DOI: 10.1111/cgf.14338
Eurographics Symposium on Rendering 2021
A. Bousseau and M. McGuire
(Guest Editors)

Volume 40 (2021), Number 4

Real-time Monte Carlo Denoising with Weight Sharing Kernel Prediction Network

Hangming Fan, Rui Wang†, Yuchi Huo†, Hujun Bao

State Key Lab of CAD&CG, Zhejiang University
†Corresponding author: rwang@cad.zju.edu.cn, huo.yuchi.sc@gmail.com

Abstract

Real-time Monte Carlo denoising aims at removing severe noise under low samples per pixel (spp) in a strict time budget. Recently, kernel-prediction methods use a neural network to predict each pixel's filtering kernel and have shown a great potential to remove Monte Carlo noise. However, the heavy computation overhead blocks these methods from real-time applications. This paper expands the kernel-prediction method and proposes a novel approach to denoise very low spp (e.g., 1-spp) Monte Carlo path traced images at real-time frame rates. Instead of using the neural network to directly predict the kernel map, i.e., the complete weights of each per-pixel filtering kernel, we predict an encoding of the kernel map, followed by a high-efficiency decoder with unfolding operations for a high-quality reconstruction of the filtering kernels . The kernel map encoding yields a compact single-channel representation of the kernel map, which can significantly reduce the kernel-prediction network's throughput. In addition, we adopt a scalable kernel fusion module to improve denoising quality. The proposed approach preserves kernel prediction methods' denoising quality while roughly halving its denoising time for 1-spp noisy inputs. In addition, compared with the recent neural bilateral grid-based real-time denoiser, our approach benefits from the high parallelism of kernel-based reconstruction and produces better denoising results at equal time.

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

1. Introduction

Monte Carlo path tracing is a technique used to synthesize realistic images, and it has the attribute of using a unified rendering pipeline to simulate various physically-based visual effects, which is appealing to both offline and real-time applications. But the use of Monte Carlo integration requires thousands of samples per pixel to produce a visually noise-free path traced image, which leads to significant computational cost. Given the computation budget of desktop GPU in the near future, only one sample per pixel (spp) is allowed in real-time path tracing application [CKS*17]. Therefore, using image-based denoising techniques to remove Monte Carlo noise would be a practical solution for production-ready applications.

Typically the low-spp Monte Carlo path traced image provides limited lighting information for denoising, and it is challenging to reconstruct unbiased results. Most denoising techniques introduce bias to reduce the variance, such as gathering the information from nearby pixels to reconstruct the center pixel color. Besides, the noise-free auxiliary feature buffers such as shading normals and texture albedo colors can provide geometric and material information about the scene, which can be used as inputs to guide the denoising algorithm. These buffers can be obtained in the path tracing phase with a negligible time cost.

Given the success of convolutional neural network (CNN) in solving computer vision and graphics tasks, lots of recent researchers focus on using a CNN to build their denoising architecture. Bako et al. [BVM*17] proposed to use a neural network to predict per-pixel filtering kernels instead of using hand-crafted kernels. Vogels et al. [VRM*18] then extended this architecture to predict small kernels in hierarchical resolutions for saving computations. This multi-resolution denoising architecture was also used in the layer-based denoiser [MH20] and the temporal adaptive sampling method [HMS*20]. These methods have shown the capability of the kernel prediction method in Monte Carlo denoising. However, they mostly target offline applications and rarely achieve real-time frame rates. The high computational cost of predicting large filtering kernels makes applying such techniques to real-time infeasible.

In this paper, we propose a novel approach to reduce the kernel prediction methods' overhead and achieve real-time Monte Carlo denoising under low spp. The core idea of our approach is to reduce the throughput of neural networks by introducing a kernel

construction module. Instead of directly predicting per-pixel filtering weights, we treat the neural network as an encoder to output a compact single-channel format of the filtering weights, denoted as importance maps. Then, we feed the importance maps to a hand-crafted decoder to construct the complete kernel map. In practice, we utilize an unfolding operation and a normalization step to build up the filtering kernel construction module and use the end-to-end training manner to make the network cooperate with this hand-crafted decoder. As a final step, we use the constructed filtering kernels to filter the noisy image, which can be efficiently implemented as screen-space post-processing. Our approach benefits from the high parallelism of kernel-based reconstruction and is easy to integrate into existing Monte Carlo denoising modules.

The decoding architecture we designed has the ability to construct specific-sized filtering kernels . We further improve the capacity of the decoder with a kernel fusion module. Specifically, the improved decoder constructs filtering kernels from a set of importance maps, then independently applies the kernels to the noisy input and combined the denoised results with learned weights. Practically we construct filtering kernels of different sizes to adapt to various noise frequencies. In addition, we add a temporal accumulation operation before the denoising phase just the same as [KIM*19] does. The increasing effective spp counts of the input image can help stabilize the denoising procedure for consecutive frames and suppress temporal artifacts.

Comprehensive experiment results show that the proposed approach is good at denoising 1-spp frames at a real-time frame rate. Our method can preserve kernel prediction methods' denoising quality while roughly halving its denoising time. In addition, compared with the recent neural bilateral grid-based real-time denoiser [MZV*20], our approach benefits from the high parallelism of kernel-based reconstruction and produces better denoising results at equal time. To summarize, our approach makes the following contributions:

- A novel kernel prediction architecture that uses a compact representation of the kernel map and a scalable decoder for the filtering kernel reconstruction, which significantly reduces neural network throughput and memory requirement.
- Achieving comparable rendering quality using only roughly half of denoising time compared with the state-of-the-art real-time kernel prediction Monte Carlo denoising method.

2. Related Work

Monte Carlo denoising is a long-standing computer graphics research topic in both industrial productions and academic studies, and we refer to the survey from Zwicker et al. [ZJL*15] for a complete overview. In this section, we will mainly discuss deep learning and real-time denoising techniques, which are most related to our method. For a comprehensive study of deep learning-based Monte Carlo denoising techniques, we refer to the recent survey of Huo et al. [HY21].

Image-space Denoising. The filtering-based methods are based on using the auxiliary feature buffers to guide the construction of image-space filters [RW94, McC99, SD12]. Li et al. [LWC12] proposed to use the measurements of SURE metric [Ste81] to construct the cross-bilateral filter. Besides, the non-local means filter [BCM05, KS13] and a joint filtering scheme [RKZ12, ZRJ*15] had also been proposed.

Deep Learning-Based Monte Carlo Denoising. Kalantari et al. [KBS15] first introduced the supervised learning method for Monte Carlo denoising. They used a multi-layer perceptron (MLP) to estimate the parameters of a fixed-function filter (e.g., cross-bilateral filter). They also computed a rich set of secondary features as inputs to improve the quality. Recent state-of-the-art denoising techniques have leveraged the convolutional neural networks (CNN) and noise-free auxiliary feature buffers to reconstruct Monte Carlo renderings. Bako et al. [BVM*17] proposed the kernel prediction method, which utilizes a CNN to predict the weights of per-pixel filtering kernel, and this allows the filtering kernels to be more complex and robust. However, predicting large filtering kernels is time-consuming and memory-exhausting, while using a small-size kernel leads to a quality decrease. Given that filtering with a small-size kernel in down-scaled resolution corresponds to filtering in a large receptive field of the original resolution, Vogels et al. [VRM*18] proposed a multi-resolution filtering architecture to approximate the behavior of a large kernel. They also decomposed the denoising pipeline with task-specific modules, independently extracting the source-aware and spatio-temporal information. Dahlberg et al. [DAN19] deployed this modular hierarchical kernel prediction method to the existing commercial Monte Carlo path tracing engines and showed its practical ability and flexibility in denoising the feature film productions. However, the original kernel prediction method is designed mainly for offline applications with 16-64 spp and runs in seconds. Hasselgren et al. [HMS*20] and Munkberg et al. [MH20] used the hierarchical kernel prediction architecture to denoise the re-sampled Monte Carlo images and the samples-splatted layers, respectively, and they achieved an interactive speed. Besides, Thomas et al. [TVLF20] also utilized the hierarchical architecture with a feature extraction network, which is resilient to quantization errors, to explore the feasibility of a heavily quantized network for image reconstruction. Unlike them directly using the kernel prediction architecture, our approach extends it to real-time denoising with 1-spp input by operating on the encoding of the kernel map to reduce neural network inference overhead.

Besides, Xu et al. [XZW*19] proposed to use an adversarial learning approach and emphasize the guidance of feature buffers with a novel conditioned auxiliary feature modulation method. Huo et al. [HWZ*20] denoised incident radiance fields to guide unbiased path tracing. Kettunen et al. [KHL19] utilized the CNN to replace the screened Poisson solver for gradient-domain path tracing and notably improved the image quality with the help of gradient samples. Gharbi et al. [GLA*19] proposed a kernel-splatting architecture to estimate the contribution of each raw Monte Carlo sample. This sample splatting method is more natural and robust than the pixel gathering manner in denoising some specific visual effects, but the drawback is that it has a linear complexity of performance and memory with the sample count. Munkberg et al. [MH20] extended this method by splatting samples to several layers, thus converting the sample-based computation to layer-based. They take only a fraction of computational cost and memory requirements than the previous per-sample method and preserve a similar quality.

Real-time Denoising. Recently, many real-time Monte Carlo reconstruction methods have been proposed. They generally take 1-spp noisy data as input and denoise in a tough time budget. Mara et al. [MMBJ17] used approximations of the bilateral filter, and Schied et al. [SKW*17] used a hierarchical filter expanded with a customized edge-stopping function to filter the temporally accumulated frames progressively. Schied et al. [SPD18] further estimated temporal gradient to guide the temporal accumulation factors. Koskela et al. [KIM*19] applied augmented QR factorization and stochastic regularization to image blocks to do a block-wise feature regression, and their method runs extremely fast under GPU implementation. All of the above methods rely on accumulating projected frames [YNS*09] to obtain a greater effective spp, which inspires us to incorporate temporal accumulation as our data preprocessing operation.

Chaitanya et al. [CKS*17] proposed to use a U-Net [RFB15] convolutional neural network architecture to predict the pixel colors directly. Hasselgren et al. [HMS*20] proposed a neural adaptive sampling and denoising architecture with a spatio-temporal joint optimization, where the temporal optimization helps the sample predictor to learn spatio-temporal sampling strategies. As a result, they can achieve real-time rates with optimized network architecture. Compare with their adaptive sampling framework, our method focus on improving kernel-predicting modules' denoising performance and yields faster speed in low-spp (e.g., 1-spp) configurations. Besides, Hofmann et al. [HHCM21] also utilized the neural temporal adaptive sampling architecture with a novel sparse voxel tree data structure to render participating media. Meng et al. [MZV*20] integrated a differentiable bilateral grid to CNN architecture to get a neural bilateral grid denoiser. They gather adjacent pixels to a 3D bilateral space with a learned mapping function and then denoise in this 3D space at real-time frame rates. However, the bilateral grid-based reconstruction suffers heavy computational overhead compared with our kernel-based reconstruction approach, as demonstrated in Figure 6.

3. Problem Statement

Our goal is to reconstruct noise-free images from 1-spp path traced images in real-time frame rates, and we achieve this with a supervised learning method. We first generate a set of Monte Carlo path tracing data $\mathcal{S} = ((r_1, \mathbf{f}_1, t_1), ..., (r_N, \mathbf{f}_N, t_N))$ where r stands for noisy image rendered with low spp, $\mathbf{f}$ is the noise-free auxiliary feature (e.g. albedo, normal, depth) obtained in the rendering process, t is the reference image with high spp. We use a denoising function $\mathcal{D}$ with parameters Θ to reconstruct the denoised image. We notate the loss function $\mathcal{L}$ measuring the difference between the denoised image and its reference image, and then minimize the loss function with gradient descent algorithm across the dataset $\mathcal{S}$ with N samples to get the optimal parameters Θ_{opt}:

$$\Theta_{\text{opt}} = \arg\min_{\Theta} \sum_{i=1}^{N} \mathcal{L}(\mathcal{D}_{\Theta}(r_i, \mathbf{f}_i), t_i). \tag{1}$$

In this paper, the denoising function $\mathcal{D}$ stands for our weight sharing kernel prediction approach, and the trainable parameters Θ refer to the convolutional neural network weights.

Our approach extends kernel prediction convolutional network (KPCN) [BVM*17] with a kernel map decoding stage. Directly predicting per-pixel large-size filtering kernel weights spends too much time in network inference for the real-time application. A solution is to prune the hierarchy kernel prediction architecture [VRM*18] for real-time application. However, the solution suffers serious quality decrease. We observe that the total denoising time of the original kernel prediction network is hard to shrink, mostly because of the high throughput of the last convolutional layer. For example, there are 169 convolutional kernels in the output layer for predicting 13×13 filtering kernels , which is too heavy for a real-time architecture. Given this, we propose to predict the encoded compact representation of the kernel map and then use an efficient and scalable decoder to reconstruct the complete kernel map. In this manner, we can avoid predicting a heavy $k \times k$ channels kernel map for a filtering window size k. Instead, we only need to predict a single-channel kernel map encoding, which dramatically reduces the network throughput. With an efficient and scalable decoder, we can use a lightweight architecture of neural network encoder to make our denoising pipeline run at real-time speed while preserving the denoising quality.

Specifically, our neural network $\mathcal{C}$ takes noisy image r and auxiliary feature $\mathbf{f}$ as input, and then estimates a single scalar for each pixel p, which we denote as importance term and the whole single-channel image is denoted as importance map:

$$I(p) = \mathcal{C}_{\Theta}(r, \mathbf{f})(p). \tag{2}$$

This 2D single-channel importance map is the predicted encoding of the kernel map. We then use it to construct the filtering kernel with a hand-crafted decoder. Letting Ω_p denote the $k \times k$ neighborhood centered around pixel p, which is also the filtering window for p, we compute the normalized filtering kernel weight $w_p(q)$, where $q \in \Omega(p)$, in a splatting manner:

$$w_p(q) = \frac{\exp(I(q))}{\sum_{q' \in \Omega_p} \exp(I(q'))}. \tag{3}$$

The softmax normalization used here is a common practice [BVM*17, HMS*20] to enforce that $0 \leq w_p(q) \leq 1$ and it contributes to a more steady training process. Then, we apply the same weights to each RGB color channel and compute the denoised pixel color as

$$R(p) = \sum_{q \in \Omega_p} w_p(q) r(q). \tag{4}$$

The filtering kernel size k is pre-defined along with the other hyperparameters (e.g., CNN layer count, convolutional kernel size), and this means the importance term we estimated is a relative value for a given filtering window size. Inspired by Vogels et al. [VRM*18] who used same sized filtering kernels in different resolutions to improve the denoising quality, we design a kernel fusion module by filtering with different sized kernels in a same resolution, which has also improved our decoder's scalability. Precisely, we predict M importance maps to construct filtering kernels with different filtering window sizes k_i. Then we independently filter the noisy input and average the results R^{k_i} with the learned weights,

$\alpha_i(p)$, as:

$$\hat{R}(p) = \sum_{i=1}^{M} \alpha_i(p) R^{k_i}(p), \tag{5}$$

where $0 \leq \alpha_i(p) \leq 1$ and $\sum_{i=1}^{M} \alpha_i = 1$. This kernel fusion module helps our denoiser distinguish the frequency of noise region and essentially improves the scalability of our decoder architecture.

4. Weight Sharing Kernel Prediction Architecture

In this section, we describe the overall architecture of our weight sharing kernel prediction neural denoiser. As illustrated in Figure 1, our architecture consists of the prediction phase and the reconstruction phase. At the prediction phase (Section 4.1), the input data are sent to an *ImportanceNet* to predict the importance map, i.e., a compact encoding of the kernel map. At the reconstruction phase (Section 4.2), we construct the filtering kernels from the importance map to filter the noisy input. Besides, we design a kernel fusion architecture that constructs filtering kernels with different window sizes to filter the input independently, and then fuses the results with a weighted average.

4.1. Prediction Phase

At the prediction phase, we use a CNN, denoted as *ImportanceNet*, to predict the encoding of the kernel map. Our *ImportanceNet* takes the 1-spp path traced image and its auxiliary feature buffers as inputs. It outputs a single-channel importance map, which is used for constructing the filtering kernel map at the reconstruction phase. We build our *ImportanceNet* with a variant of *RepVGG Block* structure [DZM*21]. Experimentally we use two network architectures to denoise 1-spp input: the first one employs six *RepVGG Blocks* for a better quality (Figure 2, left) and another employs three for a faster execution speed (Figure 2, right). They all run at real-time speed.

The *RepVGG Block* structure [DZM*21] can improve the capacity of a fully convolutional neural network during training and will not introduce additional time cost to network inference, and this is implemented with a parameter conversion. One *RepVGG Block* has multiple branches at training-time: 1×1, 3×3, 5×5 convolution layer (Conv) branch, and identity branch. These branches are combined with an element-wise addition and sent to a ReLU layer. After training, the multi-branches structure is equivalently transformed to a single 5×5 Conv layer for inference. Please see the supplementary document for the details of this structure. The converted network architecture is an efficient single-branch fully convolution network and does the same computation before conversion. The over-parameterization property of *RepVGG Block* is practically beneficial to training, please see Section 7.1 for a detail comparison.

Besides, temporally accumulating consecutive 1-spp frames can effectively improve the temporal stability and increase the effective spp of each frame. We employ a temporal accumulation preprocessing step before sending the noisy inputs to the denoising pipeline just like [SKW*17, KIM*19, MZV*20]. We first reproject the previous frame to the current frame with the motion vector and then judge their geometry consistency by world position and shading normal feature buffers. Current frame pixels that passed the consistency test are blended with their corresponding pixels in the previous frame, while the failed pixels remain original 1 spp.

4.2. Reconstruction Phase

Kernel Construction. The original kernel prediction method [BVM*17] directly predicts the kernel map with a resolution $H \times W \times (k * k)$ for the per-pixel filtering kernels of window size $k \times k$. While achieving high quality, this scheme leads to a heavy inference overhead when predicting a large-size kernel, limiting its application in real-time application. In order to address this limitation, our method predicts the encoding of the kernel map with a purpose $H \times W \times 1$, namely the importance map. Then we manually construct the $H \times W \times (k * k)$ kernel map from the encoding. The kernel construction process is illustrated in Figure 3. For each pixel, we leverage the neighborhood scalar values in the importance map to construct its filtering kernel weights. Specifically, we first unfold the importance map with a sliding window with window size k to obtain the unnormalized kernel map with resolution $H \times W \times (k * k)$. Then we normalize it with a softmax function along the channel axis. The softmax normalization helps our constructed filtering kernels have conserving energy, and it ensures well-behaved gradients for the network training [BVM*17, HMS*20]. The constructed kernel map stores filtering kernel weights for each pixel, which are used to filter the noisy input. Note that there are no learnable parameters in our kernel construction process.

As mentioned above (Section 3), we extend the kernel prediction method to a real-time architecture by enabling a lightweight network design with a scalable kernel construction module (Figure 1 II) For the kernel construction part, we observe negligible time cost compared with the total prediction module (Figure 1 I), attributing to the efficient kernel construction scheme. Note that our reconstruction module has a constant inference time with respect to the size of the kernel map. Hence, our method runs faster in constructing large-sized kernel maps compared with the original kernel prediction method.

Kernel Fusion. While filtering kernels are widely varying, encoding very complex filtering information into only one single-channel importance map has limitations because the filtering weights are highly correlated for nearby pixels. For example, it might not be able to capture signal changes in different directions around object edges, while this could be achieved with KPCN [BVM*17] by independently estimating a separate kernel at each pixel. Our alternative solution is to reconstruct multiple channels of importance maps at each pixel to decompose directional filtering information then fuse the denoised to achieve the final results. Hence, we further design a kernel fusion architecture to improve the denoising ability of our method. Specifically, given the scalability of our *ImportanceNet* and the reconstruction module, we can decompose the filtering information to multiple importance maps to encode more information. As shown in Figure 1, instead of predicting only one importance map and constructing one **fusing kernel**, our *ImportanceNet* predict multiple separate importance maps, and then we construct a set of kernel maps with different filtering sizes k_i and $i \in \{0, ..., M-1\}$. We independently filter the noisy image with

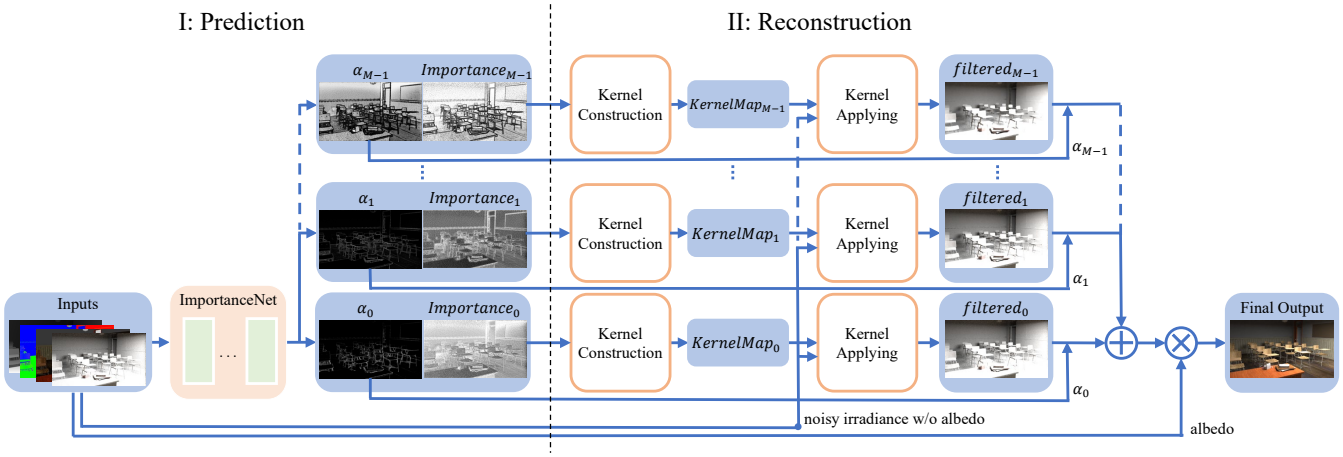

Figure 1: *An overview of our kernel prediction denoising framework. The input data are preprocessed and sent to the ImportanceNet, and we construct the filtering kernels with varying sizes from the evaluated importance maps. Then we fuse the independently filtered images with a weighted average and multiply back the albedo to obtain the final denoised image.*

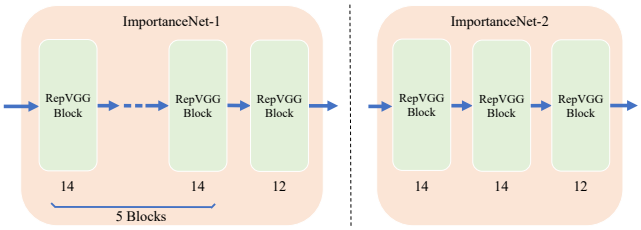

Figure 2: *Architectures of ImportanceNet for 1-spp input. One has six RepVGG Blocks (left) and another employed three (right). The output channels of each layer is shown below each block. The structure of RepVGG is in the supplementary document.*

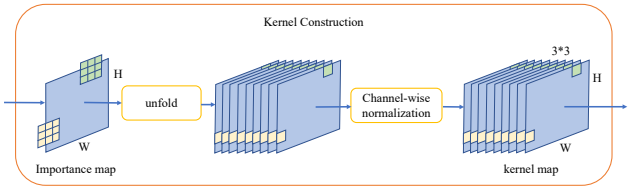

Figure 3: *Framework of kernel construction module. We construct a kernel map with 3×3 filtering window size for illustration. For the input single-channel importance map, we use a sliding window of size 3 to unfold it to 9 channels in a splatting manner, then we apply a channel-wise normalization to obtain the kernel map. The yellow and green color cells represent data flow of two example positions.*

filtering kernels of different sizes and then fuse the results with a weighted average to form the final filtered result as in Equation 5, where the per-pixel average weights α_i are also predicted by our *ImportanceNet* in the last convolutional layer and normalized by a softmax activation function. In our implementation, we use a sequence of odd-sized ($\{3, 5, 7, ...\}$) filtering kernels, which is a common selection of kernel size like that in [DZM*21]. Please see Section 7.1 for the evaluation of our kernel fusion module.

Furthermore, with the kernel fusion architecture, filtering kernels of different sizes respond to filtering on different frequencies. Our averaging weight map is trained to blend the filtered results on each pixel independently. It gives higher weight to large-sized kernel filtered results in low-frequency regions and vice versa. While fusing more filtering kernels could improve the denoising quality, it costs more time. We need to choose an appropriate fusing kernel count to have a good trade-off between quality and performance. Ablation study for the number of kernels in Section 7.1 is conducted for the appropriate number we chose in our implementation.

As shown in Figure 1, we filter the noisy input irradiance without albedo, and at the last step, we multiply back the albedo to get the final denoised radiance image. The albedo demodulation and re-modulation is a common practice [ZRJ*15, BVM*17, KIM*19, MZV*20] because the effective irradiance buffer is smoother and has simpler noise patterns. Thus denoising with the irradiance can avoid over-blurring the detail of the texture.

5. Experimental Setup

5.1. Datasets

We adopt an existing dataset from the work of Koskela et al. [KIM*19] called the BMFR dataset. This dataset comprises 1-spp path traced noisy images, 4096-spp reference images, and the related feature buffers (albedo, shading normal, word position, and camera-space depth) from six scenes with rendering effects like glossy reflections and soft shadows. Each scene has 60 consecutive frames with smooth camera movement. The similar setup is also adapted by Meng et al. [MZV*20].

To evaluate our generalization ability to high spp input, we also

use another high-quality Tungsten dataset made public by Meng et al. [MZV*20]. This dataset is rendered with eight publicly available Tungsten scenes [Bit16] covering complex geometry information and lighting conditions. See Figure 4 for some example scenes. Each scene contains a consecutive sequence of 100 frames, and the noisy image is rendered with 64 spp.

Figure 4: *Example scenes of Tungsten dataset. The reference images are rendered with 4096 spp.*

Our inputs to the *ImportanceNet* are comprised of low dynamic range (LDR) 3-channel color modulated with 3-channel albedo, 3-channel normal, and 1-channel depth (10-channel in total). We linearly scale normal buffer and depth buffer to range $[0,1]$. We apply tone mapping for the original high dynamic range (HDR) input radiance (gamma correction for the BMFR dataset and Filmic tone mapping for the Tungsten dataset) before it is sent to the *ImportanceNet*. In the kernel applying step, we filter the HDR irradiance without albedo to preserve the original value distribution.

For both datasets, we hold out frames of one scene as test set and use the other scenes as the training data. During training, we uniformly sample 80 image patches with resolution 128×128 from each 1280×720 frame of training data to form a training dataset and sample another 20 patches each frame for the validation dataset.

5.2. Training

For the BMFR dataset, we trained our network with the *symmetric mean absolute percentage error* (SMAPE) [VRM*18] as suggested by [VRM*18, MH20]. Given denoised image R and reference t in HDR domain, SMAPE computes:

$$SMAPE(R,t) = \frac{1}{3N} \sum_{p \in N} \sum_{c \in C} \frac{|R_{p,c} - t_{p,c}|}{|R_{p,c}| + |t_{p,c}| + \varepsilon}, \tag{6}$$

where N is the number of pixels in the image, C denotes color channels, and ε is a small number 0.01.

For the BMFR dataset , we use *RepVGG Block* to build our *ImportanceNet* as described in Section 4.1 for real-time speed. For the Tungsten dataset, which is not used for real-time application, we use a more complex network with multi-resolution architecture [VRM*18, MZV*20] (the network architecture details can be found in the supplementary document). This network has a U-net structure [RFB15], and we construct kernels and filter the input in 3 resolutions. For two adjacent resolutions, we denote the coarse-resolution image, fine-resolution image, and the predicted pixel-wise blending weight with $\mathbf{i}^c$, $\mathbf{i}^f$, and α, respectively. $\mathbf{D}$ and $\mathbf{U}$ are 2×2-downsampling and nearest-neighbor upsampling operators, respectively. Then the denoised results of different resolutions are combined progressively from the coarsest resolution as:

$$\mathbf{o}_p = \mathbf{i}^f_p - \alpha_p \left[\mathbf{U}\mathbf{D}\mathbf{i}^f\right]_p + \alpha_p \left[\mathbf{U}\mathbf{i}^c\right]_p. \tag{7}$$

5.3. Implementation

Our denoiser is implemented in PyTorch [PGC*17]. We implemented filtering kernel construction and fusion modules in CUDA as extended operators of PyTorch. We use one single function to realize kernel construction, filtering, and fusing in a streaming manner, which avoid explicitly constructing very large kernel maps stored in global memory as the basic KP method does. In this manner, the memory footprint when denoising one 720p-frame is reduced from 3.95G to 1.25G compared with KP. The weights of our neural network are initialized with the default uniform distribution. We use a batch size of 64 and train the network with Adam [KB14] optimizer with a learning rate of 0.001. For the BMFR dataset, we construct six filtering kernels for kernel fusion module, which means that our *ImportanceNet* predicts 6 importance maps and six corresponding averaging weight maps. We use base kernel size $k_b = 3$ and kernel size incremental step $k_s = 2$, so our minimum kernel size is 3 and maximum kernel size is 13. For the Tungsten dataset, we fuse two filtering kernels with sizes 3 and 5 for each resolution. For each test case, we hold out one scene as test set and train the network with remaining scenes for 500 epochs, which takes 12 hours on an NVIDIA RTX 2080 Ti GPU.

6. Results

6.1. Evaluation Metrics and Compared Algorithms

We use Peak Signal-to-Noise Ratio (PSNR) and Structural SIMilarity (SSIM) [WSB03] as metrics to evaluate our results. We compare our method to a range of existing real-time denoisers: Neural Bilateral Grid Denoiser (NBGD) [MZV*20], BMFR approach [KIM*19], OptiX Neural Network Denoiser (ONND) [CKS*17], Spatio-temporal Variance-Guided Filtering method (SVGF) [SKW*17], and two variants of Kernel Prediction Convolutional Network [BVM*17] (KPCN) with a small network size (KP) or a Multi-Resolution architecture (MR-KP) [VRM*18] modified by Meng et al. [MZV*20]. We also include a comparison with the traditional offline filter-based denoiser NFOR [BRM*16]. We use the implementations provided by their authors except for ONND and KP.

We use two KPCN variants for a comprehensive evaluation. The first variant KP has the same lightweight network architecture as ours (six *RepVGG Blocks*) and predicts filtering kernels with size 13×13. The second variant MR-KP adapted by Meng et al. [MZV*20] has a similar multi-resolution network architecture as described in Section 5.2 and predicts filtering kernels with size 5×5 in three resolutions. We trained the variants using the same loss function and dataset as ours, and both of them run at interactive frame rates.

Following the setup of the original paper, we evaluate the NBGD [MZV*20] with two convolutional layers for the BMFR dataset and seven convolutional layers with dense connections for the Tungsten dataset, respectively. Both the network architectures have the same multi-scale framework with three bilateral grids. We keep it the same as the original paper when doing the experiments.

ONND [CKS*17] is provided as a black box module in OptiX 5.1, and it performs albedo demodulation and re-modulation in the module itself. Because the neural network model of ONND is trained with toned mapped 1-spp input data without temporal accumulation, we send input data without temporal accumulation to avoid introducing bias.

For NBGD, BMFR, SVGF, MR-KP, and KP, we use the same temporal accumulation and albedo demodulation and re-modulation preprocessing steps as ours. Because we do not observe improvement on NBGD with the SMAPE loss, we keep the original loss function of NBGD, i.e. L_1, unchanged to achieve its best for a fair comparison.

6.2. Quality Comparisons

We present some of the visual results and the average PSNR and SSIM in this section. Please see our supplementary document for more comparisons using other error metrics.

BMFR dataset. We use a lightweight 6-layer convolutional neural network and six fused filtering kernels to denoise the 1-spp BMFR dataset in real-time speed. The example denoised images in test scenes are shown in Figure 5. Overall, kernel prediction-based methods (KP, MR-KP) produced visually pleasing results, and our method achieved a similar visual and numerical quality on par with them. Compared with one of the state-of-the-art real-time neural denoisers NBGD, our method constructed more realistic and clear shadows using less denoising time (Figure 5, crops row 1, 3, 9, 10). The grid splatting method of NBGD had difficulty suppressing outlier pixels and left some firefly artifacts for glossy material, while our method reconstructed more smooth glossy reflection results (Figure 5, crops row 7, 8). Besides, our method performed better at preserving object edges and handling occlusions with fewer artifacts (Figure 5, crops row 2, 4, 5, 6).

We report the numerical results in Table 1 and Table 2. The corresponding average execution time is reported in Table 3. Typically, kernel prediction-based methods (KP, MR-KP) have better quantitative quality than other denoising approaches, albeit some of them are faster. Our method achieves a comparable result with the kernel prediction-based methods and runs at a roughly 2-times faster speed. Compared with the recent real-time denoiser NBGD, our method has lower quantitative errors while runs faster than NBGD (12.9 ms vs. 14.0 ms). Besides, given the scalability of our architecture, we adopted a more efficient *ImportanceNet* with a 3-layer convolutional neural network and six fused filtering kernels (Figure 2, right) for better run-time performance. As shown in Table 1, 2, and 3, the denoising quality of our optimized architecture is on par with NBGD, but ours only needs 6.6 ms to denoise one 720p frame, which is roughly a half of NBGD's 14.0 ms.

As discussed in Section 2, our kernel-based reconstruction scheme is more efficient compared with that of NBGD's. Here we perform an ablated study to compared the time cost of the reconstruction parts of NBGD and ours: the grid creating and slicing of NBGD, and the kernel construction and filtering of our method, precisely.

As shown in Figure 6, the grid-based operations are more time-consuming than our kernel-based operations. For the single resolution architecture and similar reception field comparison, NBGD with a window of size 8 takes 2.15 ms while our method of size 9 takes 0.31 ms. For the complete architecture comparison, NBGD with 3 grids takes 5.35 ms, while our method with 6 kernels fused takes 1.10 ms. In addition, the reconstruction time cost of NBGD grows significantly faster over guide channel size (grid count or kernel count) than ours, demonstrating that our approach has better performance for scalable network outputs. Compared with NBGD, the efficiency of our reconstruction part allows our denoising pipeline to use a deeper network under the same time budget.

Tungsten dataset. To denoise the 64-spp Tungsten dataset, we use a multi-resolution neural network architecture with two fused filtering kernels at each resolution as described in Section 5.2. We show the visual comparisons of denoised results in Figure 7 and the average error metrics in Table 4. Note that for the Tungsten dataset, which is generated for offline applications, we do not use the temporal accumulation step. Overall, our method has the denoised results on par with the compared methods in both visual quality and quantitative metrics.

Due to the complex configuration of Tungsten scenes and the limited sampling rates, in both 64-spp noisy input and the 4096-spp reference image, there exist outlier pixels whose values are significantly larger than their neighborhoods. Meng et al. [MZV*20] suggested an outlier removal preprocessing step to the noisy input. However, this operation would change the energy distribution of the original path traced image and have a negative influence on error metrics, so we did not include it in our comparison experiments. The blue insets of the *Dining room* (row 2 of Figure 7) show that NBGD and NFOR could not effectively remove the outliers while ours, MR-KP, and ONND reduced most outliers to an unnoticeable level. Besides, as shown in the insets of *Kitchen* scene (row 1 of Figure 7), our approach excelled at reconstructing high-frequency glossy reflection and smooth shadows, while MR-KP and NBGD still left some residual noise.

We report the average denoising time of our multi-resolution 2-kernel architecture in the third data row of Table 3. Our method runs at 44 FPS in this 64-spp offline rendering application, showing a possible application to preview high-spp offline rendering.

7. Analysis

7.1. Modular Evaluations

In this subsection, we evaluate the effectiveness of our design choices. We modified the configuration of each compositing module and compared the denoising quality with our complete architecture. Specifically, the complete architecture consists of a 6-layer convolutional neural network with *RepVGG Block* and six fused fil-

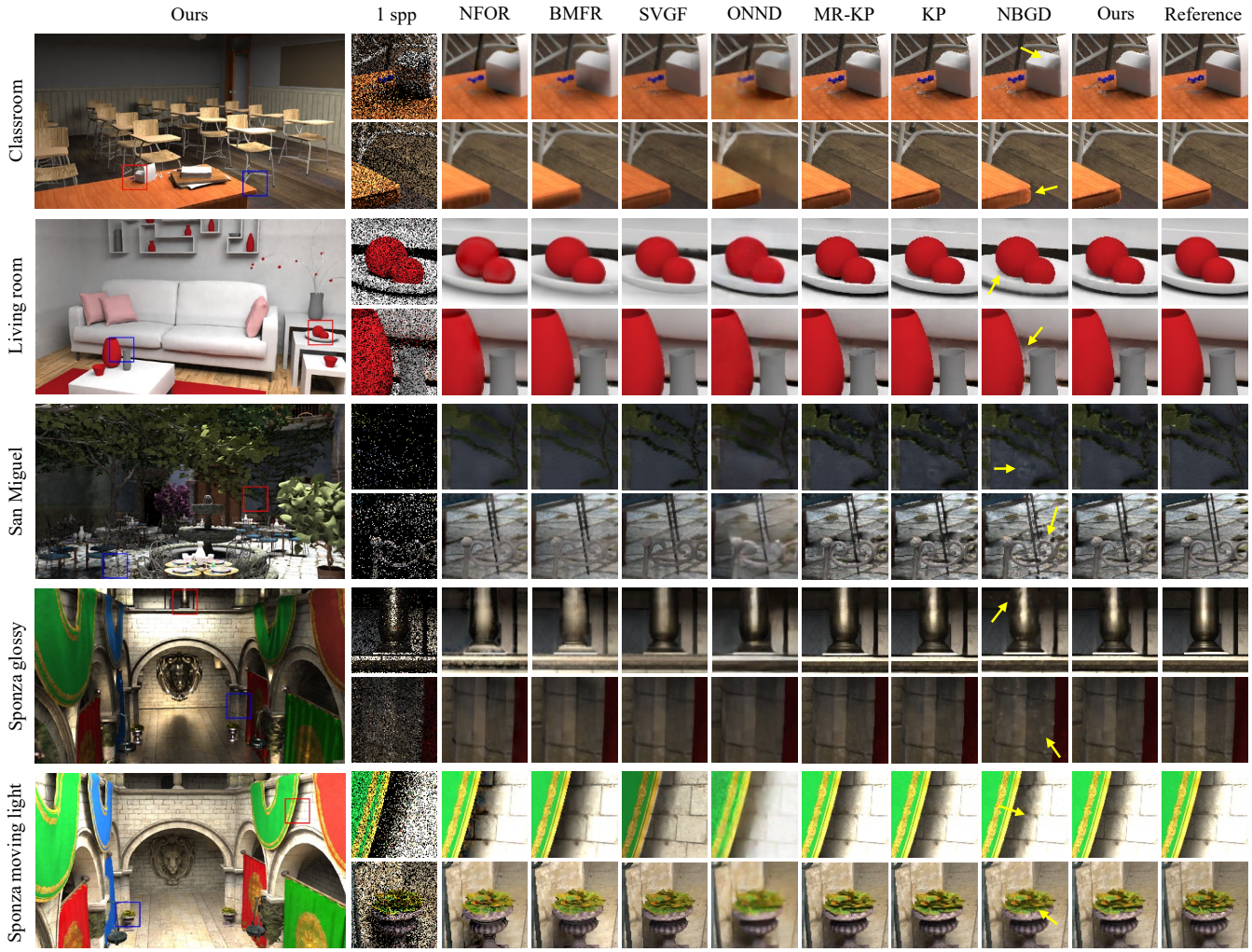

Figure 5: *Visual comparisons of denoising quality on the 1-spp BMFR test data. Each row represents an independent experiment where the displayed scene is held up for test and other scenes are used for training. Closeups highlight the visual differences between our method and other compared methods.*

tering kernels . The kernel sizes range from 3 to 13. Other training settings are untouched in the ablation study.

RepVGG Block. Practically, the denoising quality of our method is essentially influenced by the capacity of our *ImportanceNet*, as can be seen in the quality comparison of *Ours 6-layer* and *Ours 3-layer* in Table 1 and Table 2. However, the time budget of real-time applications has restricted the network architecture to be lightweight with a few narrow layers. As mentioned in Section 4.1, the *RepVGG Block* [DZM*21] utilizes the structural re-parameterization technique to convert one single branch CNN architecture to multiple branches with increased number of trainable parameters, and at the same time keeps the original inference speed. We present the error metrics of our 6-layer neural network and the KP method trained with and without *RepVGG Block* in Table 5. In practice, *RepVGG Block*'s over-parameterization property could avail the training process of small-sized networks, which brings us about 0.5dB improvement in PSNR for the presented test scenes. While the improvements to the KP method are limited due to its network parameter number is already large without the *RepVGG Block* (nearly 3× larger than ours). Besides, we experimentally found that this structure has limited benefits for more shallow networks, specifically no improvement on average for the 2-layer networks of NBGD.

Kernel Construction and Kernel Prediction. To further compare the proposed method with the original kernel prediction method KPCN [BVM*17] under real-time application time budget, we design another two architecture variants of KPCN: a 2-layer network with filtering kernel size 13 (KP-1) and a 5-layer network with filtering kernel size 7 (KP-2). As presented in Table 6, the comparison results show that there is a sharp decline in numerical quality if we reduce the network size of KP to satisfy the time budget (KP-1). In addition, the KP architecture with a proper network size and a

Table 1: *Average PSNR comparison (higher is better) on 1-spp BMFR test data. We also added the metric averaged with all test scenes at the last row and noted that we use hold-out dataset partition, this averaged metric is just for a rough analysis. Ours 6-layer represents the 6-layer convolutional neural network architecture and Ours 3-layer represents the 3-layer convolutional neural network architecture.*

Scene	PSNR								
	NFOR	BMFR	ONND	SVGF	MR-KP	KP	NBGD	Ours(6-layer)	Ours(3-layer)
Classroom	29.872	28.965	27.312	25.034	33.030	**33.047**	31.534	32.827	32.220
Living room	31.304	30.025	25.586	27.239	33.832	**34.090**	32.506	34.063	32.913
San Miguel	21.811	20.969	20.172	18.736	24.047	24.215	23.807	**24.269**	23.860
Sponza	30.377	31.111	24.698	24.401	**34.730**	34.595	33.412	34.600	33.796
Sponza (glossy)	25.974	25.005	23.460	20.917	**30.923**	30.719	29.678	30.805	29.981
Sponza (mov. light)	21.999	17.377	22.291	17.260	25.372	25.324	24.866	**25.374**	25.050
Average	26.889	25.575	23.920	22.264	30.322	**30.332**	29.474	30.323	29.637

Table 2: *Average SSIM comparison (higher is better) on 1-spp BMFR test data. We also added the metric averaged with all test scenes at the last row and noted that we use hold-out dataset partition, so this averaged metric is just for a rough analysis. Ours 6-layer represents the 6-layer convolutional neural network architecture and Ours 3-layer represents the 3-layer convolutional neural network architecture.*

Scene	SSIM								
	NFOR	BMFR	ONND	SVGF	MR-KP	KP	NBGD	Ours(6-layer)	Ours(3-layer)
Classroom	0.956	0.955	0.924	0.952	**0.978**	**0.978**	0.967	0.977	0.973
Living room	0.967	0.965	0.953	0.950	0.977	0.978	0.971	**0.979**	0.973
San Miguel	0.799	0.789	0.744	0.790	**0.851**	**0.851**	0.833	0.849	0.841
Sponza	0.939	0.948	0.852	0.927	0.982	0.982	0.975	**0.983**	0.979
Sponza (glossy)	0.901	0.907	0.867	0.913	**0.962**	0.960	0.946	0.961	0.955
Sponza (mov. light)	0.895	0.858	0.811	0.876	**0.961**	0.958	0.947	0.958	0.953
Average	0.910	0.904	0.858	0.901	**0.952**	0.951	0.940	0.951	0.946

Table 3: *Run-time performance of each denoising approach at a resolution of* 1280×720*. Ours 3-layer, Ours 6-layer, and NBGD 2-layer are designed for 1 spp denoising, while Ours MR and NBGD 7-layer are designed for 64-spp denoising. For neural denoising methods, we decompose the total denoising time into prediction time and reconstruction time, and they are shown in the bracket, respectively. We ran 300 iterations for each method and reported the average timing for one single frame to decrease random fluctuations.*

Method	Timing(ms)	Device
Ours 3-layer	6.58 (5.47 / 1.11)	Nvidia RTX 2080 Ti
Ours 6-layer	12.89 (11.75 / 1.14)	Nvidia RTX 2080 Ti
Ours MR	22.70 (21.85 / 0.85)	Nvidia RTX 2080 Ti
NBGD 2-layer	13.97 (7.84 / 6.13)	Nvidia RTX 2080 Ti
NBGD 7-layer	50.28 (44.16 / 6.12)	Nvidia RTX 2080 Ti
KP	27.86 (26.23 / 1.22)	Nvidia RTX 2080 Ti
MR-KP	24.73 (23.73 / 1.00)	Nvidia RTX 2080 Ti
BMFR	1.60	Nvidia RTX 2080
SVGF	4.40	Nvidia Titan X
ONND	55.00	Nvidia Titan X
NFOR	370.00	Intel Core i7-8700 CPU

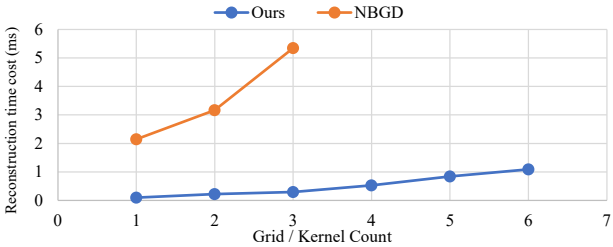

Figure 6: *The reconstruction timing change over guide channel size. For NBGD, the x-axis represents the number of bilateral grid used, and the grid resolutions vary from* $320 \times 180 \times 64$ *to* $80 \times 45 \times 16$ *with scale ratio* 2*. For our method, the x-axis represents the number of filtering kernel fused, and the kernel sizes vary from* 3×3 *to* 13×13 *with incremental step* 2*.*

small filtering kernel size has a similar execution speed with ours but produced a worse denoising quality (KP-2).

Fusion of Different Sized Kernels. Most recent neural denoisers benefit from the design of denoising the input in different spatial resolutions. For example, NBGD constructs bilateral grids with different resolutions [MZV*20], and the layer-based denoiser [MH20] splats samples to ordered layers then independently filter these layers. Likewise, we propose to denoise one pixel position with sev-

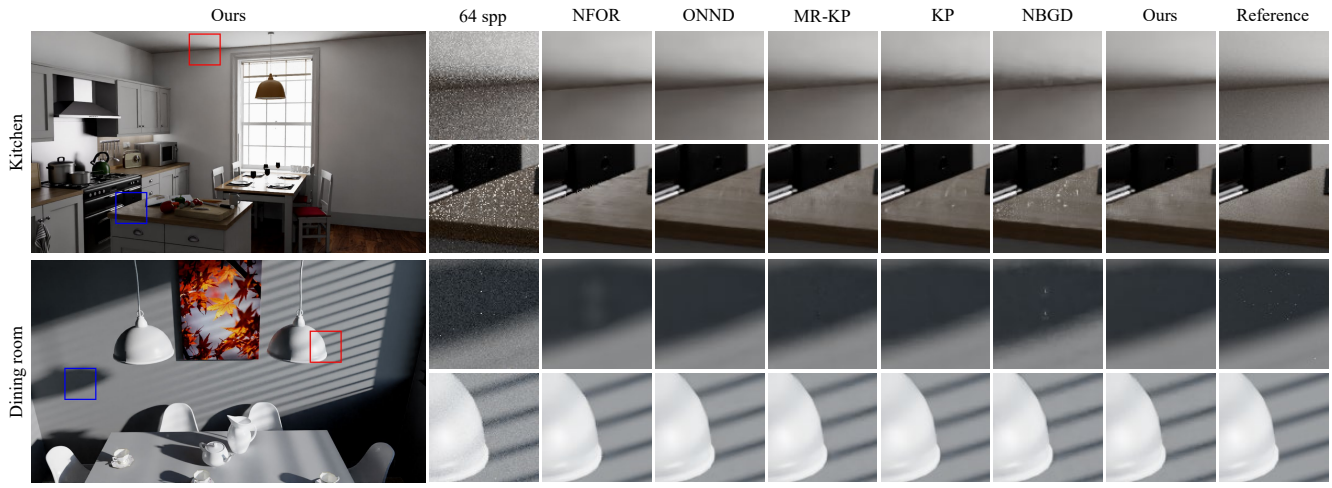

Figure 7: *Visual comparisons of denoising quality on the 64-spp Tungsten test scenes of Kitchen and Dining room. We use the multi-resolution neural network architecture with two fused kernels in each resolution.*

Table 4: *Error metrics comparisons on 64-spp Tungsten test scenes of Kitchen and Dining room, averaged with 100 consecutive frames for each scene. We use the multi-resolution neural network architecture with two fused kernels each resolution.*

Scene	PSNR						SSIM					
	NFOR	ONND	MR-KP	KP	NBGD-7	Ours MR	NFOR	ONND	MR-KP	KP	NBGD-7	Ours MR
Kitchen	34.68	34.80	**36.31**	36.03	35.53	35.88	0.973	0.973	**0.978**	0.977	0.974	0.976
Dining room	36.34	37.95	37.87	**38.11**	35.98	38.03	0.980	0.970	0.980	**0.982**	0.979	0.981

Table 5: *Error metrics comparisons for our ImportanceNet and KP method trained with and without RepVGG Block, averaged over 60 consecutive frames of each test scene.*

Scene	PSNR		SSIM	
	w/o	w/	w/o	w/
Living room (Ours)	33.470	**34.063**	0.9754	**0.9789**
Sponza (Ours)	33.925	**34.600**	0.9813	**0.9830**
Living room (KP)	33.962	**34.090**	0.9771	**0.9781**
Sponza (KP)	34.541	**34.595**	0.9829	**0.9830**

Table 6: *PSNR and SSIM comparison averaged on all test scenes, and note that we use hold-out dataset partition, so this averaged metric is just for a rough analysis. Please see the detailed comparison of each test scene in our supplementary. KP-1 represents a KPCN variant with two layers CNN and filtering kernel size 13. KP-2 represents a KPCN variant with five layers CNN and filtering kernel size 7.*

Type	KP-1	KP-2	Ours
PSNR	29.517	29.809	30.323
SSIM	0.941	0.946	0.951
Time (ms)	19.70	12.99	12.89

eral varying-sized filtering kernels and combine the independently filtered results with a weighted average. Our kernel fusion module can overcomes the limitation of our correlated filtering kernels and bring the flexibility of explicitly handling varying-frequency noise as discussed in Section 4.2. We visualize our kernel fusion scheme's prediction with kernel sizes $\{3, 7, 11\}$ in Figure 8 to intuitively check its working behaviors. The end-to-end training manner makes our *ImportanceNet* cooperate reasonably with the kernel fusion module. On the one hand, the filtering kernels use a cooperative manner to capture the signal changes from different directions. For example, around the book edges in Figure 8, the averaging weights of filtering kernels with size 3 and 7 have high values at opposite edge sides, which means they separately reconstruct pixels in different directions to contribute to the final denoised result when signal change happens. On the other hand, for high-frequency regions, our *ImportanceNet* predicted higher averaging weight for small kernel filtered results and lower averaging weight for low-frequency regions. We also provide an interactive viewer in the supplementary to visualize the fusing kernel's weights and the corresponding averaging weights to examine reconstruction behavior better.

We also took the ablation study of our kernel fusion module to further evaluate its impact. As shown in Figure 9, we took two groups of experiments with the same network architecture. The first group (the first inset row in Figure 9) included six experiments trained without the kernel fusion module, which means they

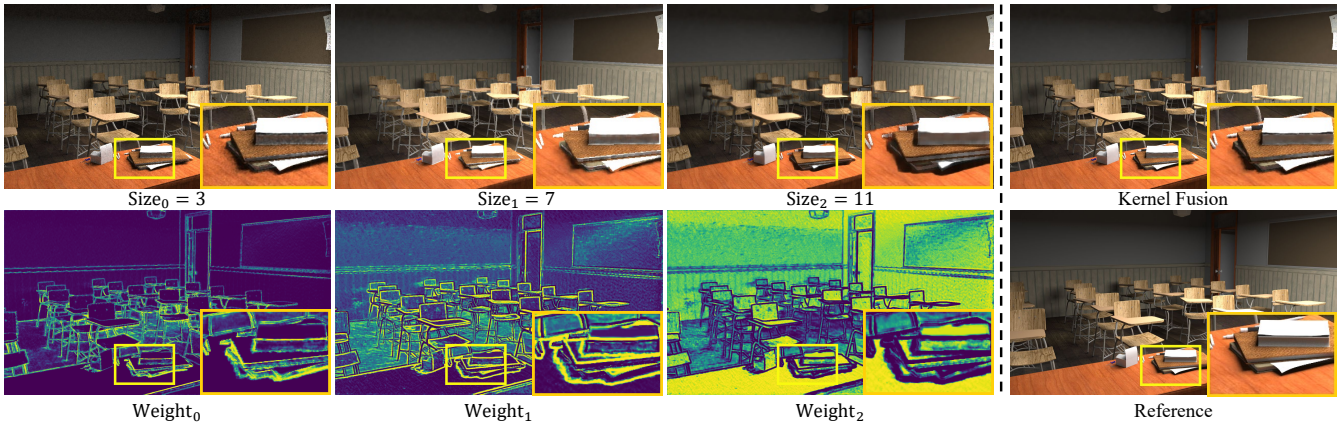

Figure 8: *Visual quality comparisons between a kernel fusion architecture and the independently filtered results for different kernel sizes. Bright colors in weight map correspond to high averaging weight. The ImportanceNet predicts higher averaging weight in high-frequency regions for small kernel filtered result while higher averaging weight in low-frequency region for large kernel filtered result, thus the fused result preserves the advantage of each sized filtering kernel.*

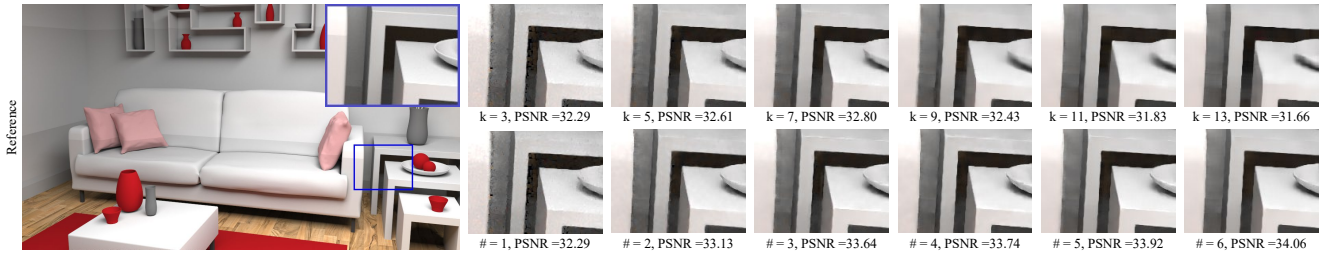

Figure 9: *Comparison of the architecture with single filtering kernel and with fused filtering kernels on Living-room test scene. The first inset row shows the reconstructed results with a single filtering kernel, and the k means the size of constructed filtering kernel. The second inset row shows the reconstructed results with the kernel fusion module. The # stands for the fused filtering kernel count, and we cumulatively add filtering kernel from size* 3 *to size* 13*. The average PSNR is computed with the* 60 *full-resolution frames.*

all constructed one single filtering kernel but with varying sizes $k_i = \{3,5,7,...,13\}$ to denoise. The second group (the second inset row in Figure 9) includes six experiments trained with the kernel fusion module. They have the increasing fusing kernel count $\#_i = \{1,2,...,6\}$ and the kernel sizes are from 3 with an incremental step of 2. For example, the third experiment has 3 fused kernels with sizes $\{3,5,7\}$. The visual results and the average error metrics presented in Figure 9 show that when using only one single filtering kernel, a small-sized filtering kernel can reconstruct sharper edges and a large-sized kernel performs better in low-frequency regions, while our kernel fusion module can take advantages of the varying-sized kernels with the weighted average. The average PSNR summarizes that the kernel fusion module outperforms any of its single compositing kernels in numerical error metrics. Besides, the first inset row also shows a failure case with leaking colors around the object edges when trained with only one single filtering kernel, and the second inset row shows that the leaking artifacts have decreased when trained with our kernel fusion module.

Varying kernel sizes benefit filtering on different frequencies. In order to evaluate the impact, we also compared our method to the basic KP method modulated with the kernel fusion module, and to one variant of our method by fusing 6 kernels with the same size $k_i = 13$ in the supplementary document.

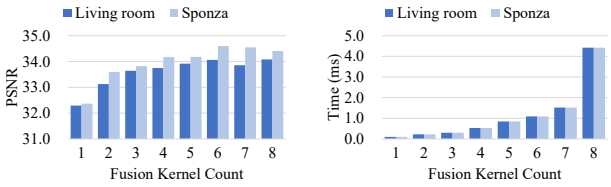

Figure 10: *Left: the denoising quality over different fusion kernel count, measured by average PSNR (higher is better) on the BMFR test scene. The kernel counts are cumulatively added with base size $k_b = 3$ and incremental step $k_s = 2$. Right: the corresponding execution time over fusion kernel count. Results show there are limited quality improvements but non-negligible performance reductions while fusing more than six kernels.*

For each fusing kernel count, we also evaluated the average ex-

ecution time of the corresponding kernel construction and fusion modules and present them in Figure 10. It shows that increasing the fusing kernel count beyond six brings very limited quality improvement but significant time cost, so our implementation utilizes six filtering kernels, which is a good trade-off between denoising quality and execution speed.

7.2. Limitations

Temporal Stability. Our method uses the widely adapted temporal accumulation preprocessing step to handle the temporal stability. Therefore, it has similar behaviors and limitations compared to other denoisers in terms of temporal stability. Please see the temporal stability measurement metric comparisons in the supplementary document. There exist some recent techniques worth trying in the future. For example, Xin et al. [XZXY20] proposed to use a temporal loss between the warped outputs of the adjacent frames without introducing additional operations in their pipeline.

Large size kernel reconstruction. The PSNR in Figure 9 experimentally shows that aggressively compressing a large-sized kernel, e.g., over a size of 9, in a single-channel importance map degenerates quality. Because the compression ratio grows squarely over the kernel size k, our *ImportanceNet* will predict an inadequate encoding for a large-size kernel and lead to a poorly behaved reconstructed result. Besides, Figure 10 shows that when using the kernel fusion module, the quality improvement grows slowly if the kernel size is larger than 13. This limitation constrains our approach from directly using very large kernels for offline applications (e.g., 21×21 in KPCN). However, it has little impact on the real-time application we aim for.

Generalization to unseen effects. Similar to other neural denoisers, our method has a common generalization issue. While denoising specific rendering effects outside the training dataset, the method produces artifacts. This issue can be alleviated by enlarging the diversity of the training dataset.

8. Conclusions

We have presented our novel and practical weight sharing kernel prediction denoiser, which can denoise extreme low-spp Monte Carlo path traced images in real-time. At the core of our approach, we utilize an efficient neural network, *ImportanceNet*, to learn to predict an encoding of the filtering kernel weights. Then we construct the filtering kernels with a hand-crafted decoder in a splatting and fusing manner. The proposed weight sharing kernel prediction denoiser is scalable and allows us to tailor the design for different real-time applications. With this hand-crafted kernel constructor, our method can reduce nearly half the run-time cost and memory requirement of the basic kernel prediction method, and at the same time, produce comparable denoised results.

Acknowledgements

We would like to thank all reviewers for their insightful comments. This research was partially funded by NSFC (No. 61872319), Zhejiang Provincial NSFC (No. LR18F020002), National Key R&D Program of China (No. 2017YFB1002605), and Zhejiang University Education Foundation Global Partnership Fund.

References

[BCM05] BUADES A., COLL B., MOREL J.-M.: A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation 4*, 2 (2005), 490–530. 2

[Bit16] BITTERLI B.: Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 6

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J. A., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 107–117. 6

[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 Trans. Graph. 36*, 4 (July 2017). URL: https://doi.org/10.1145/3072959.3073708, doi:10.1145/3072959.3073708. 1, 2, 3, 4, 5, 6, 8

[CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZAHRAI D., AILA T.: Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph. 36*, 4 (July 2017). URL: https://doi.org/10.1145/3072959.3073601, doi:10.1145/3072959.3073601. 1, 3, 6, 7

[DAN19] DAHLBERG H., ADLER D., NEWLIN J.: Machine-learning denoising in feature film production. In *ACM SIGGRAPH 2019 Talks* (New York, NY, USA, 2019), SIGGRAPH '19, Association for Computing Machinery. URL: https://doi.org/10.1145/3306307.3328150, doi:10.1145/3306307.3328150. 2

[DZM*21] DING X.-H., ZHANG X.-Y., MA N.-N., HAN J.-G., DING G., SUN J.: Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2021). 4, 5, 8

[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 (July 2019). URL: https://doi.org/10.1145/3306346.3322954, doi:10.1145/3306346.3322954. 2

[HHCM21] HOFMANN N., HASSELGREN J., CLARBERG P., MUNKBERG J.: Interactive path tracing and reconstruction of sparse volumes. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 4*, 1 (2021), 1–19. 3

[HMS*20] HASSELGREN J., MUNKBERG J., SALVI M., PATNEY A., LEFOHN A.: Neural temporal adaptive sampling and denoising. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 147–155. 1, 2, 3, 4

[HWZ*20] HUO Y., WANG R., ZHENG R., XU H., BAO H., YOON S.-E.: Adaptive incident radiance field sampling and reconstruction using deep reinforcement learning. *ACM Transactions on Graphics (TOG) 39*, 1 (2020), 1–17. 2

[HY21] HUO Y., YOON S.-E.: A survey on deep learning-based monte carlo denoising. *Computational Visual Media* (2021), 1–17. 2

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 6

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph. 34*, 4 (2015), 122–1. 2

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

[KIM*19] KOSKELA M., IMMONEN K., MÄKITALO M., FOI A., VIITANEN T., JÄÄSKELÄINEN P., KULTALA H., TAKALA J.: Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Trans. Graph. 38*, 5 (June 2019). URL: https://doi.org/10.1145/3269978, doi:10.1145/3269978. 2, 3, 4, 5, 6

[KS13] KALANTARI N. K., SEN P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum (Proceedings of Eurographics 2013) 32*, 2 (2013). 2

[LWC12] LI T.-M., WU Y.-T., CHUANG Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–9. 2

[McC99] MCCOOL M. D.: Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics (TOG) 18*, 2 (1999), 171–194. 2

[MH20] MUNKBERG J., HASSELGREN J.: Neural denoising with layer embeddings. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 1–12. 1, 2, 6, 9

[MMBJ17] MARA M., MCGUIRE M., BITTERLI B., JAROSZ W.: An efficient denoising algorithm for global illumination. In *Proceedings of High Performance Graphics* (New York, NY, USA, July 2017), ACM. doi:10/gfzndq. 3

[MZV*20] MENG X., ZHENG Q., VARSHNEY A., SINGH G., ZWICKER M.: Real-time Monte Carlo Denoising with the Neural Bilateral Grid. In *Eurographics Symposium on Rendering - DL-only Track* (2020), Dachsbacher C., Pharr M., (Eds.), The Eurographics Association. doi:10.2312/sr.20201133. 2, 3, 4, 5, 6, 7, 9

[PGC*17] PASZKE A., GROSS S., CHINTALA S., CHANAN G., YANG E., DEVITO Z., LIN Z., DESMAISON A., ANTIGA L., LERER A.: Automatic differentiation in pytorch. 6

[RFB15] RONNEBERGER O., FISCHER P., BROX T.: U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241. 3, 6

[RKZ12] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive rendering with non-local means filtering. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–11. 2

[RW94] RUSHMEIER H. E., WARD G. J.: Energy preserving non-linear filters. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), pp. 131–138. 2

[SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31*, 3 (May 2012). URL: https://doi.org/10.1145/2167076.2167083, doi:10.1145/2167076.2167083. 2

[SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*. 2017, pp. 1–12. 3, 4, 6

[SPD18] SCHIED C., PETERS C., DACHSBACHER C.: Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 1–16. 3

[Ste81] STEIN C. M.: Estimation of the mean of a multivariate normal distribution. *The annals of Statistics* (1981), 1135–1151. 2

[TVLF20] THOMAS M. M., VAIDYANATHAN K., LIKTOR G., FORBES A. G.: A reduced-precision network for image reconstruction. *ACM Trans. Graph. 39*, 6 (2020). doi:10.1145/3355089.3356565. 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 Trans. Graph. 37*, 4 (July 2018). URL: https://doi.org/10.1145/3197517.3201388, doi:10.1145/3197517.3201388. 1, 2, 3, 6

[WSB03] WANG Z., SIMONCELLI E. P., BOVIK A. C.: Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003* (2003), vol. 2, Ieee, pp. 1398–1402. 6

[XZW*19] XU B., ZHANG J., WANG R., XU K., YANG Y.-L., LI C., TANG R.: Adversarial monte carlo denoising with conditioned auxiliary feature modulation. *ACM Trans. Graph. 38*, 6 (2019), 224–1. 2

[XZXY20] XIN H., ZHENG S., XU K., YAN L. Q.: Lightweight bilateral convolutional neural networks for interactive single-bounce diffuse indirect illumination. *IEEE Transactions on Visualization and Computer Graphics* (2020), 1–1. doi:10.1109/TVCG.2020.3023129. 12

[YNS*09] YANG L., NEHAB D., SANDER P. V., SITTHI-AMORN P., LAWRENCE J., HOPPE H.: Amortized supersampling. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 1–12. URL: https://doi.org/10.1145/1618452.1618481, doi:10.1145/1618452.1618481. 3

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer graphics forum* (2015), vol. 34, Wiley Online Library, pp. 667–681. 2

[ZRJ*15] ZIMMER H., ROUSSELLE F., JAKOB W., WANG O., ADLER D., JAROSZ W., SORKINE-HORNUNG O., SORKINE-HORNUNG A.: Path-space motion estimation and decomposition for robust animation filtering. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 131–142. 2, 5