Volume xx (200y), Number z, pp. 1–12

Path-based Monte Carlo Denoising Using a Three-Scale Neural Network

Weiheng Lin[1,2], Beibei Wang[1,2], Jian Yang[1,2], Lu Wang[3],Ling-Qi Yan[4]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology
[2]Key Lab of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education
[3] Shandong University
[4] University of California, Santa Barbara

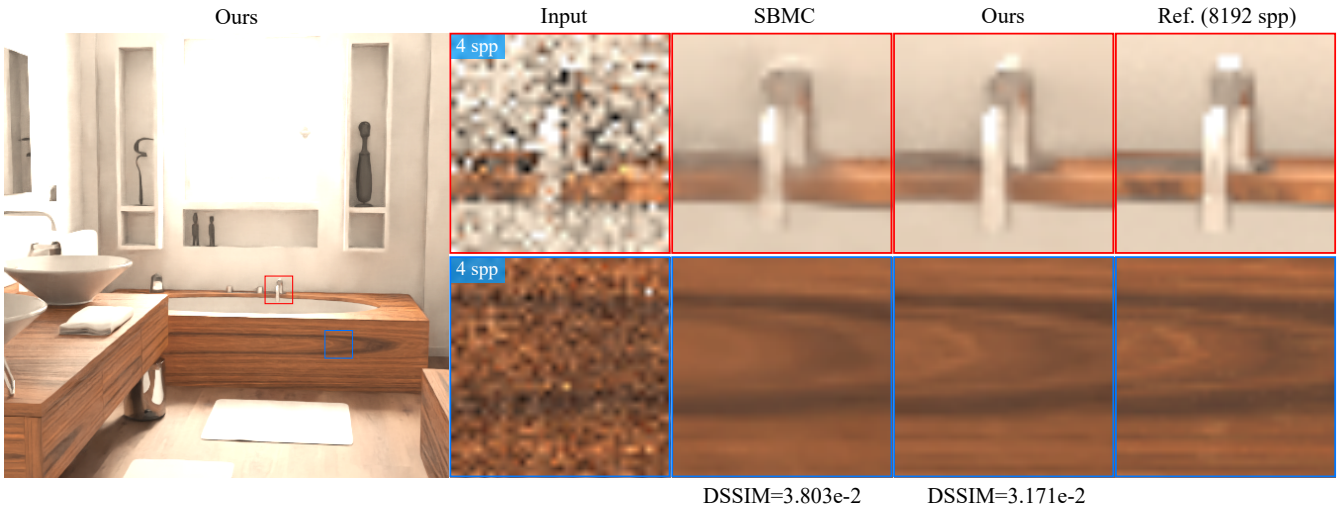

Figure 1: *Comparison between our network and sample-based Monte Carlo denoising network (SBMC) [GLA*19]. Both methods use the same dataset for training. Our method preserves details better, thanks to our novel network structure.*

Abstract

Monte Carlo rendering is widely used in the movie industry. Since it is costly to produce noise-free results directly, Monte Carlo denoising is often applied as a post-process. Recently, deep learning methods have been successfully leveraged in Monte Carlo denoising. They are able to produce high quality denoised results, even with very low sample rate, e.g. 4 spp (sample per pixel). However, for difficult scene configurations, some details could be blurred in the denoised results. In this paper, we aim at preserving more details from inputs rendered with low spp. We propose a novel denoising pipeline that handles three-scale features - pixel, sample and path - to preserve sharp details, uses an improved Res2Net feature extractor to reduce the network parameters and a smooth feature attention mechanism to remove low-frequency splotches. As a result, our method achieves higher denoising quality and preserves better details than the previous methods.

CCS Concepts

• ***Computing methodologies*** → *Neural network; Ray tracing;*

1. Introduction

Monte Carlo based rendering methods are widely used in the movie production, as they are physically based and produce realistic images. However, they usually have difficulties in generating noise-free results, especially when using very low sample rate (e.g. 4spp).

One solution is *Monte Carlo denoising*, which is performed as a post-process to remove the noise.

Deep learning based methods ([BVM*17], [VRM*18] and [GLA*19]) have been successfully exploited for Monte Carlo denoising. They are able to produce high-quality denoised results. Recently, Gharbi et al. [GLA*19] proposed a sample-based method, calculating the contribution of each sample to nearby pixels, instead of operating in the image space, which significantly improves the denoising quality, even when input noisy images have low sample rate. However, some details could not be preserved well, as seen in Figure 1.

In this paper, we design a novel deep neural network framework for Monte Carlo denoising. Specifically, we first introduce *path* features (e.g. lighting, probability density function for each bounce along the path) and group all the features into three scales: pixel, sample, and path. Then we design a novel neural network framework that uses a combined structure between Res2Net [GCZ19] and U-Net to extract features from three scales of buffers and fuse these features. The smooth features (like G-Buffer) are enhanced using extra connections. Finally, the network outputs the filter kernel for each path, and calculates the final denoised pixel radiance by a splatting operation. We also introduced the camera-path light transport covariance to better preserve high-frequency details. Our network is suitable for denoising low sample rate (e.g. 4 spp) rendered results, and significantly reduces error and enhances details as compared to the previous methods. To summarize, our contributions are:

- a three-scale neural network architecture : pixel, sample and path to enhance both geometric and lighting details;
- a hybrid feature extractor named Res2U-Net based on Res2Net and U-Net to improve the multi-scale feature extraction capability and reduce the number of network parameters;
- a smooth feature attention mechanism to reduce the low frequency splotches.

In the next section, we review some of the previous work on Monte Carlo denoising and deep neural networks. Then, we recap the theoretical basis of our method in Section 3. In Section 4, we present our method. We explain implementation details in Section 5. We present our results, compare with previous works and analyze performances in Section 6, and then conclude in Section 7.

2. Previous work

In this section, we first review the closest work to ours using machine learning, and then briefly go over general image space denoising methods.

2.1. Machine learning based Monte Carlo denoising

Kalantari et al. [KBS15] introduced a multilayer perceptual neural network to predict the parameters of a fixed-function filter and then used the filter with learned parameters to denoise Monte Carlo renderings. [CSS*17] et al. proposed a recurrent neural network (RNN) model to denoise under-sampled video renderings. Hasselgren et al. [HMS*20] also presented a deep neural network to ensure temporal stability in the context of interactive path tracing in which they co-train end-to-end over multiple consecutive frames.

Bako et al. [BVM*17] proposed a nine-layer convolutional neural network (CNN) model to predict the local weighting kernels for diffuse and specular components respectively. The network was further improved by Vogels et al. [VRM*18], combining with a number of task-specific modules, e.g. source-aware encoder, resulting in a more robust solution. Yang et al. [YWY*19] fused features with a fusion sub-network, fed the fused feature and the rendered radiance into a Dual-Encoder network and then reconstructed a clean image by a decoder network. Lin et al. [LWWH20] also extracted auxiliary features and radiance separately, and included light transport covariance from the light source to improve high-frequency lighting details. Wong et al. [WW19] proposed a deep residual network to directly map the noisy input pixels to the smoothed output. Xu et al. [XZW19] introduced a generative adversarial network (GAN) for better perceptual quality.

The prior methods all worked on pixels, which are insufficient to represent the complexity of local light distribution in low sample rate. Thus, Gharbi et al. [GLA*19] proposed a sample-based denoising method (SBMC), by splatting each sample onto nearby pixels to produce denoised results, since samples include more information. SBMC significantly improved the denoising quality, at low sample rate. However, some details were not well preserved. Our method is inspired by SBMC, but pushes further, using path information, to preserve more details. Munkberg and Hasselgren [MH20] proposed a layering embedding approach, separating samples into different layers, filtering them respectively and then compositing. Compared to their work, our method separates one sample into several paths and considers their feature individually, but we do not denoise the paths / layers separately. Also they aim at decreasing the computational and memory cost while preserving similar denoised quality, but our method aims at improving the denoising quality.

Deep learning has also been utilized for reconstruction in gradient domain rendering. Kettunen et al. [KHL19] proposed a dense variant of the U-Net and an additional perceptual loss (E-LPIPS) to improve the quality of denoised images. Guo et al. [GLL19] proposed an unsupervised deep neural network to reconstruct noisy images with the corresponding image gradients generated by gradient-domain renderers, which avoids the expensive renderings of ground truth images.

2.2. Image space Monte Carlo denoising

Image space based approaches have achieved impressive results at a reduced sampling rate [SZR*15]. They treated denoising as a regression problem, and used different regression models for filtering: zero-order linear regression model ([SD12], [RMZ13], [MJL*13], [ZRJ*15]), first-order or higher-order models ([MCY14], [BRM*16], [MMMG14]). The zero-order models have less flexibility, due to the limitations of their explicit filters. The first order methods have problem dealing with low frequency noise, and high-order methods might suffer from over-fitting.

Boughida et al. [BB17] proposed a non-local Bayesian collaborative filter, which produced globally high denoising quality, especially in dark areas.

These traditional methods often have a fixed set of parame-

ters, while our method dynamically selects the parameters, thus is content-aware, which is the advantage of using our path-based neural network.

2.3. Path space filtering

Path space filtering approaches [KDB16, DK18] could also reduce Monte Carlo rendering noises, however, they are different from denoising methods. These methods reduce noise during the rendering process in a progressive manner, while Monte Carlo denoising methods reduce noise as a postprocess.

3. Theoretical Background

Monte Carlo denoising aims at finding a reasonable filter Φ and corresponding parameters θ, given input data x from a rendering pipeline, to output a noise-free image $\hat{c}$:

$$\hat{c} = \Phi(x;\theta). \tag{1}$$

Where x has different meanings for different denoising approaches. For example, for pixel-based methods, it consists of pixels' radiance c and optional auxiliary feature buffers f. For sample-based method, it consists of the samples' radiance cs and other buffers f. Deep learning based Monte Carlo denoising methods utilize a neural network as the denoising filter Φ.

Most of the prior works [BVM*17] [VRM*18] are pixel-based, as they used the pixel-level features, and reconstructed the denoising result with the image pixels. Gharbi et al. [GLA*19] presented a sample-based solution, which splatted the samples to get the denoised radiance. We discuss the basic theory behind sample-based method in Section 3.1, and then present our path-based denoising in Section 3.2.

3.1. Sample-based Monte Carlo denoising

Gharbi et al. [GLA*19] proposed a sample-based method with a kernel-splatting network. They focused on samples instead of pixels and built a model to output a kernel indicating how much each sample contributes to nearby pixels. The reconstruction process of the method is expressed as the following formula:

$$L_{uv} = \frac{\sum_{x,y,s} K_{xyuvs} L_{xys}}{\sum_{x,y,s} K_{xyuvs}}, \tag{2}$$

where L_{uv} is the denoised result at pixel (u,v), L_{xys} is the noisy radiance of the s_{th} sample at pixel (x,y), K_{xyuvs} is the kernel that encodes the contribution from L_{xys} to I_{uv}.

We briefly review the network of SBMC [GLA*19]. Each input sample is a 74-d vector of features which contain sample coordinates, radiance, geometry, materials and lighting data. The network consists of a sample embedding module and a context propagation module. The sample embedding model extracts the samples' features with a simplified fully connected network. Then the context propagation module averages the sample features to per-pixel context features, feeds them into a U-Net model, and concatenates the U-Net outputs with sample features. After repeating the above process twice, the sample splatting is performed to output the kernel for each sample.

3.2. Path-based Monte Carlo denoising

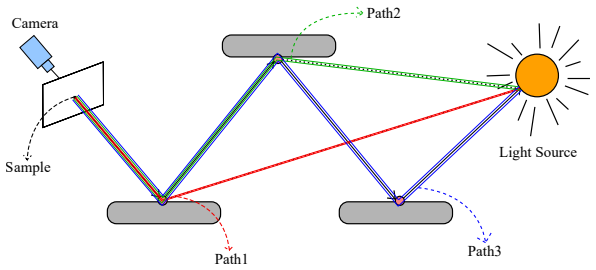

Figure 2: *From one sample, three different paths are generated under the NEE configuration.*

In this paper, our method uses path-level information. Each sample corresponds to one path when next event estimation (NEE) is not used in path tracing, and corresponds to P paths when using NEE, where P represents the number of bounces. In our paper, we assume that NEE is used in the rendering process. We aim at computing a set of weighted kernels through the neural network to represent the contribution of each path to the final pixel color. Similarly to Gharbi et al. [GLA*19], we also use the splatting operation.

The final denoised pixel color is obtained by a splatting operation for path noise radiance:

$$\hat{L} = \sum_{p}^{P} \frac{\sum_{n}^{N} K_{p,n} L_{p,n}}{\sum_{n}^{N} K_{p,n}} \tag{3}$$

where $\hat{L}$ is the denoised result, N is the sample count, $L_{p,n}$ is the noisy radiance of path p of sample n, $K_{p,n}$ is the kernel of path p from sample n.

Similar to the most previous works based on deep learning, we cast the Monte Carlo denoising problem as a supervised learning problem. We calculate the loss relationship between the network output and ground truth (renderings with high spp), and then use the learning algorithm to optimize the parameters of the neural network according to the error.

4. Path-based Monte Carlo denoising using a three-Scale neural network

We introduce a novel path-based Monte Carlo denoising neural network (Section 4.4), by separating different paths from each sample and splatting paths for denoised radiance. More specifically, we exploit three-scale features (Section 4.1) – pixel, sample and path, extract features from three-scale buffers separately with two hybrid feature extractors (Section 4.2 and Section 4.3).

4.1. Three-scale features

Inspired by the sample-based Monte Carlo denoising, we propose to exploit path related information in our network. Our method is positioned in the framework of path tracing with NEE, thus each sample involves several paths (see Figure 2). The key insight of the separation of the paths from a single sample is that the energy

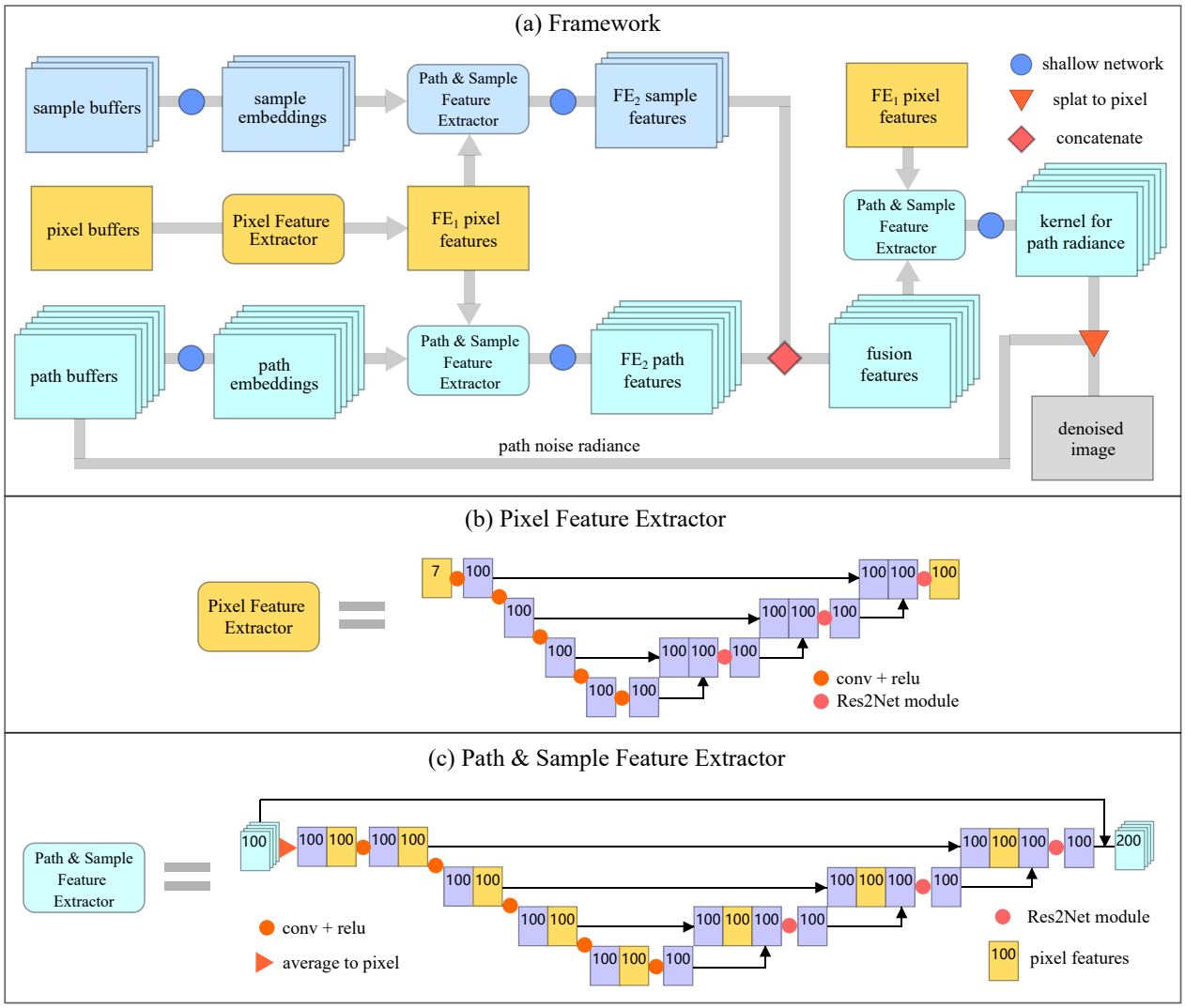

Figure 3: *(a) Our framework. Firstly, we encode the pixel buffers with the pixel feature extractor (FE_1), filter the path & sample buffers with a shallow 3-layer network (SN), and then fed them to the path & sample feature extractor (FE_2) respectively together with pixel features. Next, we filter the path & sample features from two FE_2 with SN and concatenate them to fusion features which are later processed by FE_2 and SN to output kernels for denoising path radiance images. Finally, we apply kernel-splatting operation to path radiance images to get the denoised result. (b) The architecture of pixel feature extractor (FE_1). (c) The architecture of path & sample feature extractor (FE_2). The number on the feature map indicates the channel count.*

distribution tends to decrease as the path length increases. Therefore, both the high frequency illumination from short paths and low frequency illumination from long paths could benefit from the separation. In our implementation, we denote each path using the last vertex before connecting to the light source.

In addition to the path information, we also keep both the sample and pixel information. Thus, the inputs can be grouped into three scales: pixels, samples and paths. Each group is stored in a buffer named correspondingly, pixel buffer B_{px}, sample buffer B_{sp} and path buffer B_{pt}.

- Pixel buffer consists of the G-Buffer data, e.g. normal and albedo.
- Sample buffer consists of the diffuse color, specular color, coordinates and the light transport covariance of each sample. The light transport covariance [BBS14] is evaluated from the pixel to the light source, to represent features' frequency.
- Path buffer consists of radiance for each path, the direction of incoming light from the light source, the material properties of the vertices. Similar to the sample buffer, we calculate the camera-path light transport covariance of each path to identify the high-frequency features.

The detailed content of each group is described in Section 5.

4.2. Feature extractor with Res2U-Net

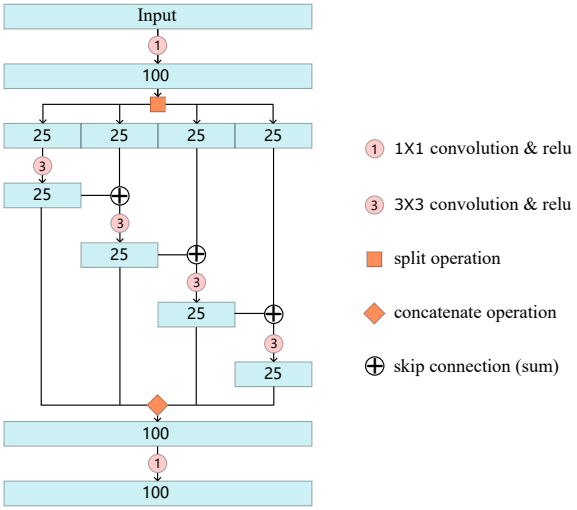

Figure 4: *The structure of Res2Net in our implementation.*

The above-mentioned buffers can be obtained during the rendering pipeline, so the next step is to extract information from these buffers. We designed two types of feature extractors. We propose a pixel feature extractor for pixel buffer, combining Res2Net [GCZ19] and U-Net (Figure 3(b)). For path buffer and sample buffer, we use a path & sample feature extractor with a feature attention mechanism (Figure 3(c)).

The pixel feature extractor (FE_1) network (Figure 3(b)) is a simplified U-Net. In each of the first five layers, convolution & Relu is applied to the previous layer's output, and in the other layers, a Res2Net [GCZ19] module is applied to the output, and skip connections are made between the symmetric layers of the FE_1 network. The output of each layer is calculated by:

$$f_{i+1}^{1} = \begin{cases} CR(f_i^1), 0 \leq i < 5 \\ R_2([f_i^1, f_{8-i}^1]), others. \end{cases} \quad (4)$$

Where f_i^1 is the feature of layer i in FE_1, [] means concatenating operation, CR is the convolution & Relu activation function and R_2 is a Res2Net module. In the Res2Net module (Figure 4), the input is first split into four small features. Then, convolution and skip connection are applied to the small features one by one, so each small feature has a different size of receptive field, and finally they are merged into a feature with the original size.

Res2Net reduces the parameters of network, and extracts multi-scale features, thus improves the training efficiency of the network and the quality of the results. U-Net has deep network layers, and it also has a high ability to reuse low-dimensional features, thanks to its symmetric skip connections. Combining Res2Net and U-Net can extract multi-scale information of both low-dimensional and high-dimensional features, which greatly improves the network's sensitivity to sharp features. In our method, this combination helps to better identify the feature information in various auxiliary feature buffers. And we name our combined module as Res2U-Net.

4.3. Feature extractor with Res2U-Net considering pixel attention

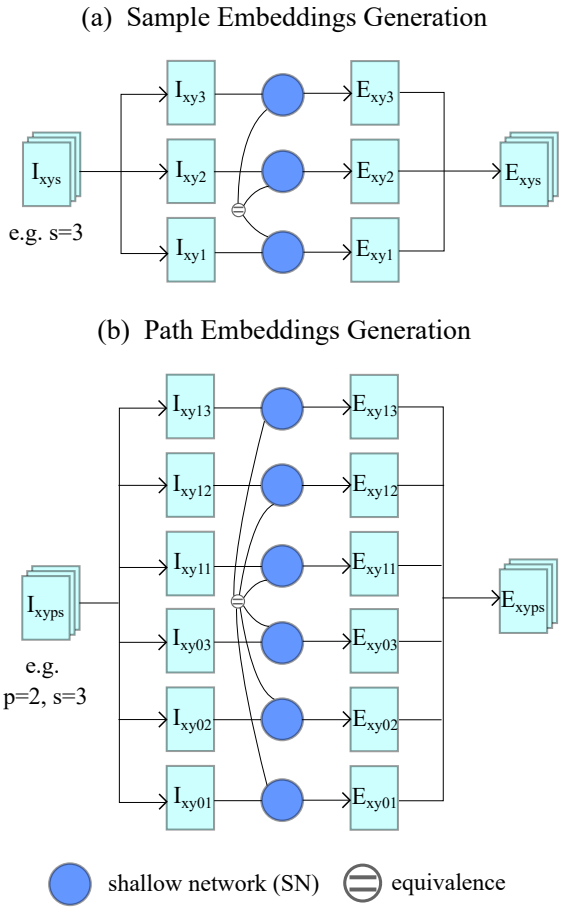

Figure 5: *A shallow network takes a sample buffer (a) or path buffer (b) into an embedding. Given a sample count or path count, the SN processes each buffer repeatedly, thus our model is able to handle arbitrary sample count or path count.*

We first encode the sample / path buffers to embeddings, similar to Gharbi et al. [GLA*19] to support arbitrary input sample count or path length, and then process the embeddings with our novel feature extractor. Then on top of the Res2U-Net, we further proposed a Res2U-Net with pixel attention as the path & sample feature extractor (FE_2). The insight behind it is that: the path and sample buffer contain high-frequency information, and processing them explicitly can effectively solve the over-blur; in contrast, the pixel buffers, like normal and albedo, are smooth compared to other

features (radiance or color). The previous method [GLA*19] suffers from low frequency splotches, which could be reduced by enhancing the impact of pixel features.

Similar to Gharbi et al. [GLA*19], we also use sample or path embeddings which are non-linear feature space for individual samples or paths. More specifically, we used a shallow network (SN, Section 4.4) with path buffers, sample buffers or embeddings as input (I_{xys} or I_{xyps}), and path or sample embeddings as output (E_{xys} or E_{xyps}), as shown in Figure 5, where x and y represent the width and height of the input, and s represents the number of samples and the p represents the path count. In Figure 3(a), we have five shallow networks: top left one with sample buffers as input, bottom left with path buffers as input, and the right three with embeddings as input.

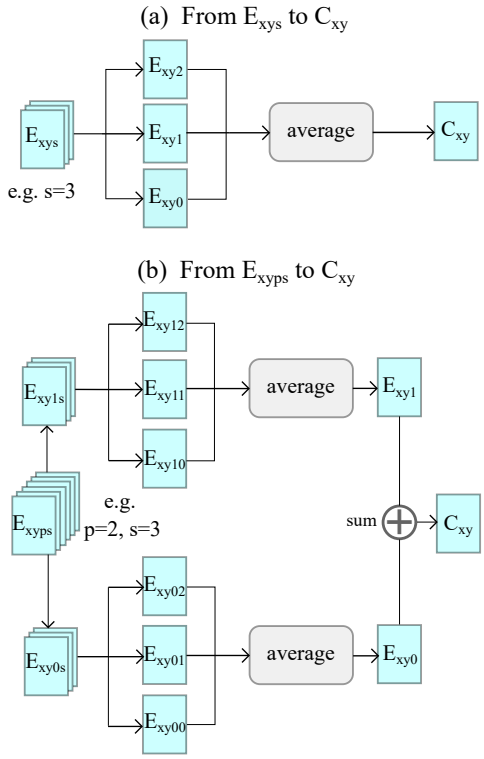

Figure 6: *The per-pixel context feature generation for sample embeddings (a) and path embeddings (b). The sample embeddings for one pixel are averaged into a pixel feature. The path embeddings with the same path length for one pixel are averaged and then summed into a pixel feature.*

After encoding the path and sample buffers to embeddings, the embeddings are fed to path & sample feature extractor (Figure 3(c)). In the feature extractor, we first turn path or sample embeddings into per-pixel context features. For sample embeddings E_{xys} (Figure 6(a)), we average them to the per-pixel context features C_{xy} across the sample axis s:

$$C_{xy} = reduce\_mean_s(E_{xys}) \tag{5}$$

For path embeddings (Figure 6(b)), we average the path embeddings E_{xyps} which belong to the same sample across the sample axis s and then take the sum across the path axis p to obtain the per-pixel context features C_{xy}:

$$C_{xy} = sum_p(reduce\_mean_s(E_{xyps})) \tag{6}$$

The per-pixel context features inform samples or paths about their neighborhood and represents the relevance of information between samples or paths. Then we feed per-pixel context features into a simplified U-Net with similar structure to FE_1. The difference to FE_1 is that, the inputs of each layer are concatenated with pixel features, to make full use of the smoothness and shape information in the pixel buffer:

$$f_{i+1}^2 = \begin{cases} CR([f_i^2, f_{px}]), 0 \leq i < 5 \\ R_2([f_i^2, f_{8-i}^2, f_{px}]), others. \end{cases} \tag{7}$$

Where f_i^2 is feature of layer i in FE_2. Finally the feature of final layer is concatenate with each input embedding at the end of path & sample feature extractor.

4.4. Network architecture

With our feature extractors (FE_1 and FE_2) defined, now we describe our full network architecture. As shown in Figure 3(a), the pixel, sample and path buffers are the inputs to the neural network, and then are processed separately. The pixel buffers are encoded with the pixel feature extractor (FE_1) to get the pixel features f_{px}:

$$f_{px} = FE_1(B_{px}). \tag{8}$$

Sample buffers and path buffers are first filtered with a shallow network (SN) respectively. The SN consists of three convolutional layers, where each layer contains 3×3 convolution, a constant bias and Relu activation function, and each layer outputs a feature map with 100 channels. The outputs of the SN are path or sample embeddings, which are the features obtained by the SN filtering each individual path or sample input. Then we average the path or sample embeddings to per-pixel context features and extract the path and sample features with the path & sample feature extractor:

$$f_{sp} = FE_{2sp}(SN(B_{sp}), f_{px}), \tag{9}$$

$$f_{pt} = FE_{2pt}(SN(B_{pt}), f_{px}), \tag{10}$$

where FE_{2sp} and FE_{2pt} are the feature extractors for sample buffer and path buffer, SN is the shallow network, f_{sp} is the output of FE_{2sp} , f_{pt} is the output of FE_{2pt}.

The output features f_{sp} and f_{pt} are filtered by SN again and concatenated to form the *fusion features*:

$$f_{fs}^{p,n} = [SN(f_{sp}^n), SN(f_{pt}^{p,n})] \tag{11}$$

where $f_{fs}^{p,n}$ is the fusion feature of path p from sample n, f_{sp}^n is the sample feature of sample n, $f_{pt}^{p,n}$ is the path feature of path p from sample n. The fusion features are sent to FE_2 and SN, then output kernels for each path noisy radiance image:

$$K_{p,n} = SN(FE_2(f_{fs}^{p,n})) \tag{12}$$

In the end, we get the denoised pixel result according to Equation 3.

5. Implementation details

5.1. Data creation

We use Tungsten renderer [Bit] to generate our training and validation dataset with a fixed path length of 5. We organize our three-scale buffers as follows:

- The pixel buffer contains G-buffers: normal (3 channels), albedo (3 channels) and depth (1 channel).
- The sample buffer contains diffuse radiance (3 channels), specular radiance (3 channels), visibility (1 channel), light-path light transport covariance (1 channel) and sample coordinates (the sub-pixel sample position and coordinates of the ray's intersection with the camera lens, 4 channels).
- We consider a connection from the camera to the light source as one path. In the context of path tracing with next event estimation, there are several paths for one sample. The path buffer contains radiance (3 channels), camera path light transport covariance per path (1 channel), conditional log-probabilities of sampling this light direction according to the BRDF and the direct light sampling algorithm (4 channels), light's direction in the camera's spherical coordinates (2 channels), and boolean features of the last vertex along the path (reflection, transmission, diffuse, glossy, specular, 5 channels).

As a preprocess, we scale the all depth and light transport covariance buffer to the range [0,1], and apply logarithm transform to all color buffers. We modify publicly available scenes [Bit16] (see Figure 7) by varying camera parameters, materials, and light sources. In the training dataset, the noisy images are rendered with 4 spp, and the reference images are rendered with 4096 spp. The resolution of these images is 1280×720. We rendered 483 images as our training set and 22 images as our validation set.

Although we use a fixed length 5, our network is able to handle either longer or shorter lengths. In both cases, thanks to our design, our model does not require retraining, while SBMC needs to be retrained, although more running time is required for longer path. If some paths do not reach the fixed path length, the value of the unused buffers has the initial value 0.

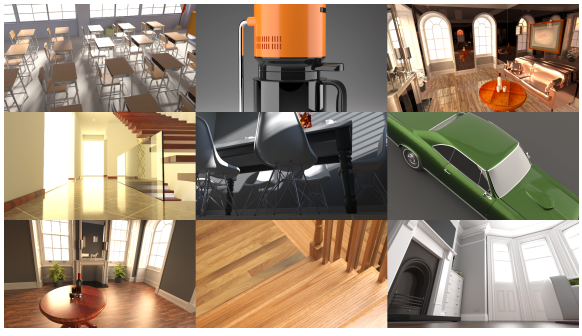

Figure 7: *Example images from our dataset.*

Table 1: *Error comparison between our method, SBMC and KPCN on the validation set with varying sample rate. All the models were trained with the same training dataset.*

		Ours	SBMC	KPCN
4spp	RMSE	**2.219e-2**	2.341e-2	2.752e-2
	DSSIM	**6.145e-2**	6.535e-2	7.065e-2
8spp	RMSE	**1.949e-2**	2.037e-2	2.259e-2
	DSSIM	**5.654e-2**	5.924e-2	6.259e-2
16spp	RMSE	**1.787e-2**	1.868e-2	1.962e-2
	DSSIM	**5.288e-2**	5.611e-2	5.905e-2
32spp	RMSE	**1.653e-2**	1.698e-2	1.761e-2
	DSSIM	**5.051e-2**	5.211e-2	5.310e-2
64spp	RMSE	**1.532e-2**	1.547e-2	1.598e-2
	DSSIM	**4.805e-2**	4.904e-2	4.923e-2
128spp	RMSE	**1.466e-2**	1.477e-2	1.474e-2
	DSSIM	4.658e-2	4.675e-2	**4.605e-2**

5.2. Training details

The loss function in our network is RelMSE, the same as Gharbi et al. [GLA*19]:

$$l = \text{RelMSE}(t(\hat{I}) - t(I_{gt})) \tag{13}$$

where $\hat{I}$ is the denoised result of our method, I_{gt} is the corresponding ground truth, and $t(x) = \frac{x}{1+x}$ is the tonemapping operator.

We split the processed data into 128×128 patches, then shuffle and feed them into the network. We use TensorFlow [AAB15] to implement our network and use ADAM [PB14] optimizer to optimize the parameters. Weights are initialized using the Xavier method [GB10]. Our network is trained for approximately 4.5 days on a RTX 2080Ti graphics card with learning rate 10^{-4} and batch size 1.

The memory consumption for both training and validating is about 11 GB. For validating, we stream the samples or paths between the GPU and the main RAM or disk to bound the memory usage as the same as Gharbi et al. [GLA*19]. The streaming operation applies to the per-sample or per-path processing steps.

6. Result

We use RMSE (relative mean squared error) and structural dissimilarity, DSSIM (1 - SSIM) as metrics to evaluate quality of the results. Same as in training, the input images are rendered with 4 spp, and the references are rendered with 4096 spp.

6.1. Comparison to previous work

We compare our method with KPCN [BVM*17], AD-VMCD [XZW19] and SBMC [GLA*19]. We implemented SBMC and KPCN with TensorFlow. The input data structure and training parameters of the two networks are set according to the

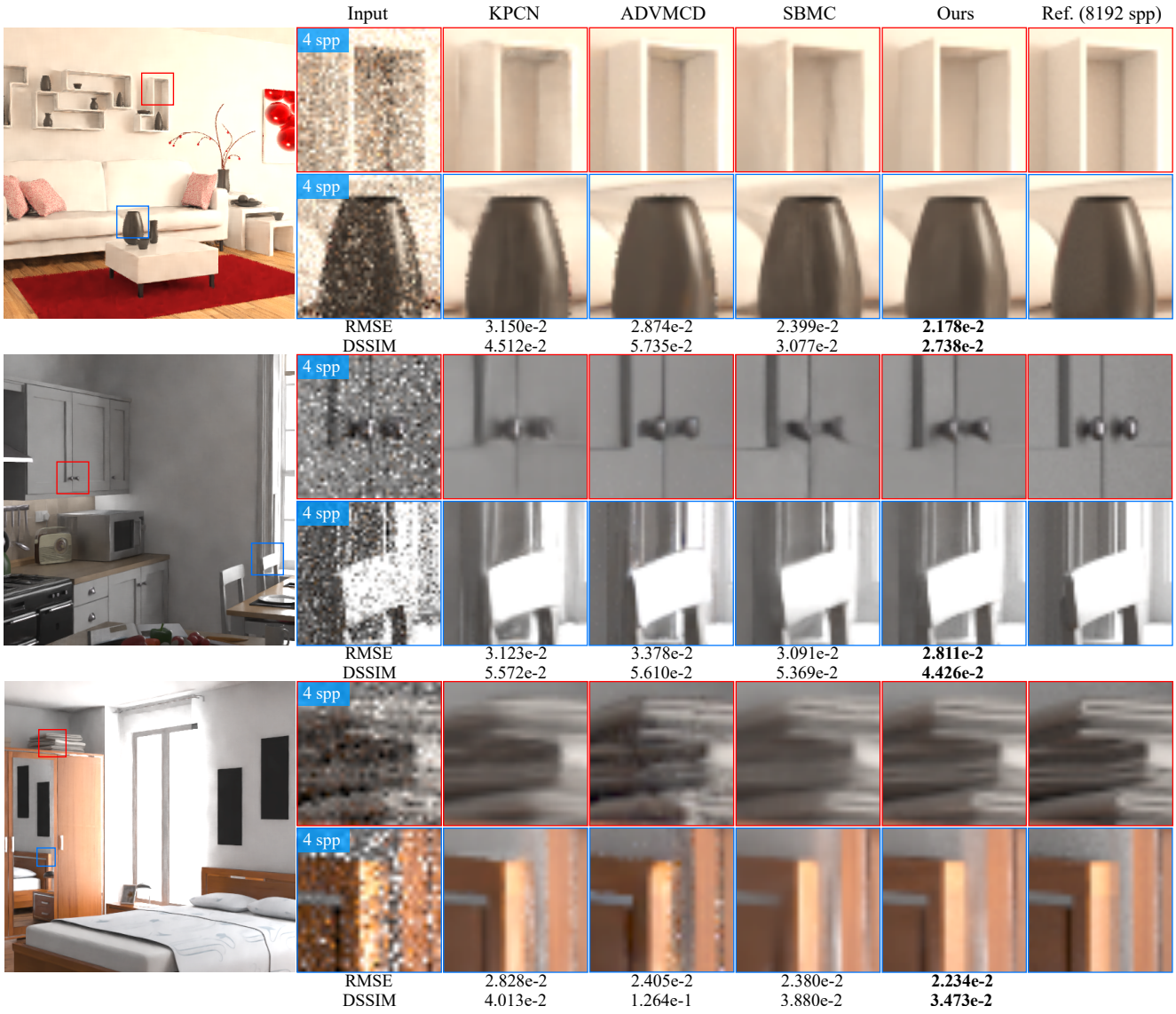

Figure 8: *Comparison of our method to other state-of-the-art methods KPCN, ADVMCD, SBMC and reference images.*

Table 2: *Error comparison between our method, SBMC and KPCN in two scenes with 4 spp.*

Scene		Ours	SBMC	KPCN
The Wooden Staircase	RMSE	**1.822e-2**	1.887e-2	2.406e-2
	DSSIM	**4.361e-2**	4.686e-2	6.250e-2
Modern Hall	RMSE	**2.156e-2**	2.491e-2	2.916e-2
	DSSIM	**7.279e-2**	8.228e-2	8.843e-2

descriptions in their papers. Both methods are trained with our training set. SBMC is trained for 4 days and KPCN was trained for 1.5 days. For ADVMCD, we use the provided trained model to test on our validation data.

As shown in Figure 1 and 8, our method produces the highest quality both perceptually and quantitatively, compared to the other three methods. KPCN and ADVMCD suffers from noise and aliases, e.g. the edge of the vase. SBMC produces higher quality than KPCN, but overblurs some geometric details (e.g. the bed frame in the bottom row) and specular highlights (e.g. the glossy handle in the middle row). In contrast, our method preserves both the geometric details and the highlights. Thanks to the feature extraction of the pixel buffer and the dense connection with subsequent modules, our network is more sensitive to geometric and texture details, thus it is able to preserve sharper geometric details and restore texture information better. Using the light transport covari-

Table 3: *The sampling rates for three methods (Ours, SBMC and KPCN) at equal time in Figure 9.*

Time (s)	20	40	60	80	100	120	140	160	180
Ours	4	8	12	16	20	24	28	32	36
SBMC	4	8	16	20	24	28	32	40	44
KPCN	4	12	20	28	34	42	50	57	64

ance to the sample and path buffer also helps to improve the quality of high-frequency effects.

We compare the quality quantitatively between our method, SBMC and KPCN in Table 1 over varying sampling rates on several scenes chosen from the validation dataset. For almost all the sampling rates, our method produces the highest quality. As the sampling rate increases, the difference between our method and others becomes smaller. When the sampling rate reaches 128 spp, the error with DSSIM of KPCN is the smallest. Therefore, our method is suitable for denoising renderings with very low sampling rates. In Table 2, we show the error of our method, SBMC and KPCN in two scenes from the validation dataset. The Staircase scene is almost diffuse materials dominant, and the Hall scene contains more high-frequency effects. In both cases, our method has the lowest error compared to the other two methods.

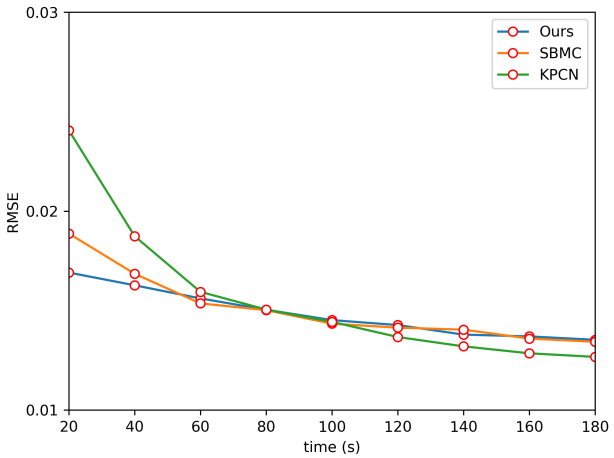

Figure 9: *Error (RMSE) curves of three methods (our method, SBMC and KPCN) over varying time budgets. The corresponding sampling rate of each method are shown in Table 3.*

Sample-based denoising methods have expensive computational cost, thus we compare our method with other methods (SBMC, KPCN) with the same time budget, which includes both the rendering and denoising time. In Figure 9, we show the error (RMSE) curves of three methods (our method, SBMC and KPCN) over varying time budgets. The error is the average of three scenes. To ensure equal time, different sampling rates are applied for different methods, shown in Table 3. With small time budget (e.g. 20s), our results have the highest quality. Thus, our method is suitable for low sampling rates. As the time budget increases, KPCN produces results with the highest quality, since its performance overhead is irrelevant to the sampling rate. We also compared with path tracing with equal time. As expected, the error of path tracing is much larger than others. We provide this comparison in the supplemental material.

6.2. Ablation study

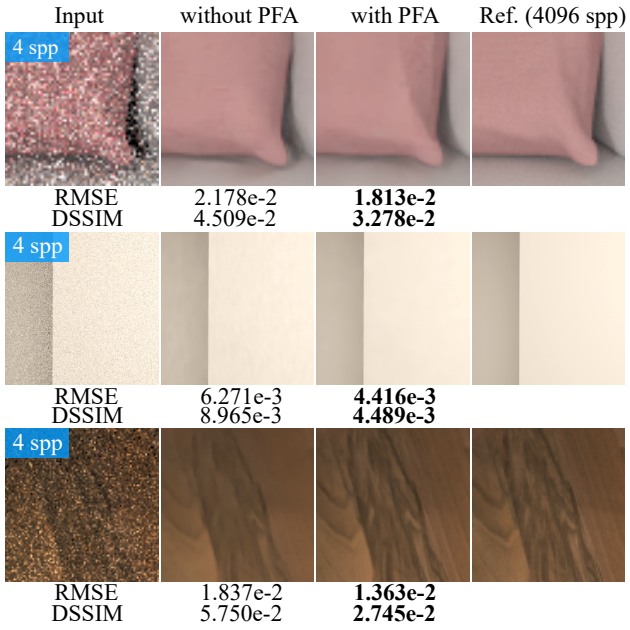

Figure 10: *Comparison of our method trained with and without pixel buffer attention mechanism.*

There are several important components in our network: pixel feature attention mechanism, Res2Net based feature extractor and light transport covariance. We validate the impacts of these components.

Pixel feature attention mechanism. In order to validate the influence of the pixel feature attention mechanism (PFA), we implement a solution without pixel feature attention, removing the pixel feature extractor and adding the g-buffer in the pixel buffer to the sample buffer. We compare the results with / without PFA in Figure 10. The results with PFA mechanism are better: in the first row, the result with PFA keeps the sharp geometric details; in the second row, the low-frequency area is smoother with PFA; the third row shows the improvement of the texture details, thanks to the attention of the albedo feature.

Res2Net module validation. We replace the Res2Net module with a traditional convolutional layer and retrained with the same dataset, and compare this simplified version with our full method in Figure 11. The details are preserved much better with the Res2Net. Res2Net decomposes the feature map into small feature maps for

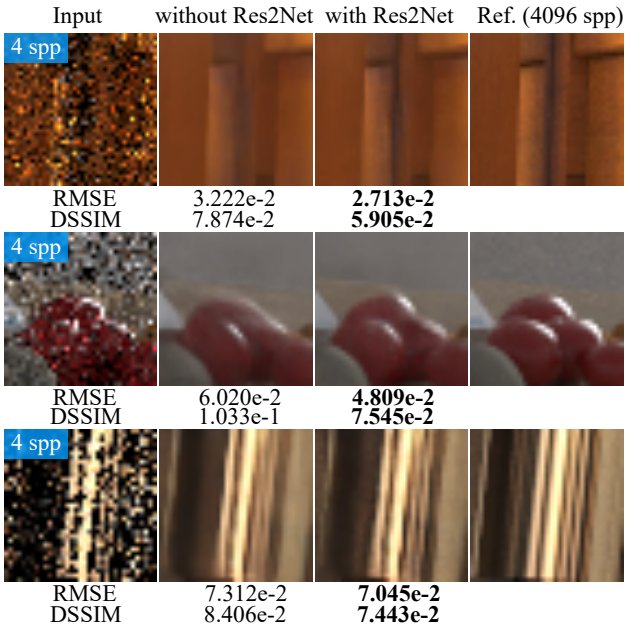

Figure 11: *Comparison of our method trained with and without Res2Net module.*

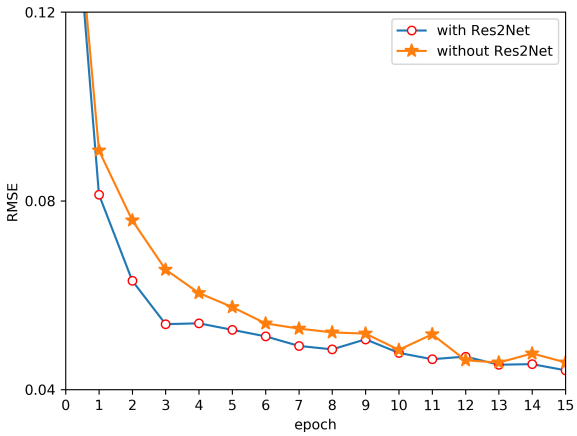

Figure 12: *RelMSE as a function of training iterations for our method with and without Res2Net.*

processing, and performs skip-connection one by one, therefore increases the receptive field. The Res2Net structure also allows multi-scale feature extraction, thereby enhancing the sensitivity to the details from various auxiliary features and improving the sharpness of the details of the results.

In Figure 12, we compare the RMSE of our network trained with and without the Res2Net module as a function of iterations. At the beginning of the training, the error reduction rate of the network trained with the Res2Net module is faster, and it reaches the quasi-convergent state faster. After converging, the difference between the two models becomes smaller. Thus, Res2Net speeds up the network training process.

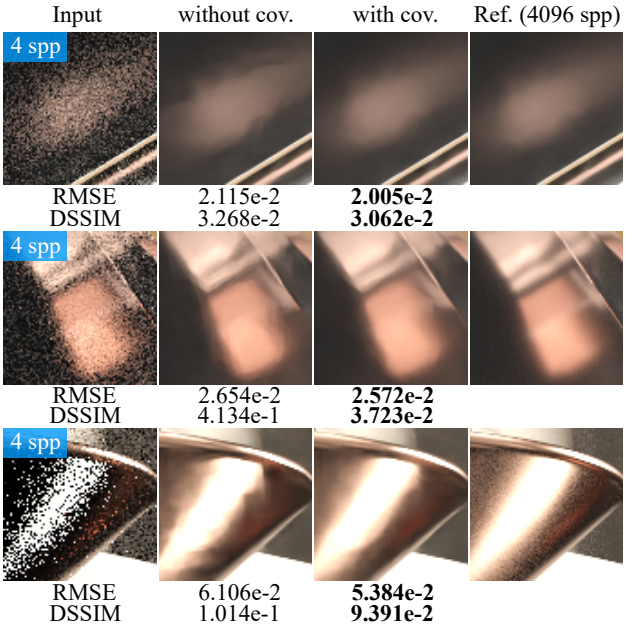

Figure 13: *Comparison of our method trained with and without light transport covariance.*

Light transport covariance validation. We validate the impact of the light transport covariance for scenes with complex lighting. In Figure 13, we compare denoised results with models trained with / without light transport covariance. The light transport covariance improves the highlight details significantly. Light transport covariance faithfully represents the frequency of the light transport, so the neural networks can learn more detailed features from high frequency illumination.

Sample buffer validation. In our network, we keep both the sample buffers and the path buffers. The sample buffers store common features for the paths, e.g. diffuse radiance, specular radiance and visibility. If we remove the sample buffers, these features have to be stored repeatedly in the path buffers, which is more redundant. Otherwise, if we simply drop all the features in the sample buffers, the denoising quality will degrade significantly. In Table 4, we compare the error between "with sample buffers" and "without sample buffers" over a variety of scenes. "With sample buffers" results in higher quality, as the sample buffers include important features.

6.3. Performances

We compare the runtime performance between our method and other methods in Table 5. Both our method and SBMC have an increasing cost, as the sampling rate increases. Compared with SBMC, our denoising cost is slightly higher than SBMC over all the sample rates, since our method has to process more images.

Table 4: *RMSE comparison between our method with and without sample buffers (sam. means sample buffer). All the input images in the table are rendered with 4 spp.*

Scene	With sam.	Without sam.	Improvement
bathroom	**1.511e-2**	1.636e-2	7.6%
bedroom	**1.998e-2**	2.096e-2	4.7%
kitchen	**2.396e-2**	2.480e-2	3.4%
classroom	**2.987e-2**	3.081e-2	3.1%

Table 5: *Runtime cost of different methods (in seconds) to denoise a 1280×720 image.*

spp	4	8	16	32	64	128
Rendering	10.9	21.3	42.3	83.9	168.9	332.5
Ours	12.1	21.7	38.1	72.4	133.9	254.5
SBMC	8.6	15.48	27.1	52.3	94.1	178.9
KPCN	N/A	N/A	N/A	N/A	N/A	11.2

6.4. Limitation

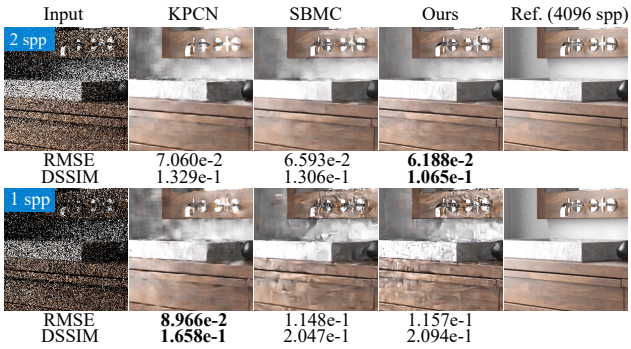

Figure 14: *Comparison between our method, KPCN, and SBMC on input images rendered with very low sampling rate (2 and 1 spp).*

We also applied our method at extremely low sampling rates, e.g. 1 spp and 2 spp, and compared our results with other methods (KPCN and SBMC) in Figure 14. With 2 samples per pixel, our method still produces the highest denoising quality and preserves the sharp details compared to other methods. However, at 1 spp, both our method and SBMC have obvious artifacts near the specular regions. Also, our method inherits the limitations of sample-based methods: missing scalability with sample count in terms of computation time (Table 5)

7. Conclusions

We have proposed a novel path space network architecture for Monte Carlo denoising. We designed a three-scale network framework considering three-scale features (pixel, sample and path), introduced a hybrid feature extractor based on Res2Net and U-Net, and further proposed a pixel feature attention mechanism to enhance the impact of smooth features. Our proposed method achieves higher denoising quality and preserves better details than the previous methods.

In the future, we will try to combine our method with unsupervised learning model to avoid the expensive ground-truth images rendering. It is also useful to solve the scalability issue with embedding layers, is also interesting to consider temporal denoising.

References

[AAB15] ABADI M., AGARWAL A., BARHAM P.: Tensorflow: Large scale machine learning on heterogeneous systems. http://tensorflow.org/, 2015. 7

[BB17] BOUGHIDA M., BOUBEKEUR T.: Bayesian collaborative denoising for Monte Carlo rendering. *Computer Graphics Forum (Proc. EGSR 2017) 36*, 4 (2017), 137–153. 2

[BBS14] BELCOUR L., BALA K., SOLER C.: A local frequency analysis of light scattering and absorption. *ACM Transactions on Graphics (TOG) 33*, 5 (2014), 163. 4

[Bit] BITTERLI B.: Tungsten renderer. http://noobody.org/tungsten.html. 7

[Bit16] BITTERLI B.: Rendering resources. https://benediktbitterli.me/resources/, 2016. 7

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., A.IGLESIAS-GUITIÁN J., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. *Computer Graphics Forum 35*, 4 (2016), 107–117. 2

[BVM*17] BAKO S., VOGELS T., MCWILLIAMS B., MEYER M., NOVÁK J., HARVILL A., SEN P., DEROSE T., ROUSSELLE F.: Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017) 36*, 4 (July 2017). 2, 3, 7

[CSS*17] CHAITANYA C. R., S.KAPLANYAN A., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZAHRA D., AILA T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph. 36*, 4 (July 2017), 98:1–98:12. 2

[DK18] DAHM K., KELLER A.: Learning light transport the reinforced way. In *Monte Carlo and Quasi-Monte Carlo Methods* (Cham, 2018), Owen A. B., Glynn P. W., (Eds.), Springer International Publishing, pp. 181–195. 3

[GB10] GLOROT X., BENGIO Y.: Understanding the difficulty of training deep feedforward neural networks. *In International conference on artificial intelligence and statistics* (2010), 249–256. 7

[GCZ19] GAO S H., CHENG M M., ZHAO K.: Res2net: A new multi-scale backbone architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019). 2, 5

[GLA*19] GHARBI M., LI T.-M., AITTALA M., LEHTINEN J., DURAND F.: Sample-based Monte Carlo denoising using a kernel-splatting network. *ACM Trans.Graph. 38*, 4 (2019), 125:1–125:12. 1, 2, 3, 5, 6, 7

[GLL19] GUO J., LI M., LI Q.: Gradnet: Unsupervised deep screened poisson reconstruction for gradient-domain rendering. *ACM Transactions on Graphics 38*, 6 (2019), 1–13. 2

[HMS*20] HASSELGREN J., MUNKBERG J., SALVI M., PATNEY A., LEFOHN A.: Neural temporal adaptive sampling and denoising. *Computer Graphics Forum 39*, 2 (2020), 147–155. 2

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015) 34*, 4 (2015). 2

[KDB16] KELLER A., DAHM K., BINDER N.: Path space filtering. In *Monte Carlo and Quasi-Monte Carlo Methods* (Cham, 2016), Cools R., Nuyens D., (Eds.), Springer International Publishing, pp. 423–436. 3

[KHL19] KETTUNEN M., HRKNEN E., LEHTINEN J.: Deep convolutional reconstruction for gradient-domain rendering. *ACM Transactions on Graphics 38*, 4 (2019), 1–12. 2

[LWWH20] LIN W., WANG B., WANG L., HOLZSCHUCH N.: A detail preserving neural network model for monte carlo denoising. *Computational Visual Media Journal* (2020). 2

[MCY14] MOON B., CARR N., YOON S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph 33*, 5 (2014), 170:1–170:14. 2

[MH20] MUNKBERG J., HASSELGREN J.: Neural denoising with layer embeddings. *Computer Graphics Forum 39*, 4 (2020), 1–12. 2

[MJL*13] MOON B., JUN J. Y., LEE J., KIM K., HACHISUKA T., YOON S.-E.: Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Computer Graphics Forum 32*, 1 (2013), 139–151. 2

[MMMG14] MOON B., MCDONAGH S., MITCHELL K., GROSS M.: Adaptive polynomial rendering. *ACM Trans. Graph* (2014), 10. 2

[PB14] P.KINGMA D., BA J.: Adam:a method for stochastic optimization. http://arxiv.org/abs/1412.6980, 2014. 7

[RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7 (2013), 121–130. 2

[SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactionson Graphics 31*, 3 (2012), 15. 2

[SZR*15] SEN P., ZWICKER M., ROUSSELLE F., YOON S.-E., KALANTARI N.: Denoising your Monte Carlo renders: Recent advances in image-space adaptive sampling and reconstruction. *ACM SIGGRAPH 2015 Courses* (2015). 2

[VRM*18] VOGELS T., ROUSSELLE F., MCWILLIAMS B., RÖTHLIN G., HARVILL A., ADLER D., MEYER M., NOVÁK J.: Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018) 37*, 4 (2018), 124:1–124:15. 2, 3

[WW19] WONG K.-M., WONG T.-T.: Deep residual learning for denoising monte carlo renderings. *Computational Visual Media 5*, 3 (2019), 239–255. 2

[XZW19] XU B., ZHANG J., WANG R.: Adversarial monte carlo denoising with conditioned auxiliary feature modulation. *ACM Transactions on Graphics 38*, 6 (2019), 1–12. 2, 7

[YWY*19] YANG X., WANG D., YIN B., WEI X., HU W., ZHAO L., ZHANG Q., FU H.: Demc: A deep dual-encoder network for denoising monte carlo rendering. *Journal of Computer Science and Technology 34*, 5 (2019), 1123–1135. 2

[ZRJ*15] ZIMMER H., ROUSSELLE F., JAKOB W., WANG O., ADLER D., JAROSZ W., SORKINE-HORNUNG O., SORKINE-HORNUNG A.: Path-space motion estimationand decomposition for robust animation filtering. *Computer Graphics Forum 34*, 4 (2015), 131–142. 2