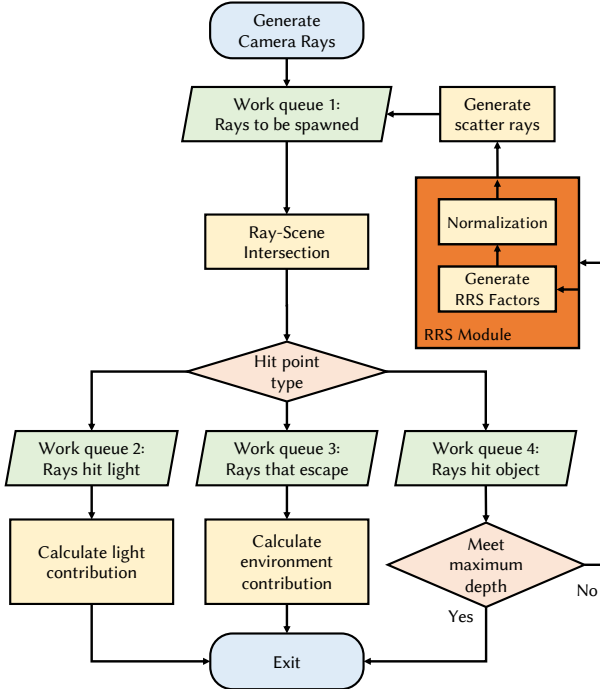


Fig. 2. **Wavefront Architecture with RRS module.** The essence of the wavefront path tracing architecture lies in decomposing the control flow of path tracing into a staged dataflow. Leveraging a work-queue-based mechanism, it is expected to achieve highly parallel, low-divergence, and load-balanced execution. Our RRS module regulates the generation and termination of paths, maintaining stable work queue sizes and providing a predictable workload scheduling mechanism for the GPU. The rectangles in this flowchart indicate different kernels.

h) *Discussion on Neuralization of RRS*: A straightforward way to incorporate neural networks into RRS is to replace the local cache structure with a neural network. For ADDRIS [2], only the estimation of the mean of the outgoing radiance $L_o(\mathbf{x}, \omega_o)$ is required, where $\mathbf{x}$ is the prefix path ending at point $\mathbf{p}$ and ω_o is the outgoing direction. However, ADDRIS only considers the expected contribution of suffix paths $\mathbf{z}$ starting from $\mathbf{p}$, without accounting for their variance or computational cost. For EARS [4], it needs to estimate both the mean and variance of $L_o(\mathbf{x}, \omega_o)$, as well as the expected computational cost of tracing suffix paths $\mathbf{z}$. It is challenging to accurately estimate all three quantities simultaneously using a single simple neural network, as discussed in detail in Sec. VI-B3. Moreover, both ADDRIS and EARS cannot be easily adapted to our normalization architecture by simply modifying their theoretical formulations. In particular, for EARS, the normalization process undermines its theoretical foundation, making it no longer optimal. In the following section, we introduce a neural network-based RRS method designed to address this issue.

IV. METHODS

In this section, we first propose a theoretical RRS framework for wavefront UPT. Next, we introduce two neural RRS methods, called NRRS and AID-NRRS, specifically designed to operate within this framework. Finally, we present Mix-Depth, a method for combining multiple RRS techniques to further improve efficiency.

A. Normalization RRS framework for Wavefront

1) *Problem Statement*: Traditional RRS algorithms generate a stochastic number of secondary samples determined by per-path factors q_{orig} . While this flexibility is acceptable in recursive path tracing, it becomes problematic in wavefront architectures, which process path states in large, preallocated batches on the GPU. Since wavefront path tracing relies on a fixed-size workqueue and synchronized execution, an uncontrolled number of samples causes severe instability, memory overflow, and unpredictable scheduling. A simple truncation of excessive samples, however, introduces statistical bias, particularly when a valid contribution is discarded entirely. For convenience, Tab. I summarizes the key notation used in the following sections.

2) *Normalization Process*: To resolve this structural incompatibility, we introduce a normalization step that globally constrains the total number of samples. Hence, we define the normalization operation as:

$$q_{\text{norm}}^{(i)} = F_{\text{norm}} \cdot q_{\text{orig}}^{(i)}, \quad (10)$$

where the normalization factor F_{norm} is computed dynamically as

$$F_{\text{norm}} = \frac{N_{\text{px}}}{\sum_i q_{\text{orig}}^{(i)}}. \quad (11)$$

This guarantees that

$$\sum_i q_{\text{norm}}^{(i)} = N_{\text{px}}, \quad (12)$$

TABLE I

SUMMARY OF KEY NOTATIONS USED IN OUR METHOD. THE UPPER PART OF THE TABLE SUMMARIZES SYMBOLS RELATED TO THE RRS NORMALIZATION PROCESS, WHILE THE LOWER PART LISTS NOTATION USED IN THE THEORETICAL ANALYSIS AND IMPLEMENTATION.

Symbol	Description
N_{px}	Number of pixels (workqueue size)
$q_{\text{orig}}^{(i)}$	Original RRS factor for path i
F_{norm}	Normalization factor (Eq. 11)
$q_{\text{norm}}^{(i)}$	Normalized RRS factor for path i
$q_{\text{norm}}^{\prime(i)}$	Rate-controlled RRS factor for path i (Eq. 13)
X_i	Random variable: scattered samples from path i
k_i, r_i	Integer/fractional parts of $q_{\text{norm}}^{(i)}$
S	Total number of generated samples $\sum_i X_i$
$\mathcal{I}$	Index set of active parent paths
f_{rate}	Rate-control factor
S'	Total number of generated samples after rate control
α	Adaptive scaling factor for f_{rate}

where N_{px} denotes the total number of pixels (or equivalently, the preallocated size of the primary workqueue). Intuitively, F_{norm} acts as a global scaling factor enforcing a conservation constraint on sample count, ensuring that the expected total number of scattered rays matches the rendering budget per iteration.

3) *Boundedness and Architectural Stability*: The normalization process guarantees that the expected number of generated samples equals N_{px} . Hence, the algorithm achieves bounded sample growth, allowing stable GPU execution and predictable memory allocation within the wavefront pipeline.

a) *Statement*: Under the normalization defined in Eq. (10)–Eq. (11), let $X_i \in \mathbb{R}^+$ denotes the random variable representing the number of scattered samples generated from the i -th parent path in one wavefront iteration. X_i follows the distribution given by Eq. (3) with $q = q_{\text{norm}}^{(i)}$. Let $\mathcal{I}$ be the index set of active parent paths. The total number of generated samples is

$$S = \sum_{i \in \mathcal{I}} X_i.$$

Then, the expected total number of samples is bounded as

$$\mathbb{E}[S] = N_{\text{px}},$$

and the conditional variance admits the upper bounds

$$\text{Var}(S \mid \{q_{\text{norm}}^{(i)}\}) \leq \rho \leq N_{\text{px}}, \quad \text{Var}(S \mid \{q_{\text{norm}}^{(i)}\}) \leq \frac{1}{4}|\mathcal{I}|,$$

where $\rho = \sum_{i \in \mathcal{I}} r_i$ and $q_{\text{norm}}^{(i)} = k_i + r_i$ with $k_i = \lfloor q_{\text{norm}}^{(i)} \rfloor \in \mathbb{N}$ and $r_i \in [0, 1)$. See the proofs in the appendix.

b) *Discussion*: The above results indicate that: (1) the expected sample count is strictly bounded by the pixel number N_{px} , and (2) its variance grows linearly with the number of active paths. Consequently, workqueues can be safely preallocated with fixed size N_{px} , ensuring stable GPU memory usage and predictable wavefront scheduling.

4) *Rate-Control Factor*: Although normalization ensures that the expected number of generated samples satisfies $\mathbb{E}[S] = N_{\text{px}}$, the stochastic nature of RRS can still cause the realized count S to slightly exceed N_{px} with a small probability. To further suppress this overflow probability, we

introduce a mildly scaled *rate-control factor* $f_{\text{rate}} \in (0, 1)$ that adjusts the normalized RRS factors.

a) *Definition*: Given the normalized factors $q_{\text{norm}}^{(i)}$, define the rate-controlled RRS factors as

$$q_{\text{norm}}^{\prime(i)} = f_{\text{rate}} \cdot q_{\text{norm}}^{(i)}. \quad (13)$$

The expected number of generated samples becomes

$$\mathbb{E}[S'] = f_{\text{rate}} N_{\text{px}}, \quad (14)$$

where S' denotes the total number of scattered samples after rate control.

b) *Overflow probability*: By Bernstein's inequality, the probability that S' exceeds the queue capacity N_{px} is exponentially small:

$$\mathbb{P}(S' \geq N_{\text{px}}) \leq \exp\left(-\frac{(1 - f_{\text{rate}})^2 N_{\text{px}}}{2f_{\text{rate}} + \frac{2}{3}(1 - f_{\text{rate}})}\right). \quad (15)$$

For instance, with $f_{\text{rate}} = 0.85$, the exponent is approximately $-0.0125 N_{\text{px}}$. We provide the detailed derivation in the supplemental material.

c) *Adaptive variant*: In practice, we also introduce an adaptive mechanism that updates the rate factor as $f_{\text{rate}} \leftarrow \alpha f_{\text{rate}}$ for finer control. Here, $\alpha \in (0, 1]$ is reduced upon detecting overflow:

$$\alpha \leftarrow \alpha \cdot (1 - \epsilon), \quad \epsilon \approx 0.01. \quad (16)$$

When overflow occurs, f_{rate} is accordingly reduced, and the current ray generation step is restarted; the adjustment remains in effect for the rest of the rendering process. As overflow is rare in practice, this adaptive mechanism primarily serves as a safeguard to maintain stability and unbiasedness in extreme cases.

Overall, normalization ensures that the sample count is bounded in expectation, while rate control further constrains the realized count, making it unlikely to exceed the preallocated capacity. Together, they provide a unified framework that achieves both theoretical guarantees and practical stability for RRS in wavefront path tracing.

5) *Extended Discussion*: Normalization serves as a critical bridge between the theoretical design of RRS and the practical constraints of wavefront path tracing. Architecturally, it transforms an unbounded stochastic sampling process into a bounded, resource-aware one, ensuring predictable GPU scheduling and stable memory utilization. Statistically, normalization enforces a global budget constraint on sample generation, preserving expectation consistency and mitigating the catastrophic bias that would otherwise result from naive sample truncation.

Nevertheless, normalization also introduces theoretical challenges. Because F_{norm} is dynamic and scene-dependent, the normalized factors q_{norm} are no longer linearly proportional to their original counterparts q_{orig} , potentially inverting the behavior of Russian Roulette and splitting. This violates the proportional-invariance assumption underlying existing analytical RRS schemes such as EARS, thereby compromising their theoretical consistency. Hence, while normalization is indispensable for boundedness and stability, it simultaneously

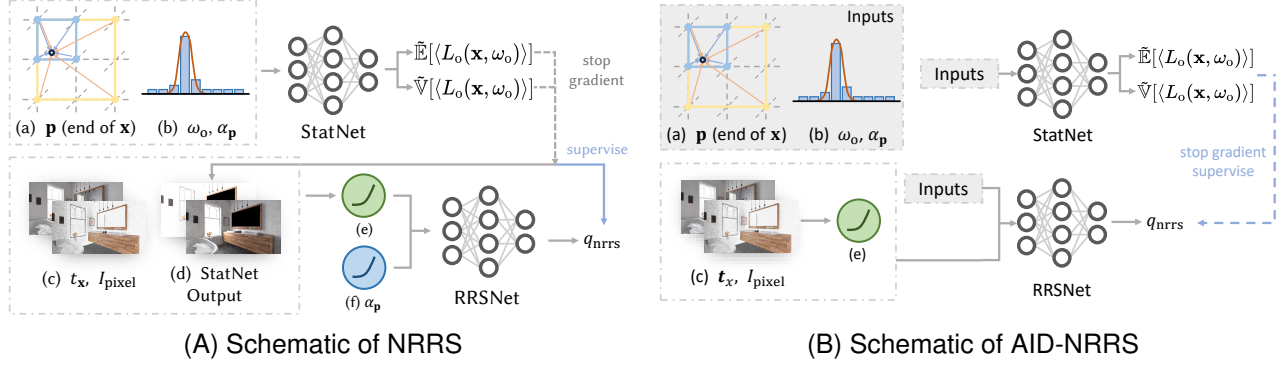


Fig. 3. (A) NRRS consists of two neural networks: **StatNet** and **RRSNet**. StatNet predicts the local radiance statistics $L_o(\mathbf{x}, \omega_o)$, while RRSNet predicts the RRS factors q_{nrrs} . StatNet takes as input the position $\mathbf{p}$, the outgoing direction ω_o , and the surface roughness α_p , and outputs both the mean and variance of the local radiance. These predicted statistics serve both as direct inputs to RRSNet (d) and as supervisory signals for training its output. (a) The position $\mathbf{p}$ is encoded using a learnable multi-resolution hash grid [34]. (b) The direction ω_o and roughness α_p are encoded using one-blob encoding [9]. The predicted radiance statistics (after Box-Cox transformation) (e), together with throughput t_x , pixel radiance I_{pixel} , and remapped roughness (f), are fed into RRSNet to predict the resampling factor q_{nrrs} for point $\mathbf{p}$. (B) AID-NRRS is similar to NRRS, except that RRSNet omits the StatNet outputs and directly takes the position $\mathbf{p}$ and outgoing direction ω_o as inputs. Note that StatNet and RRSNet each have their own Inputs module with separate trainable parameters.

demands a reformulation of RRS that remains coherent under dynamic scaling.

To address this limitation, we introduce a learning-based paradigm in which a neural network directly models the normalized RRS process. Unlike handcrafted heuristics that rely on static probabilistic formulations, our proposed neural RRS (NRRS and AID-NRRS) learns a data-driven mapping from local path statistics and scene attributes to normalized RRS factors, naturally adapting to the stochastic constraints imposed by F_{norm} . This neural formulation implicitly maintains statistical coherence under normalization, while preserving the architectural efficiency and memory stability of the wavefront framework. In essence, the neural approach provides a principled and flexible pathway to integrate normalization into RRS, achieving practical scalability within a unified framework.

B. Neural RRS

With the success of neural methods [5], [6], [33] in rendering, we further employ neural networks to predict RRS factors, leveraging their ability to approximate target functions continuously and more accurately than discrete heuristics. Furthermore, by designing appropriate loss functions (see Sec. IV-B2), we can mitigate issues arising from the behavioral changes of RRS factors before and after normalization. In other words, the side effect of the normalization process can be explicitly incorporated into the loss design. Note that our neural RRS methods are scene-specific.

1) *Network Structure*: We refer to our neural RRS methods as NRRS and AID-NRRS, which differ only in the design of RRSNet (see Fig. 3). Both methods share the same training strategy.

The main challenge in NRRS and AID-NRRS is that the ground truth for RRS factors yielding optimal efficiency is unknown, making supervised training infeasible. Fortunately, under our new RRS framework, the cost of rendering each frame is nearly constant, as normalization is performed at every bounce. With this fixed cost, maximizing efficiency can

be reformulated as minimizing the variance of pixel radiance. A similar idea is also adopted by Bolin and Meyer [23] with simplified dependencies. Since the relationship between pixel variance and the RRS factors is known from Eq. (7) and Eq. (9), we can design a loss function to minimize pixel variance and propagate gradients to the RRS factors accordingly.

As shown in Eq. (7) and Eq. (9), we need certain statistics of the local radiance $L_o(\mathbf{x}, \omega_o)$ to enable gradient transfer. Therefore, we design our network to consist of two components: StatNet and RRSNet. The details of these two networks are as follows:

a) *StatNet*: StatNet predicts the local radiance $\mathbb{E}[\langle L_o(\mathbf{x}, \omega_o) \rangle]$ and its variance $\mathbb{V}[\langle L_o(\mathbf{x}, \omega_o) \rangle]$, both of which are essential for propagating gradients from pixel variance to the RRS factors. Inspired by NRC [5], we design StatNet to take as input the outgoing direction ω_o , the position $\mathbf{p}$, and auxiliary information such as the roughness at $\mathbf{p}$.

b) *RRSNet*: RRSNet is responsible for predicting the RRS factors, $q_{nrrs}(\mathbf{x})$. We have two design choices: (1) using the outputs of StatNet as inputs and supervisory signals or (2) using the outputs of StatNet only as supervisory signals. We refer to the first design as NRRS (see Fig. 3A and the second as AID-NRRS (see Fig. 3B).

For NRRS (Fig. 3A), we provide RRSNet with the following necessary information:

- (1) the prefix path $\mathbf{x}$, specifically its throughput t_x .
- (2) the endpoint $\mathbf{p}$, specifically its roughness α_p .
- (3) the full path, specifically the pixel value I_{pixel} to which $\mathbf{x}$ contributes.
- (4) the suffix path $\mathbf{y}$, specifically the local radiance statistics estimations $\mathbb{E}[\langle L_o(\mathbf{x}, \omega_o) \rangle]$, $\mathbb{V}[\langle L_o(\mathbf{x}, \omega_o) \rangle]$ from StatNet.

For AID-NRRS (Fig. 3B), the first three items are the same as NRRS. As for the last item, we directly use the position $\mathbf{p}$ and the outgoing direction ω_o instead of the local radiance

statistics. Note that StatNet and RRSNet have its own Inputs module (trainable parameters) respectively.

For NRRS, we do not use $\mathbf{p}$ or ω_0 as inputs, so that RRSNet learns the relationship between the input features and the RRS factors, rather than memorizing the distribution of RRS factors across the scene. This design choice improves the generalization ability of RRSNet and makes the learning process more efficient. However, it requires StatNet to be executed during inference (unlike AID-NRRS), which introduces additional overhead. Therefore, NRRS is more suitable for scenes where the RRS factors change frequently throughout the scene.

The Loss Design (Sec. IV-B2) and Mix-Depth (Sec. IV-C) is the same for both NRRS and AID-NRRS, so we use NRRS as an example in the following subsections.

2) *Loss Design*: For StatNet, we use the local radiance estimation $\langle L_o(\mathbf{x}, \omega_0) \rangle$ obtained during the path tracing process as ground truth. StatNet is trained to predict the average local radiance $\mathbb{E}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$ and its statistic $\mathbb{V}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$. We use relative MSE [35] as our loss function, as shown in Eq. (17). Here, $\tilde{\mathbb{E}}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$ and $\tilde{\mathbb{V}}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$ are the outputs of StatNet, and $\text{sg}(\cdot)$ indicates that its argument is treated as a constant during optimization. As for $\mathbb{V}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$, we regard $\tilde{\mathbb{E}}[\langle L_o(\mathbf{x}, \omega_0) \rangle]$ as mean value and calculate one-sample variance of $\langle L_o(\mathbf{x}, \omega_0) \rangle$ as the ground truth.

$$\begin{aligned} \mathcal{L}_{\text{StatNet}} = & \mathcal{L}_2(\tilde{\mathbb{E}}[\langle L_o(\mathbf{x}, \omega_0) \rangle], \langle L_o(\mathbf{x}, \omega_0) \rangle) \\ & + \mathcal{L}_2(\tilde{\mathbb{V}}[\langle L_o(\mathbf{x}, \omega_0) \rangle], \mathbb{V}[\langle L_o(\mathbf{x}, \omega_0) \rangle]), \quad (17) \end{aligned}$$

where $\mathcal{L}_2(a, b) = \frac{(a - b)^2}{\text{sg}(b)^2 + \epsilon}$.

As for RRSNet, the loss function with three components is as:

$$\mathcal{L}_{\text{RRSNet}} = \gamma_1 \mathcal{L}_{\text{avg}} + \gamma_2 \mathcal{L}_{\text{min}} + \gamma_3 \mathcal{L}_{\text{rrs}}. \quad (18)$$

First, we explain how to transfer the gradient from the pixel error to the RRS factors. Then, we discuss each of the three loss functions in detail. Fig. 4 illustrates how three components of the loss function interact and contribute to the overall optimization. The loss function of RRSNet consists of three components: Minimize Error Loss ($\mathcal{L}_{\text{min}}$), Average Error Loss ($\mathcal{L}_{\text{avg}}$), and Bound Loss ($\mathcal{L}_{\text{rrs}}$). Wherein, solid arrows represent the forward process: RRSNet predicts the original RRS factors q , which are then normalized to q_{norm} by the RRS module. Here, q_{norm} is scaled by rate-control factor. For clarity, we skip the “’” notation, i.e., using q_{norm} instead of q'_{norm} . The normalized RRS factors are used to generate scattered paths, whose radiance contributions are accumulated to the corresponding pixel, and the error of pixels is computed. Dashed arrows indicate the backward process: gradients from the pixel error are propagated back to the RRS factors via the Minimize Error Loss and Average Error Loss, while the Bound Loss directly constrains the original RRS factors.

a) *Gradient Transfer*: The relationship between the pixel variance and a specific prefix path $\mathbf{x}$ is defined by Eq. (7) and Eq. (9). Using these two equations, we derive how the

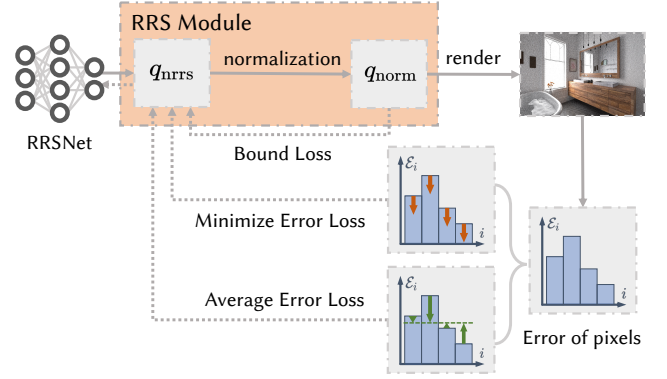


Fig. 4. **Loss Function of RRSNet**. The loss function of RRSNet includes three components: Minimize Error Loss, Average Error Loss, and Bound Loss. Solid arrows indicate the forward process—from RRS factor prediction and normalization to scattered path generation and pixel error computation. Dashed arrows represent the backward process, where gradients propagate through Minimize Error Loss and Average Error Loss, while Bound Loss directly constrains the original RRS factors.

pixel variance depends on the RRS factors under our network design.

We first consider RR. The input (denoted as θ) of RRSNet can correspond to many different prefix paths $\mathbf{x}$, which we denote as the set Θ . Therefore, the derivative of the pixel variance $\mathbb{V}[\langle I_{\text{px}} \rangle]$ with respect to the RR factors $p_{\text{rr}}(\mathbf{x})$ is the integral of $\frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dp_{\text{rr}}(\mathbf{x})}$ over the set Θ , as shown in Eq. (19):

$$\frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dp_{\text{rr}}(\theta)} = \int_{\Theta} \frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dp_{\text{rr}}(\mathbf{x})} d\mathbf{x}. \quad (19)$$

We use the Monte Carlo method to estimate this integral, collecting all necessary information during the UPT process. The one-sample Monte Carlo estimation is shown in Eq. (20). Notably, the unstable term $p(\mathbf{x})$ in Eq. (9) disappears, which makes the training process more stable. See additional discussion of this term in the appendix.

$$\left\langle \frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dp_{\text{rr}}(\theta)} \right\rangle = - \left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \frac{\mathbb{E}[\langle H(\mathbf{x}) \rangle^2]}{p_{\text{rr}}^2(\mathbf{x})}. \quad (20)$$

The deduction for the splitting case is similar. We can obtain the derivative of the pixel variance with respect to the splitting factors $n_s(\mathbf{x})$ as shown in Eq. (21). Note that, as in EARS [4] and MARS [13], we ignore the variance introduced by stochastic rounding.

$$\left\langle \frac{d\mathbb{V}[\langle I_{\text{px}} \rangle]}{dn_s(\theta)} \right\rangle = - \left(\frac{g(\mathbf{x})}{p(\mathbf{x})} \right)^2 \frac{\mathbb{V}[\langle H(\mathbf{x}) \rangle]}{n_s^2(\mathbf{x})}. \quad (21)$$

With the help of Eq. (20) and Eq. (21), we only need to design the loss function with respect to the pixel variance, and then the gradient can be propagated to the RRS factors for optimization.

b) *Minimize Error Loss*: $\mathcal{L}_{\text{min}}$ is the loss function used to minimize the pixel error. We use the relative MSE to measure the pixel error. Because our method is unbiased, the MSE is equal to the variance. Here, $\epsilon = 0.01$ is a small constant to maintain numerical stability.

$$\mathcal{L}_{\min} = \sum_{i=1}^{N_{\text{px}}} \mathcal{E}_i, \text{ where } \mathcal{E}_i = \frac{\mathbb{V}[\langle I_i \rangle]}{I_i^2 + \epsilon}. \quad (22)$$

c) *Average Error Loss*: $\mathcal{L}_{\text{avg}}$ aims to make the error across pixels more uniform. This helps prevent too many rays from concentrating on a few pixels.

$$\mathcal{L}_{\text{avg}} = \sum_{i=1}^{N_{\text{px}}} \left(\mathcal{E}_i - \mathcal{E}_{\text{avg}} \right)^2, \text{ where } \mathcal{E}_{\text{avg}} = \sum_{i=1}^{N_{\text{px}}} \frac{\mathcal{E}_i}{N_{\text{px}}}. \quad (23)$$

d) *Bound Loss*: $\mathcal{L}_{\min}$ will cause the RRS factors to grow toward infinity, as its gradient is always negative (see Eq. (20) and Eq. (21)). Moreover, $\mathcal{L}_{\text{avg}}$ only fine-tunes the RRS factors. Therefore, a loss function is needed to bound the RRS factors. To this end, we design $\mathcal{L}_{\text{rrs}}$ as shown in Eq. (24). Here, N_{samples} denotes the number of training samples in the current frame.

$$\mathcal{L}_{\text{rrs}} = \sum_{i=1}^{N_{\text{samples}}} \left(q_{\text{nrrs}}(\theta) - q_{\text{norm}}(\theta) \right)^2. \quad (24)$$

Note that $\mathcal{L}_{\text{rrs}}$ is defined on the sample points, similar to StatNet. Here, $q_{\text{norm}}(\theta)$ refers to the RRS factors after normalization (see Sec. IV-A). $\mathcal{L}_{\text{rrs}}$ can also help alleviate the issue that the behavior of RRS factors changes after normalization, as discussed in Sec. IV-A.

It is worth noting that the ground truth for the “Bound Loss” is set as q_{norm} rather than another value in our method. **Specifically, the actual RRS factor used to generate scatter rays is q_{norm} , not q_{nrrs} .** However, propagating gradients from q_{norm} to q_{nrrs} is non-trivial due to behavioral changes introduced by normalization. In our implementation, both “Minimize Error Loss” and “Average Error Loss” treat q_{nrrs} as the final RRS factor for generating scatter rays. Therefore, by setting the ground truth of the “Bound Loss” to q_{norm} , we help bridge the gap between q_{norm} and q_{nrrs} , thus enabling RRSNet to better minimize the final image error upon the normalization constraint.

As for $\mathcal{L}_{\min}$ and $\mathcal{L}_{\text{avg}}$, the pixel loss is equally shared among its samples. That is, if there are k_i training samples for pixel i , then each training sample receives $\frac{1}{k_i}$ of the original loss.

C. Mix-Depth

Network inference introduces additional overhead, so the results of NRRS may not always be optimal. In some cases, the efficiency gains from NRRS are insufficient to compensate for this overhead, making it advantageous to disable NRRS at certain bounces. Inspired by traditional heuristic strategies, we propose a new mechanism called Mix-Depth to further enhance the performance of NRRS.

In some traditional RR methods, RR is heuristically disabled at the first few bounces [20]; in other words, these approaches combine fixed RR (1.0) with their RR strategies. Inspired by this, we can also integrate NRRS with other RRS methods. With the help of StatNet, we obtain a neural version of ADRRS (denoted as ADRRS (NN)), which allows us to combine NRRS with ADRRS (NN) as well.

The key idea of Mix-Depth is to select a specific RRS method at each bounce. In our implementation, we enhance NRRS with ADRRS (NN) and fixed RR. To achieve this, we propose two algorithms: Brute-Force Search and Heuristic Search.

a) *Brute-Force Search*: This approach is suitable for small maximum depths. It tries all possible combinations of NRRS, ADRRS (NN), and fixed RR at each bounce, and selects the best one based on efficiency estimation from one-sample-per-pixel result. The time complexity of this algorithm is $O(3^B)$, where B is the maximum path depth. For UPT with a max depth of 6, the number of combinations is 729, making this method feasible. However, for a max depth of 10, the number of combinations increases to 59,049, which is too large to search exhaustively.

b) *Heuristic Search*: This approach is more practical for large maximum depths. We segment the maximum depth into several parts, divided by a parameter T_d . For each segment, starting from the shallower bounces, we search for the best combination using Brute-Force Search and set the earlier bounces to the selected combination. The remaining bounces are set to fixed RR.

For example, if $T_d = 6$ and the maximum depth is 10, we first set the latter bounces [7, 10] to fixed RR, and then perform Brute-Force Search on bounces [1, 6]. After finding the best combination, we fix bounces [1, 6] to the selected combination and then perform Brute-Force Search on bounces (6, 10]. The number of combinations to be tested drops to $3^6 + 3^4 = 810$, which is much more manageable.

Note that Mix-Depth is a post-processing step and does not affect the training process of NRRS. It can be used to combine multiple RRS methods without additional overhead, provided they share the same cache structure.

V. IMPLEMENTATION

a) *System*: We implement our algorithm using Kiraray [6], a GPU renderer based on CUDA and OptiX [36]. The neural network components of NRRS and AID-NRRS are implemented using tiny-cuda-nn [34].

b) *Network Architecture*: The hidden layers for StatNet, and RRSNet share the same architecture. They are all MLP with 3 linear layers of 32 neurons, with Leaky ReLU activation [37] after each layer.

c) *Input Encoding*: StatNet uses three input features (see Sec. IV-B1). The position $\mathbf{p} \in \mathbb{R}^3$ is encoded with a trainable multiresolution hash grid [38] G with 8 levels, and each level has 2 features. The coarsest level resolution of G is 16, and the resolution doubles at each finer level. The outgoing direction $\omega_o \in \mathbb{R}^2$ (spherical coordinates) is encoded using one-blob encoding [9] with 4 bins. The roughness $\alpha_p \in \mathbb{R}$ is encoded with one-blob encoding with 8 bins after applying the remapping function $1 - e^{-\alpha_p}$ [5].

For RRSNet, NRRS has five input features (see Fig. 3A). The high dynamic range features $\tilde{\mathbb{E}}[\langle L_o(\mathbf{x}, \omega_o) \rangle] \in \mathbb{R}^3$, $\tilde{\mathbb{E}}[\langle L_o^2(\mathbf{x}, \omega_o) \rangle] \in \mathbb{R}^3$, $t_{\mathbf{x}} \in \mathbb{R}^3$, and $I_{\text{pixel}} \in \mathbb{R}$ (mean value) are encoded with the box-cox transform [39] with $\lambda = 0.5$.

The roughness $\alpha_p \in \mathbb{R}$ is encoded by the remapping function $1 - e^{-\alpha_p}$.

AID-NRRS also have five input features (see Fig. 3B). The encodings of $\mathbf{p} \in \mathbb{R}^3$ and $\omega_o \in \mathbb{R}^2$ are the same as in StatNet. The encodings of $t_x \in \mathbb{R}^3$ and $I_{\text{pixel}} \in \mathbb{R}$ are the same as NRRS. The roughness $\alpha_p \in \mathbb{R}$ is encoded with one-blob encoding with 4 bins after applying the remapping function $1 - e^{-\alpha_p}$.

d) Output & Activation: The output of StatNet is $L_o \in \mathbb{R}^3$ and $L_o^2 \in \mathbb{R}^3$. In our experiments, StatNet works well without any activation functions on their outputs. For RRSNet, the output is the RRS factor $q \in \mathbb{R}$. We use the activation function defined in Eq. (25) to ensure the output is positive. This function is a modified version of softplus [40] to improve computational stability.

$$y = \begin{cases} \ln(1 + \exp(x)) & \text{if } x < 0 \\ 0.5x + \ln 2 & \text{if } x \geq 0. \end{cases} \quad (25)$$

e) Training: Since RRSNet requires the output of StatNet to compute gradients, we train StatNet first. The training process is separated into two steps:

- 1) Train StatNet with the ground truth $\langle L_o \rangle$ and $\langle L_o^2 \rangle$ from UPT. At the same time, train RRSNet to output 1.0 using the relative MSE loss $\mathcal{L}_2$.
- 2) Continue training StatNet as in step (1). For RRSNet, use the loss described in Sec. IV-B2. First, calculate the variance of the current frame for each pixel and use this to compute the loss $\mathcal{L}_{\text{RRSNet}}$. Then, calculate the derivative of $\mathcal{L}_{\text{RRSNet}}$ with respect to the variance. Using the Monte Carlo estimation from Eq. (20) and Eq. (21), propagate the gradient to the output of RRSNet. The statistics required for Eq. (20) and Eq. (21) are provided by StatNet.

Note that the gradient of RRSNet does not flow to StatNet; they are trained separately. We train StatNet and RRSNet using the Adam optimizer [41] with fixed learning rates of 0.005 and 0.0003, respectively, and a batch size of 2^{19} . During inference, we use exponential moving average (EMA) as [5] to reduce noise from Monte Carlo estimation. For $\mathcal{L}_{\text{RRSNet}}$, we empirically use $\gamma_1 = 0.01$, $\gamma_2 = 0.05$, and $\gamma_3 = 0.01$ for all scenes. We perform grid search on Bathroom scenes (not included in the test set), and use these hyperparameters for all other scenes. See more details of grid search in the appendix.

f) Compromise: During the training process, we do not have the ground truth I_{px} for each pixel, so we use the accumulated result from the previous frame as the ground truth. To support infinite depth in UPT, we cannot normalize the RRS factors (Sec. IV-A) at every depth. Therefore, if the normalization factor $F_{\text{norm}} \geq 1$ (Eq. (11)), we do not perform normalization. In other words, we only *downscale* the RRS factors.

g) Tackle Low Spp: During the RRSNet training in step (2), we need to compute the per-pixel variance of the current frame. However, with only 1 sample per pixel (spp), the image is too noisy, which makes the training process unstable. To address this, we employ both temporal and spatial denoising.

First, we use $I_{\text{px}}^{\text{acc}}$ instead of the current frame $I_{\text{px}}^{\text{cur}}$ to calculate the error $\mathcal{E}'_{\text{px}}$, where $I_{\text{px}}^{\text{acc}}$ is updated as:

$$I_{\text{px}}^{\text{acc}} = (1 - \alpha_{\text{mix}})I_{\text{px}}^{\text{acc}} + \alpha_{\text{mix}}I_{\text{px}}^{\text{cur}}. \quad (26)$$

We set $\alpha_{\text{mix}} = 0.5$ in our implementation. Next, we use the OptiX denoiser [26] to denoise $\mathcal{E}'_{\text{px}}$ and obtain $\mathcal{E}_{\text{px}}$ for use in Eq. (18). The denoiser is provided with the normal buffer but no albedo buffer, as including the albedo buffer can cause the error to resemble albedo rather than variance. Denoising introduces bias into the error estimation; therefore, we disable the OptiX denoiser during the final 5 seconds of training.

VI. RESULTS AND DISCUSSION

All experiments are conducted on an Intel Core i7-12700F CPU and an NVIDIA RTX 3080 GPU. For the baseline settings, we enable next event estimation (NEE) and set the maximum path length to 6 by default, unless otherwise stated, as the majority of the contribution comes from the first few bounces. For a fair comparison, reflectance factorization is not used in our experiments, as it is not employed in the compared methods. In all main experiments, the pixel estimate for each method is computed from the accumulated buffer without denoising.

All images are rendered at a resolution of 1280×720 , and image quality is evaluated using mean relative squared error (RelMSE) [35].

A. Comparisons

We compare our method with ADRRS [2] and EARS [4], evaluating both the original tree-based implementations and their neural network variants. We re-implement these methods in Kiraray [6], following the open-source code of EARS [42], and use settings as close as possible to those in the original papers. Notably, all methods are adapted to the normalization RRS framework and consistently implemented within the wavefront architecture. In this section, a "+" suffix appended to each method denotes the incorporation of the Mix-Depth module described in Sec. IV-C. For instance, NRRS+ denotes NRRS with Mix-Depth applied.

1) Equal-time Comparison: All methods are allocated 60 seconds for training and 60 seconds for rendering to ensure a fair comparison. This budget is reasonable because the competing approaches generally achieve near-converged performance within this time frame. Note that for NRRS+, the training time includes both the training of StatNet and RRSNet, as well as the time spent searching for the best strategy. Please refer to the appendix for a detailed breakdown of the training time distribution.

The comparison is performed across a variety of scenes. Apartment1 is a night scene illuminated by several small light sources. Apartment2 is a complex indoor environment containing many objects and lit by an area light positioned outside. Apartment3 features intricate indoor geometry and is illuminated by an area light outside. Bistro Interior is an indoor scene with diverse materials and numerous objects, illuminated by small light sources. Sponza Caustic is an outdoor scene exhibiting caustics produced by wave surfaces above the floor,

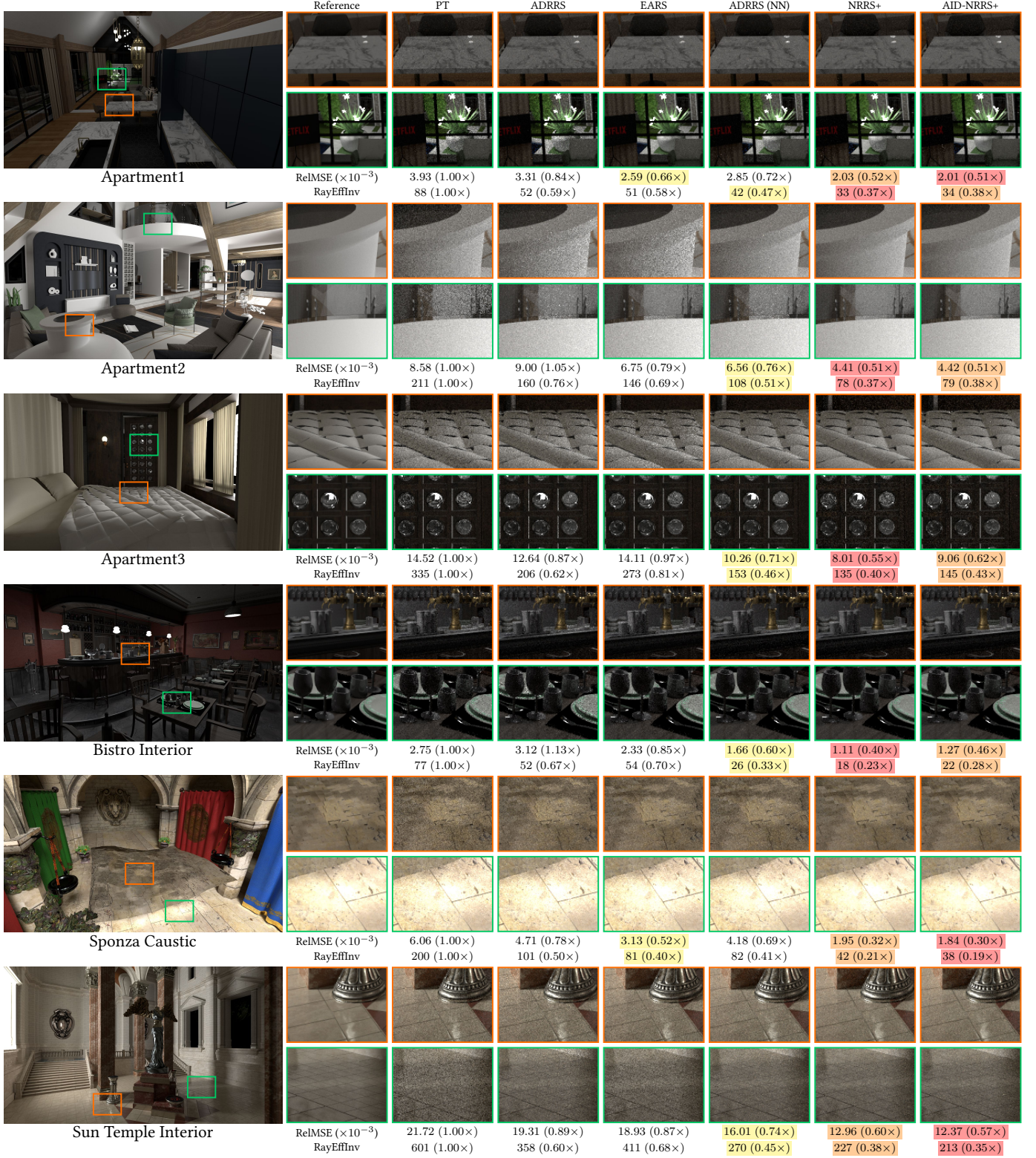


Fig. 5. **Methods Evaluation and Comparisons on diverse scenes.** Each scene is trained for 60 seconds and then inferred for 60 seconds across all methods. All images are rendered at a resolution of 1280×720 . Procedural results generated during training are not included due to excessive noise. The first row shows the RelMSE error ($\times 10^{-3}$) over the entire image, where lower values indicate better performance. The second row presents RayEffInv, as defined in Sec. VI-A. Ray count is calculated as the total number of rays traced during the 60-seconds inference period, divided by the number of pixels. We color code the first, second, and third lowest numbers. Our approach consistently shows the best results using fewer rays, indicating that each traced ray was utilized as effectively as possible. RelMSE values are rounded to two decimal places, and RayEffInv values are clamped to integers.

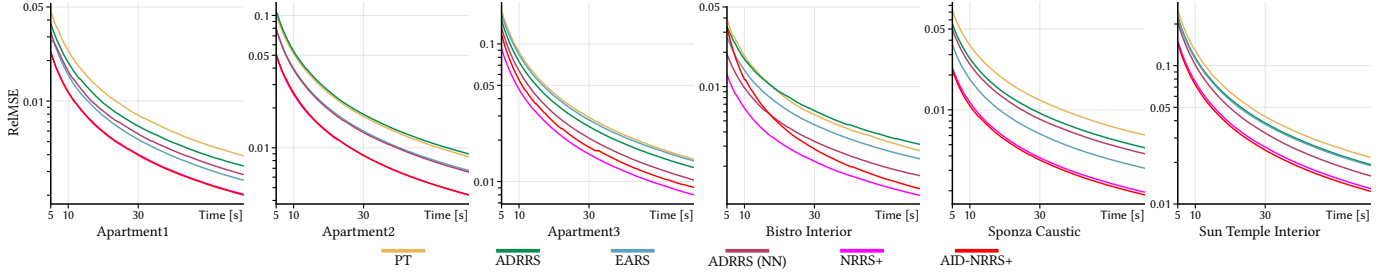


Fig. 6. RelMSE convergence curves for different methods corresponding to Fig. 5.

TABLE II

MAIN EXPERIMENT FOR DEPTH 10. THE EXPERIMENT SETTING ARE THE SAME WITH FIG. 5, EXCEPT THE MAXIMUM PATH DEPTH IS SET TO 10. THE DATA IN THE TABLE IS RELMSE ($\times 10^{-3}$). WE ALSO COLOR CODE THE FIRST, SECOND, AND THIRD LOWEST NUMBERS.

Scene	PT	ADRRS	EARS	ADRRS (NN)	NRRS+	AID-NRRS+
Apartment1	5.60 (1.00 $\times$)	4.56 (0.81 $\times$)	3.25 (0.58 $\times$)	3.34 (0.60 $\times$)	2.69 (0.48 $\times$)	2.37 (0.42 $\times$)
Apartment2	12.38 (1.00 $\times$)	9.36 (0.76 $\times$)	9.27 (0.75 $\times$)	6.96 (0.56 $\times$)	5.77 (0.47 $\times$)	5.80 (0.47 $\times$)
Apartment3	20.86 (1.00 $\times$)	14.66 (0.70 $\times$)	21.04 (1.01 $\times$)	11.55 (0.55 $\times$)	8.94 (0.43 $\times$)	9.33 (0.45 $\times$)
Bistro Interior	4.49 (1.00 $\times$)	3.83 (0.85 $\times$)	3.10 (0.69 $\times$)	2.04 (0.45 $\times$)	1.30 (0.29 $\times$)	1.30 (0.29 $\times$)
Sponza Caustic	8.78 (1.00 $\times$)	5.27 (0.60 $\times$)	3.54 (0.40 $\times$)	4.57 (0.52 $\times$)	2.17 (0.25 $\times$)	2.15 (0.24 $\times$)
Sun Temple Interior	35.43 (1.00 $\times$)	24.87 (0.70 $\times$)	27.76 (0.78 $\times$)	18.48 (0.52 $\times$)	16.06 (0.45 $\times$)	15.59 (0.44 $\times$)

and is illuminated by an area light. Sun Temple Interior contains a variety of glossy and metallic materials and is illuminated by an area light on the left. We also provide more scenes in the appendix.

The results are presented in Fig. 5 and the upper half of Tab. III. It can be observed that NRRS+ and AID-NRRS+ consistently outperform the other methods. With the help of neural networks, ADRRS also shows improvement, but still does not match the performance of our approach. Fig. 6 shows the corresponding convergence curves, illustrating how RelMSE evolves over time. We also provide error maps in Fig. 7 (see appendix for more scenes), which highlight the differences in error distribution between EARS and NRRS+.

2) *Ray Efficiency Comparison:* Here, we compare the ray efficiency of different methods. We define the metric *RayEff* as the reciprocal of the product of the number of rays and RelMSE within the same rendering time, and use its inverse, *RayEffInv*, for evaluation. This is similar to the equal-spp comparison in path guiding [6], [8], as we ignore the time spent generating rays across all methods. The results are shown in Fig. 5, using the same experimental settings as in the *Equal Time Comparison*. The results demonstrate that our method can achieve the same RelMSE with fewer rays.

3) *Deeper Depth:* We also explore the performance of our method with deeper depth settings. We set the maximum path length to 10 and perform the same experiments as Sec. VI-A1. The results are shown in Tab. II and the lower half of Tab. III. It can be observed that NRRS+ and AID-NRRS+ still outperform the other methods. Here we only show the numerical results, the detailed images can be found in the appendix.

B. Ablations

In this section, we conduct ablation studies to evaluate the effectiveness of essential components in our method, including

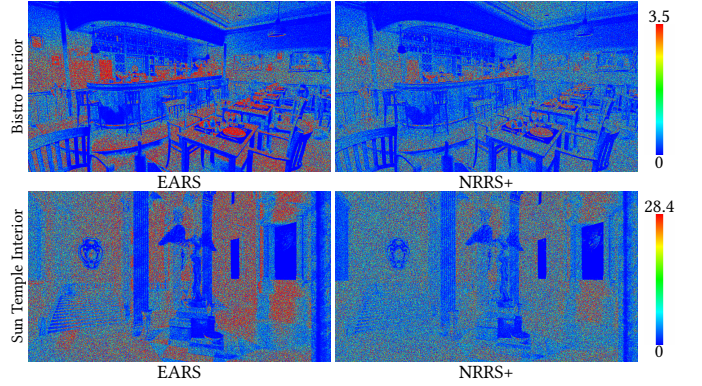


Fig. 7. **EARS Vs. NRRS+ in terms of RelMSE ($\times 10^{-3}$).** For each scene, the maximum tint value (corresponding to red) is set to 1.5 times the largest RelMSE across all methods.

the Mix-Depth module, Denoise module, RRSNet and loss design.

1) *Mix-Depth Module:* We perform Mix-Depth ablation studies on NRRS, AID-NRRS, and EARS. EARS+ denotes a combination of EARS, ADRRS, and fixed RR. Since EARS and ADRRS share the same cache structure, ADRRS can be incorporated into EARS. The results are presented in Tab. III. We can see they are all improved by the Mix-Depth module in most scenes. For EARS, the reasons are twofold: first, certain approximations are made in the implementation for practical purposes, such as tree-structured caching; second, under the normalization framework, EARS is no longer the optimal solution.

However, Mix-Depth does not always lead to performance improvements. For instance, in the Sponza Caustic scene with maximum path length set to 10, EARS+ performs slightly worse than EARS. This is because efficiency estimation based on one sample per pixel is inherently noisy, and Heuristic

TABLE III

ABLATION STUDY FOR MIX-DEPTH MODULE. THE EXPERIMENT SETTING ARE THE SAME WITH FIG. 5, EXCEPT THE MAXIMUM PATH DEPTH FOR THE SCENES IN THE UPPER HALF OF THE TABLE IS 6, WHILE FOR THOSE IN THE LOWER HALF IT IS 10. THE DATA IN THE TABLE IS RELMSE ($\times 10^{-3}$). WE ALSO COLOR CODE THE FIRST, SECOND, AND THIRD LOWEST NUMBERS.

Scene	PT	NRRS	NRRS+	AID-NRRS	AID-NRRS+	EARS	EARS+
Apartment1	3.93 (1.00 $\times$)	2.34 (0.60 $\times$)	2.03 (0.52 $\times$)	2.36 (0.60 $\times$)	2.01 (0.51 $\times$)	2.59 (0.66 $\times$)	2.58 (0.66 $\times$)
Apartment2	8.58 (1.00 $\times$)	5.24 (0.61 $\times$)	4.41 (0.51 $\times$)	5.20 (0.61 $\times$)	4.42 (0.51 $\times$)	6.75 (0.79 $\times$)	5.47 (0.64 $\times$)
Apartment3	14.52 (1.00 $\times$)	8.85 (0.61 $\times$)	8.01 (0.55 $\times$)	9.43 (0.65 $\times$)	9.06 (0.62 $\times$)	14.11 (0.97 $\times$)	11.81 (0.81 $\times$)
Bistro Interior	2.75 (1.00 $\times$)	1.96 (0.71 $\times$)	1.11 (0.40 $\times$)	1.64 (0.60 $\times$)	1.27 (0.46 $\times$)	2.33 (0.85 $\times$)	2.32 (0.85 $\times$)
Sponza Caustic	6.06 (1.00 $\times$)	2.82 (0.47 $\times$)	1.95 (0.32 $\times$)	2.87 (0.47 $\times$)	1.84 (0.30 $\times$)	3.13 (0.52 $\times$)	3.08 (0.51 $\times$)
Sun Temple Interior	21.72 (1.00 $\times$)	14.92 (0.69 $\times$)	12.96 (0.60 $\times$)	14.86 (0.68 $\times$)	12.37 (0.57 $\times$)	18.93 (0.87 $\times$)	17.13 (0.79 $\times$)
Apartment1	5.60 (1.00 $\times$)	3.39 (0.60 $\times$)	2.69 (0.48 $\times$)	3.16 (0.56 $\times$)	2.37 (0.42 $\times$)	3.25 (0.58 $\times$)	3.20 (0.57 $\times$)
Apartment2	12.38 (1.00 $\times$)	7.89 (0.64 $\times$)	5.77 (0.47 $\times$)	7.90 (0.64 $\times$)	5.80 (0.47 $\times$)	9.27 (0.75 $\times$)	7.50 (0.61 $\times$)
Apartment3	20.86 (1.00 $\times$)	14.63 (0.70 $\times$)	8.94 (0.43 $\times$)	14.47 (0.69 $\times$)	9.33 (0.45 $\times$)	21.04 (1.01 $\times$)	17.83 (0.86 $\times$)
Bistro Interior	4.49 (1.00 $\times$)	2.76 (0.62 $\times$)	1.30 (0.29 $\times$)	2.26 (0.50 $\times$)	1.30 (0.29 $\times$)	3.10 (0.69 $\times$)	2.47 (0.55 $\times$)
Sponza Caustic	8.78 (1.00 $\times$)	4.34 (0.49 $\times$)	2.17 (0.25 $\times$)	4.96 (0.57 $\times$)	2.15 (0.24 $\times$)	3.54 (0.40 $\times$)	4.22 (0.48 $\times$)
Sun Temple Interior	35.43 (1.00 $\times$)	23.69 (0.67 $\times$)	16.06 (0.45 $\times$)	25.79 (0.73 $\times$)	15.59 (0.44 $\times$)	27.76 (0.78 $\times$)	25.46 (0.72 $\times$)

Search can only find a locally optimal strategy as the search space is large.

2) *Denoise Module*: The denoise module is introduced to address instability during training at low samples per pixel (see Sec. V). Here, we focus on spatial denoising using the OptiX denoiser. Fig. 8 shows the error before and after applying the OptiX denoiser. It can be seen that the OptiX denoiser effectively reduces noise while preserving the main structure, and also decreases the absolute value of the error.

We conduct ablation studies on NRRS to evaluate its functionality. The results are shown in Tab. IV. It can be observed that always using the denoiser leads to a significant drop in performance, as the denoiser introduces bias into the error estimation. Conversely, disabling the denoiser results in unstable training, which often leads to NaN values. Our strategy achieves superior results in most scenes, demonstrating its effectiveness. However, we have not yet found the optimal allocation strategy for dividing time between denoising and non-denoising phases; we leave this for future work. For example, in Sponza Caustic, the denoiser is not beneficial.

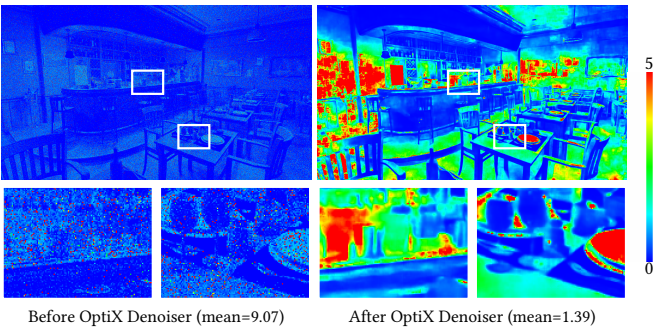


Fig. 8. **Training Errors visualization before/after applying OptiX denoiser.** The average error over the entire image is included.

3) *RRSNet*: Simply replacing the tree structure in EARS with a neural network can not achieve better performance, as shown in Tab. IV. We use two variants of EARS based on how the mean of the cost is handled: EARS (NN) predicts it jointly with StatNet (by adding a relative loss for the cost to Eq. (17)), while EARS (NT) uses an additional tree to

cache it. EARS (NN) performs poorly, likely because it is challenging for a simple MLP to accurately predict all three values. Although EARS (NT) can provide better RRS factors, the additional overhead of querying the tree means it still does not outperform EARS, let alone NRRS. Additionally, neither method can address the normalization problem mentioned in Sec. IV-A.

4) *Loss Design*: There are three loss functions for RRSNet: $\mathcal{L}_{\min}$, $\mathcal{L}_{\text{avg}}$, and $\mathcal{L}_{\text{rrs}}$. As mentioned in Sec. IV-B2, $\mathcal{L}_{\min}$ is necessary to reduce the error in the final image. $\mathcal{L}_{\text{rrs}}$ is used to bound the RRS factors and narrow the gap between pre- and post-normalization. Fig. 9 shows that, although the problem is not perfectly solved, the gap between pre- and post-normalization for NRRS is smaller than for EARS. This demonstrates that $\mathcal{L}_{\text{rrs}}$ is helpful. We also conduct an ablation study to show that our RRSNet does not simply learn a defensive mapping for RRS factors. Specifically, we replace $\mathcal{L}_{\text{rrs}}$ with

$$\mathcal{L}_{\text{defensive}} = \sum_{i=1}^{N_{\text{samples}}} (q_{\text{nrrs}}(\theta) - 1.0)^2,$$

where the ground truth for the bound loss is set to 1.0. As shown in Tab. IV, our design of $\mathcal{L}_{\text{rrs}}$ provides clear benefits. Furthermore, we conduct an ablation study (See Tab. IV) to show that $\mathcal{L}_{\text{avg}}$ can slightly improve performance.

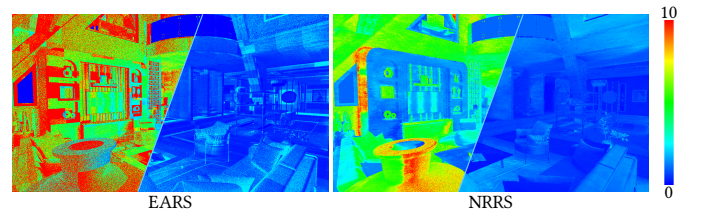


Fig. 9. **Visualization of the RRS factors for EARS (left) and NRRS (right), respectively.** For each method, the upper-left region shows the original RRS factors q_{orig} before applying the RRS module, and the lower-right region shows the factors q_{norm} after it. Both are computed at the primary ray intersection point prior to the RRS stage (Fig. 2).

5) *Pixel Estimations*: In the original EARS implementation, the accumulated buffer is denoised before calculating the

TABLE IV

ABLATION STUDY AND MORE COMPARISONS. "DENOISE ALWAYS" DENOTES THAT DENOISING IS ENABLED, WHEREAS "NO DENOISE" DENOTES THAT DENOISING IS DISABLED. EARS (D) CALCULATES THE RRS FACTOR USING THE ACCUMULATED BUFFER AFTER DENOISING. EARS (NN) AND EARS (NT) ARE TWO VARIANTS OF EARS: EARS (NN) USES A NEURAL NETWORK TO CACHE ALL STATISTICS, WHILE EARS (NT) USES A NEURAL NETWORK TO CACHE THE MEAN AND VARIANCE OF RADIANCE AND A TREE TO CACHE THE MEAN OF THE COST. " $-\mathcal{L}_{\text{avg}}$ " CORRESPONDS TO NRRS WITH $\mathcal{L}_{\text{avg}}$ DISABLED. " $+\mathcal{L}_{\text{defensive}}$ " INDICATES THAT $\mathcal{L}_{\text{RRS}}$ IS REPLACED WITH $\mathcal{L}_{\text{defensive}}$. ERRORS ARE REPORTED AS RELMSE ($\times 10^{-3}$). THE BEST RESULT FOR EACH ROW IS **HIGHLIGHTED**.

	NRRS	Denoise Always	No Denoise	EARS (D)	EARS	EARS (NN)	EARS (NT)	$-\mathcal{L}_{\text{avg}}$	$+\mathcal{L}_{\text{defensive}}$
Apartment1	2.34	4.05	nan	2.50	2.59	4.33	3.55	2.45	2.58
Apartment2	5.24	8.95	nan	6.44	6.75	7.35	6.54	5.31	5.55
Apartment3	8.85	15.01	nan	13.33	14.11	17.28	14.48	9.20	9.35
Bistro Interior	1.96	3.14	2.26	2.23	2.33	2.95	2.77	2.11	1.99
Sponza Caustic	2.82	6.34	2.74	3.11	3.13	5.46	4.84	2.92	3.16
Sun Temple Interior	14.92	23.04	nan	18.26	18.93	23.29	23.56	14.95	15.26

RRS factors. We also tests this setting. The results are shown in the EARS (D) column of Tab. IV. On average, this yields about 5% better performance than using the accumulated buffer without denoising. However, our NRRS method still performs better. More results are provided in the appendix.

C. Evaluation & Validation

1) *Encoding of Position*: In Fig. 5, a vase near the camera plane in Apartment2 is visible. Traditional tree-based structures struggle to capture statistical variations in such a small region under limited memory, whereas our neural network with hash-grid encoding handles this efficiently. Fig. 10A and Fig. 10B show the radiance cache from the tree structure and the neural network, respectively. The radiance from neural network is more accurate than that from the tree structure.

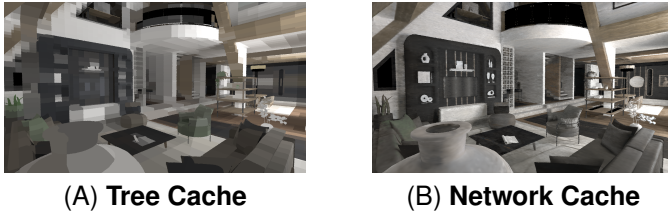


Fig. 10. **Radiance Cache Visualization.** Visualization of the radiance cache from the tree structure and the neural network. The left shows the radiance cache from the tree structure, and the right shows the radiance cache from the network. Both caches are queried at the primary ray intersection point.

2) *StatNet*: StatNet outputs the mean and variance of radiance. In our implementation, both the encoding and all layers of StatNet are shared between these outputs. We design several sharing strategies and compare them in Tab. V. The memory usage of each method is set to be approximately equal (see the appendix for details). The less the two outputs share, the less interference occurs between them; however, this increases both training and inference time. Each design represents a trade-off between accuracy and efficiency. For NRRS/NRRS+, StatNet is required during inference, and the results show that sharing all layers achieves the best performance. For AID-NRRS/AID-NRRS+, StatNet is not required during inference, so the trade-off only affects training. The results indicate that sharing all layers is not always the best choice in this case. We have not further explored which sharing strategy is optimal under

TABLE V

DIFFERENT DESIGN FOR STATNET. STATNET OUTPUTS THE MEAN AND VARIANCE OF RADIANCE. "ALL" INDICATES THAT BOTH THE ENCODING AND ALL LAYERS OF STATNET ARE SHARED; "PARTNET" MEANS THE ENCODING AND THE FIRST SEVERAL LAYERS ARE SHARED; "ENCODING" MEANS ONLY THE ENCODING IS SHARED; AND "NONE" MEANS NO SHARING. ERRORS ARE REPORTED AS RELMSE ($\times 10^{-3}$). THE BEST RESULTS ARE **HIGHLIGHTED** FOR EACH COLUMN.

	NRRS	NRRS+	AID-NRRS	AID-NRRS+
Bistro Interior				
All	1.96	1.11	1.64	1.27
PartNet	2.27	1.24	1.62	1.21
Encoding	2.18	1.13	1.96	1.11
None	2.50	1.30	2.12	1.19
Sun Temple Interior				
All	14.92	12.96	14.86	12.37
PartNet	16.38	17.45	15.49	13.02
Encoding	15.92	17.53	15.45	12.93
None	17.23	18.13	15.12	13.55

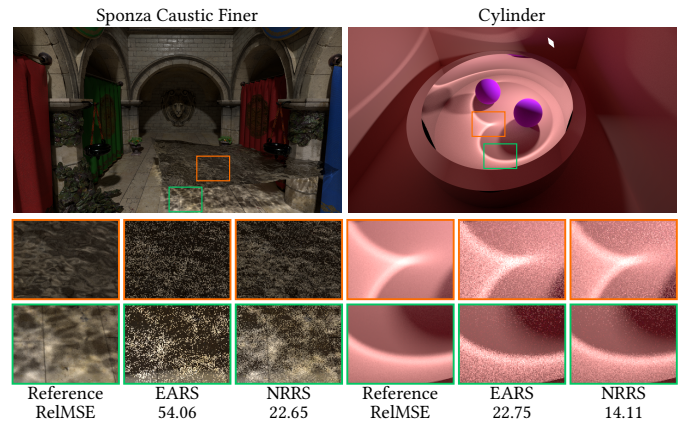


Fig. 11. **Detailed Caustics Scene.** Comparison of different methods in two detailed caustics. The settings are the same as in Fig. 5, and the RelMSE ($\times 10^{-3}$) is calculated **over the entire image**.

specific time constraints for AID-NRRS/AID-NRRS+, leaving this for future work.

3) *Caustics*: We evaluate our method on more challenging scenes featuring detailed caustics; the results are shown in Fig. 11. Sponza Caustic Finer is a more complex version of Sponza Caustic, with higher precision in the wave surface and a smaller area light. Cylinder is a scene with a mirror

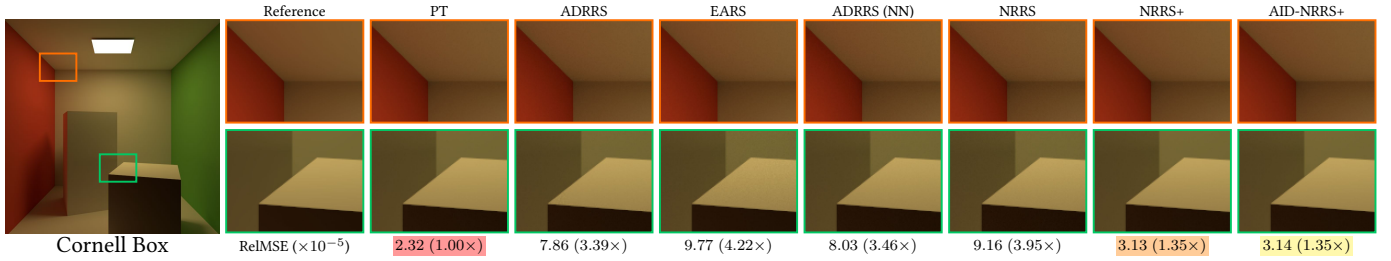


Fig. 12. **Failure Case.** Cornell Box is a simple scene with a light source. The experimental setting is the same as in Fig. 5, except the resolution is 750×750 . Under such simple geometry and illumination conditions, the original path tracing outperforms the other methods.

cylinder above a plane, illuminated by an area light. The results demonstrate that our method consistently outperforms EARS in these two challenging scenarios.

4) *Training Frames and Inference Frames:* Under the settings of Fig. 5, NRRS processes $\sim 1K$ frames during training, while EARS processes $\sim 2K$ frames. For inference, both methods process $\sim 2K$ frames. See the appendix for more details.

5) *Failure Cases:* Fig. 12 shows the result for the Cornell Box scene under simple lighting conditions. It can render approximately 160 frames per second with 1 ray per pixel in Kiraray. In this case, simple path tracing outperforms the other methods. With hardware acceleration, the overhead of the other techniques cannot be compensated. However, with our Mix-Depth design, NRRS+ can approximately fall back to simple path tracing.

VII. CONCLUSION, LIMITATION, AND FUTURE WORK

a) *Conclusion:* In this paper, we propose a new framework for integrating RRS techniques into wavefront path tracing. Since the theoretical foundations of traditional RRS methods do not hold under this framework, we introduce two neural RRS methods: NRRS and AID-NRRS. Both are carefully designed to address the challenges outlined above, with only subtle differences in their implementation. Our Mix-Depth technique can also combine multiple RRS methods with little overhead. To our knowledge, we are the first to introduce neural networks into RRS techniques, and this surpasses the state-of-the-art RRS method [4]. Our approach consistently shows the best results using fewer rays when compared to the alternative, indicating that each traced ray was utilized as effectively as possible in our approach.

b) *Limitation:* We summarize the main limitations of our method as follows:

- Practical approximations in NRRS may introduce suboptimality relative to the original formulation. First, estimating pixel variance from one sample per pixel (spp) is inherently noisy and may lead to unstable training without denoising. Second, while denoising improves stability, it inevitably introduces bias into the error estimation. Third, the predictions from StatNet, particularly variance estimate, can be inaccurate, which may prevent the RRS factors from converging to optimal values. Efficiently caching variance within the network remains challenging and is left for future work. Lastly, the one-spp estimation used in Mix-Depth can be inaccurate, potentially leading

to suboptimal strategy selection and preventing fallback to plain path tracing, as discussed in Sec. VI-C5. Importantly, these compromises affect only the efficiency of NRRS and do not compromise its unbiasedness. This is because, as long as the RRS factors are valid, the estimator defined in Eq. (3) remains an unbiased Monte Carlo estimator.

- Although our Mix-Depth technique is effective in practice, selecting an optimal strategy becomes increasingly difficult as the number of candidate strategies grows; even heuristic approaches may become impractical in such cases.
- NRRS, as well as prior RRS methods, is primarily designed for static scenes, as it relies on estimating pixel statistics in a manner similar to traditional approaches.

c) *Future Work:* Our work offers several straightforward extensions. ADRRS [2] can be further optimized by integrating path guiding with minimal overhead, as they share the same cache structure. NRRS can also be combined with other neural path guiding methods [6], [7], which can improve the quality of radiance samples and also enable joint optimization for better performance. Meyer et al. [13] learns to distribute samples across multiple sampling techniques, which we have not explored; this could further improve the efficiency of RRS methods, especially since NEE can be ineffective in many scenarios. As for Mix-Depth, a better mixing strategy would be to determine the RRS method at each scene intersection, rather than at each bounce. Currently, NRRS queries the network at every bounce. If we could instead query the network in primary space, we would only need to query once per path, allowing for more complex network designs and potentially better performance. This is also an interesting direction for future research. NRRS and prior RRS methods are primarily designed for static scenes, as they rely on the estimation of pixel values. Extending our method to dynamic scenes represents an important direction for future work, particularly in light of the promising results achieved by neural approaches such as NRC [5].

ACKNOWLEDGMENT

This work is partially supported by the Southern Marine Science and Engineering Guangdong Laboratory (Zhuhai) (No.SML2021SP101). It is also partially supported by the Shenzhen Science and Technology Program (No.

KJZD20230923114114028) and the Shandong Key Research and Development Program (No. 2024CXGC010801).

REFERENCES

- [1] J. T. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 143–150, Aug. 1986. [Online]. Available: <https://doi.org/10.1145/15886.15902>
- [2] J. Vorba and J. Krivánek, "Adjoint-driven russian roulette and splitting in light transport simulation," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016. [Online]. Available: <https://doi.org/10.1145/2897824.2925912>
- [3] S. Laine, T. Karras, and T. Aila, "Megakernels considered harmful: wavefront path tracing on gpus," in *Proceedings of the 5th High-Performance Graphics Conference*, ser. HPG '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 137–143. [Online]. Available: <https://doi.org/10.1145/2492045.2492060>
- [4] A. Rath, P. Grittmann, S. Herholz, P. Weier, and P. Slusallek, "Ears: efficiency-aware russian roulette and splitting," *ACM Trans. Graph.*, vol. 41, no. 4, Jul. 2022. [Online]. Available: <https://doi.org/10.1145/3528223.3530168>
- [5] T. Müller, F. Rousselle, J. Novák, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Trans. Graph.*, vol. 40, no. 4, Jul. 2021. [Online]. Available: <https://doi.org/10.1145/3450626.3459812>
- [6] H. Dong, G. Wang, and S. Li, "Neural parametric mixtures for path guiding," in *ACM SIGGRAPH 2023 Conference Proceedings*, ser. SIGGRAPH '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3588432.3591533>
- [7] J. Huang, A. Iizuka, H. Tanaka, T. Komura, and Y. Kitamura, "Online neural path guiding with normalized anisotropic spherical gaussians," *ACM Trans. Graph.*, vol. 43, no. 3, Apr. 2024. [Online]. Available: <https://doi.org/10.1145/3649310>
- [8] T. Müller, M. Gross, and J. Novák, "Practical path guiding for efficient light-transport simulation," *Computer Graphics Forum*, vol. 36, no. 4, pp. 91–100, 2017. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13227>
- [9] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák, "Neural importance sampling," *ACM Trans. Graph.*, vol. 38, no. 5, Oct. 2019. [Online]. Available: <https://doi.org/10.1145/3341156>
- [10] L. Ruppert, S. Herholz, and H. P. A. Lensch, "Robust fitting of parallax-aware mixtures for path guiding," *ACM Trans. Graph.*, vol. 39, no. 4, Aug. 2020.
- [11] A. Rath, M. Manzi, S. Weiss, T. Portenier, F. Salehi, S. Hadadan, and M. Papas, "Neural Resampling with Optimized Candidate Allocation," in *Eurographics Symposium on Rendering*, B. Wang and A. Wilkie, Eds. The Eurographics Association, 2025.
- [12] P. Figueiredo, Q. He, and N. K. Kalantari, "Neural Path Guiding with Distribution Factorization," in *Eurographics Symposium on Rendering*, B. Wang and A. Wilkie, Eds. The Eurographics Association, 2025.
- [13] J. Meyer, A. Rath, O. Yazici, and P. Slusallek, "Mars: Multi-sample allocation through russian roulette and splitting," in *SIGGRAPH Asia 2024 Conference Papers*, ser. SA '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: <https://doi.org/10.1145/3680528.3687636>
- [14] T. Müller, F. Rousselle, A. Keller, and J. Novák, "Neural control variates," *ACM Trans. Graph.*, vol. 39, no. 6, 11 2020. [Online]. Available: <https://doi.org/10.1145/3414685.3417804>
- [15] Q. Hua, P. Grittmann, and P. Slusallek, "Revisiting controlled mixture sampling for rendering applications," *ACM Trans. Graph.*, vol. 42, no. 4, Jul. 2023. [Online]. Available: <https://doi.org/10.1145/3592435>
- [16] I. Kondapaneni, P. Vevoda, P. Grittmann, T. Skřivan, P. Slusallek, and J. Krivánek, "Optimal multiple importance sampling," *ACM Trans. Graph.*, vol. 38, no. 4, 7 2019. [Online]. Available: <https://doi.org/10.1145/3306346.3323009>
- [17] R. West, I. Georgiev, A. Gruson, and T. Hachisuka, "Continuous multiple importance sampling," *ACM Trans. Graph.*, vol. 39, no. 4, aug 2020. [Online]. Available: <https://doi.org/10.1145/3386569.3392436>
- [18] R. West, I. Georgiev, and T. Hachisuka, "Marginal multiple importance sampling," in *SIGGRAPH Asia 2022 Conference Papers*, ser. SA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: <https://doi.org/10.1145/3550469.3555388>
- [19] J. Arvo and D. Kirk, "Particle transport and image synthesis," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: Association for Computing Machinery, 1990, pp. 63–66. [Online]. Available: <https://doi.org/10.1145/97879.97886>
- [20] W. Jakob, "Mitsuba Renderer," 2010. [Online]. Available: <https://mitsuba-renderer.org/>
- [21] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
- [22] L. Szirmay-Kalos, G. Antal, and M. Sbert, "Go with the winners strategy in path tracing," in *Journal of WSCG, Vol.13*, 01 2005, pp. 49–56.
- [23] M. R. Bolin and G. W. Meyer, "An error metric for monte carlo ray tracing," in *Rendering Techniques '97*, J. Dorsey and P. Slusallek, Eds. Vienna: Springer Vienna, 1997, pp. 57–68.
- [24] E. Veach, "Robust monte carlo methods for light transport simulation," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1998, aA19837162.
- [25] F. Su, B. Li, Q. Yin, Y. Zhang, and S. Li, "Proxy tracing: Unbiased reciprocal estimation for optimized sampling in bdpt," *ACM Trans. Graph.*, vol. 43, no. 4, Jul. 2024. [Online]. Available: <https://doi.org/10.1145/3658216>
- [26] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Noworzehzahr, and T. Aila, "Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017. [Online]. Available: <https://doi.org/10.1145/3072959.3073601>
- [27] A. T. Afra, "Intel® Open Image Denoise," 2025, <https://www.openimagedenoise.org>.
- [28] H. Fan, R. Wang, Y. Huo, and H. Bao, "Real-time monte carlo denoising with weight sharing kernel prediction network," *Computer Graphics Forum*, vol. 40, no. 4, pp. 15–27, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14338>
- [29] J. Yu, Y. Nie, C. Long, W. Xu, Q. Zhang, and G. Li, "Monte carlo denoising via auxiliary feature guided self-attention," *ACM Trans. Graph.*, vol. 40, no. 6, Dec. 2021. [Online]. Available: <https://doi.org/10.1145/3478513.3480565>
- [30] H. Xin, S. Zheng, K. Xu, and L.-Q. Yan, "Lightweight bilateral convolutional neural networks for interactive single-bounce diffuse indirect illumination," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 4, pp. 1824–1834, 2022.
- [31] M. Dereviannykh, D. Klepikov, J. Hanika, and C. Dachsbacher, "Neural two-level monte carlo real-time rendering," *Computer Graphics Forum*, vol. 44, no. 2, p. e70050, 2025. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.70050>
- [32] S. Fujieda, C. C. Kao, and T. Harada, "Neural Intersection Function," in *High-Performance Graphics - Symposium Papers*, J. Bikker and C. Gribble, Eds. The Eurographics Association, 2023.
- [33] P. Weier, A. Rath, E. Michel, I. Georgiev, P. Slusallek, and T. Boubekeur, "N-bvh: Neural ray queries with bounding volume hierarchies," in *ACM SIGGRAPH 2024 Conference Papers*, ser. SIGGRAPH '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: <https://doi.org/10.1145/3641519.3657464>
- [34] T. Müller, "tiny-cuda-nn," NVIDIA, 4 2021. [Online]. Available: <https://github.com/NVlabs/tiny-cuda-nn>
- [35] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2Noise: Learning image restoration without clean data," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2965–2974. [Online]. Available: <https://proceedings.mlr.press/v80/lehtinen18a.html>
- [36] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: a general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, Jul. 2010. [Online]. Available: <https://doi.org/10.1145/1778765.1778803>
- [37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: <http://arxiv.org/abs/1502.01852>
- [38] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022. [Online]. Available: <https://doi.org/10.1145/3528223.3530127>
- [39] G. E. P. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 26, no. 2, pp. 211–243, 12 2018. [Online]. Available: <https://doi.org/10.1111/j.2517-6161.1964.tb00553.x>
- [40] C. Dugas, Y. Bengio, F. Bélice, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," in *Proceedings of the 14th International Conference on Neural Information*

Processing Systems, ser. NIPS'00. Cambridge, MA, USA: MIT Press, 2000, p. 451–457.

- [41] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. [Online]. Available: <https://arxiv.org/abs/1412.6980>
- [42] A. Rath, P. Grittmann, S. Herholz, P. Weier, and P. Slusallek, “Ears: Efficiency-aware russian roulette and splitting,” 4 2022. [Online]. Available: <https://github.com/iRath96/ears>
- [43] B. Burley, “Extending disney’s physically based brdf with integrated subsurface scattering,” *SIGGRAPH’15 Courses*, 2015.



Sheng Li is currently a Professor at the School of Computer Sciences, Peking University. He serves as vice director of Graphics & Intelligence Technology Lab. and has published over 70 refereed papers in prestigious journals and conferences, including ACM TOG, SIGGRAPH/Asia, IEEE TVCG, CVPR, etc. His research interests include computer graphics and virtual reality. He is a member of the ACM and IEEE.



Haojie Jin is a doctoral student at the School of Computer Science, Peking University. He previously earned a Bachelor of Science degree from the School of Electronics Engineering and Computer Science at Peking University. His research focuses on photorealistic rendering and high-performance real-time rendering.



Jierui Ren is a doctoral student at College of Future Technology, Peking University. He previously earned a B.E. degree from Beihang University. His research focuses on neural rendering, global illumination, and 3D scene representation.



Yisong Chen is currently an Associate Professor at the School of Computer Sciences, Peking University. He works as a member of the Graphics & Intelligence Technology Lab. He has published over 50 refereed papers in prestigious journals and conferences, including IEEE TIP, IEEE TVCG, CVPR, ICCV, etc. His research interests include computer vision and pattern recognition.



Guoping Wang is currently a professor at Peking University, where he is also the director of Graphics & Intelligence Technology Laboratory. He achieved the National Science Fund for Distinguished Young Scholars in 2009. His research interests include Virtual Reality, Computer Graphics, Human-Computer Interaction, and Multimedia.

APPENDIX

A. PREFIX AND SUFFIX PATHS

These formulations are adopted from EARS [4]; similar formulations can be found in its “Background” section. Here, we explain the details of Eq. (4).

The pixel value can be written as an integral over all paths $\bar{\mathbf{z}}_k = \mathbf{z}_0 \cdots \mathbf{z}_k$, summed over all path lengths k ,

$$I_{\text{px}} = \sum_k \int_{\mathcal{Z}_k} W_{\text{px}}(\mathbf{z}_0, \mathbf{z}_1) T(\bar{\mathbf{z}}_k) L_e(\mathbf{z}_{k-1}, \mathbf{z}_k) d\mu(\bar{\mathbf{z}}_k), \quad (27)$$

where

$$T(\bar{\mathbf{z}}_k) = GV(\mathbf{z}_0, \mathbf{z}_1) \left[\prod_{j=1}^{k-1} f_r(\mathbf{z}_{j-1}, \mathbf{z}_j, \mathbf{z}_{j+1}) GV(\mathbf{z}_j, \mathbf{z}_{j+1}) \right], \quad (28)$$

with

$$GV(\mathbf{z}_l, \mathbf{z}_m) = V(\mathbf{z}_l, \mathbf{z}_m) \frac{|\cos \theta_{l,m}| |\cos \theta_{m,l}|}{\|\mathbf{z}_l - \mathbf{z}_m\|^2}. \quad (29)$$

The subscript of the integral domain denotes the path space of all paths with length k , L_e denotes the emission from $\mathbf{z}_k$ to $\mathbf{z}_{k-1}$, f_r is the bidirectional scattering distribution function (BSDF), $V(\mathbf{z}_l, \mathbf{z}_m)$ is the visibility term between $\mathbf{z}_l$ and $\mathbf{z}_m$, $d\mu(\bar{\mathbf{z}}_k) = dA(\mathbf{z}_0) \cdots dA(\mathbf{z}_k)$ is the differential area product, and $\theta_{l,m}$ represents the angle between the normal at $\mathbf{z}_l$ and the direction from $\mathbf{z}_l$ to $\mathbf{z}_m$.

Unidirectional path tracing (UPT) solves the integral by incrementally constructing a path $\bar{\mathbf{x}}$, starting from $\mathbf{x}_0$ on the camera. Its estimator can be expressed as Eq. (5). At depth k , we have a path prefix $\bar{\mathbf{x}}_k$, and the pixel value can be rewritten as

$$I_{\text{px}} = \int_{\mathcal{X}_k} W_{\text{px}}(\mathbf{x}_0, \omega_{i,0}) B(\bar{\mathbf{x}}_k) L_o(\mathbf{x}_k, \omega_{o,k}) d\bar{\mathbf{x}}_k, \quad (30)$$

where

$$B(\bar{\mathbf{x}}_k) = \prod_{j=1}^{k-1} f_r(\mathbf{x}_j, \omega_{o,j}, \omega_{i,j}) |\cos \theta_{j,j+1}|, \quad (31)$$

and

$$d\bar{\mathbf{x}}_k = \prod_{j=1}^k d\omega_{i,j}. \quad (32)$$

Here, $\omega_{o,j}$ and $\omega_{i,j}$ denote the outgoing and sampled incident directions at vertex $\mathbf{x}_j$, respectively.

The term L_o can also be expanded recursively to form the suffix $\mathbf{y}$ in a similar fashion.

Corresponding to Eq. (4) and Eq. (5),

$$g(\bar{\mathbf{x}}_k) = W_{\text{px}}(\mathbf{x}_0, \omega_{i,0}) B(\bar{\mathbf{x}}_k), \quad (33)$$

$$p(\bar{\mathbf{x}}_k) = p(\omega_{i,0} | \mathbf{x}_0) \prod_{j=1}^{k-1} p(\omega_{i,j} | \omega_{o,j}, \mathbf{x}_j), \quad (34)$$

and

$$H(\bar{\mathbf{x}}_k) = L_o(\mathbf{x}_k, \omega_{o,k}). \quad (35)$$

Note that UPT solves the rendering equation in a recursive manner. The L_o term in Eq. (30) represents the outgoing radiance, while the L_e term in Eq. (27) denotes the emission radiance. Therefore, the summation term does not appear in Eq. (30).

B. BOUNDEDNESS PROOFS FOR NORMALIZATION

(1) Modeling and single-path statistics. Each normalized RRS factor is decomposed as

$$q_{\text{norm}}^{(i)} = k_i + r_i, \quad r_i \in [0, 1),$$

and the number of generated children is sampled via the “integer-splitting + residual Bernoulli” rule:

$$X_i = k_i + B_i, \quad B_i \sim \text{Bernoulli}(r_i),$$

with $\{B_i\}$ conditionally independent given $\{q_{\text{norm}}^{(i)}\}$. Then,

$$\mathbb{E}[X_i | q_{\text{norm}}^{(i)}] = k_i + r_i = q_{\text{norm}}^{(i)}, \quad (36)$$

$$\text{Var}(X_i | q_{\text{norm}}^{(i)}) = r_i(1 - r_i) \leq \frac{1}{4}. \quad (37)$$

(2) Expectation boundedness. Using the normalization $q_{\text{norm}}^{(i)} = F_{\text{norm}} q_{\text{orig}}^{(i)}$ with $F_{\text{norm}} = \frac{N_{\text{px}}}{\sum_i q_{\text{orig}}^{(i)}}$, we have

$$\mathbb{E}[S | \{q_{\text{orig}}^{(i)}\}] = \sum_i \mathbb{E}[X_i | q_{\text{norm}}^{(i)}] = \sum_i q_{\text{norm}}^{(i)} = N_{\text{px}}.$$

Taking total expectation yields

$$\boxed{\mathbb{E}[S] = N_{\text{px}}}.$$

Thus, the expected number of generated samples is globally bounded and independent of scene complexity.

(3) Variance upper bound. Under conditional independence,

$$\text{Var}(S | \{q_{\text{norm}}^{(i)}\}) = \sum_i \text{Var}(X_i | q_{\text{norm}}^{(i)}) \quad (38)$$

$$= \sum_i r_i(1 - r_i) \leq \sum_i r_i = \rho. \quad (39)$$

Because $\sum_i q_{\text{norm}}^{(i)} = \sum_i k_i + \rho = N_{\text{px}}$, we obtain $\rho \leq N_{\text{px}}$, leading to

$$\boxed{\text{Var}(S | \{q_{\text{norm}}^{(i)}\}) \leq \rho \leq N_{\text{px}}}.$$

Using $r_i(1 - r_i) \leq \frac{1}{4}$ further gives a universal bound independent of scene statistics:

$$\boxed{\text{Var}(S | \{q_{\text{norm}}^{(i)}\}) \leq \frac{1}{4} |I|}.$$

□

C. DERIVATION OF THE OVERFLOW PROBABILITY BOUND.

After rate control, each normalized RRS factor is scaled as $q_{\text{norm}}'^{(i)} = f_{\text{rate}}(k_i + r_i)$. Each path generates samples following the integer-splitting model:

$$X'_i = k'_i + B'_i, \quad k'_i = \lfloor q_{\text{norm}}'^{(i)} \rfloor, \quad (40)$$

$$B'_i \sim \text{Bernoulli}(r'_i), \quad r'_i = q_{\text{norm}}'^{(i)} - k'_i \in [0, 1). \quad (41)$$

Let $S' = \sum_i X'_i = K' + Y'$, where $K' = \sum_i k'_i$ and $Y' = \sum_i B'_i$. From normalization, the expected number of samples is

$$\mathbb{E}[S'] = \sum_i q_{\text{norm}}'^{(i)} = f_{\text{rate}} \sum_i q_{\text{norm}}^{(i)} = f_{\text{rate}} N_{\text{px}}.$$

a) *Variance bound*: Since B'_i are conditionally independent,

$$\text{Var}(S') = \text{Var}(Y') = \sum_i r'_i(1-r'_i) \leq \sum_i r'_i =: \rho' \leq f_{\text{rate}}N_{\text{px}},$$

because $\sum_i q_{\text{norm}}^{(i)} = \sum_i k'_i + \rho' = f_{\text{rate}}N_{\text{px}}$.

b) *Deviation reformulation*: To analyze overflow,

$$S' \geq N_{\text{px}} \iff S' - \mathbb{E}[S'] \geq t,$$

where $t := N_{\text{px}} - \mathbb{E}[S'] = (1 - f_{\text{rate}})N_{\text{px}}$.

c) *Applying Bernstein's inequality*: For independent bounded random variables $B'_i \in [0, 1]$, Bernstein's one-sided inequality gives

$$\mathbb{P}(S' - \mathbb{E}[S'] \geq t) \leq \exp\left(-\frac{t^2}{2\text{Var}(S') + \frac{2}{3}t}\right).$$

Substituting $\text{Var}(S') \leq f_{\text{rate}}N_{\text{px}}$ and $t = (1 - f_{\text{rate}})N_{\text{px}}$ yields

$$\mathbb{P}(S' \geq N_{\text{px}}) \leq \exp\left(-\frac{(1 - f_{\text{rate}})^2 N_{\text{px}}}{2f_{\text{rate}} + \frac{2}{3}(1 - f_{\text{rate}})}\right),$$

which corresponds to Eq. (15). For $f_{\text{rate}} = 0.85$, the exponent is approximately $-0.0125 N_{\text{px}}$, indicating that overflow is virtually negligible even in moderate-resolution scenes.

D. IMPLEMENTATION DETAILS

A. Grid-Search of Hyperparameters

For RRSNet, we use three hyperparameters, γ_1 , γ_2 , and γ_3 , to control the weights of different loss components. We perform a grid search over Γ^3 (see Eq. (42)), enumerating all possible $|\Gamma|^3 = 1000$ combinations of γ_1 , γ_2 , and γ_3 . For each combination, we train StatNet for 10 seconds and then jointly train RRSNet and StatNet for 50 seconds. After training, we evaluate the efficiency and select the combination with the highest efficiency. The best combination we found is $\gamma_1 = 0.01$, $\gamma_2 = 0.05$, and $\gamma_3 = 0.01$.

$$\Gamma = \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5\} \quad (42)$$

Note that we select the best hyperparameters using only a single view from one scene (Fig. 13), and then apply this combination to all scenes. This single set of hyperparameters achieves good performance across different scenes, demonstrating the robustness of our approach. How to select the optimal combination of hyperparameters for each specific setting remains an interesting direction for future work.

We only choose best combination with one experimental setting, NRRS without denoising with the max depth 6. With this combination, we also achieve good performance in other settings, such as NRRS+, AID-NRRS, NRRS+ with max depth 10. It also shows the robustness of our approach.



Fig. 13. the bathroom scene used for grid-search.

B. Encoding for Roughness

The encodings of roughness differ slightly between StatNet and RRSNet, as described in Sec. V. This is because tcnn [34] pads the neural network input to a multiple of 16. We adjust the number of dimensions used for the one-blob encoding of the roughness feature to fill the remaining dimensions after encoding the other features.

For StatNet, the position $\mathbf{p} \in \mathbb{R}^3$ uses $16 = 8 \times 2$ dimensions, and the outgoing direction $\omega_o \in \mathbb{R}^2$ uses $8 = 4 \times 2$ dimensions. Therefore, we allocate 8 bins for the one-blob encoding of roughness $\alpha_p \in \mathbb{R}$, resulting in a total of 32 dimensions.

For AID-NRRS, the position $\mathbf{p} \in \mathbb{R}^3$ uses $16 = 8 \times 2$ dimensions, the outgoing direction $\omega_o \in \mathbb{R}^2$ uses $8 = 4 \times 2$ dimensions, throughput $t_x \in \mathbb{R}^3$ uses 3 dimensions, and the mean pixel color $I_{\text{pixel}} \in \mathbb{R}$ uses 1 dimension. Thus, we use 4 bins for the one-blob encoding of roughness $\alpha_p \in \mathbb{R}$ to pad the total to 32 dimensions.

For NRRS, we use a single kernel for all encodings, so we do not apply one-blob encoding to roughness for simplicity.

C. Training Time Distribution

In the main experiment, we train each method for 60 seconds. The main routine is described in Sec. V. The detailed time distribution are as follows:

- NRRS/AID-NRRS: 10 seconds for training StatNet; 45 seconds for training RRSNet and StatNet with denoising enabled; and 5 seconds for training RRSNet and StatNet without denoising. RRS is disabled during the first 10 seconds and then enabled.
- NRRS+/AID-NRRS+: 10 seconds for training StatNet; 35 seconds for training RRSNet and StatNet with denoising enabled; 5 seconds for training RRSNet and StatNet without denoising; and 10 seconds for Mix-Depth search. RRS is disabled during the first 10 seconds and then enabled.
- ADDRIS (NN): LNet is trained for 60 seconds. RRS is disabled during the first 10 seconds and then enabled.
- ADDRIS/EARS: The tree cache is trained for 60 seconds. RRS is disabled during the first 3 iterations and then enabled.
- EARS (NN): StatNet is trained for 60 seconds. RRS is disabled during the first 3 iterations and then enabled.

- EARS+: 50 seconds for training the tree cache and 10 seconds for Mix-Depth search. RRS is disabled during the first 3 iterations and then enabled.

D. Updates of Cache Structures

For our neural RRS methods, we update the neural network after every frame. For traditional RRS methods, we update the tree cache after every frame. For EARS, the number of frames in iteration i is 64×2^i .

E. Different Designs of StatNet

We provide additional details regarding the memory usage of different StatNet design. StatNet outputs both the mean and variance of radiance. The various sharing strategies are described as follows. Note that only the MLP size of StatNet is changed; the encoding size remains fixed, with an output dimension of 32. Since we use tiny-cuda-nn [34], the output dimension of each MLP is padded to a multiple of 16.

- **All:** Both the encoding and all layers of StatNet are shared. The network is an MLP with three linear layers of 32 neurons each, totaling 3584 floats ($3584 = 32 \times 32 + 32 \times 32 \times 2 + 32 \times 16$).
- **PartNet:** The encoding and the first two hidden linear layers of 32 neurons are shared between the mean and variance outputs, while each output has its own final layer of 32 neurons. This configuration uses 5120 floats in total ($5120 = 32 \times 32 + 32 \times 32 + 2 \times (32 \times 32 + 32 \times 16)$).
- **Encoding:** Only the encoding is shared. The mean and variance each have their own network, with two hidden linear layers of 32 neurons each. This design also uses 5120 floats in total ($5120 = 2 \times (32 \times 32 + 32 \times 32 + 32 \times 16)$).
- **None:** No sharing is used. The mean and variance each have their own separate encoding and MLP, with two hidden linear layers of 32 neurons each, resulting in 5120 floats in total ($5120 = 2 \times (32 \times 32 + 32 \times 32 + 32 \times 16)$).

E. MORE RESULTS OF MAIN EXPERIMENT

In addition to the error maps shown in Fig. 7, we supplement the error maps of the other four scenes in Fig. 16. We also visualize the typical error distribution for the Sponza Caustic scene, which shows that the error distribution of EARS is lower and smoother than PT in the Fig. 20.

F. ADDITIONAL SCENES

A. Diverse Settings

We validate our approach using additional scenes with the same setting as the main experiment (Fig. 5).

- Measure One is an indoor scene featuring metallic and specular surfaces, illuminated by several area lights.
- Pool Large is an **outdoor** scene with a large swimming pool, lit by an environment map.
- San Miguel is also an **outdoor** scene lit by an environment map.

- Sponza Glossy is a modified version of the original Sponza scene, featuring a glossy floor and a glossy dragon, illuminated by an area light.
- Sun Temple is the same scene as used in the main experiment, but shown from a different exterior viewpoint.
- Tropical Bedroom is a simple indoor scene illuminated by an area light.

The result are shown in Fig. 14, and Fig. 15 is the corresponding RelMSE convergence curve.

B. Detailed Texture Ablations

We also test our methods on scenes with smooth texture details by modifying the Sponza Caustic scene. For all texture images (except normal maps), we set each opaque texel's value to the average of all opaque texel values in the image, resulting in smooth rather than high-frequency textures.

We use the same setting as the main experiment (Fig. 5). The result is shown in Fig. 17. We can see that both our methods and EARS achieve better performance, with a larger relative reduction in relMSE compared to PT than in the original scene. This is because neither traditional nor neural methods perform well at capturing high-frequency details (note that our StatNet does not use albedo or normal as input). Under these circumstances, our methods can also outperform EARS.

G. RESULTS FOR DEEPER DEPTH

We provide the visual result in Fig. 18, corresponding to Tab. II. The error maps for the main experiment with a maximum depth of 10 are shown in Fig. 21, and the RelMSE convergence curve is presented in Fig. 19.

H. UNSTABLE TERM $p(\mathbf{x})$

Our material model is the “Disney Principled BSDF” [43], and we use BSDF sampling as our base sampling technique. We test the Bathroom scene with a path depth of 6, obtaining $p(\mathbf{x})$ values under RRS= 1.0 (i.e., the simplest UPT setting). The 5%, 20%, 50%, 80%, and 95% quantiles are 4.27e-6, 2.18e-5, 1.31e-4, 1.19e-3, and 1.79e-2, respectively. We can see that $p(\mathbf{x})$ exhibits a wide range of values even when RRS is off, which is not suitable for network training. With our “Gradient Transfer” design, the $p(\mathbf{x})$ term is removed, making the training process much more stable.

I. TRAINING FRAMES AND INFERENCE FRAMES

Here, we report the number of training frames and inference frames used under the main experimental setup (Fig. 5), where both training and inference are performed for 60 seconds. The experimental results shown in Fig. 5 only consider the frames obtained during the 60 seconds of inference. This is partly because images generated during training are very noisy, and partly because as the inference time increases, the influence of frames obtained during training on the final result diminishes.

The statistics are shown in Tab. VII. During inference, our neural method NRRS generate fewer frames compared to traditional approaches like EARS, since neural network

	Reference	PT	ADRRS	EARS	ADRRS (NN)	NRRS+	AID-NRRS+
Measure One							
	RelMSE ($\times 10^{-3}$) RayEffInv	23.25 (1.00 $\times$) 610 (1.00 $\times$)	20.70 (0.89 $\times$) 392 (0.64 $\times$)	19.52 (0.84 $\times$) 404 (0.66 $\times$)	19.25 (0.83 $\times$) 330 (0.54 $\times$)	17.68 (0.76 $\times$) 305 (0.50 $\times$)	17.10 (0.74 $\times$) 346 (0.57 $\times$)
Pool Large							
	RelMSE ($\times 10^{-3}$) RayEffInv	1.93 (1.00 $\times$) 46 (1.00 $\times$)	2.09 (1.08 $\times$) 38 (0.81 $\times$)	1.53 (0.79 $\times$) 35 (0.75 $\times$)	1.93 (1.00 $\times$) 33 (0.71 $\times$)	1.36 (0.71 $\times$) 31 (0.66 $\times$)	1.37 (0.71 $\times$) 31 (0.66 $\times$)
San Miguel							
	RelMSE ($\times 10^{-3}$) RayEffInv	1.65 (1.00 $\times$) 34 (1.00 $\times$)	1.53 (0.93 $\times$) 21 (0.63 $\times$)	1.51 (0.92 $\times$) 21 (0.63 $\times$)	1.10 (0.67 $\times$) 14 (0.43 $\times$)	0.81 (0.49 $\times$) 13 (0.38 $\times$)	0.80 (0.49 $\times$) 13 (0.39 $\times$)
Sponza Glossy							
	RelMSE ($\times 10^{-3}$) RayEffInv	1.92 (1.00 $\times$) 63 (1.00 $\times$)	1.98 (1.03 $\times$) 41 (0.66 $\times$)	2.13 (1.11 $\times$) 55 (0.88 $\times$)	1.82 (0.95 $\times$) 33 (0.53 $\times$)	1.36 (0.71 $\times$) 32 (0.51 $\times$)	1.30 (0.68 $\times$) 29 (0.46 $\times$)
Sun Temple							
	RelMSE ($\times 10^{-3}$) RayEffInv	16.27 (1.00 $\times$) 412 (1.00 $\times$)	16.72 (1.03 $\times$) 290 (0.70 $\times$)	18.27 (1.12 $\times$) 358 (0.87 $\times$)	14.44 (0.89 $\times$) 235 (0.57 $\times$)	10.53 (0.65 $\times$) 205 (0.50 $\times$)	10.27 (0.63 $\times$) 199 (0.48 $\times$)
Tropical Bedroom							
	RelMSE ($\times 10^{-3}$) RayEffInv	2.61 (1.00 $\times$) 93 (1.00 $\times$)	1.57 (0.60 $\times$) 39 (0.42 $\times$)	1.55 (0.59 $\times$) 43 (0.46 $\times$)	1.58 (0.60 $\times$) 34 (0.37 $\times$)	1.29 (0.50 $\times$) 31 (0.34 $\times$)	1.29 (0.49 $\times$) 31 (0.33 $\times$)

Fig. 14. Additional testing scenes using the same setting as main experiment (Fig. 5).

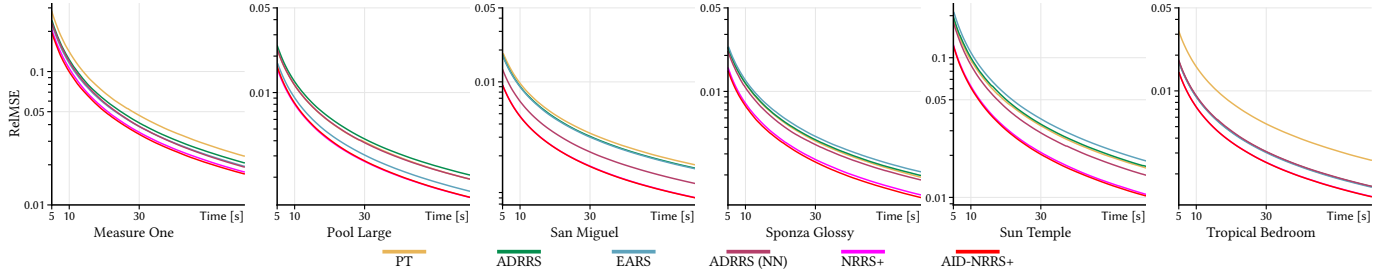


Fig. 15. RelMSE convergence curve corresponds to Fig. 14. Notably, in the Tropical Bedroom scene, the curves of the different methods almost completely overlap.

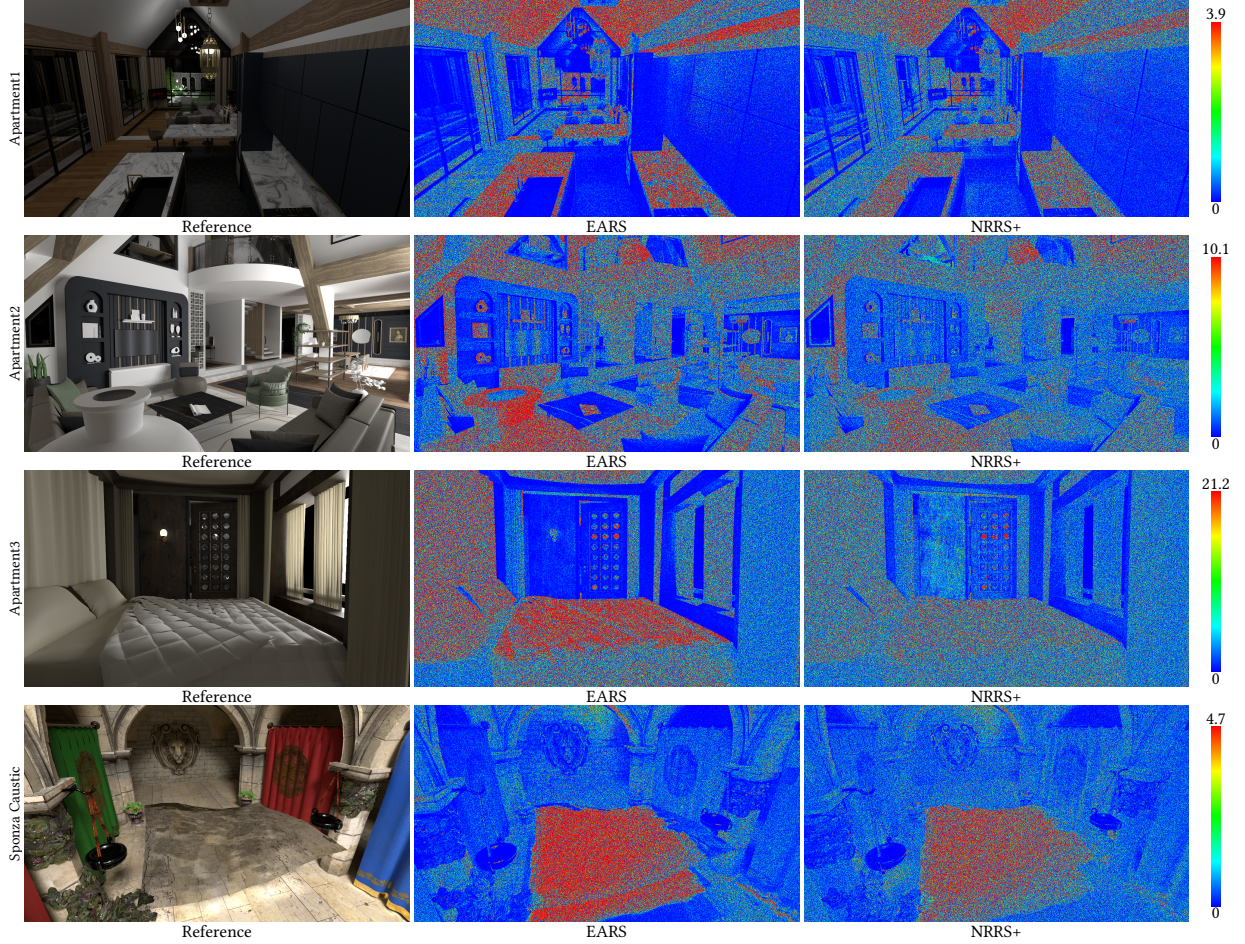


Fig. 16. **EARS Vs. NRRS+ in terms of RelMSE ($\times 10^{-3}$)**. For each scene, the maximum tint value (corresponding to red) is set to 1.5 times the largest RelMSE across all methods.

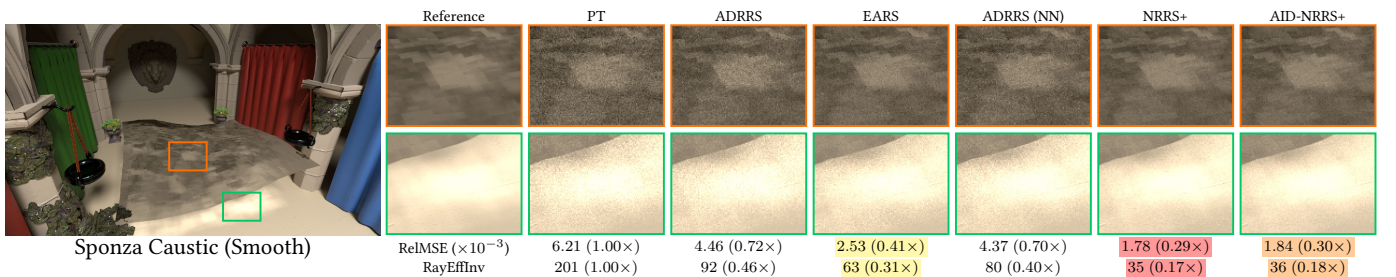


Fig. 17. Evaluation on scenes with smooth textures under the same setting as the main experiment (Fig. 5)

	Reference	PT	ADRRS	EARS	ADRRS (NN)	NRRS+	AID-NRRS+
Apartment1							
	RelMSE ($\times 10^{-3}$)	5.60 (1.00 $\times$)	4.56 (0.81 $\times$)	3.25 (0.58 $\times$)	3.34 (0.60 $\times$)	2.69 (0.48 $\times$)	2.37 (0.42 $\times$)
	RayEffInv	123 (1.00 $\times$)	65 (0.53 $\times$)	62 (0.50 $\times$)	45 (0.37 $\times$)	40 (0.33 $\times$)	36 (0.29 $\times$)
Apartment2							
	RelMSE ($\times 10^{-3}$)	12.38 (1.00 $\times$)	9.36 (0.76 $\times$)	9.27 (0.75 $\times$)	6.96 (0.56 $\times$)	5.77 (0.47 $\times$)	5.80 (0.47 $\times$)
	RayEffInv	299 (1.00 $\times$)	158 (0.53 $\times$)	198 (0.66 $\times$)	110 (0.37 $\times$)	99 (0.33 $\times$)	96 (0.32 $\times$)
Apartment3							
	RelMSE ($\times 10^{-3}$)	20.86 (1.00 $\times$)	14.66 (0.70 $\times$)	21.04 (1.01 $\times$)	11.55 (0.55 $\times$)	8.94 (0.43 $\times$)	9.33 (0.45 $\times$)
	RayEffInv	477 (1.00 $\times$)	224 (0.47 $\times$)	409 (0.86 $\times$)	162 (0.34 $\times$)	134 (0.28 $\times$)	140 (0.29 $\times$)
Bistro Interior							
	RelMSE ($\times 10^{-3}$)	4.49 (1.00 $\times$)	3.83 (0.85 $\times$)	3.10 (0.69 $\times$)	2.04 (0.45 $\times$)	1.30 (0.29 $\times$)	1.30 (0.29 $\times$)
	RayEffInv	125 (1.00 $\times$)	56 (0.45 $\times$)	72 (0.58 $\times$)	28 (0.23 $\times$)	21 (0.17 $\times$)	21 (0.17 $\times$)
Sponza Caustic							
	RelMSE ($\times 10^{-3}$)	8.78 (1.00 $\times$)	5.27 (0.60 $\times$)	3.54 (0.40 $\times$)	4.57 (0.52 $\times$)	2.17 (0.25 $\times$)	2.15 (0.24 $\times$)
	RayEffInv	286 (1.00 $\times$)	102 (0.36 $\times$)	85 (0.30 $\times$)	82 (0.29 $\times$)	43 (0.15 $\times$)	44 (0.15 $\times$)
Sun Temple Interior							
	RelMSE ($\times 10^{-3}$)	35.43 (1.00 $\times$)	24.87 (0.70 $\times$)	27.76 (0.78 $\times$)	18.48 (0.52 $\times$)	16.06 (0.45 $\times$)	15.59 (0.44 $\times$)
	RayEffInv	969 (1.00 $\times$)	435 (0.45 $\times$)	590 (0.61 $\times$)	296 (0.31 $\times$)	290 (0.30 $\times$)	281 (0.29 $\times$)

Fig. 18. Main Experiment with Depth 10, corresponding to Tab. II.

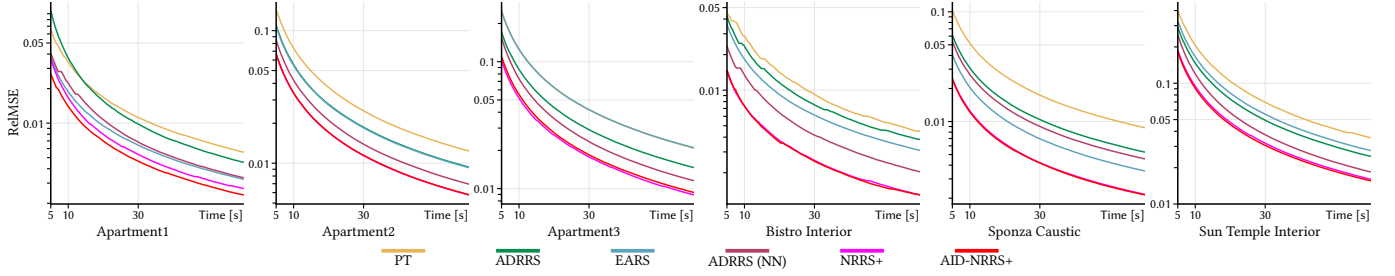


Fig. 19. RelMSE convergence curve corresponds to Fig. 18.

TABLE VI

MORE RESULTS OF PIXEL ESTIMATES. EARS (D) CALCULATES THE RRS FACTOR USING THE ACCUMULATED BUFFER AFTER DENOISING. ADRRS (D) CALCULATES THE RRS FACTOR USING THE DENOISED ACCUMULATED BUFFER IN ADRRS. THE SETTINGS ARE THE SAME WITH FIG. 5. ERRORS ARE REPORTED AS RELMSE ($\times 10^{-3}$). THE BEST RESULT FOR EACH ROW IS **HIGHLIGHTED**.

	PT	NRRS	ADRRS	ADRRS (D)	EARS	EARS (D)
Apartment1	3.93 (1.00 $\times$)	2.34 (0.60$\times$)	3.31 (0.84 $\times$)	3.16 (0.80 $\times$)	2.59 (0.66 $\times$)	2.50 (0.64 $\times$)
Apartment2	8.58 (1.00 $\times$)	5.24 (0.61$\times$)	9.00 (1.05 $\times$)	8.28 (0.97 $\times$)	6.75 (0.79 $\times$)	6.44 (0.75 $\times$)
Apartment3	14.52 (1.00 $\times$)	8.85 (0.61$\times$)	12.64 (0.87 $\times$)	11.77 (0.81 $\times$)	14.11 (0.97 $\times$)	13.33 (0.92 $\times$)
Bistro Interior	2.75 (1.00 $\times$)	1.96 (0.71$\times$)	3.12 (1.13 $\times$)	2.85 (1.04 $\times$)	2.33 (0.85 $\times$)	2.23 (0.81 $\times$)
Sponza Caustic	6.06 (1.00 $\times$)	2.82 (0.47$\times$)	4.71 (0.78 $\times$)	4.41 (0.73 $\times$)	3.13 (0.52 $\times$)	3.11 (0.51 $\times$)
Sun Temple Interior	21.72 (1.00 $\times$)	14.92 (0.69$\times$)	19.31 (0.89 $\times$)	18.06 (0.83 $\times$)	18.93 (0.87 $\times$)	18.26 (0.84 $\times$)

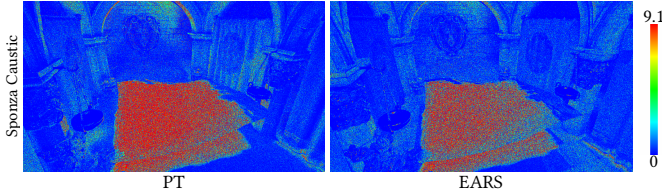


Fig. 20. **EARS vs. PT in terms of RelMSE ($\times 10^{-3}$).** For each scene, the maximum tint value (corresponding to red) is set to 1.5 times the largest RelMSE across all methods.

TABLE VII

TRAINING FRAMES AND INFERENCE FRAMES. SHOWN ARE THE NUMBERS OF TRAINING FRAMES AND INFERENCE FRAMES CONSUMED BY NRRS AND EARS IN THE MAIN EXPERIMENTAL SETTING (WITH 60S FOR TRAINING AND 60S FOR INFERENCE) IN FIG. 5.

Scene	NRRS		EARS	
	training	inference	training	inference
Apartment1	985	2146	2177	2426
Apartment2	684	2071	2057	2371
Apartment3	748	1867	1832	2030
Bistro Interior	902	2335	2355	2799
Sponza Caustic	1039	2498	2601	3212
Sun Temple Interior	666	1988	1954	2250

inference is slower than querying a tree structure. When it comes to training, NRRS processes even fewer frames because backpropagation and network parameter updates are more time-consuming. Nonetheless, as demonstrated in Fig. 5, our neural method NRRS outperforms EARS when the inference time is kept the same.

J. MORE RESULT OF PIXEL ESTIMATES

In Sec. VI-B5, we present the results for EARS (D), which uses the denoised accumulated buffer to calculate RRS factors. Here, we additionally provide results for ADRRS (D), which

denotes ADRRS using the denoised accumulated buffer for RRS factor calculation. The results are shown in Tab. VI, under the same settings as Fig. 5. We can see that our method, NRRS, still outperforms all other methods even without using the denoised accumulated buffer.

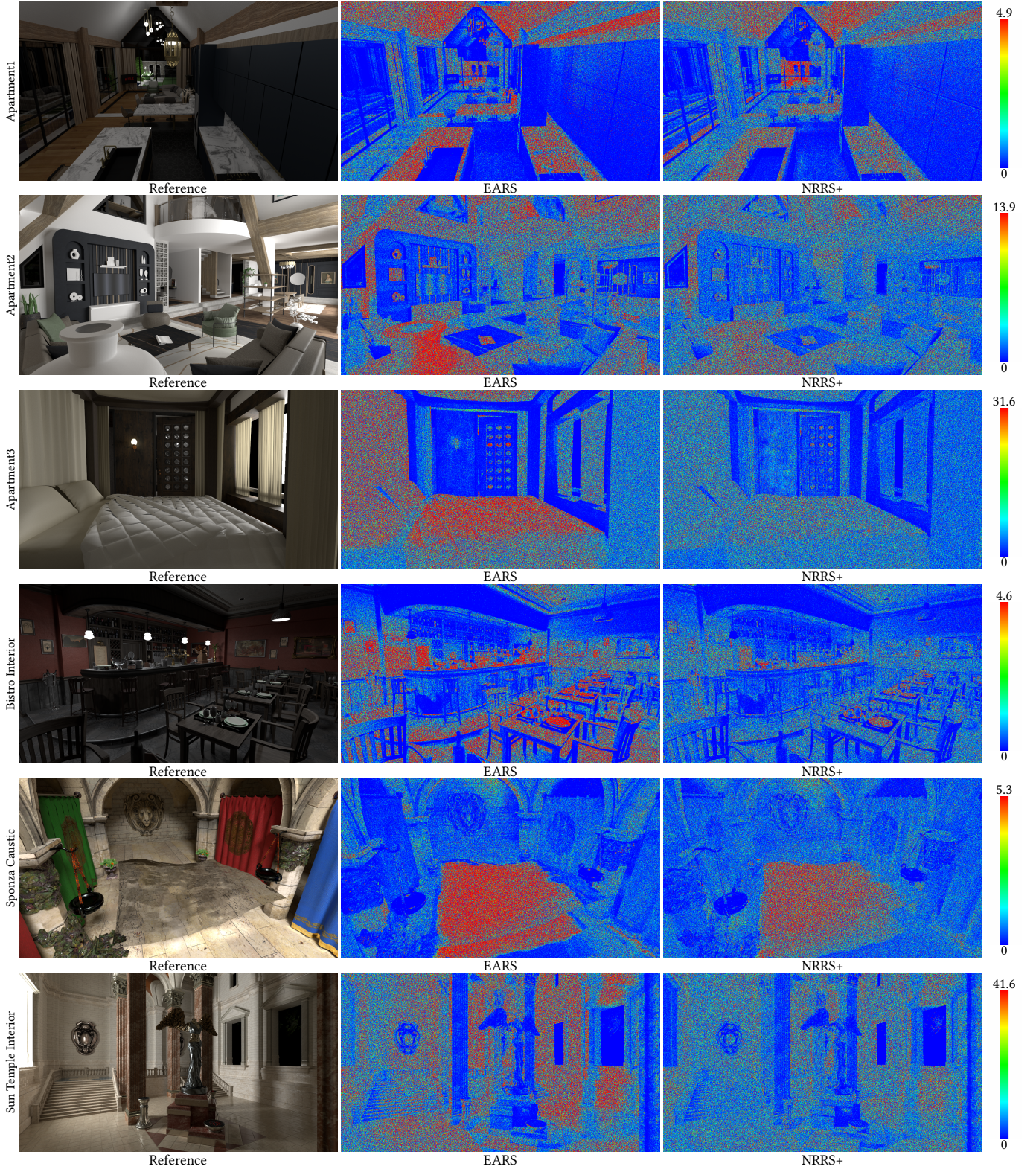


Fig. 21. EARS Vs. NRRS+ with Depth 10 in terms of RelMSE ($\times 10^{-3}$). For each scene, the maximum tint value (corresponding to red) is set to 1.5 times the largest RelMSE across all methods.