Deep Adaptive Sampling and Reconstruction using Analytic Distributions

FARNOOD SALEHI*, Disney Research | Studios, Switzerland
MARCO MANZI*, Disney Research | Studios, Switzerland
GERHARD ROETHLIN, Disney Research | Studios, Switzerland
ROMANN WEBER, Disney Research | Studios, Switzerland
CHRISTOPHER SCHROERS, Disney Research | Studios, Switzerland
MARIOS PAPAS, Disney Research | Studios, Switzerland

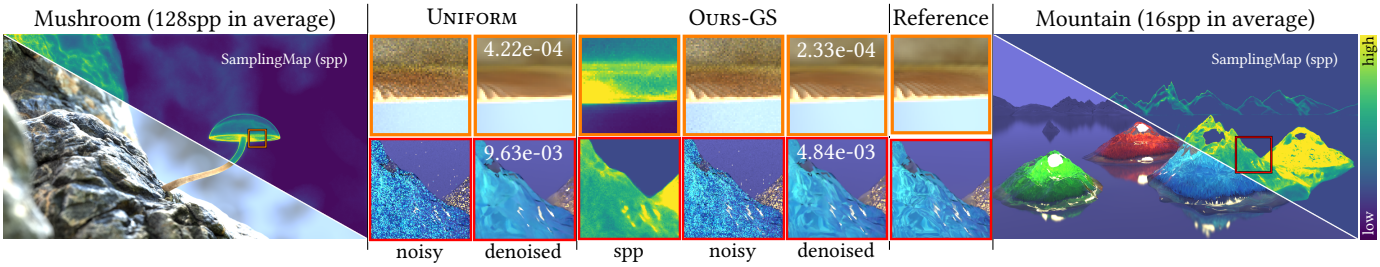


Fig. 1. We compare our proposed approach—a jointly trained adaptive sampling and reconstruction method using analytic distributions and a global summary module (Ours-GS)—to uniform sampling followed by a neural denoiser, akin to the one proposed by Vogels et al. [2018]. For both methods, we show insets before and after the reconstruction. For Ours-GS, we additionally show the sampling distribution, where brightness scales with sample count. We show the samples per pixel (spp) across the entire image, and the average mean relative squared error (MRSE) over the entire reconstructed image.

We propose an adaptive sampling and reconstruction method for offline Monte Carlo rendering. Our method produces sampling maps constrained by a user-defined budget that minimize the expected future denoising error. Compared to other state-of-the-art methods, which produce the necessary training data on the fly by composing pre-rendered images, our method samples from analytic noise distributions instead. These distributions are compact and closely approximate the pixel value distributions stemming from Monte Carlo rendering. Our method can efficiently sample training data by leveraging only a few per-pixel statistics of the target distribution, which provides several benefits over the current state of the art. Most notably, our analytic distributions' modeling accuracy and sampling efficiency increase with sample count, essential for high-quality offline rendering. Although our distributions are approximate, our method supports joint end-to-end training of the sampling and denoising networks. Finally, we propose the addition of a global summary module to our architecture that accumulates valuable information from image regions outside of the network's receptive field. This information discourages sub-optimal decisions based on local information. Our evaluation against other state-of-the-art neural sampling methods demonstrates denoising quality and data efficiency improvements.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Neural networks**.

Additional Key Words and Phrases: Adaptive sampling, denoising, path tracing

ACM Reference Format:
Farnood Salehi, Marco Manzi, Gerhard Roethlin, Romann Weber, Christopher Schroers, and Marios Papas. 2022. Deep Adaptive Sampling and Reconstruction using Analytic Distributions. *ACM Trans. Graph.* 41, 6, Article 259 (December 2022), 16 pages. https://doi.org/10.1145/3550454.3555515

1 INTRODUCTION

Rendering high-quality images with accurate lighting currently relies on running Monte Carlo (MC) simulations of the light transport. Unfortunately, these methods are expensive, as rendering a single image in high quality in offline setups can require dozens or hundreds of CPU hours. At its core, MC rendering methods repeatedly sample random light paths to estimate the color values of pixels. However, these estimates suffer from variance that is inversely proportional to the number of evaluated samples. Interestingly, not all pixel colors are equally difficult to estimate and may thus require a different number of samples to achieve acceptable quality.

One common strategy to reduce render costs is *adaptive sampling*, which takes the varying estimation error of pixels into account by adjusting the sample densities over the image plane during rendering. Another is *denoising*, which is a cheap post-processing reconstruction step that removes remaining noise in renderings. Modern rendering systems make extensive use of both strategies to keep the rendering cost low. Note that since the sample distribution influences the denoising quality, an ideal adaptive sampling strategy

*Both authors contributed equally to the paper

Authors' addresses: Farnood Salehi, Disney Research | Studios, Zurich, Switzerland, farnood.salehi@disneyresearch.com; Marco Manzi, Disney Research | Studios, Zurich, Switzerland, marco.manzi@disneyresearch.com; Gerhard Roethlin, Disney Research | Studios, Zurich, Switzerland, gerhard.roethlin@disneyresearch.com; Romann Weber, Disney Research | Studios, Zurich, Switzerland, romann.weber@disneyresearch.com; Christopher Schroers, Disney Research | Studios, Zurich, Switzerland, christopher.schroers@disneyresearch.com; Marios Papas, Disney Research | Studios, Zurich, Switzerland, marios.papas@disneyresearch.com.

0730-0301/2022/12-ART259 $15.00
https://doi.org/10.1145/3550454.3555515

should be aware of how additional samples will affect the denoised image.

Adaptive sampling and reconstruction based on data-driven neural methods has been shown to outperform non-neural counterparts by a large margin [Kuznetsov et al. 2018]. Current denoiser-aware adaptive sampling methods that are compatible with neural denoisers either do not take into account the effect of future samples on the denoising error, such as in the case of direct error prediction (DEP) [Vogels et al. 2018], or require significantly more data to train, such as the deep adaptive sampling and reconstruction approach (DASR) [Kuznetsov et al. 2018].

We propose a novel, end-to-end trained neural adaptive sampling and reconstruction approach that yields results superior to DASR and DEP when using commensurate training datasets. Further, our approach produces results equivalent to or better than DASR when using the same volume of data for training as required by DEP or a neural denoiser such as KPCN [Bako et al. 2017; Vogels et al. 2018]. As a result, smaller legacy datasets used to train denoisers may be used to train our sampler and denoiser in an end-to-end fashion.

Our method performs on-the-fly synthesis and sampling of plausible future renderings using per-pixel analytic distribution models that are easy and efficient to construct and sample from. Further, we propose the addition of a global summary module that provides the network with non-local information about the input data, further improving the sampling distributions.

We designed our method for the specific use case of high quality offline rendering of single frames, where the sampling distribution is refined iteratively during rendering.

We evaluate the effect of our design decisions through ablation studies and provide an extensive evaluation with qualitative and quantitative comparisons against state-of-the-art methods, showing that our method can achieve superior results with equal training data. In our experiments targeting high-quality renderings, we observed a reduction of $26-37\%$ in the samples needed to reach the same average quality as other state-of-the-art adaptive sampling methods. Such a reduction amortizes to considerable cost savings in demanding production settings.

Our main contributions can be summarized as follows;

- We propose a method to jointly train a neural sampler and denoiser in an end-to-end fashion by constructing future renderings from analytic distributions instead of assembling them from large amounts of rendered data as in previous work.
- We propose a global summary module that allows the sampler to leverage non-local information to improve sampling decisions.
- We provide a thorough theoretical analysis, proving that under certain assumptions on the loss and the denoiser we can use an analytical distribution to generate data while still yielding an exact solution to our problem formulation.
- We demonstrate that the approximation error of our chosen distribution vanishes at high sample counts even when our assumptions do not hold.

2 RELATED WORK

Early work on adaptively sampling within the image plane was based on contrast [Kirk and Arvo 1991; Mitchell 1987], confidence bounds of the mean [Tamstorf and Jensen 1997], or on image discontinuities [Bala et al. 2003; Guo 1998]. While easy to apply, these methods do not adjust for variance or bias introduced by the denoising techniques and can therefore lead to sub-optimal sampling maps when the goal is to minimize the rendering error after denoising.

Methods to couple denoising with adaptive sampling have been explored extensively in the past decade under the umbrella of *adaptive sampling and reconstruction*. While many such methods have been proposed, most adaptive sampling methods are specifically tailored to the denoising method with which they are coupled [Bauszat et al. 2015, 2011; Bitterli et al. 2016; Li et al. 2012a; Moon et al. 2014, 2015, 2016; Overbeck et al. 2009; Rousselle et al. 2011, 2013].

Some exceptions exist that can operate on arbitrary reconstruction methods. One such example is adaptive sampling based on the empirical two-buffer variance after denoising [Rousselle et al. 2012]. This approach requires rendering multiple independent images that are then denoised separately. However, this is not always practical, as it can lead to noisy variance estimates and cannot detect bias in the denoised image due to under-sampling. Another general adaptive sampling method is based on Stein's unbiased risk estimator (SURE) [Li et al. 2012a; Rousselle et al. 2013]. Unfortunately, SURE requires the partial derivative of the output of the denoiser with respect to its input during inference, which can be very expensive to obtain. Alternative adaptive sampling methods operate on additional dimensions other than the image plane [Hachisuka et al. 2008] or rely on analytic properties of the integrand to determine the sampling budget [Durand et al. 2005].

Neural networks have had tremendous success in recent years in the task of denoising Monte Carlo renderings. Chaitanya et al.'s [2017] interactive method utilizes a network that directly outputs the denoised image. It utilizes a recurrent convolutional neural network that leverages temporal information from past frames. Kalantari et al. [2015] learn optimal parameters to denoise using traditional filters. Bako et al. [2017] introduce a kernel-predicting convolutional network (KPCN), which, instead of learning parameters for traditional filters, predicts individual per-pixel kernels. This offline denoising method was then extended to use an efficient multi-scale reconstruction and to leverage temporal information [Vogels et al. 2018]. A further extension of the method decomposes the image automatically into easier-to-denoise components [Zhang et al. 2021]. More recently, Isik et al. [2021] proposed an interactive denoiser that produces temporal kernels based on pairwise affinity between predicted features of neighboring frames. Although any differentiable denoiser would work with our proposed end-to-end method, we decided to use an offline kernel-predicting denoiser akin to KPCN [Bako et al. 2017; Vogels et al. 2018; Zhang et al. 2021].

Our approach is most closely related to two recent adaptive sampling methods based on deep neural networks using supervised learning. The first one is the direct error prediction approach (DEP) described in Section 5 of Vogels et al. [2018]. In a first step, DEP learns to predict the remaining error after denoising by training on pairs of denoised images and their corresponding denoising error

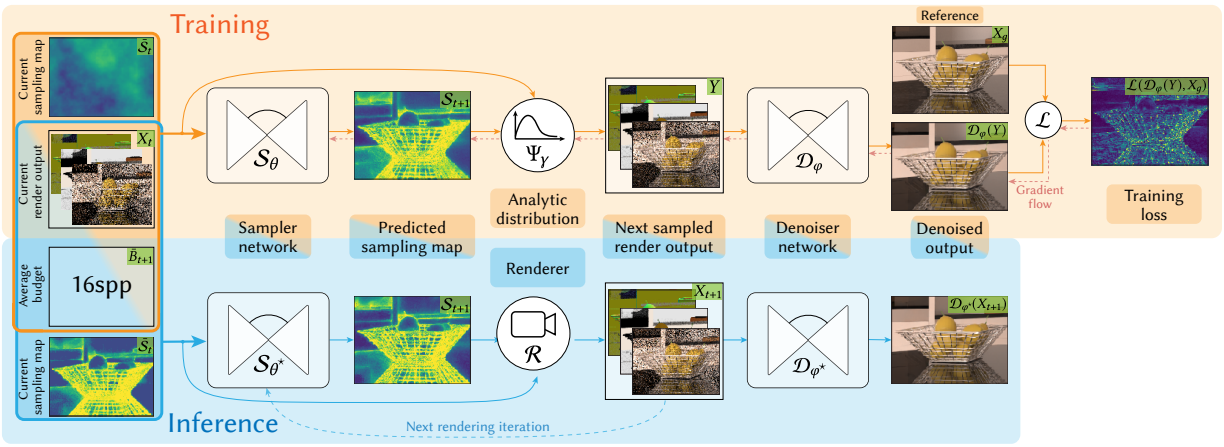


Fig. 2. A schematic representation of our training (top) and inference (bottom) pipelines. Training: Given the rendered data at the current iteration, X_t, rendered with a sampling map $\bar{S}_t$ and the desired total budget B_{t+1}, we utilize an analytic noise distribution Ψ_Y to synthesize noisy data Y for jointly training an adaptive sampling network $\mathcal{S}_\theta$ and a denoiser $\mathcal{D}_\varphi$ to minimize the loss $\mathcal{L}$ of denoised result given ground-truth reference X_g. Inference: During rendering we can use the optimized sampler $\mathcal{S}_{\theta^\star}$ to produce adaptive sampling maps given the current rendered data X_t and current sampling map $\bar{S}_t$ along with the next iteration budget B_{t+1}. The jointly trained denoiser $\mathcal{D}_{\varphi^\star}$ can be used to denoise the adaptively sampled result from the renderer $\mathcal{R}$.

maps. In a second step, DEP obtains sampling maps by normalizing the predicted error image and re-scaling it to obtain a desired sampling budget. DEP assumes that the future error reduction at each pixel is proportional to its current denoising error. While intuitively reasonable, this is only a heuristic that can lead to sub-optimal solutions. For instance, this heuristic does not account for how additional samples in a pixel may affect the error in surrounding pixels. Another drawback of DEP is that it only allows a one-way coupling between the denoiser and sampler. That is, the sampler optimizes its distributions for the denoiser, but the denoiser does not learn to optimally denoise renderings with the sampling distributions obtained from the sampler.

The second method closely related to our work is deep adaptive sampling and reconstruction (DASR) originally introduced by Kuznetsov et al. [2018], which minimizes the expected future denoising error for a given budget and was originally designed for interactive rendering. DASR has been extended to the temporal domain by Hasselgren et al. [2020] and has also been used in the two-stage denoising method of Xiang et al. [2021]. DASR and related methods train a sampler and a denoiser simultaneously in an end-to-end fashion: Given a rendered noisy input, the sampler produces a sampling map used to render a less noisy image that is then denoised and compared against a ground-truth reference. Such end-to-end training requires the rendering after sampling to be computed on the fly, since it depends on the output of the sampler being trained. The naïve way to obtain those renderings is to include an actual renderer in the training loop. However, this is typically too expensive and poses challenges for the backpropagation step. DASR circumvents the need to include a renderer in the training loop by constructing the renderings after sampling from pre-computed data. More specifically, *sample count cascades* in powers of 2 are generated for every training example, which allows for efficient composition of plausible renderings and for simple analytical computation of the gradients for backpropagation. The drawback of this method compared to DEP [Vogels et al. 2018] is that it requires significantly more training data and longer training times due to excessive I/O disk usage and limited GPU memory. We found these issues to be amplified in the high-sample-count regimes targeted by our work.

While our method is similar to DASR, we sidestep the need to store and load sample count cascades by synthesizing new renderings from analytic noise distributions. This makes our approach significantly faster to train and less data reliant.

3 METHODOLOGY

In this section, we describe our method for predicting optimal sampling maps for an iterative unbiased Monte Carlo renderer. We provide an overview of our pipeline in Figure 2.

In this work, we assume images are progressively generated by dividing the rendering process into $T+1$ consecutive iterations, each refining the previous result by allocating additional samples that improve the pixel estimates. Based on these estimates, we progressively improve the distribution of the rendering budget over the image plane. To each rendering iteration $t \in [0, T]$ we assign a *budget* B_t that controls how many samples will be computed for all pixels P in that iteration. A *sampling map* $\mathcal{S}_t = \{s_p(t)\}$ with $\sum_{p \in P} s_p(t) = B_t$ controls the specific number of path samples to compute in any pixel $p \in P$ during iteration t. In the first iteration, samples are distributed uniformly, i.e. according to $s_p(0) = B_0/|P| \ \forall p \in P$, yielding the initial pixel estimates. In all subsequent iterations, the sampling maps $\mathcal{S}_t$ are generated according to the procedure described in the following subsections.

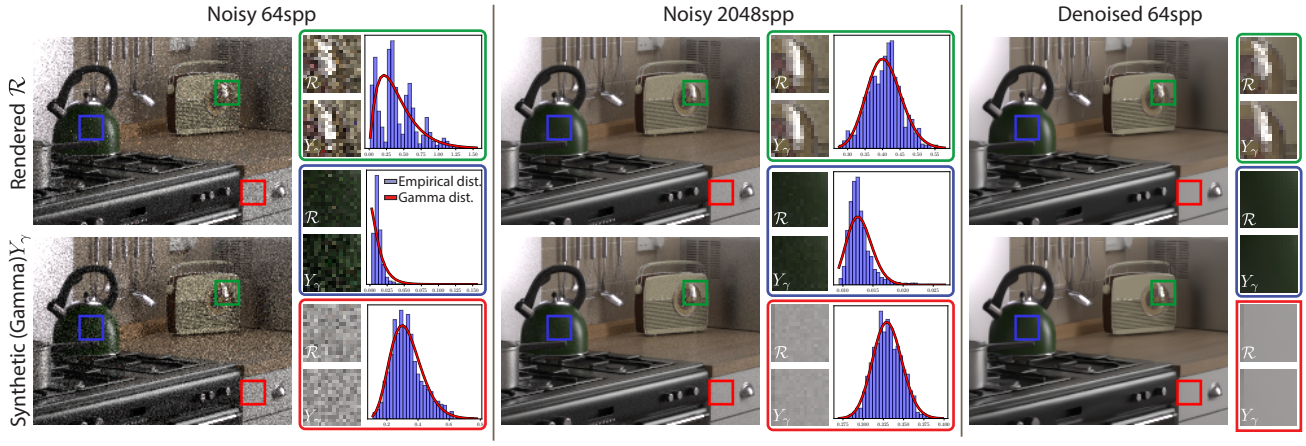


Fig. 3. We compare the pixel means from the empirically rendered distribution $\mathcal{R}$ (top) with ones synthesized from our proposed gamma distribution Y_γ (bottom) across low and high sample counts (left and middle column). For the central pixel in each inset we show the histograms of rendered pixel means collected over 1024 independent renderings (blue) and our proposed analytic approximation based on the gamma distribution (red). Some of the distributions at low sample count are poorly approximated by our gamma distribution (green inset), leading to different noise characteristics in the synthesized images. However, at higher sample counts all distributions become Gaussian-like and can therefore be represented well by our approximation. The right-most column shows how our denoiser discussed in Section 4.5 can properly denoise real and synthetic data.

3.1 Optimizing the Sampling Map

Starting from an iteration t, we define the *optimal sampling map for the next rendering iteration* $t+1$ as

$$\mathcal{S}_{t+1}(X_t, B_{t+1}) = \underset{S}{\operatorname{argmin}}\, \mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}(Y), X_g)\right] \quad (1)$$

subject to

$$\sum_{p\in P} s_p = B_{t+1} \;\text{ and }\; s_p \geq 0,\ \forall p \in P, \quad (2)$$

where $\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)$ is the distribution of rendered outputs obtained *after* adding samples to X_t according to sampling map $\mathcal{S}$, $\bar{\mathcal{S}}_t = \{\sum_{j=0}^{t} s_p(j)\}$ is the accumulated sampling map up to iteration t, and $\mathcal{L}$ is a per-pixel loss aggregated over the image, with denoiser $\mathcal{D}$ and ground-truth reference image X_g.

Sampling from the distribution $\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)$ to produce the next iteration rendering Y can be performed by calculating[1]

$$Y = \frac{\bar{\mathcal{S}}_t X_t + \mathcal{S}\bar{Y}_\mathcal{S}}{\bar{\mathcal{S}}_t + \mathcal{S}}, \quad (3)$$

where every pixel p in $\bar{Y}_\mathcal{S}$ is the average of s_p newly rendered i.i.d. samples. The solution to Equation (1) is the sampling map $\mathcal{S}$ that minimizes the loss of the denoised rendering after the next iteration. Since the values of the samples drawn in the next iteration $t+1$ are not known during the current iteration t, $\mathcal{S}$ is optimized in expectation over the distribution of the next iteration's renderings.

The method described by Kuznetsov et al. [2018] samples Equation (1) by assembling the next iteration rendering Y from precomputed sample count cascades with different seeds. This requires additional data for every training example, which is expensive to generate, store, and process. In the next section, we present our analytic noise synthesis method, which does not have such additional data requirements but still allows us to approximately sample and minimize the loss (Equation (1)).

[1]Throughout the text, all products and quotients on multidimensional arguments are assumed to be element-wise.

3.2 Analytic Noise Distribution Synthesis

Instead of composing pixel values for the next iteration from expensive pre-rendered data as in DASR, we approximate the distribution of the rendered mean values with an easy-to-sample analytic distribution that can directly produce a sampled mean value. With the help of the ground-truth mean and variance of each pixel, and under the assumption that the renderer is using i.i.d. samples, we can analytically model the expected mean and variance of both the rendered and analytic distributions as a function of sample count. We can then directly generate as many samples as needed at each pixel with the desired sample count.

We start by analyzing the behavior of the next iteration's expected denoising error, $\mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}(Y), X_g)\right]$, in a more constrained setup. Let $\bar{\mathcal{S}}_t$ be the cumulative samples distributed up until iteration t, and let $X_t = [I_t, F_t]$ be the rendered (noisy) data composed of color I_t and features F_t at iteration t. We propose the following theorem:

Theorem 1. *For a kernel-based denoiser $\mathcal{D}$ with kernels that do not depend on the specific values of future samples $\bar{Y}_\mathcal{S}$, the following equality holds when either the mean squared error or mean relative squared error is used as the loss (i.e. $\mathcal{L}$ = MSE or $\mathcal{L}$ = MRSE):*

$$\mathbb{E}_{\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}(Y), X_g)\right] = \mathbb{E}_{\Psi(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}(Y), X_g)\right], \quad (4)$$

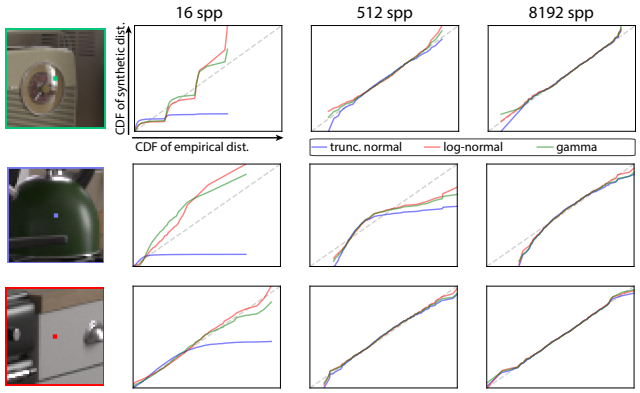


Fig. 4. We assess the similarity of our distribution models with the empirically rendered distribution of sample means. To do so, we show Q-Q plots, i.e. we plot the CDF of the empirical distribution against the CDF of the modelled distribution. The distributions are over the luminance values of the color means for various pixels across different sample counts. Green shows the gamma distribution, blue the truncated normal distribution, and red the log-normal distribution (See Section 3.3 and Section 5.4.) The closer the Q-Q plots to the diagonal line, the more similar the distributions are to the empirical one. We can see that the log-normal and gamma distribution fit the empirical data reasonably well, even at lower sample counts.

where $\Psi(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)$ *is* any *distribution with the same mean and variance as the true rendered distribution* $\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)$. *That is,*

$$\begin{aligned}\mathbb{E}_{\Psi(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y] &= \mathbb{E}_{\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y],\\ \mathbb{V}_{\Psi(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y] &= \mathbb{V}_{\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y].\end{aligned} \tag{5}$$

Our proof, provided in appendix A.2, relies on a distribution-agnostic bias-variance decomposition of the expected future error (Equation (1)) when MSE is used as the loss. We show that when samples are generated from a distribution with the same mean and variance as the true rendered distribution, we recover a loss with identical bias and variance.

Note that in Theorem 1 the denoising kernel is still allowed to be a function of the current data X_t, the current sampling map $\bar{\mathcal{S}}_t$, the future sampling map $\mathcal{S}$, and any other rendered statistics that future renderings will have. As an example, the result of Theorem 1 is still valid for the optimal expected denoiser

$$\varphi_E^\star(\mathcal{S},\bar{\mathcal{S}}_t,X_t) = \underset{\varphi}{\operatorname{argmin}}\,\mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}_\varphi(Y),X_g)\right].$$

The optimal expected denoiser $\varphi_E^\star$ is a function of the current values $(\bar{\mathcal{S}}_t,X_t)$ and the next iteration's sampling map $\mathcal{S}$. However, for Theorem 1 to hold, the denoiser cannot react to outliers (e.g. fireflies) sampled in the future. Since the effect of outliers decreases with the sample count, Theorem 1 holds in the limit as the sample count increases to infinity in a more general setting.

Thus, under the aforementioned assumptions, we can exactly solve the original problem in Equation (1) by replacing the rendered samples with samples from any analytic distribution $\Psi(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)$ whose first two moments match those of the rendering distribution.

For each pixel, we sample from a distribution with the same mean and variance as the rendered pixel distribution. The expectation of the future rendered mean at each data channel (e.g. color, normal, albedo) after allocating $\mathcal{S}$ additional samples can be analytically computed from Equation (3) as

$$\mathbb{E}_{\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y] = \frac{\bar{\mathcal{S}}_t X_t + \mathcal{S}X_g}{\bar{\mathcal{S}}_t+\mathcal{S}}, \tag{6}$$

where X_g is the ground-truth. The variance of each channel is then given by

$$\mathbb{V}_{\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}[Y] = \frac{\bar{\mathcal{S}}_t^2\hat{\mathbb{V}}[X_t] + \mathcal{S}\mathbb{V}_{X_g}}{(\bar{\mathcal{S}}_t+\mathcal{S})^2}, \tag{7}$$

where $\hat{\mathbb{V}}[X_t]$ is a numerical estimate of the variance of the current data and $\mathbb{V}_{X_g}$ is the reference sample variance.

Note that the relation in Equation (4) is exact only when the loss is MSE or MRSE, and for other losses it is only an approximation. We will show that by judiciously choosing an approximate distribution Ψ that closely matches the rendering distribution, we can efficiently produce high-quality results with a fraction of the data required by competing methods.

3.3 Choice of Distribution

The noise distribution of rendered pixel mean values has been studied by Elek et. al [2019] in detail. They observe that the shapes of rendered pixel mean distributions commonly resemble gamma distributions.

In Figure 3 we provide a comparison between rendered pixel mean values and a gamma distributions with corresponding statistics, while in Figure 4 we provide Q-Q plots to illustrate the quality of fit for several analytic distributions. We observe that despite some discrepancies between the rendered and analytical distributions in low sample count cases, these discrepancies vanish with higher sample counts. This can be justified by the central limit theorem, which states that the averages of random variables converge to Gaussian distributions. Indeed, the sample-averages from our analytic distributions and rendered pixel means both converge to Gaussian distributions with the same mean and variance. Furthermore, for the gamma distribution it is easy and efficient to analytically compute the distribution of the average of N such samples ($\sum_i^N x_i/N$), as this is also gamma distributed. This allows us to directly sample the mean with only a single sample from the "mean distribution" instead of sampling from the sample distribution N times and averaging.

Thus, to approximately sample Y in Equation (3), we generate new samples from a gamma distribution and combine each sample $Y_\gamma \sim \text{gamma}\,(\alpha(\mathcal{S}),\beta(\mathcal{S}))$ with the current data:

$$\frac{\bar{\mathcal{S}}_t X_t + \mathcal{S}Y_\gamma(\mathcal{S})}{\bar{\mathcal{S}}_t+\mathcal{S}} \sim \Psi_\gamma(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t). \tag{8}$$

The combined sample distribution (8) will have the same expected mean (6) and variance (7) as the rendered data when the gamma distribution has the ground truth value X_g as its mean and a variance of $\mathbb{V}_{X_g}/\mathcal{S}$. The shape α and rate β parameters of a gamma distribution with these properties can be computed as a function of the new samples $\mathcal{S}$, the ground truth value X_g, and the ground

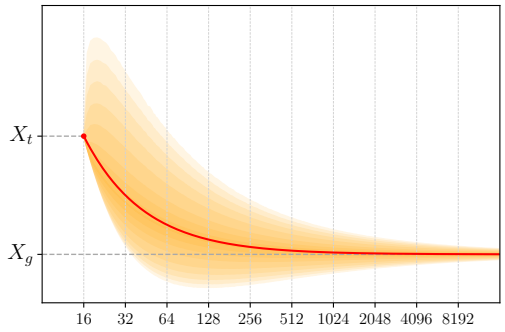


Fig. 5. For a pixel with ground truth mean $X_g = 0.38$ and sample variance $\mathbb{V}_{X_g} = 0.9$, we show the combined sample distribution from Equation (8) at different sample counts $\mathcal{S}$ given a sampled mean of $X_t = 0.78$ at $\bar{\mathcal{S}}_t = 16$ samples. The solid red line shows the expected mean of the resulting distribution (Equation (6)) and the various shades of yellow show the different quantile ranges.

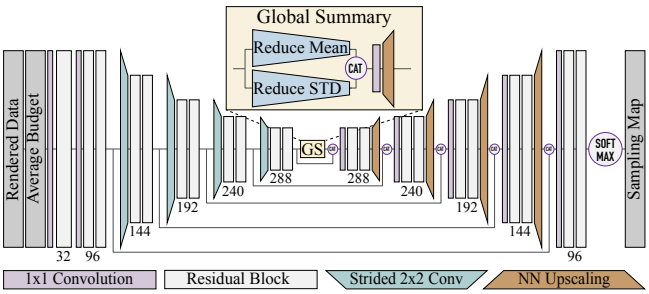


Fig. 6. Schematic representation of our 5-scale U-net architecture for sampling map prediction. The network receives as input rendered data and the desired sampling budget for the next iteration. The output is a scalar value per pixel that is soft-maxed and then multiplied by the desired total sampling budget. Each residual block consists of two 3x3 convolutions each prefaced with a ReLU. At the coarsest scale, our *global summary* module extracts statistics from the encoder's output which we can provide as input to the following decoder.

truth sample variance $\mathbb{V}_{X_g}$:

$$\alpha(\mathcal{S}) = \frac{{X_g}^2 \odot \mathcal{S}}{\mathbb{V}_{X_g}}, \ \beta(\mathcal{S}) = \frac{X_g \odot \mathcal{S}}{\mathbb{V}_{X_g}}. \tag{9}$$

In Figure 5 we visualize an example of the combined sample distribution (8) generated by the aforementioned procedure.

We can now replace the rendering distribution $\mathcal{R}(Y|\mathcal{S}, \bar{\mathcal{S}}_t, X_t)$ with $\Psi_Y(Y|\mathcal{S}, \bar{\mathcal{S}}_t, X_t)$, sample from it efficiently, and use it to optimize Equation (1).

3.4 Jointly Optimizing Sampling and Denoising

So far we discussed how to optimize the sampling map $\mathcal{S}$ to minimize the expected loss of the denoised rendering after the next iteration for a given fixed pre-trained denoiser $\mathcal{D}$ (Equation (1)). However, a lower loss can be achieved by optimizing the sampler $\mathcal{S}$ and the denoiser $\mathcal{D}$ jointly. To that end, we model both as convolutional neural networks parameterized by trainable parameters θ and φ, respectively. We can optimize them by solving

$$\underset{\theta,\varphi}{\operatorname{argmin}} \sum_{\iota_t \in \mathcal{T}} \mathbb{E}_{Y \sim \Psi_Y(Y|\mathcal{S}_\theta(\bar{\mathcal{S}}_t, X_t, B_{t+1}), \bar{\mathcal{S}}_t, X_t)} \left[\mathcal{L}(\mathcal{D}_\varphi(Y), X_g) \right] \tag{10}$$

over a large training set $\mathcal{T}$ comprised of tuples $\iota_t = (X_t, \bar{\mathcal{S}}_t, X_g, B_{t+1})$ made up of rendered data, the corresponding sampling maps, ground-truth data, and the desired sampling budget for the next iteration. Note that Equation (10) is also subject to the constraints Equation (2).

4 IMPLEMENTATION

We implement our method in TensorFlow [Abadi et al. 2015], which we utilize both for training and evaluation. In this section, we describe in detail the architectures of the denoising and sampling networks, our global summary module, how to sample and back-propagate through our analytic distributions, our datasets, and our training procedure.

4.1 Denoiser and Sampler Architectures

We use a two-pass kernel-predicting denoiser with a U-net architecture, similar to KPAL-DS in Zhang et al. [2021] but with minor changes to the architecture. By "two-pass", we mean that we denoise the specular image contribution separately from the diffuse image contribution. We do not perform albedo division on the diffuse channel as proposed in [Vogels et al. 2018].

We use a U-net with five scales instead of three. The convolution bandwidth starts at 64 at the finest scale and doubles at each coarser scale until a maximum of 256, which is repeated for the remaining scales.[2] For the color input to the denoiser (I), we use either the mean of the diffuse (I_d) or specular (I_s) channel, depending on the pass; in both passes we use the albedo F_a and the surface normal F_n. Both albedo and color are log transformed, leading to the input $X_t = [\log(1 + I), \log(1 + F_\mathrm{a}), F_\mathrm{n}]^\top$.

The sampler networks for all adaptive sampling methods compared in this paper employ a U-net architecture similar to our denoiser, but predict a single scalar value per pixel instead of kernels. We provide a schematic of the shared sampler architecture in Figure 6. We also visualize the addition of our global summary module, which aims to provide context from the entire input image to our sampling network (see Section 4.2).

To produce a sampling map, we normalize the output of the sampler network Φ_θ with a softmax and multiply it by the next iteration's budget B_{t+1}:

$$\mathcal{S}_\theta(X_t, \bar{\mathcal{S}}_t, B_{t+1}) = B_{t+1} \operatorname{softmax}(\Phi_\theta(X_t, \bar{\mathcal{S}}_t, \bar{B}_{t+1})), \tag{11}$$

where $\bar{\mathcal{S}}_t$ is the current accumulated sampling map and $\bar{B}_{t+1} = B_{t+1}/|P|$ is the desired average pixel sample budget. This tells the network how many samples will be distributed over the image on average. Note that this is an important input feature, since changing the iteration budget has a non-linear effect on the optimal sampling map (Figure 11). After applying our transforms, the input to the

[2] That is, the bandwidths are 64, 128, 256, 256, 256.

sampler network is:

$$\tilde{X}_t = \begin{bmatrix} \log(1+I_\mathrm{s}) \\ \log(1+I_\mathrm{d}) \\ \log(1+F_\mathrm{a}) \\ F_\mathrm{n} \\ \log(1+\bar{\mathcal{S}}_t) \\ \log(1+\bar{B}_{t+1}) \end{bmatrix},$$

where I_s and I_d are the specular and diffuse components of the color buffer.

4.2 Global Summary Module

One issue we observe while evaluating our implementation of DASR [Kuznetsov et al. 2018] and our method (Ours) is the occasional presence of halos in the predicted sampling maps (e.g. Figure 10). We later demonstrate that this undesired behavior is due to the narrow receptive field of the network and can be ameliorated by providing context from distant regions.

Our solution to this problem utilizes our *global summary* (GS) module as depicted in Figure 6. We place this module between the encoder and decoder in the coarsest U-net scale. It computes statistics on the output of the encoder, which, during inference, are aggregated over the pixels of the entire input. We first compute the mean and standard deviation on each channel of the output of the coarsest scale encoder and concatenate the result along the channel dimension. A 1×1 convolution then reduces the number of channels, and we broadcast the result to the same image shape as the initial input via nearest-neighbor upscaling. Finally, we concatenate the global summary and encoder output and provide it as input to the subsequent decoder.

Although this module produces statistics over pixels of finite patches during training, we can apply it over the entire image during inference. These extracted statistics provide valuable context to the sampler about the input values outside of the network's receptive field and lead to higher quality sampling maps without the halo artifacts we noted above. We provide an analysis of the benefits of our global summary module in Section 5.4.

4.3 Analytic Noise Distributions

To compute the gradient of analytical distributions (Gaussian and log-normal), we use the re-parameterization trick [Kingma and Welling 2013]. For gamma distributed random variables, we found that although their gradients can be computed with the implicit re-parameterization [Figurnov et al. 2018], such gradients are often unstable and result in divergence of the training. For better training behavior, we adopt the finite difference technique introduced by Kuznetsov et al. [2018] for DASR to estimate the gradient of the rendered values with respect to the sampling map.

Our method relies on high-quality estimates of the per-pixel sample variance $\mathbb{V}_{X_g}$. We estimate these quantities during the rendering of our ground truth training data. We remove the remaining noise from our reference sample variance estimates with a specially trained kernel-based denoiser on our training dataset. This variance denoiser follows a similar structure to our color denoiser, and we train it by minimizing the SMAPE error [Vogels et al. 2018] between the denoised sample variance estimate of X_t, namely $\bar{\mathcal{S}}_t \cdot \mathbb{V}(X_t)$, and the ground-truth variance $\mathbb{V}_{X_g}$.

The transformed input to our variance denoising network is then

$$\left[\log(1+\bar{\mathcal{S}}_t \cdot \mathbb{V}(X_t)), \log(1+F_\mathrm{a}), F_\mathrm{n}\right].$$

Note that we only use denoised sample variance from reference estimates during training of the sampler network to generate samples for color and feature channels. The RGB components of color samples are often assumed to be correlated. Hence, to enforce the correlation, we use the same seed when sampling the distribution with the desired statistics for each channel of Y_γ in Equation (8). Note that for the surface normal feature F_n we are using a Gaussian distribution clipped between −1 and 1, since the gamma distribution cannot produce negative values.

In Figure 13, we evaluate various sampler networks trained with different analytic distributions. We observe that the choice of distribution is more important at lower sample counts and that the gamma distribution yields the highest quality results. Experimentally, we found that lowering the variance of the noise distribution also lowers the variance of the gradients during backpropagation and helps convergence, although this adds bias to the predicted sampling map.

We settled on dividing the model variance by a factor of 5 as a trade-off between improving training convergence and reducing the quality of the results at low sample counts.

4.4 Dataset

As explained earlier, one of the tested baselines (DASR) requires cascades of the pre-rendered images to generate future states. Hence, we created two types of datasets: (1) a dataset with such cascades used only by DASR, and (2) another dataset without cascades used to train our method and the other baseline (DEP). For all methods, we use the second dataset to train the denoisers. We ensured that each method used an equal volume of data.

We have 17 base scenes, from which we generate training examples by perturbing the base scenes for the two datasets. The perturbations modify the camera, lighting, and material parameters of the base scenes and spawn new objects and area light sources at random locations, scales, and orientations. We use 4000 images for both datasets, resulting in approximately 800 gigabytes of data. Unless stated otherwise, the dataset used by DASR contains about 200 scene perturbations, and the dataset used to train the denoiser and our method contains about 800.

After generating the scene perturbations, we render them with Mitsuba [Jakob 2010] at various sample count levels. We increase the diversity of our data by rendering each scene perturbation four times with different procedurally generated sampling maps with an average sample count of 16, 64, 256 and 1024 samples per pixel. The sampling maps were generated by combining Perlin noise images with different frequencies. We emphasize that spatial variation in the sampling map of the training examples is crucial for both the sampler and the denoiser to work well for iterative adaptive sampling. Without it, both networks perform poorly in later sampling iterations due to the sampling map $\bar{\mathcal{S}}_t$ having large spatial variation. In the DASR training dataset, for each perturbation of the base scene, we generate an additional cascade consisting of

$K \in \{0, 1, 2, \cdots, 13\}$ independently rendered images with uniform sampling at 2^K samples per pixel. This cascade allows us to generate rendered data with up to 16000 samples. To increase the data efficiency, we share the same cascade for all four training examples of a scene perturbation. Despite this cascade-reuse optimization, DASR still requires roughly four times more disk space per training example compared to our method and DEP. Without increasing the dataset size, using additional cascades will lower the number of training examples for DASR, and we ultimately decided to prioritize diversity in our training examples (i.e., more training examples from a scene perturbation) over cascades.

We further augment our data during training by rotating the patches, permuting the color channels and re-scaling the color channels by a random scale r and the corresponding variances by r^2. Finally, we render for every perturbation a reference image with $2^{17} = 131072$ samples per pixel.

Our evaluation set consists of 24 hand-crafted scenes, and we perform quantitative comparisons with a variety of metrics in Section 5. In this report we show qualitative results for only a small subset of the test scenarios. The full evaluation set can be inspected in our supplemental viewer.

4.5 Training

Our training is composed of two training phases. First we train the denoiser network, and then we train the sampler network with the pre-trained denoiser. For our method and DASR, the sampler and denoiser networks are jointly trained end-to-end, whereas for DEP the pre-trained denoiser is fixed and is not jointly trained. During all phases, we extract patches of size 128×128 from each frame and use mini-batches of size 4.

Denoiser Training. First, our five-scale kernel-based denoiser for color is trained with the training loss proposed in DASR [Kuznetsov et al. 2018],

$$\ell(x', {X_g}') = 0.5 \left\| \frac{x' - {X_g}'}{{X_g}' + \epsilon} \right\|_1 + 0.5 \left\| \frac{L(x') - L({X_g}')}{{X_g}' + \epsilon} \right\|_1, \qquad (12)$$

where L is the *Laplacian of Gaussian* operator, and $\epsilon = 0.01$. As in DASR, we use the log transform to control the range of colors, $x' = \log(1 + x)$. We refer to this loss as logLoss in the following sections. This loss penalizes the mismatch in the edges and can be seen as a sort of perceptual loss. To train the variance denoiser, we use the more robust Symmetric Mean Absolute Percentage Error (SMAPE) [Vogels et al. 2018]. Our training schedule uses an Adam optimizer with 2 million iterations at a learning rate of 2×10^{-5}. This is followed by two fine-tuning phases of 250K iterations, with a learning rate reduction factor of 10 in each phase. We provide more details on the denoiser training in Section A.1.

Joint training of Sampler and Denoiser. Second, we train the sampler and denoiser jointly. We use the pre-trained denoiser in the previous step as the starting checkpoint for the denoiser. We follow the same training schedule as for the denoiser with logLoss as the error metric. This is analogous to the joint training step in DASR [Kuznetsov et al. 2018]. However, there are significant differences. First, we found that directly minimizing Equation (10) degrades the performance of the denoiser on real sampled data since loss Equation (10) operates on synthesized data. Thus we chose to optimize the following combined loss for our training-set example $[X_t, \bar{S}_t, X_g, B_{t+1}]$:

$$\mathbb{E}_{Y \sim \Psi_Y(Y | \mathcal{S}_\theta(\bar{S}_t, X_t, B_{t+1}), \bar{S}_t, X_t)} \left[\mathcal{L}(\mathcal{D}_\varphi(Y), X_g) + \mathcal{L}(\mathcal{D}_\varphi(X_t), X_g) \right]. \qquad (13)$$

The first term is the loss from Equation (10) while the second term ensures that the denoiser still performs well on real data (see Appendix Equation (14)). $\mathcal{L}$ is the mean of the logLoss (Equation (12)) over all image pixels. Second, during refinement we update the parameters of both networks in every iteration step instead of in an alternating fashion as described in Kuznetsov et al. [2018]. We did not observe any degradation by doing so and can reduce the training time by lowering the total number of iterations.

5 RESULTS

5.1 Baselines

Our proposed method operates on single-frame inputs and works with iterative sampling schedules and arbitrary sample counts. We adapt two state-of-the-art neural adaptive sampling methods to this use case and compare them to our method. Namely, we compare to direct error prediction (DEP) [Vogels et al. 2018] and deep adaptive sampling and reconstruction (DASR) [Kuznetsov et al. 2018]. Generally applicable non-neural adaptive sampling approaches such as SURE-based methods [Li et al. 2012b] or more traditional variance-based methods were not considered due to their not being competitive with the state of the art [Kuznetsov et al. 2018].

The original authors' implementations of DASR and DEP differ significantly from each other in terms of network architecture, training data, training schedule, target use case and input features provided to the networks. Comparing the original versions of these methods directly with our method would thus make attribution of any performance differences challenging. Since we are mainly interested in studying the effect of incorporating analytic distributions, discussed in Section 3, we modified the competing methods to use the same input features and network architectures as our method. Further, we kept the training schedule as similar as possible across the methods (Section 4.5) unless explicitly stated otherwise. We verified that these changes did not degrade the performance of either competing method relative to their original setup and even observed *improvements* in the quality and performance of these methods when compared to the authors' original implementations. We describe those changes in detail below.

DEP. A noteworthy change to DEP [Vogels et al. 2018] is that we provide the auxiliary features as additional input to the algorithm and do not provide the denoised color as input. We did not observe any quality degradation as a result. We speculate that this is because we use a U-net based neural network to model the sampler instead of a less-expressive sequence of residual blocks as proposed in Section 5 of Vogels et al.'s [2018] work. Despite the more complex sampler network architecture, this change significantly reduces the inference time of DEP, since no denoising pass is required for preparing the input of the sampler network during rendering. In contrast to our method and DASR, DEP does not perform joint training of the denoiser and sampler. This also allows for faster training iterations

Table 1. This table compares the percentage of the sample budget required by various sampling methods to reach the same quality as uniform sampling (Uniform) in terms of various error metrics over our entire evaluation set. We show the percentages with respect to three quality settings for Uniform, roughly corresponding to 16spp (low), 64spp (mid), and 512spp (high). For all methods, adaptive sampling starts at 8spp. Ours-GS denotes our method with the global summary module, which yields the best overall improvement.

	Relative budget to same average error as Uniform				
	Quality	DEP	DASR	Ours	Ours-GS
log-loss	low	75%	79%	64%	**62%**
	mid	67%	69%	56%	**52%**
	high	60%	61%	50%	**45%**
1-SSIM	low	72%	76%	60%	**58%**
	mid	60%	66%	50%	**45%**
	high	52%	59%	44%	**37%**
MRSE	low	68%	69%	62%	**60%**
	mid	49%	51%	44%	**40%**
	high	40%	43%	36%	**31%**

since the denoised images required to compute the loss can be cached.

DASR. The authors of DASR [Kuznetsov et al. 2018] originally applied their method to *interactive* setups. In their described use case, DASR is only used to predict a sampling map with three samples per pixel on average from an image rendered with exactly one sample per pixel. Our method, by contrast, targets iterative adaptive sampling schemes for *offline* rendering. To directly compare DASR with our approach and our use case, DASR needs to be able to predict sampling maps from already adaptively sampled images with a wide range of sample counts that it may encounter during later rendering iterations.

As explained in Section 4.4, to make this possible we trained DASR on data with various, spatially varying sample counts. Recall that unlike DEP and our method, DASR requires an additional cascade of rendered images for every training example, which affects storage requirements and training time.

Kuznetsov et al. [2018] proposed a three-step training scheme, consisting of first training the denoiser in isolation, then training the sampler and finally both the denoiser and sampler jointly. We found that skipping the second step and training DASR with the two-step training scheme described in Section 4.5 performed slightly better in terms of denoising quality while also reducing training time. We use this simpler training scheme for all our versions of DASR.

The training time for $2.8M$ iterations with DASR is 20 days, Ours is 12 days and DEP is 5 days. During inference, the running time of the denoiser is about 0.8 seconds, and all the sampler networks require about 0.4 seconds for an image of size 720×1280 on a GeForce GTX 1080 Ti.

5.2 Quantitative Analysis

For this experiment, we train all methods as described in Section 4.5. All methods use a comparable amount of training data in terms of disk space to keep comparisons fair. In particular, this means that DASR uses roughly a quarter of the training examples used by DEP and our method, since each training example for DASR requires roughly four times the amount of data due to the sample count cascades. To keep the variety of training examples comparable across all methods, we reduced the number of perturbations per training base scene (see Section 4.4) but did not change the number of base scenes. We compare our method to DEP, DASR, and uniform sampling in an iterative rendering setup. All sampling methods are tested in the same progressive rendering setup, where the initial iteration uniformly distributes 8 samples per pixel and each subsequent iteration doubles the sample budget. That is, the sample budget is given by $B_t = 2^{t+3}$ at iteration t.

Table 1 shows a quantitative comparison under the aforementioned setup. This table shows the relative sampling budget required by each sampling method to match the quality after denoising a uniformly sampled image. Specifically, we match the average denoising error of a uniformly sampled image for three quality levels achieved at roughly 16 (low), 64 (mid), and 512 (high) samples per pixel. We replicate this comparison and match the average error of uniform sampling with three different error metrics (log-loss, 1-SSIM and MRSE). We track the denoising error and sampling budget at each iteration in order to estimate for each method the budget at which it matches a target denoising error.

Note that we compare our method with and without the global summary module (Ours-GS and Ours). By comparing DASR and our method without the global summary module (Ours), we observe that most of the performance improvements can be attributed to the use of our analytic distributions during training. Overall, we measure additional but smaller improvements due to our global summary module across all loss functions and quality levels. For these equal dataset size comparisons we also observe that DASR is not able to improve over error prediction (DEP) even though it uses a more accurate problem formulation for the expected future error. This can be attributed to the higher data requirement of DASR per training example compared to both other methods. In order to keep the dataset size the same, DASR is trained with fewer training examples, which adversely affects its resulting quality. This occurs despite our efforts to improve the data efficiency of DASR by reusing the same cascade for four different training examples. Note that DASR was originally designed for low sample count scenarios requiring only small cascades, which allows for higher data efficiency than our setup. We also provide a comparison with larger dataset sizes in Section 5.4. It is important to note that all adaptive sampling methods provide higher relative improvement over uniform sampling at higher sample counts. Finally, when augmenting our method with the global summary module (Ours-GS) we observe improvement across almost all metrics and configurations.

Finally, we compare the loss of the different methods during training as a function of wall-clock time in Figure 7. We follow the training scheme described in Section 4.5 for all methods except DEP, for which we doubled the number of training iterations (5.6M iterations). During the evaluation, we perform a single adaptive sampling step that doubles the sample count of images rendered with 256 uniformly distributed samples per pixel. We measure the log-loss of every denoised image in the evaluation dataset after the adaptive sampling step. During training, DEP can process more batches per second compared to DASR or Ours, while DASR is the

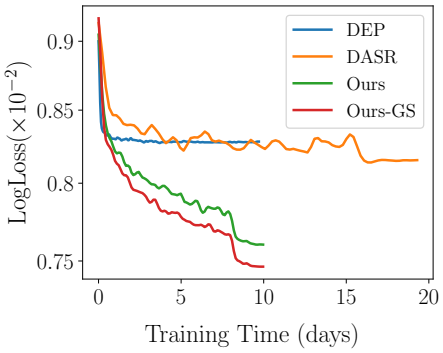


Fig. 7. Convergence plots comparing the evaluation error (logLoss) vs. training time (wall-clock) in high sample count scenarios. Each method receives rendered data with an average of 256 samples per pixel and a budget of 256 samples. We trained all methods for 2.8M iterations except for DEP, for which we used 5.6M iterations. (DEP can achieve double the speed of our method, as it only trains the sampler and does not jointly train the denoiser.) Ours and Ours-GS converge faster than DASR and reach a lower error compared to all other methods.

MRSE (full image)	8.31e-04	5.43e-04	6.05e-04	**4.77e-04**
1-SSIM (full image)	1.23e-02	9.64e-03	1.05e-02	**8.52e-03**

Fig. 8. Failure case: All compared adaptive sampling methods fail to detect the thin shadow and end up distributing roughly half as many samples in the inset region compared to uniform sampling. This leads to under-sampling and failure to reconstruct the shadow. However, despite this local failure, the error over the entire image is still significantly lower with the adaptive sampling methods than with uniform sampling.

slowest. Note that DEP's loss plateaus very quickly, indicating that increasing the number of training iterations will not improve DEP further. DASR converges more slowly but plateaus at a similar error level. Both Ours and Ours-GS reach a significantly lower evaluation loss and seem not to be fully converged at 2.8M iterations.

5.3 Qualitative Analysis

Next, we qualitatively compare the performance of the different sampling methods. We reuse the same networks and sampling schemes during rendering as in Section 5.2. In Figure 9 we compare our method with and without the global summary module (Ours-GS and Ours) to DEP, DASR and uniform sampling (Uniform) in an iterative rendering setup. We show images at various samples per pixel to cover a wider range of quality settings. Note that these examples show only a small subset of our evaluation dataset used in our quantitative analysis. In all shown examples either Ours or Ours-GS achieves the lowest MRSE and 1-SSIM error. Note that the differences between Ours and Ours-GS are very subtle. This is expected since Ours-GS should only yield benefits over Ours in specific types of scenes as discussed in Section 5.4 and showcased in Figure 10.

In GlossyKitchen, we showcase a challenging scenario with high noise at 16 samples per pixel. Despite the extreme amount of noise in this case, our methods are able to reconstruct the shadow of the knife and the darker top rim of the cupboard more faithfully than the other methods.

In Bookshelf, we show another scene with excessive noise at 16 samples per pixel. Our methods are the only ones able to reconstruct the text on the book spines and the highlight on the glossy flower vase.

In ModernKitchen, we observe that all adaptive sampling methods improve over uniform sampling, with Ours-GS creating a higher quality reconstruction of the metallic fruit basket. For the plant we observe that our methods are best at reconstructing the textures on the leaves in the center and are able to better separate the overlapping fore- and background leaves.

In LivingRoom, our methods are the only ones that detect the metal strip on the wall. In addition, the high frequency structure of the plant is best preserved by Ours-GS.

Failure Cases. A general limitation of adaptive sampling—and hence our approach—is that, by definition, it trades off quality in one region for another based on the loss it tries to minimize. Hence, the distribution of error in the final image with our approach is highly dependent on the specific training loss. While our chosen training loss (logLoss) leads to good results across many perceptive and non-perceptive metrics, sometimes the chosen trade-offs were not the ones that we believe would have led to the most perceptually pleasing results. We experimented with using perceptual error metrics such as SSIM and FLIP [Andersson et al. 2020] as training losses but found them to be less robust during training inconsistent in general relative to logLoss. We note that perceptual quality is highly subjective, and finding the ideal training loss would require a dedicated user study. We considered this to be out of scope for this project but believe it to be a interesting avenue for future work.

Another limitation is that the sampler might not be able to detect fine details in the presence of excessive noise, leading to under-sampling and ultimately failure to properly reconstruct some details. We mainly observed this in high variance regions with fine structures that are not captured by any of the auxiliary features, e.g. thin shadows or caustics. An example is shown in Figure 8. Note that all compared adaptive sampling methods fail in this situation, since they rely on the same input and, further, that the sampling distribution still leads to a significant average error reduction over the whole image, meaning that the failure happens in a few highly localized regions only. One remedy to such situations might be to input a richer set of auxiliary features to the network, such as visibility or caustic maps [Rousselle et al. 2013], to facilitate the detection of such subtle image features. More generally, this poses the question of what the best set of auxiliary features might be (similarly to Zhang et al. [2022]).

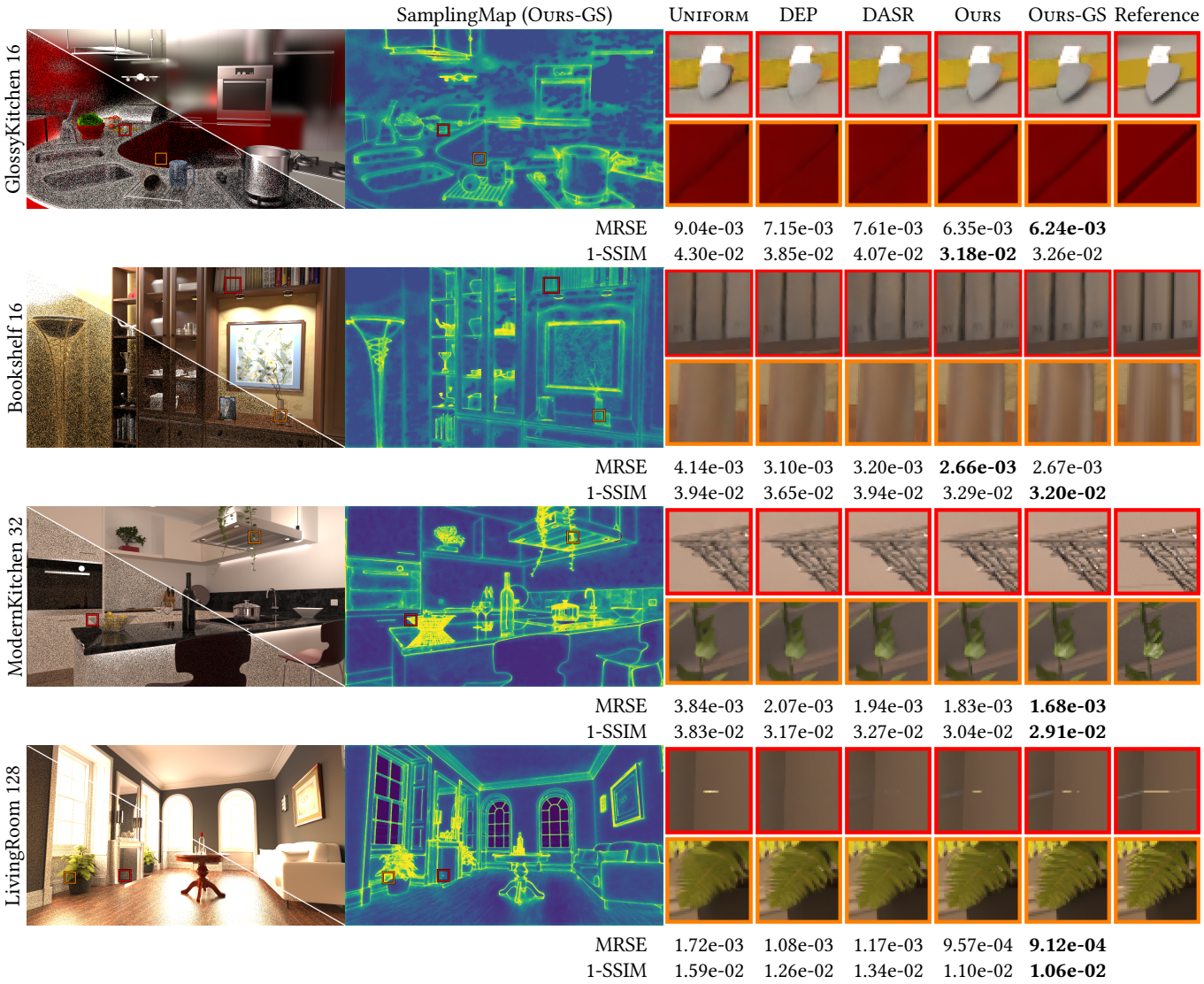


Fig. 9. Qualitative comparison of various adaptive sampling and reconstruction methods at equal sample counts. We compare uniform sampling (Uniform), direct error prediction (DEP, [Vogels et al. 2018]) and deep adaptive sampling and reconstruction (DASR, [Kuznetsov et al. 2018]) to our proposed method without and with the global summary module (Ours and Ours-GS). The noisy images on the left show the scene at the indicated sample count without adaptive sampling or denoising. The numbers below the insets show the average mean relative square error (MSRE) and structural similarity index (1-SSIM) over the *entire* images. For both metrics lower values are better.

5.4 Ablation Studies

Effect of Global Summary Module. In Section 4.2, we propose the addition of a global summary module to our architecture that accumulates valuable information from image regions outside of the network's receptive field. Table 1 shows that the GS module in Ours-GS improves the budget to equal quality compared to Ours. As previously mentioned in Section 5.2 the main improvements of our method over the baselines stem from using the analytic distribution. This suggests that both of our contributions—namely the analytic models for training and the GS module—lead to substantial improvements over the state-of-the-art in combination. We did not test the effect of the GS module in isolation without analytic distribution (i.e. DASR with a GS module) and leave this as future work.

In the majority of scenes the GS module does not affect the quality of our approach significantly (see Ours and Ours-GS in Figure 9). However, in some specific cases it can prevent our sampler from wasting significant computation in the wrong regions. We demonstrate this effect in Figure 10. In the sampling maps of the end-to-end

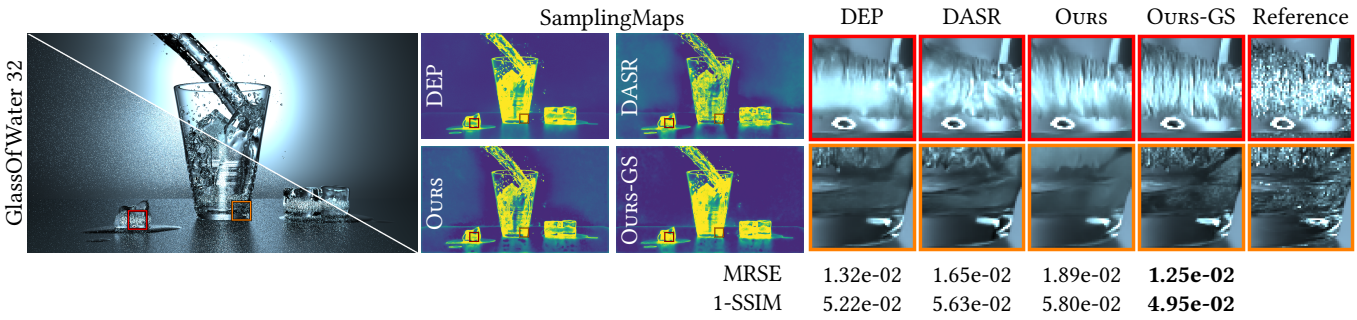


Fig. 10. The original network architectures of our method (Ours) and DASR [Kuznetsov et al. 2018] have a limited receptive field, yielding sub-optimal decisions based on local information. In this scene we observe relatively high sample allocation in the outer regions of the background at the cost of valuable samples from the most difficult regions on the ice and glass. With the addition of our global summary module, our method is able to detect such cases and save valuable samples for the most difficult image regions, yielding higher quality results.

methods without a GS module (DASR and Ours), we observe that the sampling densities increase as we move away from the challenging central region. This leads to a perceived dark halo in the sampling map around the glass of water. This behavior stems from the lack of global knowledge in the networks. Due to a limited receptive field, regions far away from each other in the image plane will not affect each other's predictions by the sampler. Together with the global averaging (with soft-max in Equation (11)) this leads to a problem, since local predictions from the sampler can have global effects in the sampling map. If the network is oblivious to distant regions, this can lead to simpler regions pulling away valuable samples from far away, more difficult regions. This is especially problematic in situations in which highly localized parts of the image require orders of magnitude more samples than the rest to converge. We note that this is not a problem for DEP, which is trained to predict a globally consistent error across images, as supported by the evidence in the figure. Applying our GS module to Ours provides the network with much-needed global context that results in more globally consistent sampling densities, similar to DEP.

Different Sampling Budget. In Figure 11, we change the sampling budget and compare the sampling maps that Ours and DEP produce. The input to the networks is rendered with uniform sampling at 32 samples per pixel and the desired average per-pixel budget for the next iteration $\bar{B}$ is either 32 or 480. The desired budget influences the sampling densities with Ours while it does not affect DEP. In the low-budget case, Ours focuses aggressively on high-error regions. As we increase the budget $\bar{B}$, Ours distributes samples more uniformly, including low-error regions as well. Note that our method is trained to minimize the expected denoising error after the budget is spent, and that is why it adapts its sampling map. On the other hand, DEP is distributing samples according to the predicted denoising error of the current image, and its output does not change significantly for different budgets. Note that DASR would exhibit a similar behavior as Ours in this setup.

Effect of Joint Training. We examined the effect of the joint training step described in Section 4.5 on Ours. To do so, we trained Ours with and without joint training. For the latter case, after training the denoiser, we only trained the sampler and kept the denoiser fixed, akin to the second training step described for DEP in Section 5.1. In Figure 12 we show side by side the evaluation losses of DEP (blue) and both flavors of Ours (orange and green). The figure's caption provides details on how the loss was measured. All training runs for this figure were stopped at 1M iterations. We clearly see that Ours with joint training performs better than Ours without joint training. On top of that, the plot shows that even without joint training, Ours improves over DEP.

In an additional experiment, we compared the performance of the jointly trained denoiser from Ours to a pre-trained denoiser (as used by DEP and Ours without joint training) on *uniformly* sampled data only. The intent of this experiment was to decouple the change of performance of the denoiser due to joint training from potential improvements due to better sampling distributions. Somewhat surprisingly, we observed an improvement of roughly five percent in in terms of 1-SSIM across varying sample counts due to joint training. We see this as evidence that jointly training the denoiser on synthetic data does not degrade the denoiser's performance on real data. In fact, these results could indicate that

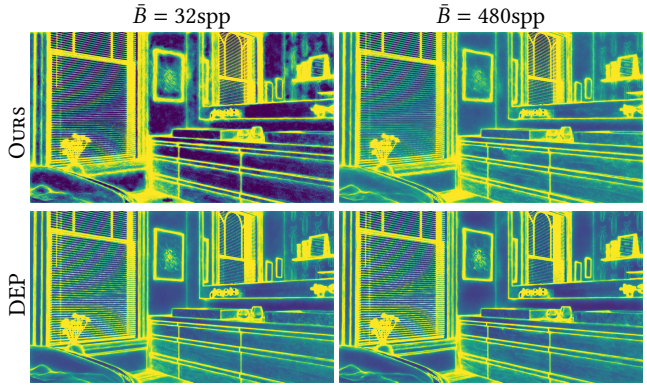


Fig. 11. Given an image rendered uniformly with 32 spp, we compare the normalized sampling maps predicted by DEP and Ours for average per-pixel budgets $\bar{B} = 32$ and $\bar{B} = 480$.

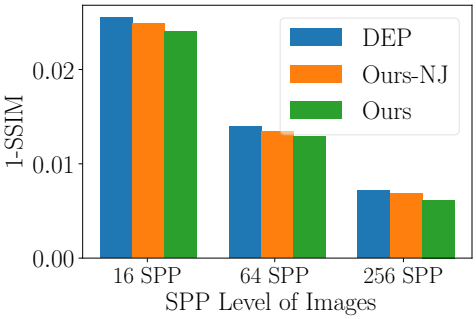


Fig. 12. We compare Ours when only the sampler is trained with a pre-trained denoiser (orange bar), and when the sampler and denoiser are jointly trained starting from a pre-trained denoiser (green bar). We use three evaluation data sets differing only in their sample counts (16, 64, and 256). The loss is measured after the sampler doubled the total sample count and the image has been denoised. Even without joint training Ours-NJ outperforms DEP and with joint training Ours improves even further.

the synthetic data is actually valuable training data for a denoiser, which might open new exciting applications.

Effect of Patch Size. To assess the effect of patch size on training, we trained Ours with two different patch sizes, namely 128×128 and 256×256. Both use a batch-size of 4. When the two networks trained with same number of iterations, the network trained with 256×256 patch size yielded slightly better results. However, training with 256×256 patch size almost quadruples the training time, and training with 128×128 patch size yields better results when trained for the same wall-clock time.

Choice of Analytic Distributions. We analyzed alternatives to our proposed gamma distributions for generating future samples for our method. We considered two additional distributions: The truncated normal distribution and the log-normal distribution.

In Figure 13, we plot the evaluation losses during training for Ours with the three considered distributions. We show three different evaluation losses for scenes at three different sample counts; 16 (left figure), 64 (middle figure), and 256 (right figure). We see that Ours trained with gamma distribution outperforms Ours trained with the other distributions across all spp levels. The difference is however more pronounced for lower sample counts (left column). The truncated normal distribution has the worst performance, the reason being that the negative-valued samples are clipped to zero, effectively biasing the mean of the distribution. At higher spp (middle and right figures), the difference is smaller. This agrees with our observations in Figure 4 that the gamma and log-normal distributions nicely fit the rendered distributions, while the truncated normal distributions does not at low sample counts.

Dataset Size. We compare Ours (without GS module) to DASR when trained with differently sized training datasets. We create three datasets; the smallest has 1K images (roughly 200 gigabytes), another one with 4k images (roughly 800 gigabytes), and the biggest one has 14k images (roughly 3 terabytes). Recall that we keep dataset sizes

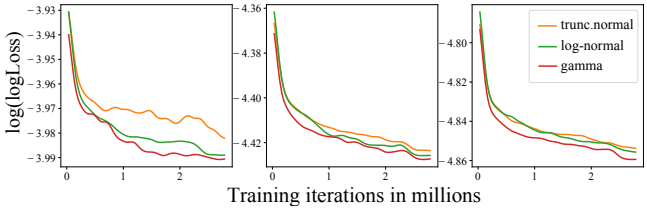


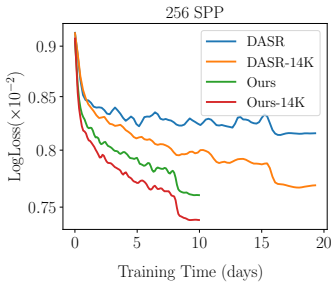
Fig. 13. We train our method with color samples from various analytical distributions (truncated normal, log-normal and gamma). The sampler networks double the sample counts on three data sets at different quality levels (from left to right; 16spp, 64spp and 256spp). The largest error difference is at low sample counts, where the gamma distribution yields the highest accuracy, closely followed by the log-normal distribution.

consistent across all methods by adjusting the number of training examples for DASR, as explained in Section 5.1.

In Figure 14 we compare the denoising error after rendering with the sampling map predicted by each method. We provide as input to all networks renderings with uniform sample counts (256spp in Figure 14(a), and 16spp (left) and 256spp (right) in Figure 14(b)) and double the budget for the next iteration. We see in Figure 14(a) that Ours has a faster convergence than DASR regardless of the dataset size.

Figure 14(b) shows that increasing the size of the dataset improves both methods. But no matter the size of dataset, Ours performs better than DASR, suggesting better data efficiency of our method. Note that even with four times more data, DASR is not able to outperform Ours. As discussed in Section 5.1, each training example of DASR requires roughly four times more data than Ours per training iteration due to the need of sample count cascades. This means a less diverse set of examples is available to train the sampler and fine-tune the denoiser at equal dataset size. In addition, in Ours, at each iteration of training we get new random future states due to the randomness of sampling from the analytic distribution. We speculate that this helps avoiding over-fitting to explicit noise patterns in the available sample count cascades for DASR. We could tackle this problem of DASR by using more random seeds and images in the cascades, but that would further decrease the data efficiency of the approach.

Evaluation at Different Sample Counts. In Figure 15 we evaluate all methods at various sample counts. To enable comparisons at low sample counts, we begin from uniformly sampled images with 2 samples per pixel, and we apply adaptive sampling iteratively while doubling the budget at each iteration up to 128 samples. It is important to note that all methods were trained with higher sample count input starting from 8 average samples per pixel. None of the methods encountered training input with low sample averages (e.g. 2 or 4 spp). We plot the average logLoss over our evaluation set for each method relative to uniform sampling. The main observation is that all methods provide an improvement over uniform sampling, even at low sample counts. Moreover, the improvement of our method (Ours) compared to other methods becomes more noticeable at higher sample counts. Note that at lower sample counts, the expected gain from adaptive sampling is smaller than at higher



(a) Evaluation error (the same loss as the training loss) as a function of time during training. Ours and DASR use 4K images.

(b) Perceptual error (SSIM) as a function of the dataset size.

Fig. 14. Denoising error after rendering the images with the predicted sampling maps as a function of the number of images in the training set. The experiment shows the effect of the dataset size on the networks. For each dataset size and input spp level, our method without the global summary module (Ours) reaches a lower error than DASR. Interestingly, DASR requires at least three to four times more data to match the quality of our method. Note that in all other experiments, we used the middle dataset configuration with roughly 4K images.

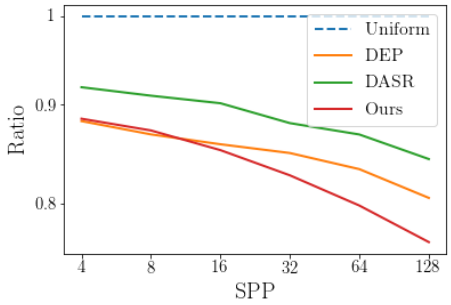


Fig. 15. The rendering starts at 2spp uniformly. We continue the rendering with adaptive sampling, each iteration doubling the sampling count. We plot the 1-SSIM error relative to Uniform sampling.

sample counts. We believe this is due to the increased noise level and thus less reliable predictions of our networks, and because our networks were trained for higher sample count inputs. Further at low sample counts, the proportion of samples distributed uniformly during the warm up phase is higher, giving less margin for the adaptive sampling methods to improve.

Stability and Temporal Coherence. We analyze the stability of the sampling maps and temporal coherency of denoised images. We render two test images with 64 uniform samples per pixel, each with a different seed. With a budget of 64 additional samples, we predict a pair of sampling maps with each method. We compute the per-pixel correlation coefficient for each sampling map pair and average it over all pixels. DEP yields a value 0.98 and DASR and Ours-GS both 0.97. This indicates high resilience to noise for all methods. We can, however, still observe some flickering between the denoised image pairs in all methods for the chosen sample count. While flickering diminishes with more samples, future work on using a temporal denoiser, a temporal loss, and information from nearby frames could improve the temporal coherency of all methods.

6 LIMITATIONS AND FUTURE WORK

Quasi-Monte Carlo Methods. Our model for future means and variances of the mean (Equations (6) and (7)) implicitly assume that the variance of our MC estimates reduces linearly with the number of samples. While this is true for i.i.d. samples, this is in general not true when quasi-Monte Carlo methods are used that change the rate at which variance reduces. However, theoretically, our models could be modified to account for this if we had a reliable way to estimate the convergence rate per pixel.

Limitations of Analytic Distribution Models. While our distributions perform well in high-sample-count regimes due to the central limit theorem, at low sample counts or for exotic multi-modal distributions the limited expressiveness of our analytic distribution model leads to higher approximation errors. Further, our models assume perfect correlation in the noise of the different color channels and features. This can lead to sub-optimal results in cases where this assumption is violated. We leave the exploration of more complex distribution models to future work.

Progressive Sampling Map Update. Similar to previous work on adaptive sampling, we update the sampling map prediction every time we double the sampling budget. As future work, it would be interesting to investigate alternative update schedules.

7 CONCLUSION

We presented a neural approach for adaptive sampling aimed at high-quality Monte Carlo renderings. Our method minimizes the future denoising error by adaptively distributing a user-defined budget to image regions that will benefit the most. Compared to previous work that relies on heuristics based on the current image error (DEP, [Vogels et al. 2018]), our method can account for the impact of new samples on the future denoising error without requiring additional training data.

We eliminate the additional data requirements of state-of-the-art adaptive sampling methods [Hasselgren et al. 2020; Kuznetsov et al. 2018] by efficiently utilizing analytic distributions to produce rendered data during training. Our distributions are parameterized by per-pixel statistics gathered during the rendering of the ground-truth data. We demonstrate their effectiveness at generating plausible renderings for joint training of high-quality adaptive sampling and denoising networks. Moreover, we demonstrate that our

distributions converge to the ground-truth Monte Carlo distribution in high-sample-count scenarios, leading to a lower approximation error in such cases. We also propose the addition of a global summary module, which accumulates information from image regions outside of the receptive field of our networks during inference. In addition to the quality and data efficiency improvements over state-of-the-art approaches [Hasselgren et al. 2020; Kuznetsov et al. 2018], our method can be trained on the same data as neural denoisers. This enables the reuse of legacy training datasets originally made for denoisers, making it an attractive solution for production environments in which creating new extensive datasets can be prohibitive.

We believe that such analytic distributions are highly compact and efficient representations of pixel value distributions stemming from Monte Carlo rendering. We are excited to see how these can be used in the future to accelerate the training of various data-driven problems that rely on huge amounts of rendered data.

ACKNOWLEDGMENTS

We thank Xianyao Zhang for creating Figure 2 and for proofreading. We thank the scene authors, Alain Hostettler for Mountain and Armadillo, Maurizio Nitti and Doriano van Essen for ModernKitchen and Mushroom. Glossykitchen and Bookshelf are based on Evermotion assets and ported to Mitsuba by Tiziano Portenier. We thank Wig42 and Disaster for LivingRoom, Grégory Schwartz and bjobernis for the scene in Figure 11, jay-artist for the Kitchen scene in Figure 3 and aXel for the GlassOfWater scene. The Armadillo model is from the Stanford 3D scanning repository. We thank Mark Meyer, David Adler, André Mazzone, Per Christensen, Sam Cordingley and Shilin Zhu for their valuable feedback throughout this project.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/

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

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graphics (Proc. SIGGRAPH)* 36, 4, Article 97 (July 2017), 14 pages.

Kavita Bala, Bruce Walter, and Donald P Greenberg. 2003. Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 631–640.

Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor. 2015. General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proc. of Eurographics EG)* 34, 2 (May 2015), 597–608. Eurographics 2015 paper.

Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Adaptive Sampling for Geometry-aware Reconstruction Filters. In *Proc. Vision, Modeling and Visualization (VMV)*. Eurographics, 183–190.

Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A. Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35, 4 (2016), 107–117. https://doi.org/10.1111/cgf.12954 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12954

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X. Sillion. 2005. A Frequency Analysis of Light Transport. *ACM Trans. Graph.* 24, 3 (July 2005), 1115–1126. https://doi.org/10.1145/1073204.1073320

Oskar Elek, Manu M. Thomas, and Angus Forbes. 2019. Learning Patterns in Sample Distributions for Monte Carlo Variance Reduction. arXiv:1906.00124 [cs.GR]

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. 2018. Implicit Reparameterization Gradients. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/92c8c96e4c37100777c7190b76d28233-Paper.pdf

Baining Guo. 1998. Progressive radiance evaluation using directional coherence maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 255–266.

Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. 2008. Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–10. https://doi.org/10.1145/1360612.1360632

Jon Hasselgren, Jacob Munkberg, Marco Salvi, Anjul Patney, and Aaron Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. *Computer Graphics Forum* (2020). https://doi.org/10.1111/cgf.13919

Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. 2021. Interactive Monte Carlo Denoising using Affinity of Neural Features. *ACM Transactions on Graphics (TOG)* 40, 4, Article 37 (2021), 13 pages. https://doi.org/10.1145/3450626.3459793

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4, Article 122 (July 2015), 12 pages. https://doi.org/10.1145/2766977

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

David Kirk and James Arvo. 1991. Unbiased Sampling Techniques for Image Synthesis. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. Association for Computing Machinery, New York, NY, USA, 153–156. https://doi.org/10.1145/122718.122735

Alexandr Kuznetsov, Nima Khademi Kalantari, and Ravi Ramamoorthi. 2018. Deep Adaptive Sampling for Low Sample Count Rendering. *Computer Graphics Forum* 37 (07 2018), 35–44. https://doi.org/10.1111/cgf.13473

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012a. SURE-Based Optimization for Adaptive Sampling and Reconstruction. *ACM Trans. Graph.* 31, 6, Article 194 (Nov. 2012), 9 pages. https://doi.org/10.1145/2366145.2366213

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012b. SURE-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 31, 6, Article 194 (Nov. 2012), 9 pages.

Don P. Mitchell. 1987. Generating Antialiased Images at Low Sampling Densities. *SIGGRAPH Comput. Graph.* 21, 4 (Aug. 1987), 65–72. https://doi.org/10.1145/37402.37410

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *ACM Trans. Graph.* 33, 5, Article 170 (Sept. 2014), 14 pages. https://doi.org/10.1145/2641762

Bochang Moon, Jose A. Iglesias-Guitian, Sung-Eui Yoon, and Kenny Mitchell. 2015. Adaptive Rendering with Linear Predictions. *ACM Trans. Graph.* 34, 4, Article 121 (July 2015), 11 pages. https://doi.org/10.1145/2766992

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *ACM Trans. Graph.* 35, 4, Article 40 (July 2016), 10 pages. https://doi.org/10.1145/2897824.2925936

Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. 2009. Adaptive Wavelet Rendering. In *ACM SIGGRAPH Asia 2009 Papers* (Yokohama, Japan) *(SIGGRAPH Asia '09)*. Association for Computing Machinery, New York, NY, USA, Article 140, 12 pages. https://doi.org/10.1145/1661412.1618486

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive Sampling and Reconstruction Using Greedy Error Minimization. *ACM Trans. Graph.* 30, 6 (Dec. 2011), 1–12. https://doi.org/10.1145/2070781.2024193

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2012. Adaptive Rendering with Non-Local Means Filtering. *ACM Trans. Graph.* 31, 6, Article 195 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366214

Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. 2013. Robust Denoising using Feature and Color Information. *Computer Graphics Forum* 32 (10 2013). https://doi.org/10.1111/cgf.12219

Rasmus Tamstorf and Henrik Jensen. 1997. Adaptive Sampling and Bias Estimation in Path Tracing. *Eurographics Workshop on Rendering Techniques 97* (07 1997). https://doi.org/10.1007/978-3-7091-6858-5_26

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)* 37, 4, Article 124 (2018), 124:1–124:15 pages. https://doi.org/10.1145/3197517.3201388

Tiange Xiang, Hongliang Yuan, Haozhi Huang, and Yujin Shi. 2021. Two-Stage Monte Carlo Denoising with Adaptive Sampling and Kernel Pool. arXiv:2103.16115 [cs.CV]

Xianyao Zhang, Marco Manzi, Thijs Vogels, Henrik Dahlberg, Markus Gross, and Marios Papas. 2021. Deep Compositional Denoising for High-quality Monte Carlo Rendering. *Computer Graphics Forum* 40, 4 (2021), 1–13. https://doi.org/10.1111/cgf.14337 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14337

Xianyao Zhang, Melvin Ott, Marco Manzi, Markus Gross, and Marios Papas. 2022. Automatic Feature Selection for Denoising Volumetric Renderings. *Computer Graphics Forum* 41, 4 (2022), 63–77. https://doi.org/10.1111/cgf.14587 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14587

A APPENDIX

A.1 Kernel-based denoising

In our first analysis, we will assume that the denoiser follows the kernel-based convolutional network (KPCN) architecture [Bako et al. 2017; Vogels et al. 2018]. KPCNs are neural networks K_φ with trainable parameters φ that predict kernel weights for every pixel $p \in P$ over a neighborhood of pixels $q \in \mathcal{N}(p)$ from noisy rendering outputs $X = [I, F]$. A pixel $p \in P$ of the denoised image is then computed as $\mathcal{D}_\varphi(X_t)_p = \sum_{q\in\mathcal{N}(p)} K_\varphi(X_t)_{pq} I_q$. The set of feature channels F consists of specular, diffuse, albedo, and normals. The parameters φ are optimized by minimizing the following loss over a large training set $\mathcal{T}$ consisting of image pairs with noisy and ground-truth images (X_t, X_g):

$$\varphi^\star = \underset{\varphi}{\operatorname{argmin}} \sum_{(X_t,X_g)\in\mathcal{T}} \left[\mathcal{L}(\mathcal{D}_\varphi(X_t), X_g)\right], \tag{14}$$

where $\mathcal{L}$ is either SMAPE [Vogels et al. 2018] or logLoss defined in Equation (12).

A.2 Proof of Theorem 1

We start the analysis by computing a bias-variance decomposition of the loss $E_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathcal{L}(\mathcal{D}(Y), X_g)\right]$.

Lemma A.1. *Under the assumptions of Theorem 1, when the mean squared error (MSE) is used as the loss, we have*

$$\begin{aligned}\mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathrm{MSE}(\mathcal{D}(Y), X_g)\right] = &\underbrace{\sum_{p\in P}\sum_{q\in\mathcal{N}(p)} K_{pq}^2 \frac{\mathcal{S}_q}{\bar{\mathcal{S}}_{t,q}+\mathcal{S}_q}\mathbb{V}_q}_{\textit{Variance}} \\ &+ \underbrace{\sum_{p\in P}\left(\sum_{q\in\mathcal{N}(p)} K_{pq}\frac{\bar{\mathcal{S}}_{t,q}X_{t,q}+\mathcal{S}_q X_{g_q}}{\bar{\mathcal{S}}_{t,q}+\mathcal{S}_q} - X_{g_p}\right)^2}_{\textit{Bias}},\end{aligned} \tag{15}$$

where $\mathbb{V}_q$ is the 1-sample variance of a pixel q, and $q \in \mathcal{N}(p)$ represents the neighboring pixels within the denoising-kernel radius of p. All expectations are taken by averaging over new samples $Y \sim \mathcal{R}(Y|\mathcal{S}, \bar{\mathcal{S}}_t, X_t)$.

Proof. We start with substituting the kernel denoising equation in the expected loss

$$\begin{aligned}\mathbb{E}\left[\mathrm{MSE}(\mathcal{D}(Y), X_g)\right] &= \sum_p \mathbb{E}\left[\left(\sum_q K_{pq}Y_q - X_{g_p}\right)^2\right] \\ &= \sum_p \mathbb{E}\left[\left(\sum_q K_{pq}(Y_q - \mathbb{E}[Y_q]) + (\mathbb{E}[\sum_q K_{pq}Y_q] - X_{g_p})\right)^2\right] \\ &= \sum_p\sum_q K_{pq}^2\mathbb{E}\left[\left(Y_q - \mathbb{E}[Y_q]\right)^2\right] + \sum_p\left(\sum_q K_{pq}\mathbb{E}[Y_q] - X_{g_p}\right)^2 \\ &\quad + 2\sum_p \mathbb{E}\left[\sum_q K_{pq}(Y_q - \mathbb{E}[Y_q])\right]\cdot\left(\mathbb{E}[\sum_q K_{pq}Y_q] - X_{g_p}\right) \\ &= \sum_p\sum_q K_{pq}^2\mathbb{E}\left[\left(Y_q - \mathbb{E}[Y_q]\right)^2\right] + \sum_p\left(\sum_q K_{pq}\mathbb{E}[Y_q] - X_{g_p}\right)^2.\end{aligned} \tag{16}$$

In the last equality, we use the fact that $\mathbb{E}[Y_q - \mathbb{E}[Y_q]] = 0$.

The final rendered data Y is a mix of current data X_t and next rendered data $\bar{Y}_\mathcal{S}$

$$Y = \frac{\bar{\mathcal{S}}_t X_t + \mathcal{S}\bar{Y}_\mathcal{S}}{\bar{\mathcal{S}}_t + \mathcal{S}}. \tag{17}$$

The expected value of Y is

$$\mathbb{E}[Y] = \frac{\bar{\mathcal{S}}_t X_t + \mathcal{S}X_{g_p}}{\bar{\mathcal{S}}_t + \mathcal{S}}, \tag{18}$$

that is because expectation is conditioned on the current data X_t. Plugging Equation (18) in the second term of Equation (16) yields the bias term.

Substitute Equation (17) in $\mathbb{E}\left[\left(Y_q - \mathbb{E}[Y_q]\right)^2\right]$

$$\begin{aligned}\mathbb{E}\left[\left(Y_q - \mathbb{E}[Y_q]\right)^2\right] &= \frac{\mathcal{S}_q^2}{(\bar{\mathcal{S}}_{t,q}+\mathcal{S}_q)^2}\mathbb{E}[(Y_{\mathrm{next},q} - X_{g_q})^2] \\ &= \frac{\mathcal{S}_q}{(\bar{\mathcal{S}}_{t,q}+\mathcal{S}_q)^2}\mathbb{V}_q,\end{aligned} \tag{19}$$

where $\mathbb{E}[(Y_{\mathrm{next},q} - X_{g_q})^2]$ is the variance of average of $\mathcal{S}_q$ samples, that is $\mathbb{V}_q/\mathcal{S}_q$. Plugging Equation (19) in the first term of Equation (16) yields the variance term of Equation (15) and concludes the proof. □

Proof of Theorem 1. The proof of Theorem 1 follows directly from the bias-variance decomposition in Lemma A.1. Note that in Lemma A.1, we do not make any assumptions about the rendering distribution $\mathcal{R}$. Hence, any other distribution that satisfies the properties in Equation (5) results in the same bias-variance decomposition. Note that MRSE loss can be written as

$$\begin{aligned}&\mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathrm{MRSE}(\mathcal{D}(Y), X_g)\right] \\ &\qquad = \sum_p \frac{\mathbb{E}_{Y\sim\mathcal{R}(Y|\mathcal{S},\bar{\mathcal{S}}_t,X_t)}\left[\mathrm{MSE}(\mathcal{D}(Y)_p, X_{g_p})\right]}{X_{g_p}^2},\end{aligned} \tag{20}$$

from which the proof of Theorem 1 follows similar to the proof for MSE. □