EARS: Efficiency-Aware Russian Roulette and Splitting

ALEXANDER RATH, Saarland University, Germany and German Research Center for Artificial Intelligence DFKI), Germany
PASCAL GRITTMANN, Saarland University, Germany
SEBASTIAN HERHOLZ, Intel Corporation, Germany
PHILIPPE WEIER, Saarland University, Germany and German Research Center for Artificial Intelligence DFKI), Germany
PHILIPP SLUSALLEK, Saarland University, Germany and German Research Center for Artificial Intelligence DFKI), Germany

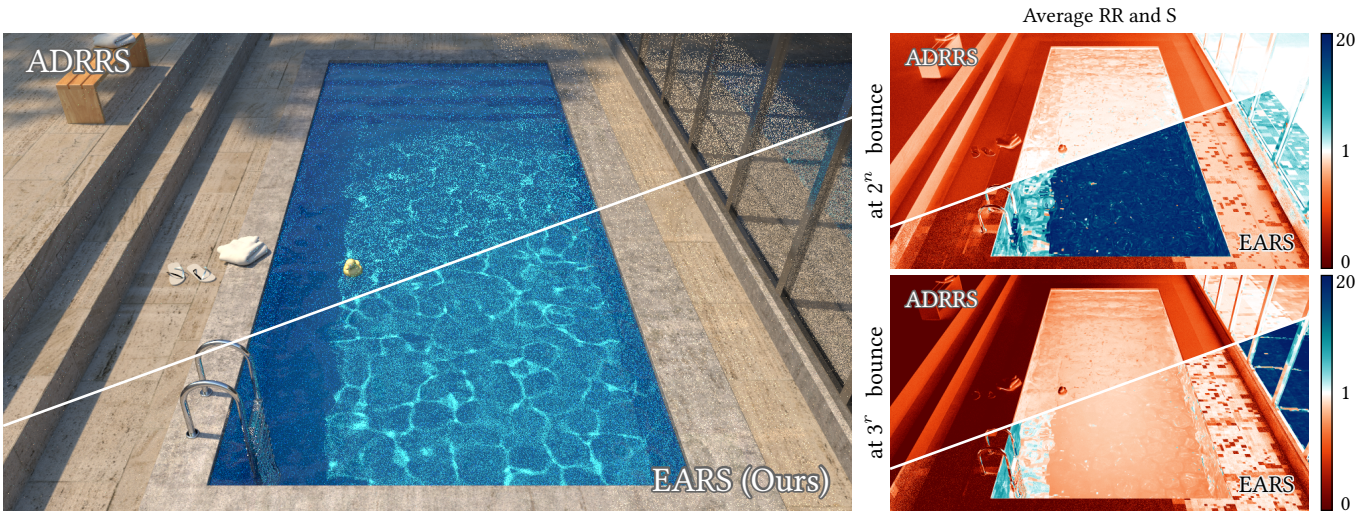


Fig. 1. We derive a fixed-point iteration to compute the Russian roulette and splitting RRS) factors that maximize the rendering efficiency. Here, we compare the rendered images in equal-time 10 min) of our method and the state of the art, *adjoint driven Russian roulette and splitting* ADRRS) [Vorba and Křivánek 2016]. The false-color images on the right visualize the average RRS factors in each pixel at the second and third bounce. Red indicates that mostly roulette is played; blue indicates that mostly splitting is done. By directly optimizing variance and cost, our method produces more efficient RRS decisions: splitting is mostly performed at the bottom of the pool and on the diffuse surface behind the window, which dominate the image variance.

Russian roulette and splitting are widely used techniques to increase the efficiency of Monte Carlo estimators. But, despite their popularity, there is little work on how to best apply them. Most existing approaches rely on simple heuristics based on, e.g., surface albedo and roughness. Their efficiency often hinges on user-controlled parameters. We instead iteratively learn optimal Russian roulette and splitting factors during rendering, using a simple and lightweight data structure. Given perfect estimates of variance and cost, our fixed-point iteration provably converges to the optimal Russian roulette and splitting factors that maximize the rendering efficiency. In our application to unidirectional path tracing, we achieve consistent and significant speed-ups over the state of the art.

CCS Concepts: • **Computing methodologies** → **Ray tracing**.

Additional Key Words and Phrases: global illumination, importance sampling, path guiding

ACM Reference Format
Alexander Rath, Pascal Grittmann, Sebastian Herholz, Philippe Weier, and Philipp Slusallek. 2022. EARS: Efficiency-Aware Russian Roulette and Splitting. *ACM Trans. Graph.* 41, 4, Article 81 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530168

Authors' addresses: Alexander Rath, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany and German Research Center for Artificial Intelligence (DFKI), Saarland Informatics Campus, Saarbrücken, Germany, rath@cg.uni-saarland.de; Pascal Grittmann, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany, grittmann@cg.uni-saarland.de; Sebastian Herholz, Intel Corporation, Karlsruhe, Germany, sebastian.herholz@intel.com; Philippe Weier, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany and German Research Center for Artificial Intelligence (DFKI), Saarland Informatics Campus, Saarbrücken, Germany, weier@cg.uni-saarland.de; Philipp Slusallek, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany and German Research Center for Artificial Intelligence (DFKI), Saarland Informatics Campus, Saarbrücken, Germany, philipp.slusallek@dfki.de.

0730-0301/2022/7-ART81
https://doi.org/10.1145/3528223.3530168

1 INTRODUCTION

In the past decades, advances to physically-based rendering have been tremendous, allowing us to render increasingly large scenes with complex illumination. Currently, most rendering algorithms are based on unidirectional or bidirectional path tracing, which uses random walks to explore the light transport in a scene.

Much research has been done on increasing the efficiency of path tracing methods, mostly focused on variance reduction through importance sampling. Efficiency can also be increased by better allocating the available computation budget. In most path tracing methods, considerable computational power is wasted to trace long paths that contribute little to the final image. This effect worsens as more complex importance sampling methods are employed, since these usually generate a high computational overhead.

Two ubiquitous methods to increase the efficiency of path tracing algorithms are Russian roulette (RR) and splitting. On the one hand, RR reduces costs by stochastically terminating paths that are expected to have a low contribution to the image, which, if done right, only leads to a slight increase in variance. On the other hand, splitting reduces the variance of a local estimate by continuing a prefix path with multiple suffix paths. If most of the variance is due to that local sampling decision, splitting greatly reduces the variance without incurring the cost of repeatedly sampling the prefix path.

Optimally performing Russian roulette and splitting (RRS) has received little attention. A very promising approach is adjoint-driven RRS [Vorba and Křivánek 2016], which bases the RRS decisions on estimates of the expected image contribution. While producing great results, the expected contribution does not consider variance or cost of local estimators, let alone the global efficiency. We, instead, derive RRS decisions that maximize efficiency and show how they can numerically be solved using a fixed-point iteration based on estimates of variance and cost stored in a simple 5D data structure. The difference is shown in Fig. 1. Our method encourages splitting when reaching the caustics at the bottom of the pool and behind the windows as they have high variance and highly benefit from splitting. On the other hand, by only considering the expected contribution of the local estimator and not its variance, ADRRS only ensures the survival of these paths but does not perform any splitting. This is visible in the false-color images on the right, comparing the average per-pixel RRS factors for both methods at different depths.

We derive a fixed-point iteration to numerically find the Russian roulette and splitting (RRS) factors that maximize efficiency, and prove that it converges (Section 4). The method is general and applies to any Monte Carlo method that can perform RRS. Applied to rendering, we introduce an online learning scheme that iteratively improves the RRS factors over time (Section 5). In practice, we achieve consistent speed-ups over ADRRS of up to 4× (Section 6).

The Mitsuba [Jakob 2010] implementation of our method is publicly available at https://github.com/iRath96/ears [Rath et al. 2022].

2 RELATED WORK

The predominant method to drive Russian roulette (RR) or splitting decisions is to base them on heuristics based on approximations of the expected path contribution. Typically, in conjunction with user-controlled parameters such as a minimum path length.

Albedo-based. The typical RR approach in practice is to base the survival probability on the surface albedo [Arvo and Kirk 1990] or currently accumulated path throughput weight [Pharr et al. 2016]. Szécsi et al. [2003] take spectral properties of the albedo into account and show that variance can be reduced by returning reflected radiance estimates upon path termination. Szirmay-Kalos [2005] combines throughput weight, albedo, and surface roughness in a heuristic that controls both RR and splitting. The underlying assumption of these approaches is that paths that have scattered across multiple dark surfaces are unlikely to yield a high image contribution. Unfortunately, this assumption is often violated in practice, e.g., due to complex indirect illumination or bright light sources illuminating dark surfaces. This can lead to premature termination of important paths, which increases variance. A typical workaround is to only use RR beyond a certain minimum path length (e.g., 5), subject to manual control by the user.

Approximated contributions. An alternative is to base the survival probability or splitting factor on an approximation of the full image contribution. Efficiency-optimized RR [Veach 1997] randomly skips shadow ray evaluations by comparing the unoccluded contribution to a threshold. Taking the idea a step further, adjoint-driven Russian roulette and splitting (ADRRS) [Vorba and Křivánek 2016] uses estimates of the expected incident radiance in a scene and compares them to approximations of the pixel value. Thereby, ADRRS resolves the problem of early path terminations faced by albedo and throughput weight-based RR methods, providing significant improvements, e.g., in scenes dominated by indirect illumination via dark surfaces. The main shortcoming of ADRRS is that it only considers the expected contributions, not the costs or variances. As a result, ADRRS cannot directly reduce the variance due to a poor sampling decision (e.g., inside a caustic). Paths are only split after the poor decision has already been made, effectively bounding the subsequent variance. Our method is conceptually very similar to ADRRS, only we directly maximize the efficiency by including information about local variances and costs. This allows us to directly split at the source of variance, resulting in up to 4× faster renderings.

Efficiency analysis. Bolin and Meyer [1997] provide an analysis of the efficiency when applying RR and splitting (RRS) to a path tracer. They derive formulas that can be used to compute the optimal RRS factor per pixel and path length. By operating in image space, they forego the potential of controlling the RRS factor based on the actual prefix path at hand. If a pixel receives contributions from an unimportant and an important region at the same depth, the RRS factor is shared for both, which is suboptimal. Our method is based in part on their derivations, but modified such that the splitting factors are optimized for each prefix path. Further, we make the computations practical by employing a fixed-point iteration instead of trying to compute the factors directly.

Adaptive sampling. Another common method to control rendering efficiency is via adaptive sampling in image space [Zwicker et al. 2015]. There is a plethora of such methods with very different goals and approaches. What they have in common is that the number of samples per pixel are controlled automatically, based on, e.g., variance estimates. Russian roulette and, in particular, splitting can be seen as a path space extension of such adaptive sampling methods.

Fixed-point iterations in rendering. Our method uses a fixed-point iteration to incrementally update the RRS factors. Fixed-point iterations have been used in other rendering applications, most prominently radiosity methods [Goral et al. 1984], that employ them to

iteratively compute the light transport in a scene. Similarly, the recent work of Deng et al. [2021] has employed fixed-point iterations in a reconstruction method that propagates contributions across paths. A key difference in our application is that the fixed-point iteration is used solely to optimize sampling decisions, which does not introduce bias in the result.

3 BACKGROUND

The rendering equation [Kajiya 1986] can be written as an integral over all paths $\bar{\mathbf{z}}$ connecting the pixel px to a point on a light source, which can be estimated via Monte Carlo integration:

$$I_{\text{px}} = \int f_{\text{px}}(\bar{\mathbf{z}})\,\mathrm{d}\bar{\mathbf{z}}, \qquad \langle I_{\text{px}}\rangle = \frac{f_{\text{px}}(\bar{\mathbf{z}})}{p(\bar{\mathbf{z}})}. \tag{1}$$

Here, $f_{\text{px}}(\bar{\mathbf{z}})$ is the contribution of the path $\bar{\mathbf{z}}$ to the pixel, and $\langle I_{\text{px}}\rangle$ is a single-sample Monte Carlo estimator using a random path $\bar{\mathbf{z}}$ with probability density $p(\bar{\mathbf{z}})$ to form an unbiased estimate of the integral.

Efficiency. The efficiency of an estimator is the inverse of the product of its variance and its expected computational cost [Pharr et al. 2016]. We extend this to multi-integral cases, such as rendering an image with path tracing, by using the mean pixel variance and expected pixel render time,

$$\epsilon^{-1} = \left(\frac{1}{N_{\text{px}}}\sum_{\text{px}} \mathbb{V}\left[\langle I_{\text{px}}\rangle\right]\right)\left(\frac{1}{N_{\text{px}}}\sum_{\text{px}} \left[c\ \langle I_{\text{px}}\rangle\right]\right), \tag{2}$$

where N_{px} is the number of pixels, and the pixel variance is the difference between the expectation of the square and the squared expectation of the estimator,

$$\mathbb{V}\left[\langle I_{\text{px}}\rangle\right] = \left[\langle I_{\text{px}}\rangle^2\right] - I_{\text{px}}^2. \tag{3}$$

The pixel render time can be modelled as the expectation of a *cost function* $c(\langle I_{\text{px}}\rangle)$ measuring the cost of evaluating the Monte-Carlo estimator $\langle I_{\text{px}}\rangle$ for a given pixel px. A simple heuristic to measure this cost is to count the number of traced rays.

Prefix paths. For our discussion of Russian roulette and splitting, we rewrite the pixel value as an integral over all *prefix paths* $\bar{\mathbf{x}}_k = \mathbf{x}_0 \ldots \mathbf{x}_k$, summed over all path lengths k,

$$I_{\text{px}} = \sum_k \int t_{\text{px}}(\bar{\mathbf{x}}_k)\, L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1})\,\mathrm{d}\bar{\mathbf{x}}_k, \tag{4}$$

where $t_{\text{px}}(\bar{\mathbf{x}}_k)$ is the throughput, i.e., the product of sensor response, pixel contribution, BSDFs, and geometry terms [Veach 1997]. L_{r} is the reflected radiance at point $\mathbf{x}_k$ towards the previous point $\mathbf{x}_{k-1}$, which is itself a high-dimensional integral given by the rendering equation.

Prefix weight. Forward path tracing operates by incrementally constructing a path $\bar{\mathbf{x}}$, starting from $\mathbf{x}_0$ on the camera. At depth k, we have a path prefix $\bar{\mathbf{x}}_k$ with corresponding prefix weight

$$T(\bar{\mathbf{x}}_k) = \frac{W_{\text{px}}(\mathbf{x}_0, \omega_{\text{i},0})}{p(\mathbf{x}_0, \omega_{\text{i},0})} \prod_{j=1}^{k-1} \frac{\rho(\mathbf{x}_j, \omega_{\text{i},j}, \omega_{\text{o},j})\,|\cos\theta_{\text{i},j}|}{p(\omega_{\text{i},j} \mid \mathbf{x}_j, \omega_{\text{o},j})}, \tag{5}$$

where W_{px} is the sensor response, $\rho(\mathbf{x}, \omega_{\text{i}}, \omega_{\text{o}})$ denotes the BSDF, and $p(\omega_{\text{i}} \mid \mathbf{x}, \omega_{\text{o}})$ the probability density of the directions sampled at each vertex $\mathbf{x}$ to continue the path. The prefix is then combined with an estimate of the reflected radiance to form an MC estimator

$$\langle I_{\text{px}}\rangle = T(\bar{\mathbf{x}}_k)\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1})\rangle. \tag{6}$$

Russian roulette (RR). RR instead randomly decides whether to construct the nested estimate, or terminate the path and return zero

$$\langle I_{\text{px}}; q\rangle = \begin{cases} T(\bar{\mathbf{x}}_k)\frac{\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1})\rangle}{q(\bar{\mathbf{x}}_k)} & \text{if survived} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $q(\bar{\mathbf{x}}_k)$ is the *survival probability*. If done right, RR increases efficiency by terminating unimportant paths, allowing more computation time to be invested in important regions. But it also increases the variance and thus has to be used carefully.

Splitting. Splitting reduces the variance by continuing the prefix path $\bar{\mathbf{x}}_k$ with $n(\bar{\mathbf{x}}_k)$ independent suffix samples for the nested estimator,

$$\langle I_{\text{px}}; n\rangle = T(\bar{\mathbf{x}}_k) \sum^{n(\bar{\mathbf{x}}_k)} \frac{\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1})\rangle}{n(\bar{\mathbf{x}}_k)}. \tag{8}$$

This reduces the variance due to the nested integral for L_{r} by a factor of $1/n(\bar{\mathbf{x}}_k)$, while avoiding the cost of generating a new prefix $\bar{\mathbf{x}}_k$ for each such suffix sample. Splitting is effective if most of the variance is due to the $\langle L_{\text{r}}\rangle$ estimate, or if an unlikely prefix found a high-contribution region.

Russian roulette and splitting (RRS). The highest efficiency increase can be achieved if both RR and splitting are combined into a single framework, which also makes it easier to optimize [Vorba and Křivánek 2016]. This can be achieved by computing a non-integer splitting factor $s(\bar{\mathbf{x}}_k) \in \mathbb{R}^+$ and using a *stochastic rounding* function r to convert it to an integer value:

$$r(s(\bar{\mathbf{x}}_k)) = \begin{cases} \lfloor s(\bar{\mathbf{x}}_k)\rfloor + 1 & \text{with probability } s(\bar{\mathbf{x}}_k) - \lfloor s(\bar{\mathbf{x}}_k)\rfloor \\ \lfloor s(\bar{\mathbf{x}}_k)\rfloor & \text{otherwise.} \end{cases} \tag{9}$$

This leads to an RRS estimator:

$$\langle I_{\text{px}}; s\rangle = T(\bar{\mathbf{x}}_k) \sum^{r(s(\bar{\mathbf{x}}_k))} \frac{\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1})\rangle}{s(\bar{\mathbf{x}}_k)}, \tag{10}$$

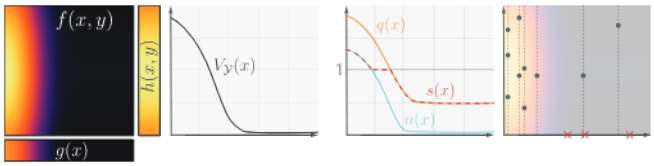
which is very similar to the pure splitting estimator (8), with the key difference that the summation uses stochastic rounding while the estimates are still divided by the real-valued RRS factor.

Nested RRS. The nested recursive L_{r} estimator can, of course, also perform RRS. We denote such an estimator as $\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); s\rangle$ and the pixel estimator with RRS at multiple depths as:

$$\langle I_{\text{px}}; s\rangle = T(\bar{\mathbf{x}}_k) \sum^{r(s(\bar{\mathbf{x}}_k))} \frac{\langle L_{\text{r}}(\mathbf{x}_k, \mathbf{x}_{k-1}); s\rangle}{s(\bar{\mathbf{x}}_k)}. \tag{11}$$

The definition of $s(\bar{\mathbf{x}})$ determines if and where splitting happens. The prefix weight of these subsequent RRS decisions then also contains the previous RRS factors,

$$T(\bar{\mathbf{x}}_k) = \frac{W_{\text{px}}(\mathbf{x}_0, \omega_{\text{i},0})}{p(\mathbf{x}_0, \omega_{\text{i},0})} \prod_{j=1}^{k-1} \frac{\rho(\mathbf{x}_j, \omega_{\text{i},j}, \omega_{\text{o},j})\,|\cos\theta_{\text{i},j}|}{s(\bar{\mathbf{x}}_j)\,p(\omega_{\text{i},j} \mid \mathbf{x}_j, \omega_{\text{o},j})}. \tag{12}$$



a) Integrand b) Variance c) RRS factors d) Samples

Fig. 2. Russian roulette and splitting in a simplified 2D example. a) the integrand $f(x, y)$ is the product of two functions, $g(x)$ and $h(y)$. b) the variance is highest in the high-contribution region on the left. c) the optimal RR, splitting, or combined RRS factor for each prefix x, computed via our fixed-point iteration. d) when using the optimized RRS factors, samples in the low-variance region are terminated red crosses), while samples in the high-variance region are split one batch of samples per dashed lines).

Note that in our notation $\langle L_r(\mathbf{x}_k, \mathbf{x}_{k\ 1}); s\rangle$ is itself still a primary estimator, i.e., only using a single sample for the incident radiance at $\mathbf{x}_k$. In our derivations, we will exclusively use nested RRS estimators.

4 EFFICIENCY-AWARE RRS

We determine the optimal RRS factors that, given a prefix path $\bar{\mathbf{x}}_k$ decide if and with how many samples to continue the path in order to maximize rendering efficiency. To this end, we show how the resulting equations can be efficiently solved using a fixed-point iteration that converges to the optimal RRS factors. In the following, we first introduce the method in a simplified setting, then show how it can be applied in a rendering context.

Without loss of generality, we first consider a 2D integration problem, illustrated in Fig. 2. The integrand is modelled after the rendering equation and consists of the product of two functions, the prefix $g(x)$ and suffix $h(x, y)$,

$$I = \int_{\mathcal{X}} g(x) \int_{\mathcal{Y}} h(x, y)\, \mathrm{d}y\, \mathrm{d}x =: \int_{\mathcal{X}} g(x) H(x)\, \mathrm{d}x. \tag{13}$$

Given a prefix x sampled with density $p(x)$, the goal is to find the optimal number of splits, or the optimal Russian roulette (RR) probability, to continue the 'path' by sampling suffixes y_i.

We start by deriving a fixed-point iteration to compute the optimal splitting factors, and prove that it converges. Then, we show that the same approach can be applied to RR, as well as a combined RRS.

4.1 Optimal splitting

A nested splitting estimator $\langle I; n\rangle$ for I takes a single prefix sample $x \in \mathcal{X}$ and combines it with $n(x)$ suffix samples $y_j \in \mathcal{Y}$:

$$\langle I; n\rangle = \frac{g(x)}{p(x)} \sum_{j=1}^{n(x)} \frac{h(x, y_j)}{n(x) p(y_j \mid x)}. \tag{14}$$

For optimal splitting at x, we need the optimal *splitting function* $n(x) : \mathcal{X} \to \mathbb{N}$, which gives us a positive integer splitting count for each prefix x.

4.1.1 Objective. Efficiency is maximized when the product of variance and cost is minimized,

$$\arg\min_{n} \mathbb{V}\left[\langle I; n\rangle\right] \quad [c(\langle I; n\rangle)]\,. \tag{15}$$

Under the assumption that the suffix samples y_i are uncorrelated, we can use the law of total variance to express the variance of Eq. (13) as the sum of two components [Bolin and Meyer 1997],

$$\mathbb{V}\left[\langle I; n\rangle\right] = V_{\mathcal{X}} + \left[\frac{V_{\mathcal{Y}}(x)}{n(x)}\right], \tag{16}$$

where

$$V_{\mathcal{X}} := \mathbb{V}\left[\frac{g(x)}{p(x)} H(x)\right] \tag{17}$$

is the variance due to sampling the prefix x, if given a ground truth value $H(x)$ for the nested integral, and

$$V_{\mathcal{Y}}(x) := \mathbb{V}\left[\langle I\rangle \mid x\right] = \left(\frac{g(x)}{p(x)}\right)^2 \mathbb{V}\left[\langle H(x)\rangle \mid x\right] \tag{18}$$

is the variance, without any splitting, due to sampling a single suffix y when given a prefix x.

The cost is modeled as the expectation of a cost function c,

$$[c(\langle I; n\rangle)] = \quad [c(x) + n(x) c(\langle H(x)\rangle)]\,, \tag{19}$$

where we assume that the cost is linear in the number of samples, i.e., it can be split into a sum of the cost $c(x)$ due to sampling of the prefix, and the cost $c(\langle H(x)\rangle)$ due to sampling the suffix.

4.1.2 Optimization. To solve Eq. (15), we introduce an approximation: We pretend that the splitting factors do not have to be integers. That is, we allow $n(x) : \mathcal{X} \to \mathbb{R}^+$. This allows us to compute derivatives and perform convex optimization, but it also introduces a small error, since the result has to be rounded to a positive integer, either exactly or stochastically. The result then has a slightly different variance than predicted during the optimization[1].

For a real-valued splitting function, the efficiency is a convex functional of $n(x)$, as it can be written as a sum of convex functions, which is by definition also convex. Hence, it must have a unique global minimum. The question is, how can we find it efficiently?

In principle, we can set the partial functional derivatives to zero,

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I; n\rangle\right] \quad [c(\langle I; n\rangle)]}{\mathrm{d}n(x)} = 0, \tag{20}$$

and solve the resulting system of equations. The derivative is easy enough to compute analytically, as shown in Appendix A, but analytically solving the system of equations is not practical, at least not in a full rendering context. Asides from requiring some integrals that are challenging to estimate, the problem is also a recursive one, as splitting can happen at other points along the prefix and suffix.

4.1.3 Fixed point iteration. Our solution is to numerically find the solution to Eq. (20) via a fixed-point iteration. A primer on using fixed-point iterations for root finding, and proving their convergence, can be found in Appendix B. The key idea is that, instead of solving (20) directly, we formulate a fixed-point iteration

$$n_i(x) = \ _S(n_{i\ 1}(x)) = \sqrt{\frac{V_{\mathcal{Y}}(x)}{\mathbb{V}\left[\langle I; n_{i\ 1}\rangle\right]} \frac{[c(\langle I; n_{i\ 1}\rangle)]}{[c(\langle H(x)\rangle) \mid x]}}, \tag{21}$$

where the ith iteration computes the splitting factor $n_i(x)$ based on the variances and costs of the previous iteration's configuration. In Appendix C, we show that the unique fixed-point $_S(n(x)) = n(x)$

[1] This approximation error is also present in the previous work of Bolin and Meyer [1997].

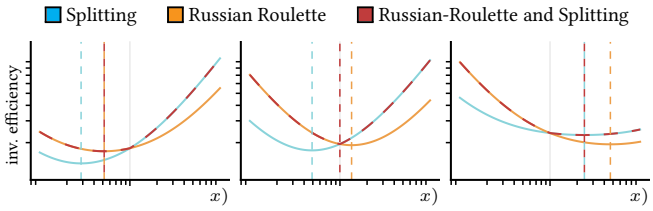


Fig. 3. Examples visualizing the shape of our objective function, here in 1D for a single point x with corresponding $s(x)$. The splitting blue) and RR orange) objectives are both convex and have a unique local minimum vertical lines). There are three possible cases, shown from left to right: RR is optimal, doing neither is optimal, and splitting is optimal. By definition, the RR objective and the splitting objective intersect at $s(x) = 1$. The RR variance is greater than the splitting variance for $s(x)$ 1, and vice versa for $s(x) > 1$. Hence, the combined objective dashed red curve) is also convex, and both local minima always lie in the correct portion of the domain i.e., below or above 1). Unless 1 is the minimum, then both lie in the opposite region. These properties guarantee the convergence of the joint fixed-point iteration.

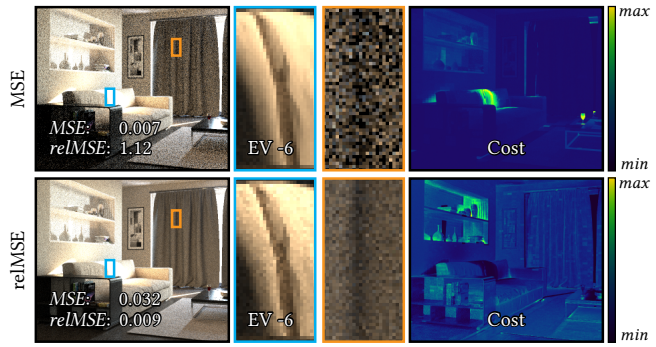


Fig. 4. Minimizing the mean-squared error MSE) or relative MSE relMSE) in EARS. Using the relMSE performs significantly better in scenes with high contrast. By decreasing the exposure EV -6) of a crop we can see that using the MSE oversamples very bright regions. Using the relMSE, on the other hand, yields better convergence across the entire dynamic range of the image, as we can see when looking at the average per-pixel cost, shown in false-color. With the MSE, most computation time is invested in the bright pixels. With the relMSE, computation time is spread more evenly.

is the optimal $n(x)$ and prove that iteratively applying $n_i(x) =$ $(n_{i\ 1}(x))$, starting with an arbitrary initial guess $n_0(x)$, converges to the optimum.

4.2 Incorporating Russian roulette

A very similar fixed-point iteration can be derived to find the optimal Russian roulette (RR) probability $q(x)$. The result can be combined with the optimal splitting $n(x)$ into an optimal RRS decision $r(x)$.

4.2.1 Optimal RR. First, consider the case where instead of splitting, only RR is allowed. Performing RR at a prefix x increases the variance [Bolin and Meyer 1997],

$$\mathbb{V}_{\mathrm{RR}}\left[\langle I; q\rangle\right] = V_{\mathcal{X}} + \left[\frac{M_{\mathcal{Y}}(x)}{q(x)} \left(\frac{g(x)}{p(x)} H(x)\right)^2\right], \tag{22}$$

where the suffix moment

$$M_{\mathcal{Y}}(x) := \left(\frac{g(x)}{p(x)}\right)^2 \left[\langle H(x)\rangle^2 \mid x\right] \tag{23}$$

is the expectation of the squared estimator value, given the prefix x.

The effect on our fixed-point function (21) is rather minor. If we re-compute the derivatives with the RR variance, the only change is that the suffix variance $V_{\mathcal{Y}}$ is replaced by the suffix moment $M_{\mathcal{Y}}$,

$$q_{i+1}(x) = \ _{\mathrm{RR}}(q_i(x)) = \sqrt{\frac{M_{\mathcal{Y}}(x)}{\mathbb{V}\left[\langle I; q_i\rangle\right]} \frac{\left[c(\langle I; q_i\rangle)\right]}{\left[c(\langle H(x)\rangle) \mid x\right]}}, \tag{24}$$

The objective is still convex, and the fixed-point iteration still converges to the unique optimum, since, from a mathematical point of view, we merely swapped one constant for another.

4.2.2 Optimal RRS. We can jointly optimize splitting and RR, by first combining them into a piece-wise variance

$$\mathbb{V}_{\mathrm{RRS}}\left[\langle I; s\rangle\right] = V_{\mathcal{X}} + \left[R(s(x))\right], \tag{25}$$

where we define

$$R(s(x)) := \begin{cases} \frac{V_{\mathcal{Y}}(x)}{s(x)} & \text{if } s(x) > 1 \\ \frac{M_{\mathcal{Y}}(x)}{s(x)} \left(\frac{g(x)}{p(x)} H(x)\right)^2 & \text{else.} \end{cases} \tag{26}$$

The product of this piece-wise RRS variance and the cost (which remains unchanged) is still a convex functional in $s(x)$, as illustrated in Fig. 3. The figure shows examples for the three possible configurations of the joint objective. The minimum can be either the optimal RR value, the optimal splitting value, or $s(x) = 1$.

The minimum can be found with a similar approach to the one suggested by Bolin and Meyer [1997]. First, we compute the splitting factor $n(x)$. If the result is greater than one, that is our optimum. Otherwise, we compute the RR factor $q(x)$ and clamp it to one, to handle the case where the optimal decision is exactly one (which is a discontinuity in our objective function). This can be written as a joint fixed-point function:

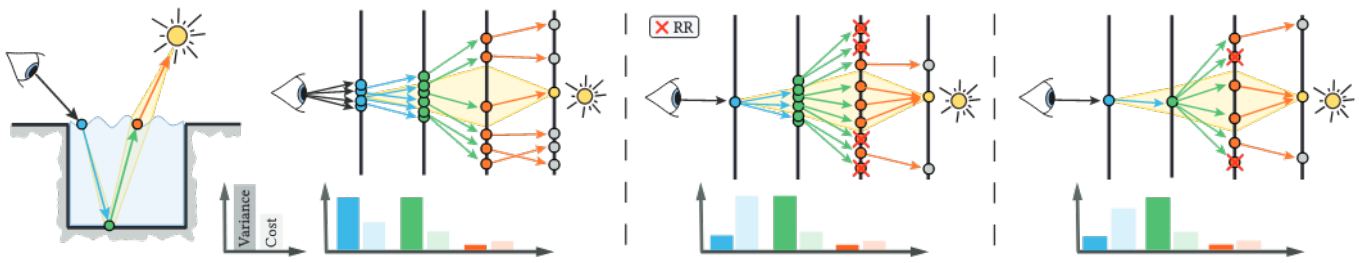
$$_{\mathrm{RRS}}(s(x)) = \begin{cases} \ _{\mathrm{S}}(s(x)) & \text{if } \ _{\mathrm{S}}(s(x)) > 1 \\ \min\{\ _{\mathrm{RR}}(s(x)), 1\} & \text{otherwise.} \end{cases} \tag{27}$$

The convexity of the joint objective, in combination with the convergence of the individual components, guarantees that the fixed-point iteration converges.

4.3 Application to rendering

The theory discussed so far can be directly applied to rendering. In the following, we discuss how to do so in the context of forward path tracing. Compared to the simplified setting, there are two major differences: (1) instead of a single integral, there is one per pixel, and (2) RRS occurs multiple times along a path.

4.3.1 Objective. A rendered image consists of multiple integrals, one for each pixel; the goal is to maximize the efficiency across all these integrals, as defined in Eq. (2). That is, the goal is to obtain local splitting factors that maximize the total efficiency over the



a) Pool scene b) Initial data / statistic pass c) Fixed-point iteration 1 d) Fixed-point iteration 2

Fig. 5. Illustration of the fixed-point update behavior for a set of nested estimators blue, green, and orange) along a caustic path. a) shows the scene setup, which is similar to the Pool scene Fig. 1). The yellow region marks the subset of paths that constitute the caustic. b), c), and d) show the sampling behavior top) and the estimated variances and costs bottom) of the nested estimators at different stages: the initial training iteration b) as well as the first c), and second d) fixed-point iteration.

entire image. Fortunately, this is still a convex objective (being a sum of convex functions), and the derivatives have the exact same form as the single integral case discussed so far.

There is, however, one more consideration to make. Using the absolute variance will overfit on bright pixels. Instead, we minimize the product of average relative variance and average per-pixel cost:

$$\bar{V}(n)\bar{C}(n) := \left(\frac{1}{N_{\text{px}}}\sum_{\text{px}}^{N}\frac{\mathbb{V}\left[\langle I_{\text{px}};n\rangle\right]}{I_{\text{px}}^2}\right)\left(\frac{1}{N_{\text{px}}}\sum_{\text{px}}^{N}\left[c(\langle I_{\text{px}};n\rangle)\right]\right). \tag{28}$$

Using the relative variance, i.e., dividing by the squared ground truth I_{px}^2, prevents an oversampling of bright regions. In other words, instead of aiming for the lowest mean squared error (MSE), we aim for the lowest *relative* mean squared error (relMSE). Fig. 4 compares the difference between the two objectives. The scene has high variance everywhere, but the directly illuminated region on the couch is very bright and completely dominates the MSE. Hence, minimizing the MSE results in splitting factors that focus most of the samples on that region. Minimizing the relMSE instead produces a much more uniform distribution of sampling cost and hence a more uniform noise.

A problem with this objective is that the ground truth pixel value is unknown in practice. We will later show that a coarse approximation of the ground truth pixel value (e.g., using denoised intermediate renders) is a sufficient surrogate of this value.

4.3.2 Fixed point function. Following the same steps as the simplified 2D example discussed before, we can formulate the RRS fixed-point functions for the ith fixed-point iteration

$$(n_i(\bar{\mathbf{x}}_k)) = \underbrace{\frac{T(\bar{\mathbf{x}}_k)}{I_{\text{px}}(\bar{\mathbf{x}}_k)}}_{\text{prefix}}\underbrace{\sqrt{\frac{R(\langle L_{\text{r}}(\mathbf{x}_k,\mathbf{x}_{k\ 1});n_i\rangle)}{\left[c(\langle L_{\text{r}}(\mathbf{x}_k,\mathbf{x}_{k\ 1});n_i\rangle)\right]}}}_{\text{local}}\underbrace{\sqrt{\frac{\bar{C}(n_i)}{\bar{V}(n_i)}}}_{\text{global}} \tag{29}$$

where

$$R(\langle L_{\text{r}}(\mathbf{x}_k,\mathbf{x}_{k\ 1});n_i\rangle) = \begin{cases}\mathbb{V}\left[\langle L_{\text{r}}(\mathbf{x}_k,\mathbf{x}_{k\ 1});n_i\rangle\right] & \text{if } n_i(x) > 1\\ \left[\langle L_{\text{r}}(\mathbf{x}_k,\mathbf{x}_{k\ 1});n_i\rangle^2\right] & \text{else}\end{cases} \tag{30}$$

is the piece-wise moment or variance function, computing the *primary*, i.e., single sample, variance or second moment of the reflected radiance estimator at point $\mathbf{x}_k$.

The fixed-point function consists of three components: the relative prefix weight $T(\bar{\mathbf{x}}_k)/I_{\text{px}}$ which is readily available, the local ratio of nested variance and cost, which can be cached in a 5D data structure, and the global variance and cost of the entire image.

The beauty of this approach is that we can perform a fixed-point update of a *continuous* RRS function $n(\bar{\mathbf{x}}_k)$ without actually storing the full continuous representation of all exact values $n(\bar{\mathbf{x}}_k)$ for all possible prefixes $\bar{\mathbf{x}}_k$, which would be prohibitive in practice. Instead, we store only the dependent quantities (i.e., variances and costs) that are used by the next iteration. Each fixed-point iteration stochastically updates some $n(\bar{\mathbf{x}}_k)$ for a set of random $\bar{\mathbf{x}}_k$. Since the probability of each $n(\bar{\mathbf{x}}_k)$ being updated is non-zero (otherwise $\bar{\mathbf{x}}_k$ is never sampled and hence irrelevant), this stochastic fixed-point iteration converges.

Unlike previous works, which rely on reducing the dimensionality of the problem to be computationally feasible (e.g., Bolin and Meyer [1997] only use the length of a prefix path), our fixed-point scheme does not suffer from the curse of dimensionality and hence enables us to solve the splitting factors without dimensionality reduction.

4.3.3 Convergence with repeated RRS. In rendering practice, RRS is performed at multiple points along a path. Fortunately, this has no effect on the convergence of our fixed-point iteration. An RRS factor $n(\bar{\mathbf{x}}_j)$ occurring at a point before or after $\mathbf{x}_k$ has a similar effect on the derivatives of the fixed-point function as the other splitting factors of unrelated $n(\bar{\mathbf{x}}'_k)$. The criteria used to prove convergence in Appendix C are unaffected and convergence is still guaranteed independent of path length.

4.3.4 Example. Fig. 5 illustrates how our fixed-point iteration behaves when applied to forward path tracing rendering a complex caustic (a). The initial training iteration (b) uses classic albedo-based RR and estimates the corresponding variance and cost of the nested $\langle L_{\text{r}}\rangle$ estimators at the positions blue (specular water), green (diffuse floor), and orange (specular water). The cost in the initial pass is given by the lengths of the suffix paths (e.g., blue = 3, green = 2, and orange = 1), as no splitting is done. The variance propagates backwards along the path, and is dominated by the diffuse surface

at green, so blue and green have the same local variance estimates, while orange, being a specular surface directly reflecting the sun, has low variance.

In the first fixed-point iteration (c), the high variance at blue and green results in splitting being done at both. Paths generated at green that do not find the sun (i.e., outside the yellow region) are terminated via RR. After the iteration, the variance estimate at blue decreases, due to the splitting at green, and its cost increases. The variance at green remains the same, while the cost is marginally reduced by the RR done at orange.

In the second fixed-point iteration no splitting is performed at blue, due to the now low variance and high cost. When the path then arrives at green, which has a high variance and low cost, a lot of splitting is done to reduce the overall variance of the pixel estimator. Again, suffixes sampled at green that do not find the sun are terminated by RR at orange.

5 IMPLEMENTATION

We implement our method in the Mitsuba renderer [Jakob 2010], using a recursive path tracer as the basis. Our fixed-point iteration requires an iterative rendering process, where each iteration creates some number of samples per pixel and estimates the global and local variances. Initially, we start with classic prefix weight-based RR, and then apply our fixed-point iteration (29) to iteratively refine the RRS factors. The required local estimates are stored in a simple 5D data structure. To alleviate the computational overhead of our method, we increase the duration of each iteration over time, thereby reducing the number of updates.

Because our RRS decisions improve over time, the rendered images of early iterations have much higher noise than later ones. Thus, we weight each iteration's rendered image with its inverse variance, which, in theory, yields the optimal combination of images [Hammersley and Handscomb 1968]. However, the variance estimates are based on the same samples as the images themselves, which introduces bias [Kirk and Arvo 1991]. But that bias is typically negligible, and it vanishes with growing iteration times.

The pseudocode in Alg. 1 provides an overview of the process. The individual steps are explained in more detail in the following.

5.1 Adapting the theory

In the following, we summarize modifications to the theory that are required for our implementation.

Next event estimation. The pixel estimator $\langle I_{\text{px}}; n\rangle$ is given by a recursive path tracer with Russian roulette and splitting, additionally performing next event estimation at each intersection. Following the approach of Vorba and Křivánek [2016], we consider next event estimation and path continuation via BSDF sampling as an atomic operation. That is, a splitting factor of 2 implies that 2 BSDF samples are traced to continue the path, and 2 shadow rays are traced for next event estimation. This integrates nicely into our theory: The computed RRS factors are optimal under the constraint that exactly as many shadow rays need to be traced as BSDF samples. Note that in a more general context, this is not optimal. There can, e.g., be regions of high variance that only benefit from BSDF sampling. Balancing the number of samples between multiple techniques in an MIS combination [Veach and Guibas 1995b] is an orthogonal problem [Sbert et al. 2019].

Handling colors. The derivations so far have glanced over the fact that the (reflected) radiance is vector-valued (RGB triplets or spectral samples). The best local RRS factor depends on spectral contributions of the prefix and the local and global estimates. There is, e.g., no point in splitting a 'red' prefix if the local reflected radiance has no red contribution. We can extend our efficiency formula Eq. (2) to a multichannel renderer by averaging the variances of individual color channels λ:

$$\epsilon^{-1} = \left(\frac{1}{N_{\text{px}} N_\lambda} \sum_{\text{px}} \sum_{\lambda} \mathbb{V}\left[\langle I_{\text{px}}\rangle_\lambda\right]\right)\left(\frac{1}{N_{\text{px}}} \sum_{\text{px}} \left[c\ \ \langle I_{\text{px}}\rangle\right]\right). \quad (31)$$

Using this as starting point, we find the splitting factors,

$$n = \sqrt{\frac{\sum_\lambda T_\lambda^2(\bar{\mathbf{x}}_k)\tilde{I}^{\,2}_{I_{\text{px}},\lambda} R_\lambda(\langle L_{\text{r}}; n_i\rangle)}{\sum_\lambda \bar{V}_\lambda(n_i)}}\sqrt{\frac{\bar{C}(n_i)}{[c(\langle L_{\text{r}}; n_i\rangle)]}}, \quad (32)$$

which differ from our monochromatic splitting factors (Eq. (29)) only in that the product of local variance estimate with the prefix weight is now performed component-wise, and that we sum up the variances over their color channels λ. Similar derivations could be carried out using other metrics as starting point, e.g., using luminance or the maximum component of the variance spectrum if desired.

Cost heuristic. To quantify the cost, we use the same simple heuristic as previous work [Bolin and Meyer 1997; Szirmay-Kalos 2005], i.e., we count the number of rays that are traced by the estimator. For simplicity, we assume that shadow rays, primary rays from the camera, and BSDF samples have the same cost.

Clamping. In practice, it is beneficial to limit the allowed range of the RRS factors. On the one hand, the theoretically optimal splitting factor can in principle be arbitrarily large. On the other hand, error in our estimates, due to noise and approximations, can also produce much too large or much too small RRS factors. Thus, we clamp each RRS factor to the interval $(0.05, 20)$ to avoid bias, which can happen if $n(\bar{\mathbf{x}}) = 0$, and to prevent excessive splitting.

5.2 Global estimates

When a pixel estimate is completed, we record its cost

$$\widetilde{C} = \frac{1}{N_{\text{spp}}} \sum_{s=1}^{N_{\text{spp}}} \frac{1}{N_{\text{px}}} \sum_{\text{px}} c(\langle I_{\text{px}}\rangle_s) \quad (33)$$

and approximate the variance using a denoised image [Áfra 2019] in lieu of the ground truth

$$\widetilde{V} = \frac{1}{N_{\text{spp}}} \sum_{s=1}^{N_{\text{spp}}} \frac{1}{N_{\text{px}}} \sum_{\text{px}} \left(\frac{\langle I_{\text{px}}; n\rangle_s \quad \tilde{I}_{\text{px}}}{\tilde{I}_{\text{px}}}\right)^2. \quad (34)$$

It is important that $\langle I_{\text{px}}; n\rangle$ is the estimator *including splitting*. That is, every path tree generated per sample is considered as a whole.

Algorithm 1. Overview of our main rendering loop. The image is rendered in iterations, each taking multiple samples per pixel. Each iteration updates the global and local variance and cost estimates.

function RENDER
  for $i \in 1..n_{\text{iterations}}$ **do**
    $\widetilde{V}^{(i)}, \widetilde{C}^{(i)} = 0$ ← initialize global statistics to zero
    $\widetilde{C}_b^{(i)}, \widetilde{E}_b^{(i)}, \widetilde{M}_b^{(i)}, n_B = 0$ ← initialize all local statistics
    while time budget of iteration not exhausted **do**
      for px in image **do** ← render one sample per pixel
        $\bar{\mathbf{x}}_1 = \text{SAMPLECAMERA}(\text{px})$ ← start a path from the camera
        $c, \langle L_\text{r} \rangle = \text{LRESTIMATE}(\bar{\mathbf{x}}_1, \tilde{I}_\text{px})$ ← Alg. 2
        $\widetilde{C}^{(i)}$ += $1 + c$ ← update cost, Eq. (33)
        $\widetilde{V}^{(i)}$ += $\left(T(\bar{\mathbf{x}}_1) \cdot \langle L_\text{r} \rangle \quad \tilde{I}_\text{px}\right)^2 / \tilde{I}_\text{px}^2$
        $\langle I_\text{px} \rangle^{(i)}$ += $T(\bar{\mathbf{x}}_1) \cdot \langle L_\text{r} \rangle$ update relative variance (34)
      N_spp += 1
    $\widetilde{C}^{(i)}, \widetilde{V}^{(i)}$ /= $N_\text{px} \cdot N_\text{spp}$ ← normalize estimates, Eqs. (33) and (34)
    $\langle I \rangle = \text{MERGEFRAMESBYVARIANCE}(\langle I \rangle, \langle I \rangle^{(i)}, \bar{V}^{(i)})$
    $\tilde{I} = \text{DENOISE}(\langle I \rangle)$
    for $B \in$ SpatialCache **do** ← normalize local statistics
      $\widetilde{C}_b^{(i)}, \widetilde{E}_b^{(i)}, \widetilde{M}_b^{(i)}$ /= n_B ← Eqs. (35), (36) and (38)
      $\widetilde{V}_b^{(i)} = \widetilde{M}_b^{(i)} \quad \left(\widetilde{E}_b^{(i)}\right)^2$ ← compute variance Eq. (37)
  return $\langle I \rangle$

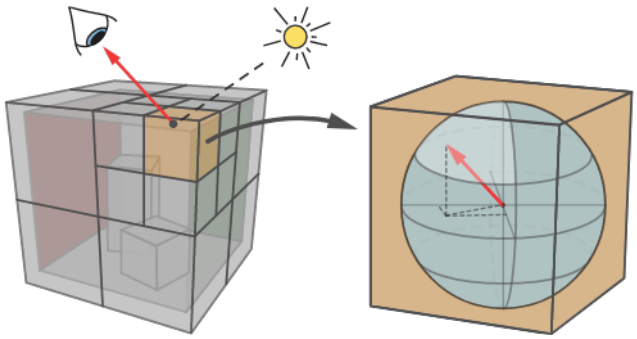


Fig. 6. The data structure used by our implementation. The scene is divided by an octree left). Each cell of which stores variance estimates for the reflected radiance estimator. The directional dependency on ω_o is handled via a simple histogram right).

Outlier removal. Even a single outlier in a single pixel can severely distort the estimate of the average image variance. We apply a simple workaround and ignore the 0.001 of all pixels that have the highest variance when computing the average image variance required by our method.

5.3 Local estimates

The local variance and cost estimates (see Section 4.3.2) are stored in a 5D data structure, illustrated in Fig. 6. The scene is partitioned by an octree, each cell of which contains a histogram over outgoing directions of fixed 4×4 resolution. In our experiments, higher resolutions showed only small improvements in the quality of splitting factors, which were offset by the disadvantage of requiring more training samples. We apply a similar approach to Müller et al. [2017], subdividing leaves of the octree after more than 40,000 samples have been accumulated.

Building the estimates. Each directional histogram bin b stores approximations of variance $\widetilde{V}_b$, second moment $\widetilde{M}_b$ and cost $\widetilde{C}_b$ of the reflected radiance estimator $\langle L_\text{r} \rangle$. The second moment is approximated as the average second moment over all points and directions in the bin b,

$$\left[\langle L_\text{r}; n_i \rangle^2\right] \approx \widetilde{M}_b = \frac{\sum_{s=1}^{n} \langle L_\text{r} \rangle_s^2}{n_b}, \tag{35}$$

where n_b is the number of samples within bin B. The variance $\widetilde{V}_b$ is approximated by additionally computing the average reflected radiance in the bin

$$\widetilde{E}_b = \frac{\sum_s^n \langle L_\text{r} \rangle_s}{n_b}, \tag{36}$$

which we can square and subtract from the second moment to approximate the variance

$$\mathbb{V}\left[\langle L_\text{r}; n_i \rangle^2\right] \leq \widetilde{V}_b = \widetilde{M}_b \quad \widetilde{E}_b^2. \tag{37}$$

Note that this approximation replaces the integral of squared L_r terms by the square of the integral. Hence, if the reflected radiance fluctuates strongly within b, the variance is overestimated and the resulting RRS factors will be too large. We discuss this approximation in the next section. The expected cost $[c(\langle L_\text{r}; n \rangle)]$ within a bin is approximated by averaging the cost of all samples from the bin,

$$[c(\langle L_\text{r}; n \rangle)] \approx \widetilde{C}_b = \frac{\sum_s^n c(\langle L_\text{r} \rangle_s)}{n_b}. \tag{38}$$

Incremental learning. According to our fixed-point scheme (Section 4.3.2), an iteration i should use the variances and costs derived from the splitting factors of the previous iteration i 1. For unbiased estimation, each iteration should hence start computing new variance and cost estimates from scratch. Doing so would require very long iterations for sufficiently converged estimates. To reduce the noise, we include samples from earlier iterations in our estimates. In practice, this slightly reduces the convergence speed of our iterative scheme but yields much more reliable estimates which improves the performance of our method.

6 EVALUATION

We compare our method and the adjoint-driven (ADRRS) approach of Vorba and Křivánek [2016] for two cases: when applied solely to Russian roulette (RR) and when applied to combined RR and splitting (RRS). We omit results for applying only splitting, as neither method performs well without RR, which is required to kill unimportant paths that can be generated by splitting. As a baseline, we include classic prefix weight-based RR (starting at the 5th bounce), of which we also include an adaptive sampling variant. Unlike ADRRS and our method, which learn their statistics on-line during rendering, the adaptive sampler is provided with a ground truth relative variance image that was computed in a pre-process not contained in the

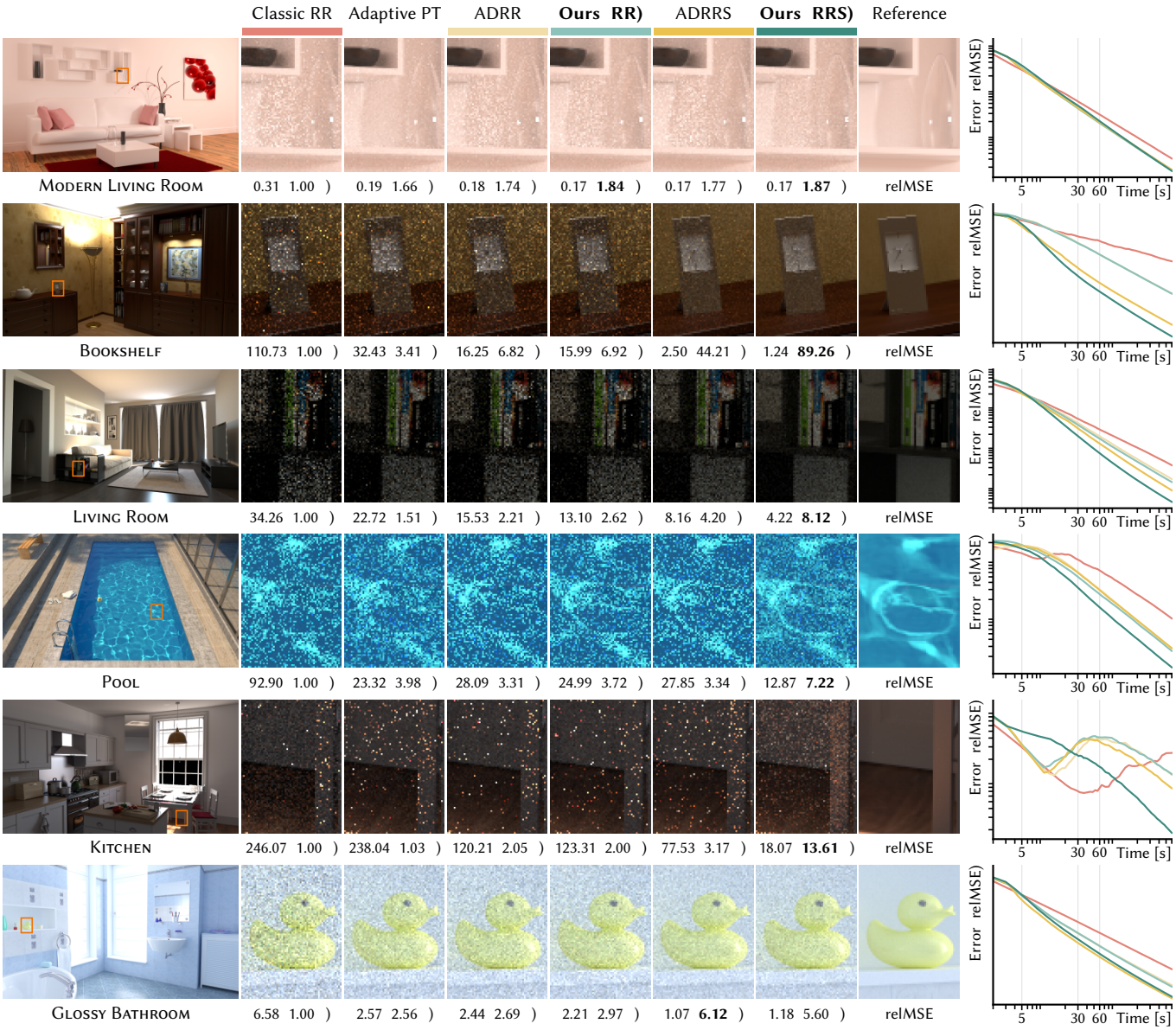


Fig. 7. We render five scenes with different Russian roulette and splitting strategies for 10 minutes each. The numbers below the crops are the relative mean-squared error relMSE, lower is better), with the speed-up compared to classic RR in parentheses higher is better).

reported render time. All images were rendered for ten minutes on an AMD Ryzen 3950X CPU with 16 cores. The maximum path length is set to 40. The full results across all 19 scenes are in the supplemental materials. Our method achieves an average speed-up factor of 1.52× when compared to ADRRS and an average speed-up factor of 5.87× when compared to classic RR, with the poorest performing scene (Glossy Bathroom) being only 9 slower than ADRRS.

Fig. 7 shows the results for some of our test scenes. The numbers below each crop are the equal-time relative mean-squared error (relMSE), lower is better, for which we discard 0.01 of the pixels with the highest error to increase robustness towards outliers. Note that this is distinct from the outlier removal discussed in section 5.2, which discards fewer outliers (0.001) and uses denoised intermediate renders (in lieu of a reference image) to estimate the relative image variance required by our method. The numbers in parentheses are the speed-up w.r.t. classic RR.

The Modern Living Room is mostly diffuse and illuminated by a large spherical light, making it simple to render with forward path tracing. Due to mostly short paths, RR has little impact here. While

Table 1. Performance statistics of all RR and RRS methods. The samples per pixel is the number of frames rendered at an equal time i.e., 10 min). The average path length values provide insights into the different termination behaviors of each method. For a better understanding of the splitting behavior of ADRRS and Ours RRS), additional information is provided: first, the average number of paths generated per primary ray using splitting, and second, the average splitting factors at the first intersection of the primary ray.

	Samples per pixel					Avg. path length					Avg. paths per sample		Avg. primary splits	
Scene	Classic	ADRR	Ours(RR)	ADRRS	Ours(RRS)	Classic	ADRR	Ours(RR)	ADRRS	Ours(RRS)	ADRRS	Ours(RRS)	ADRRS	Ours(RRS)
Modern Living Room	1800	1213	1469	1147	274	5.61	6.81	5.75	6.99	6.90	1.08	7.28	1	5.25
Bookshelf	2370	3099	3196	2699	208	4.59	3.11	3.12	3.45	3.88	1.28	26.50	1	10.62
Living Room	1805	2526	3255	2242	219	4.95	3.19	2.67	3.33	3.13	1.20	19.99	1	13.33
Pool	2814	2234	3042	2208	421	3.17	3.35	2.90	3.37	4.54	1.01	8.69	1	1.82
Kitchen	2400	2258	2438	1967	365	4.78	4.33	4.15	4.59	4.28	1.21	19.98	1	9.42
Glossy Bathroom	1522	834	1367	698	51	6.92	9.86	6.35	9.62	8.85	1.87	113.55	1	18.45

Algorithm 2. Pseudocode for the reflected radiance estimation, given a prefix path $\bar{\mathbf{x}}_k$ originating in pixel px. First, we query the corresponding bin in our data structure. Then, we apply our fixed-point function to update the RRS factor. The sample weights and costs are logged in the data structure for each sampled direction from BSDF or next event the latter was omitted for brevity). For simplicity, we assume monochromatic rendering, but the algorithm can easily be extended to multichannel rendering as discussed in section 5.1.

function LrEstimate($\bar{\mathbf{x}}_k, \tilde{I}_{\text{px}}$)
  b = SpatialCacheBin($\bar{\mathbf{x}}_k$) ← find responsible bin
  $n = \frac{T(\bar{\mathbf{x}}_k)}{\tilde{I}_{\text{px}}} \sqrt{\frac{\widetilde{C}^{(i\ 1)}}{\widetilde{V}^{(i\ 1)}}} \sqrt{\frac{\widetilde{V}^{(i\ 1)}}{\widetilde{C}^{(i\ 1)}}}$ ← splitting case (29)
  if n < 1 **then** ↶ compute Russian roulette objective
    $n = \min\left(1, \frac{T(\bar{\mathbf{x}}_n)}{\tilde{I}_{\text{px}}} \sqrt{\frac{\widetilde{C}^{(i\ 1)}}{\widetilde{V}^{(i\ 1)}}} \sqrt{\frac{\widetilde{M}^{(i\ 1)}}{\widetilde{C}^{(i\ 1)}}}\right)$
  n = clamp(n, 0.05, 20) ← clamp to a reasonable range
  $c,\ w = 0, 0$ ← initialize total cost and contribution
  for i = 1..StochasticRounding(n) **do**
    $\bar{\mathbf{x}}_{k+1}$ = SampleBsdf($\bar{\mathbf{x}}_k$) ← (next event omitted for brevity)
    $c, \langle L_{\text{r}} \rangle$ = LrEstimate($\bar{\mathbf{x}}_{k+1}, \tilde{I}_{\text{px}}$)
    $w = \frac{\rho(\mathbf{x}_k, \omega_{\text{o}}, \omega_{\text{i}})}{p(\omega_{\text{i}})} \langle L_{\text{r}} \rangle$ ← local weight times nested estimate
    $\widetilde{C}_b^{(i)}$ += c $\widetilde{E}_b^{(i)}$ += w ← Eqs. (36) and (38)
    $\widetilde{M}_b^{(i)}$ += w^2 n_b += 1 ← Eq. (35)
    c += 2 + c w += w ← accumulate cost and contribution
  return $c,\ w$

our method does not noticeably improve the mean error compared to ADRRS, it still achieves a more uniform noise distribution across the image as shown in Fig. 8. The figure shows false-color images of the per-pixel error. Similar to adaptive sampling, our method greatly reduces the error in the caustics of the glass vases, by slightly increasing the error everywhere else. In such simple scenes, the overhead of complex RRS methods like ADRRS and ours does not pay off for very short renders. As can be seen in the error over time plots, in some cases it takes at least 20 seconds to out-perform classic RR.

The Bookshelf is an example of a scene with strong, difficult diffuse indirect illumination. Due to many dark surfaces, classic RR prematurely terminates paths. When applied only to RR, our method achieves a 30 speed-up compared to ADRR, by heeding

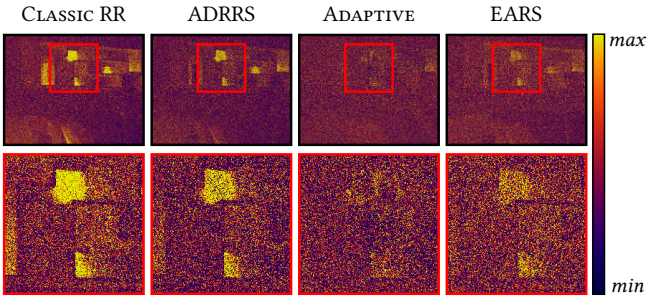


Fig. 8. Relative mean-squared error for Classic prefix weight-based) Russian roulette with and without adaptive sampling), EARS and ADRRS. While both ADRRS and our method approximately achieve the same mean error in this scene, EARS benefits from a more uniform noise distribution across the entire image. Classic Russian roulette with adaptive sampling achieves an even more uniform noise distribution, but higher mean error.

variance. Splitting can drastically increase the performance here: if an unlikely path finds the directly illuminated regions, splitting within the illuminated region increases the odds of forming a full valid path to the light. This is an important advantage over adaptive sampling, which can only split a pixel as a whole, thus wasting time on paths that do not land in the illuminated region. For full RRS, our method performs twice as fast as ADRRS. The Living Room scene is another example with similar light transport and similar results.

The Pool is an example that shows more clearly the benefit of basing splitting decisions on variance and cost (as done by our method), rather than expected contributions (as done by ADRRS). By directly considering variance, our method performs splitting on the diffuse bottom of the pool, unlike ADRRS (see Fig. 1).

The Kitchen combines the effects from Bookshelf and Pool: this scene features strong indirect illumination through a high-variance caustic underneath the table. While both ADRR and our method (RR) similarly increase performance by not killing paths that land in the caustic, even stronger speed-ups can be gained with splitting. By considering variance, our method (RRS) again performs significantly more splitting than ADRRS on the caustic, resulting in less noise in the indirect illumination and a 4.3× speed-up over ADRRS. Similar to the Bookshelf scene, adaptive sampling achieves little improvement in this scene, as noise is mostly caused by indirect illumination from a small brightly lit area.

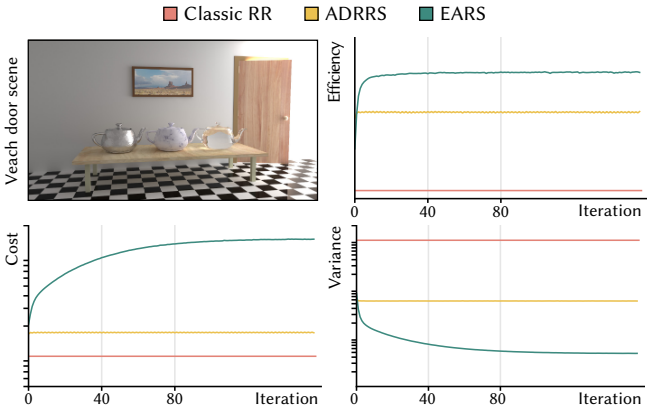


Fig. 9. Convergence of different Russian Roulette and splitting methods in the Veach Door scene. For each iteration, we measure the average time it takes to render one sample per pixel *cost*) and the average relative variance with one sample per pixel *variance*). The *efficiency* is the inverse product of the two. In all our scenes, our method converges to fixed-point for cost and variance, and hence efficiency.

The Glossy Bathroom is a failure case of our method. The homogeneous illumination makes the scene relatively simple to render, while the many glossy surfaces are problematic for our approach. The over-approximation of the local variance results in excessive splitting and hence in 9 worse performance than ADRRS in this scene. Still, our result outperforms classic RR.

6.1 Sampling statistics

To better understand the actual sampling behavior, some additional statistics are shown in Table 1: the samples per pixel (SPP), the average path length, and, for the splitting methods, the average number of generated paths per primary ray and the average splitting factor at the first bounce.

Among the RR methods, ADRR and Ours (RR) tend to generate more SPP than classic RR. This is because they generate shorter paths, which is partly due to the fact that, in our setup, Classic RR only activates after a path length of 5. But the better rendering results are not only due to more SPP. The statistics of Modern Living Room show that the termination decisions themselves are also better. Here, all RR methods have an average path length of around 5. Classic RR generates 200 400 more SPP, but still the speed-up of ADRR and Ours (RR) is up to 1.8× (see Fig. 7).

With splitting, the number of SPP naturally decreases. Our method produces up to an order of magnitude fewer SPP than classic RR, but yields vastly better rendering results. The lower number of samples produced by the RRS methods directly relates to the average number of paths generated for each primary ray, i.e., the number of leaves in the generated ray tree. Here we can see that ADRRS, on average, only generates a few additional paths per primary ray (up to 2), while this number can be much higher for ours (up to 67). The high numbers are largely due to the splitting at the primary hit point.

Our optimization objective is to maximize the full efficiency of the rendering process (28). Hence, if the variance due to aliasing and sampling the lens is low, as is the case in most of our scenes, our method tends to perform a lot of splitting at the primary hit point. Thereby, variance is reduced as if additional SPP were generated, but without the cost of the primary ray. Dividing the number of paths per sample by the number of primary splits, we see that our method usually does not perform much more splitting than ADRRS at later bounces. A notable exception is the Pool scene, where splitting happens for specular paths that arrive at a caustic.

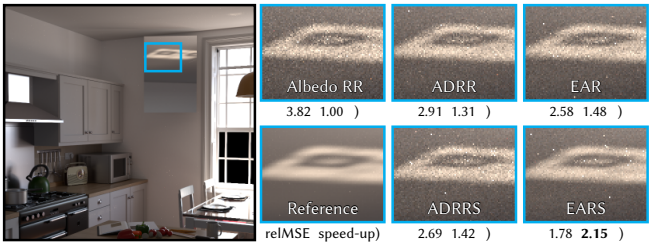


Fig. 10. Our method vs. previous work in a path guiding application. The numbers below the crops are the error relMSE) and, in parentheses, the speed-up compared to the baseline, which is classic albedo-based RR.

6.2 Overhead

To build and store the required estimates, both ADRRS and EARS incur computational overhead. In our evaluation, we have limited the memory footprint of the data structure to 24 MiB, limiting the spatial partitioning to roughly 22,000 regions. In our experiments, the benefits of higher resolution estimates were outweighed by higher computational costs of updating and traversing the data structure. In terms of computation time, the overhead of ADRRS and EARS results in an average of 15 fewer rays being traced when compared to classic RR. The primary causes of this overhead are the traversal of the tree structure (8.7 of render time) and splatting of the required statistics (5.2). Denoising the pixel estimate and adapting the data structure each amount to less than 1 of the render time. Note that our method adds no noticeable overhead over ADRRS: since the solution to our optimization problem is found implicitly through a fixed-point scheme, only a few additional arithmetic operations when splatting samples and computing the splitting factor are carried out.

6.3 Convergence of our fixed-point scheme

While a nice theoretical foundation, the convergence we proved for our fixed-point iteration in theory does not automatically imply convergence in practice. Approximations aside, there is also always noise in the estimates. We conducted an empirical verification of the convergence in a separate rendering setting, without incremental learning and with constant iteration times. Fig. 9 plots the variances and sampling costs of the image rendered in each iteration, for the Veach Door scene. Across all our scenes, the cost and variance converged nicely to a fixed-point, when given enough iterations.

6.4 Path guiding

We have also evaluated our method in the context of path guiding, specifically the method of Müller et al. [2017]. There, our method can re-use the same data structures employed by the guiding distribution,

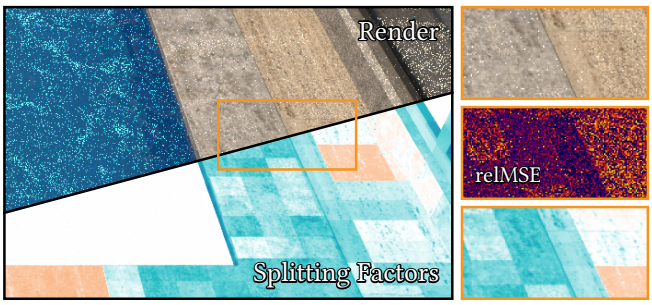


Fig. 11. Artefacts due to discretized estimates at a low sample count. The spatial caches used in the estimation of our splitting factors can create small visual artefacts in case the caches have not received enough samples to compute accurate variance estimates.

only storing a few extra values (the cost and variances) in each spatial cell. We found that using the product of surface albedos, instead of the more common throughput weight, yielded better results for classic RR. The throughput weight can be very low if guiding has a high sampling density for a path across dark surfaces in which case classic RR can undo the benefits of guiding [Vorba and Křivánek 2016]. The product of albedos does not exhibit that problem. The results were similar to the path tracing application. An example is shown in Fig. 10. In the Kitchen scene, our method outperforms ADRR(S) both with and without splitting, achieving a speed-up of 1.5× compared to ADRRS.

7 LIMITATIONS AND FUTURE WORK

In practice, our method is limited by the accuracy of the required estimates. Promising directions for future work include combinations with image space adaptive sampling and bidirectional algorithms.

Estimation error. Noise in the local estimates can cause poor RRS decisions. The worst case occurs if some spatial caches contain much lower-quality estimates than their immediate neighbors. An example is shown in Fig. 11 for a shorter rendering of the Pool scene. The severe outliers yield poor variance estimates that differ strongly between caches. As a consequence, islands of noise with hard edges in-between are visible in the image. The problem vanishes with longer renderings, due to the increasing iteration times and the fact that we keep the variance estimates of the previous iterations. In practice, this can be alleviated by spatial filtering, i.e., by sharing estimates between caches. Similar problems are encountered in path guiding methods [Müller et al. 2017; Vorba et al. 2014].

Combining with Adaptive Sampling. Our method performs Russian roulette at the primary hit point if some parts of the image have a significantly higher variance than others. An example is shown in Fig. 12. The figure shows a crop of the Pool scene and the corresponding RRS factor at the primary hit used by our method. The noise in this directly illuminated region is greatly increased by our method, due to aggressive RR at the primary hit point. While this increases the overall efficiency by focusing computation time on the difficult pixels, i.e., the caustics, it also wastes primary rays that could have been avoided altogether. Future work could combine our method with adaptive sampling in image space [Zwicker et al. 2015] to not sample such wasted primary rays in the first place. Doing so requires a small modification to our optimization objective: We should maximize the efficiency of each individual pixel rather than the average across the whole image, to benefit the most from adaptive sampling.

Bidirectional methods. We have shown that our method can successfully increase efficiency in a unidirectional path tracer. A promising area of future work would be to apply our method to bidirectional algorithms [Georgiev et al. 2012; Hachisuka et al. 2012; Lafortune and Willems 1993; Veach and Guibas 1995a]. This is significantly more complicated, because bidirectional algorithms are multi-sample MIS combinations of different sampling techniques that each construct full paths between the camera and the light sources. RRS along these paths would become part of the MIS weights, hence the splitting decisions on a camera sub-path affect the splitting decisions on the light sub-paths. This produces a non-convex objective that is much harder to optimize. Further, RRS in a bidirectional context is problematic for MIS weighting, as the classic balance heuristic ignores the covariance introduced by splitting [Grittmann et al. 2021; Popov et al. 2015].

Participating media. We have limited our discussion and implementation to light transport on surfaces. Volumetric transport can equally benefit from RRS [Herholz et al. 2019]. Our theory can be easily extended to the volumetric case, only the practical implementation, in particular the data structure, has to be extended.

Correlated sampling. For splitting, we have assumed that variance decreases with $O(n^{-1})$. This is true for uncorrelated sampling, but not for quasi-Monte Carlo (QMC) sampling. Most likely, not all samplers will yield a necessary convex objective. However, we expect the effect of this assumption to be less significant than other sources of inaccuracies (in particular discretization and noise).

Dynamic scenes. In dynamic scenes, some learned statistics could potentially be reused in later frames. Future work could look into how many iterations of our fixed-point scheme are necessary after a scene update and which parts of the data structure might need to be discarded and retrained from scratch.

8 CONCLUSION

We present a fixed-point iteration to optimize the Russian roulette and splitting (RRS) factors in a path tracing algorithm. In principle, the theory can be applied to any random walk Monte Carlo method. Assuming perfect knowledge of variances and expected costs, the fixed-point iteration is proven to converge to the optimal RRS factors that maximize the rendering efficiency. In our rendering application, we iteratively improve the RRS factors used by a forward path tracer; our implementation employs a simple 5D data structure to track variances and costs throughout the scene. Despite the simplicity, we achieve consistent speed-ups over the state of the art of 1.6× on average. Especially scenes with challenging indirect illumination benefit from the fact that our method, unlike previous work, directly minimizes variance and cost.

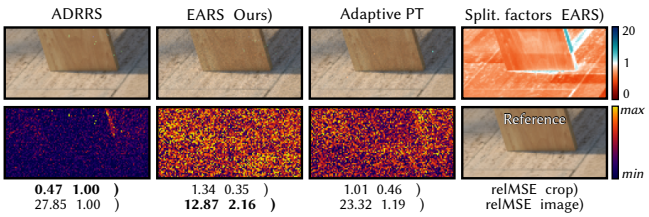


Fig. 12. A zoom-in of the Pool scene see Figs. 1 and 7), comparing the renderings of ADRRS and our method. The top-right false-color image shows the splitting factors at the primary hit used by our method. Note that aggressive RR is performed, i.e., many paths started from the camera never sample any contribution. This manifests in much higher noise in some regions of the image. The reason for this behavior is that our method focuses computation time on the much more challenging caustic. Adaptive sampling similarly suffers from increased noise, but less severely as it does not need to resort to Russian roulette to artificially lower sample counts.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956585 (https://prime-itn.eu/). We thank the anonymous reviewers for their insightful remarks, as well as the artists of our test scenes: Benedikt Bitterli [2016], Miika Aittala, Samuli Laine, and Jaakko Lehtinen (Veach Door), Evermotion and Tiziano Portenier (Bookshelf, Glossy Bathroom), Ondřej Karlík (Pool), Wig42 (Modern Living Room), Jay-Artist (Kitchen), and Ludvík Koutný (Living Room).

REFERENCES

Attila T. Áfra. 2019. Intel® Open Image Denoise. https://www.openimagedenoise.org/.
James Arvo and David Kirk. 1990. Particle transport and image synthesis. *SIGGRAPH '90*, 63–66.
Stefan Banach. 1922. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math* 3, 1 (1922), 133–181.
Vasile Berinde. 2007. *Iterative approximation of fixed points.* Vol. 1912. Springer.
Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/
Mark R Bolin and Gary W Meyer. 1997. An error metric for Monte Carlo ray tracing. In *Eurographics Workshop on Rendering Techniques*. Springer, 57–68.
Xi Deng, Miloš Hašan, Nathan Carr, Zexiang Xu, and Steve Marschner. 2021. Path graphs: iterative path space filtering. *ACM Transactions on Graphics TOG)* 40, 6 (2021), 1–15.
Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. 2012. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.* 31, 6 (2012), 192–1.
Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 213–222.
Pascal Grittmann, Iliyan Georgiev, and Philipp Slusallek. 2021. Correlation-Aware Multiple Importance Sampling for Bidirectional Rendering Algorithms. *Comput. Graph. Forum EG 2021)* 40, 2 (2021).
Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. 2012. A path space extension for robust light transport simulation. *ACM Trans. Graph. TOG)* 31, 6 (2012), 191.
J.M. Hammersley and D.C. Handscomb. 1968. *Monte Carlo Methods*. Springer, Dordrecht.
Sebastian Herholz, Yangyang Zhao, Oskar Elek, Derek Nowrouzezahrai, Hendrik P. A. Lensch, and Jaroslav Křivánek. 2019. Volume Path Guiding Based on Zero-Variance Random Walk Theory. *ACM Trans. Graph.* 38, 3, Article 25 (June 2019), 19 pages.
Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.
James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 143–150.
David Kirk and James Arvo. 1991. Unbiased Sampling Techniques for Image Synthesis. *ACM Trans. Graph. SIGGRAPH '91)* 25, 4 (jul 1991), 153–156. https://doi.org/10.1145/127719.122735
Eric P. Lafortune and Yves D. Willems. 1993. Bi-Directional Path Tracing. 93 (Dec. 1993), 145–153.
Thomas Müller, Markus H. Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Comput. Graph. Forum* 36 (2017), 91–100.
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation.* Morgan Kaufmann.
Stefan Popov, Ravi Ramamoorthi, Fredo Durand, and George Drettakis. 2015. Probabilistic Connections for Bidirectional Path Tracing. *Comput. Graph. Forum* 34, 4 (jul 2015), 75–86.
Alexander Rath, Pascal Grittmann, Sebastian Herholz, Philippe Weier, and Philipp Slusallek. 2022. *Implementation of EARS: Efficiency-Aware Russian Roulette and Splitting*. https://doi.org/10.5281/zenodo.6514751
Mateu Sbert, Vlastimil Havran, and László Szirmay-Kalos. 2019. Optimal Deterministic Mixture Sampling.. In *Eurographics Short Papers)*. 73–76.
László Szirmay-Kalos. 2005. Go with the winners strategy in path tracing. © *Journal of WSCG, 2005, vol. 13, núm. 1-3, p. 49-56* (2005).
László Szécsi, László Szirmay-Kalos, and Csaba Kelemen. 2003. Variance Reduction for Russian-roulette.
Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation.* Stanford University PhD thesis.
Eric Veach and Leonidas Guibas. 1995a. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 145–167.
Eric Veach and Leonidas J Guibas. 1995b. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *SIGGRAPH '95*. ACM, 419–428.
Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. Graph. Proceedings of SIGGRAPH 2014)* 33, 4 (2014).
Jiří Vorba and Jaroslav Křivánek. 2016. Adjoint-driven Russian Roulette and Splitting in Light Transport Simulation. *ACM Trans. Graph.* 35, 4, Article 42 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925912
Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In *Computer graphics forum*, Vol. 34. Wiley Online Library, 667–681.

A DERIVATIVES OF THE OBJECTIVE

The functional derivatives of variance and cost, respectively, are:

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I;n\rangle\right]}{\mathrm{d}n(x)} = \quad p(x)\frac{V_{\boldsymbol{y}}(x)}{n^2(x)} \tag{39}$$

$$\frac{\mathrm{d}\quad[c(\langle I;n\rangle)]}{\mathrm{d}n(x)} = p(x)\quad[c(\langle H(x)\rangle)\mid x]\,. \tag{40}$$

The derivative of their product (20) is obtained via the product rule,

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I;n\rangle\right]\quad[c(\langle I;n\rangle)]}{\mathrm{d}n(x)} = \tag{41}$$

$$p(x)\left(\quad[c(\langle H(x)\rangle)\mid x]\,\mathbb{V}\left[\langle I;n\rangle\right]\quad\frac{V_{\boldsymbol{y}}(x)}{n^2(x)}\quad[c(\langle I;n\rangle)]\right).$$

From that, we can easily obtain our fixed-point function

$$\frac{\mathrm{d}\mathbb{V}\left[\langle I;n\rangle\right]\quad[c(\langle I;n\rangle)]}{\mathrm{d}n(x)} = 0 \tag{42a}$$

$$\Leftrightarrow\quad[c(\langle H(x)\rangle)\mid x]\,\mathbb{V}\left[\langle I;n\rangle\right] = \frac{V_{\boldsymbol{y}}(x)}{n^2(x)}\quad[c(\langle I;n\rangle)] \tag{42b}$$

$$\Leftrightarrow n(x) = \sqrt{\frac{V_{\boldsymbol{y}}(x)}{\mathbb{V}\left[\langle I;n\rangle\right]}\,\frac{[c(\langle I;n\rangle)]}{[c(\langle H(x)\rangle)\mid x]}}, \tag{42c}$$

where the last equivalence holds because variance, cost, and splitting factor are always positive.

B FIXED-POINT ITERATIONS FOR ROOT FINDING

The following provides a very brief intro to fixed-point iteration methods applied to root finding problems. A more complete discussion can be found, e.g., in Berinde [2007], or in various textbooks on numerical analysis.

Consider the simple example of computing the root of

$$f(x) := \sqrt{x} \quad x = 0 \tag{43}$$

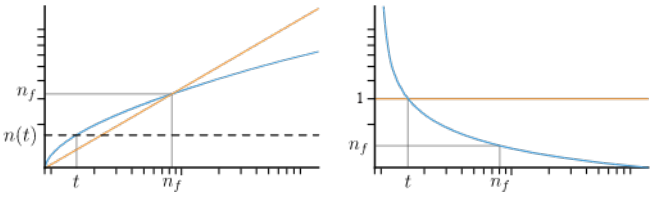


Fig. 13. The fixed-point function is strictly monotonically increasing left) and its derivatives have a convex hyperbolic shape right). The fixed-point n_f lies in a sub-domain where the first derivatives are always less than one, i.e., below the orange line on the right. And, due to the monotonicity, $(n) > n \; \forall n \quad n_f$. Together, these two properties ensure convergence.

(which, in fact, resembles our derivative). The analytical solution of this is trivially given by $x = 1$. It can also be computed numerically, using a fixed-point iteration. For that, we rewrite the equation as

$$f(x) = 0 \Leftrightarrow g(x) \quad x = 0 \Leftrightarrow g(x) = x, \tag{44}$$

where

$$g(x) := \sqrt{x} \tag{45}$$

is the equivalent *fixed-point function*. By construction, every fixed-point of g, i.e., every x for which $g(x) = x$, must be a root of $f(x)$, and vice versa. This equivalence provides a way to numerically find the roots of $f(x)$ via a *fixed-point iteration*, i.e., by repeatedly updating

$$x_i = g(x_{i\ 1}), \tag{46}$$

starting with an initial guess x_0.

The convergence of this simple example is a well-known property: repeatedly applying the square root to any number will eventually converge to 1. But how can it be proven? If the fixed-point function $g : D \to D$ is a mapping from some domain D onto itself, and if g is a *contraction*, then *Banach's fixed-point theorem* [Banach 1922] guarantees that there is a unique fixed-point $x_f \in D$ and that the fixed-point iteration above will converge to that fixed point, when initialized with any $x_0 \in D$.

For $g(x)$ to be a contraction, its first derivatives need to be less than 1 over the domain D. In our example,

$$g'(x) = (2\sqrt{x})^{\ 1} \quad 1 \Leftrightarrow x > 0.25, \tag{47}$$

this holds for $D = (0.25, \infty)$. The domain of $g(x)$, however, is the set of all positive real numbers. So this is not a sufficient criterion for convergence. To prove convergence, we have to prove two more properties:

(1) g is a mapping from D onto itself, $g(x) \in D \, \forall x \in D$
(2) g moves all points x outside D closer to it, $g(x) > x \forall x \quad D$

The first property guarantees that points within the sub-domain D will not leave it. Hence, g is a contraction mapping over D and Banach's theorem states that a unique fixed-point lies within D and our iteration will converge to it. The second property ensures that all initial guesses outside D are updated such that they eventually enter D. In our simple example here, both properties are trivially verified. Thus, convergence is guaranteed for all initial guesses $x_0 \in \ ^+$.

C PROOF OF CONVERGENCE

The convergence of the fixed-point iterations Eqs. (21) and (24) can be proven by following the exact same steps as for the simple example in Appendix B. In the following, we provide the proof for the case of a single splitting factor. This readily generalizes to the multivariate / functional case, where the same conditions can be proven along each dimension separately.

To prove convergence, we have to prove three properties: There is a sub-domain (t, ∞) where the fixed-point function has bounded derivatives,

$$\exists t : \ '(n) \quad 1 \, \forall n > t, \tag{48}$$

the points within that sub-domain are mapped to the same domain,

$$(n) > t \, \forall n > t, \tag{49}$$

and the points outside the domain move closer to it,

$$(n) > n \, \forall n \leq t. \tag{50}$$

All three conditions follow directly from a simple analysis of the fixed-point function, as visualized in Fig. 13. The first derivative of our fixed-point function (29) can be obtained easily with the chain- and quotient rules; we skip the exact equation here for brevity. It is positive everywhere and has a hyperbolic shape,

$$'(n) > 0 \qquad '(n) \in O\left(1/\sqrt{n}\right) \tag{51}$$

Therefore, is concave and strictly monotonically increasing. Further, we can trivially obtain the following limits:

$$\lim_{n\to\infty} \ '(n) = 0 \quad \lim_{n\to 0^+} \ '(n) = \infty \quad \lim_{n\to 0^+} \ (n) = 0 \tag{52}$$

The existance of the threshold (48) then follows directly from these limits and the hyperbolic shape of the first derivative.

The remaining two conditions can be proven by first asserting that the unique fixed-point is above the threshold, $n_f > t$. This can be done geometrically. By definition, the fixed-point n_f is an intersection of (n) with the diagonal of the positive quadrant. As (n) is a concave function, there are either zero or two such intersections. If there are two, basic geometry implies that the first occurs at a point n_a where approaches the diagonal from below, $'(n_a) > 1$, and the second at a point n_b where approaches the diagonal from above, $(n_b) \quad 1$. The limit $\lim_{n\to 0} \ = 0$ gives us the first intersection with the diagonal: it would have been at $n_a = 0$, but that is outside our domain. We know that the derivative at zero is unbounded, $'(0) = \infty$, so there must be another intersection where

$$'(n_b = n_f) \quad 1. \tag{53}$$

This intersection is our unique fixed-point in the domain of positive real numbers, and its derivative is always less than one. That, in turn, implies that $n_f > t$, i.e., the fixed-point is above the threshold.

This immediatly implies (50), since all $n \quad n_f$ must be mapped to a value greater than n, because is above the diagonal between 0 and n_f,

$$(n) > n \, \forall n \quad t \quad n_f. \tag{54}$$

Hence, the threshold itself is also mapped to a value greater than t, $(t) > t$, so every $n > t$ must also satisfy (49), by definition of monotonicity.