Neural Partitioning Pyramids for Denoising Monte Carlo Renderings

MARTIN BALINT, Max Planck Institute for Informatics, Germany
KRZYSZTOF WOLSKI, Max Planck Institute for Informatics, Germany
KAROL MYSZKOWSKI, Max Planck Institute for Informatics, Germany
HANS-PETER SEIDEL, Max Planck Institute for Informatics, Germany
RAFAŁ MANTIUK, University of Cambridge, United Kingdom

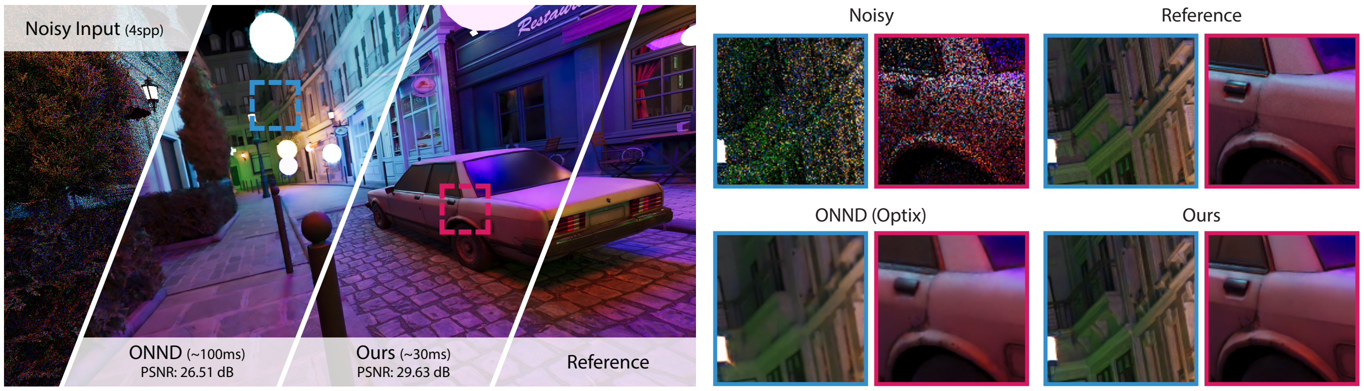

Fig. 1. Result of our denoiser compared to noisy input generated with 4 samples per pixel (spp), a reference image generated using 6144 spp, and a state-of-the-art ONND method (OptiX) [Nvidia 2022b]. Our solution produces better image quality in significantly less time. *Bistro* © 2023 Amazon Lumberyard

Recent advancements in hardware-accelerated raytracing made it possible to achieve interactive framerates even for algorithms previously considered offline, such as path tracing. Interactive path tracing pipelines rely heavily on spatiotemporal denoising to produce a high-quality output from low-sample-count renderings. Such denoising is typically implemented as multiscale-kernel-based filters driven by lightweight U-Nets operating on pixels, and encoders operating on samples. In this work, we present a novel kernel architecture in the line of low-pass pyramid filters. Our architecture avoids the issues with the low-frequency response of previous such filters, resolving ringing, blotchiness, and box-shaped artefacts while improving overall detail. Instead of using classical downsampling and upsampling approaches, which are prone to aliasing, we let our weight predictor networks learn to partition the input radiance between pyramidal layers, predict kernels for denoising each partitioned and downscaled image, and then guide the upsampling process when combining layers. We present failure cases of pyramidal scale-composition in previous work and, through Fourier analysis, show how our method resolves them. Finally, we demonstrate state-of-the-art denoising performance.

CCS Concepts: • **Computing methodologies** → *Image processing*; **Ray tracing**.

ACM Reference Format:
Martin Balint, Krzysztof Wolski, Karol Myszkowski, Hans-Peter Seidel, and Rafał Mantiuk. 2023. Neural Partitioning Pyramids for Denoising Monte Carlo Renderings. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 6–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3588432.3591562

1 INTRODUCTION

Path tracing, despite its high computational cost, has become the primary algorithm used for physically-based rendering throughout the animation and visual effects industry due to its accuracy and flexibility. Current GPUs can produce noisy low-sample-count renderings at interactive rates thanks to hardware-accelerated raytracing and reconstruct clean images using neural denoising filters, also accelerated using linear algebra hardware. While such efficient pipelines have enormous potential to improve creative workflows and unlock new applications of path tracing in the real-time domain, the quality of spatiotemporal denoisers remains a bottleneck.

Many recent works in neural denoising rely on kernel prediction; instead of predicting an output image directly, they map the final activations of their neural networks to local, per-pixel filtering kernels. Then, they compute their output by applying these kernels to the noisy input image. Kernel-predicting methods have proven to hold some significant advantages; they are more robust, train faster and offer improved performance [Bako et al. 2017].

Denoising benefits from larger kernels as weighing and averaging more pixels reduces variance. If the variance is sufficiently low, larger kernels can be more selective, only picking pixels that form local structures, thereby preserving details. Unfortunately, enlarging kernels quickly becomes impractical as the computational cost of kernel prediction and application scales quadratically with the kernel size. Hierarchical pyramid kernels provide a natural solution;

by efficiently applying small, cascaded filtering kernels, these methods can achieve large footprints at low computational costs [Vogels et al. 2018].

Although pyramidal kernel prediction holds immense potential, predicting parameters for such filters with neural networks is challenging. We identify some shortcomings of scale-composition, which limit its overall performance. We propose a novel pyramidal filter that avoids these issues by introducing jointly learnable downsampling and upsampling stages. Our downsampler learns to partition radiance between layers of our filtering pyramid, providing each layer with inputs matching their ideal noise characteristics. In conjunction, our upsampler learns to reconstruct edges and high-frequency details lost during downsampling, accurately aligning each layer for recombination. By their combined effect, each layer of our pyramid filter learns to denoise image structures of corresponding sizes, which can be recombined by simply summing up the upsampled output of each layer. Our filter is inherently energy-preserving and numerically stable. Furthermore, we take special care to use robust activation functions to predict our filtering parameters.

We also present considerable improvements to previous Monte Carlo denoising pipelines by applying prominent ideas from related fields. By adapting backpropagation through time [Hochreiter and Schmidhuber 1997] from recurrent neural networks, we train on 64-frame-long sequences instead of 8, simultaneously increasing the patch size to 256 pixels compared to 128 in previous works. By motion compensation of our training patches, we train our filter to better utilize temporal coherency. We adapt architectures from image restoration [Zamir et al. 2022] and large kernel convolution [Liu et al. 2022a] methods to scale our weight predictor network to 30 million parameters, further improving denoising performance at interactive to offline rates.

To summarise our contributions:

- We propose a pyramidal filter with learnable partitioning and upsampling stages. Through Fourier analysis, we explore how our filter resolves previous shortcomings and provides further advantages.
- We adopt training techniques previously used for recurrent networks and image restoration networks proposing small and large weight predictor networks to drive our method. We demonstrate state-of-the-art performance in multiple timing categories.
- Our implementation and dataset are available under the MIT license on our project website.[1]

2 RELATED WORK

Algorithms for denoising Monte Carlo renderings have actively been developed in recent years, with numerous solutions successful at different performance budgets. We group methods into three categories: real-time denoisers processing 1-4 samples within 30 milliseconds, interactive denoisers processing 4-8 samples in a performance budget between 30 and 100 milliseconds, and offline denoisers process over 8 samples in over 100 milliseconds. We focus on recent works relevant to our work; for a wider overview, we refer the reader to excellent surveys by Huo and Yoon [2021] on deep learning-based methods and by Zwicker et al. [2015] on classical methods.

Non-machine-learning-driven filters [Koskela et al. 2019; Kozlowski and Cheblokov 2021; Nvidia 2022a; Schied et al. 2017; Zhdan 2021] are still commonly used for high framerate real-time applications. However, optimised kernel-predicting filters [Fan et al. 2021; Meng et al. 2020; Thomas et al. 2022] take prominence when the budget allows for inference using a lightweight neural network. Most notably, Thomas et al. [2022] jointly solve denoising and supersampling using a U-Net [Ronneberger et al. 2015] inspired multiscale filter.

Neural kernel-predicting filters dominate interactive denoising methods, with two general approaches emerging: pyramidal filters and generalised bilateral filters [Tomasi and Manduchi 1998]. Our work focuses on pyramidal filtering, with a detailed discussion of its background in Section 3. As an alternative, some recent works generalise bilateral filters and combine them with learning components, building effective neural edge-preserving smoothing filters. Meng et al. [2020] use a lightweight convolutional network to guide a bilateral grid filter. Işık et al. [2021] further generalise bilateral filtering, calculating the range kernel for eight-dimensional per-pixel affinity features. Their approach avoids the issues with pyramid methods but is computationally more expensive due to higher memory bandwidth requirements and less cache-friendly memory access patterns. In addition, orthogonal extensions such as path-based filtering [Cho et al. 2021; Lin et al. 2021], compositional filtering [Zhang et al. 2021], and self-supervised post-correction denoising [Back et al. 2022] have been proposed.

Recent offline filters abandon kernel prediction for directly predicting the output image and instead focus on improving the underlying network architecture. The proposed ideas include transformer blocks [Lu et al. 2020; Yu et al. 2021], deformable convolution [Wei et al. 2021], conditioned feature modulation [Fu et al. 2021; Xu et al. 2019], dual-residual connections [Lu et al. 2021] and generative-adversarial training [Lu et al. 2021, 2020; Xu et al. 2019; Yu et al. 2021]. Zheng et al. [2021] apply ensembles of such denoisers to mitigate the artefacts produced by each individual denoiser. These methods offer much larger receptive fields than those of kernel-predicting architectures. However, we argue that denoising is still a substantially easier task when done in the parameter space of predicted linear denoising kernels; we show that our multiscale filter does not incur the same limitations in terms of kernel size and scales when driven by powerful networks, outperforming direct predicting architectures.

3 PYRAMIDAL DENOISING

Pyramid hierarchical kernels allow for an efficient, cascaded application of small denoising kernels [Hasselgren et al. 2020; Munkberg and Hasselgren 2020; Vogels et al. 2018] while achieving the footprint of large, computationally expensive kernels [Bako et al. 2017]. In this section, we describe the background of previous filter architectures and the shortcomings of their neural adaptations. These limitations serve as the core motivation for our filter design, which we describe in Section 4.

[1] https://github.com/balintio/nppd

We use various symbols to denote radiance processed at different pipeline stages. Supplementary, Table 1 and Figure 4 offer an overview to ensure clarity for our readers.

3.1 Low-pass filtering and downsampling

Pyramidal filters first need to generate a decimated low-pass pyramid [Burt and Adelson 1987] serving as input for per-layer denoising. Although the original formulation uses Gaussian filters, learning-based methods favour box filters or average pooling. Introducing our notation, we can formulate this operation in a closed form:

$$\bar{\boldsymbol{L}}^{l}_{xyt} = \sum_{u=2^l x}^{2^l(x+1)-1} \sum_{v=2^l y}^{2^l(y+1)-1} \frac{1}{2^{2l}} \cdot \bar{\boldsymbol{L}}_{uvt}, \tag{1}$$

where $\bar{\boldsymbol{L}}_{uvt}$ denotes the radiance of pixel (u, v) averaged over samples rendered for frame t (and temporally accumulated for recent frames, Equation 10), and $\bar{\boldsymbol{L}}^{l}_{xyt}$ denotes the radiance of the coarse pixel (x, y) of the l-th layer of the low-pass pyramid, where $l = 0$ stands for the full-resolution and $l = N$ the coarsest layer. We use zero-based indexing for pixel coordinates in our notation. We use a non-recursive pyramid formulation for reasons that will become clear in Section 4.1.

3.2 Denoising kernels

Next, pyramidal filters suppress noise in each layer of the pyramid by applying neurally predicted kernels K^{l}_{uvxyt}, which are unique for each pixel (x, y) and cover its neighbourhood (u, v):

$$\hat{\boldsymbol{L}}^{l}_{xyt} = \sum_{uv} \bar{\boldsymbol{L}}^{l}_{uvt} \cdot \sigma_{xy}(K^{l}_{uvxyt}), \tag{2}$$

where $\hat{\boldsymbol{L}}^{l}_{xyt}$ denotes the layered denoised radiance and σ_{xy} denotes softmax normalisation of the kernels along (x, y). Equation 2 corresponds to a kernel-splatting operation [Gharbi et al. 2019; Munkberg and Hasselgren 2020] as x and y index the denoised layers. Conversely, swapping xy and uv, essentially transposing the operation, would denote a kernel-gathering operation, which is also commonly used by previous work [Hasselgren et al. 2020; Vogels et al. 2018].

3.3 Upsampling and composition

After applying the predicted denoising kernels separately at each scale, pyramidal filters need to upscale the coarse layers and compose an output image. Unfortunately, each layer of the denoised low-pass pyramid contains some residual low-frequency noise that falls below the receptive field of the small denoising kernels. Thus, denoising filters must solve a challenging task; fusing the clean frequency bands from each layer.

Two approaches have been proposed in previous works. First, Delbracio et al. [2014] convert the low-pass pyramid to a Laplacian pyramid by subtracting subsequent layers to isolate the clean frequency bands (shown in brackets in Equation 3). The sum of these bands then gives a clean, full-resolution output. We formulate this operation recursively, where $\tilde{\boldsymbol{L}}^{N}_{xyt} = \hat{\boldsymbol{L}}^{N}_{xyt}$ starts with the coarsest layer, yielding the full-resolution output as $\tilde{\boldsymbol{L}}^{0}_{xyt}$, by applying the following equation:

$$\tilde{\boldsymbol{L}}^{l}_{xyt} = \left[\hat{\boldsymbol{L}}^{l}_{xyt} - \mathrm{U}(\mathrm{D}(\hat{\boldsymbol{L}}^{l}_{xyt}))\right] + \mathrm{U}(\tilde{\boldsymbol{L}}^{l+1}_{xyt}), \tag{3}$$

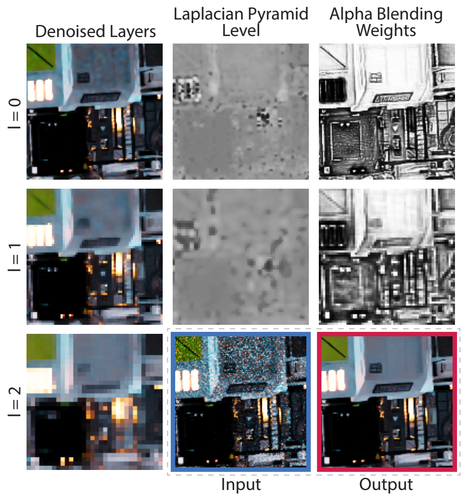

Fig. 2. Scale-composition as proposed by Vogels et al. [2018]. The denoised layers contain residual low-frequency noise. Converting the representation to a Laplacian pyramid by subtracting subsequent layers cancels this residual noise. However, small mismatches between the denoised layers cause overshoots and ringing artefacts in the Laplacian pyramid. The predicted blending weights control composition to reduce these artefacts (lower weights give more conservative composition). *Zero-Day* © 2023 Mike Winkelmann

where U and D are shorthands for the two-by-two upsampling and downsampling operators.

Delbracio et al. [2014] proposed this formulation of scale-composition for a classical method, applying their ray histogram fusion filter to all layers. Matching the response of the filters applied to each layer is critical for Laplacian pyramids, as inconsistencies between the layers where the filters' response overlaps can cause ringing and over-blurring artefacts. Such consistency is difficult to guarantee between neurally predicted kernels; thus, Vogels et al. [2018], when adapting scale-composition for neural kernel prediction, proposed a neurally predicted per-pixel blending weight, α^{l}_{xyt}, that can control composition in specific regions to reduce artefacts:

$$\tilde{\boldsymbol{L}}^{l}_{xyt} = \hat{\boldsymbol{L}}^{l}_{xyt} - \alpha^{l}_{xyt}\mathrm{U}(\mathrm{D}(\hat{\boldsymbol{L}}^{l}_{xyt})) + \alpha^{l}_{xyt}\mathrm{U}(\tilde{\boldsymbol{L}}^{l+1}_{xyt}). \tag{4}$$

While they show this workaround to be reasonably effective, it illustrates a fundamental incompatibility between Laplacian pyramidal denoising and neural kernel prediction. Figure 2 shows the Laplacian pyramid layers and blending weights for an example scene.

Recently, Munkberg and Hasselgren [2020] proposed upscaling layers to full-resolution and taking their weighted sum by neurally predicted weights. While these weights can cancel the residual noise without resorting to Laplacian pyramids, learning the residual noise's characteristics still poses a challenging task for the weight predictor network. In the following section, we present our pyramidal filter design that prevents the creation of such residual noise in the first place.

4 PARTITIONING PYRAMIDS

An overview of our pyramidal denoising is shown in Figure 4 as a grey-colored inset. While its three-stage structure is similar to existing solutions (Section 3), we propose the following innovations.

By letting our downsampler partition the input pixel radiance in a learnable manner, our weight predictor network learns to provide denoising kernels with radiance better suited for their receptive field. This way, compositing becomes as simple as summing up each layer. Our upsampler applies neurally predicted splatting kernels instead of bilinear upsampling, enhancing coarse layers with edge-preserving capabilities, which are utilised thanks to our neural partitioning stage, enhancing the reconstructed detail in noisy areas.

In this section, we introduce our partitioning downsampler and our splatting upsampler, explaining the unique properties of each and how they work in tandem to improve denoising performance.

4.1 Partitioning downsampler

Previous works weigh their pyramidal layers after denoising through learnable composition [Hasselgren et al. 2020; Munkberg and Hasselgren 2020; Vogels et al. 2018]. We aim to simplify this weighting task by bringing it to the front of our denoising filter. We use the per-layer weights to partition the radiance of each pixel before downsampling and denoising. Compared to learnable composition, our partitioning task is more straightforward as it is independent of the following denoising kernels and is not prone to ringing and blurring artefacts.

The softmax function is widely used as the final activation layer of neural networks to partition probability spaces as it provides a smooth mapping from a vector with arbitrary components to a vector whose components fall on the interval $(0, 1)$ and which sum up to one [Goodfellow et al. 2016; Sutton and Barto 2018]. Similarly, we can multiply the resulting weights with a signal to partition said signal; Munkberg and Hasselgren [2020] used softmax normalised weights to partition sample radiances between depth-wise layers for denoising defocus and motion blur, albeit without improvements beyond two layers. Nevertheless, we use a similar operation and do not observe such limitations in our application.

First, we predict partitioning weights w^l_{uvt}, all at full-resolution, which we then softmax normalise along pyramid layers. Then, we partition the radiance across the layers and downsample by average pooling the resulting partitions to produce the multiscale images ready for denoising. We formulate our operation by adding the highlighted part to Equation 1:

$$\bar{L}^l_{xyt} = \sum_{u=2^l x}^{2^l(x+1)-1} \sum_{v=2^l y}^{2^l(y+1)-1} \frac{1}{2^{2l}} \cdot \boxed{\sigma_l(w^l_{uvt})} \cdot \bar{L}_{uvt} \tag{5}$$

where σ_l denotes softmax normalisation along layers. The leftmost side of Figure 6 shows the predicted partition weights for a demonstration scene and the resulting radiance partitions. Intuitively, the partitions separate image features based on scale and noise characteristics, directing smaller or less noisy features to finer layers. Denoising each feature at the finest layer with the minimum required kernel size is beneficial as it frees up coarser layers for larger, noisier features. We show partitions for further test images in Supplementary, Fig. 3.

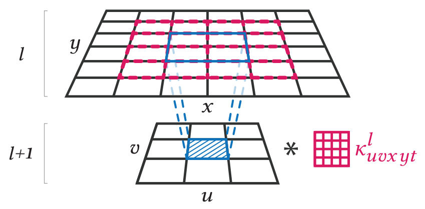

Fig. 3. We splat each coarse pixel (blue) to its four-by-four neighbourhood of one-layer finer-resolution pixels (red) according to predicted kernels κ, unique to each coarse pixel.

Once radiance is partitioned, we apply the learned kernels K^l_{uvxyt} to the radiance $\bar{L}^l_{xyt}$ in the denoising kernel stage, as described in Section 3.2 and shown in Figure 4 (grey-colored area). Specifically, we perform a 5×5 kernel-splatting operation (Equation 2) as proposed by Gharbi et al. [2019] and Munkberg and Hasselgren [2020].

4.2 Splatting upsampler

If the coarse layers of the pyramid are naively upsampled, for example, using bicubic interpolation, their utility is much limited — they cannot convey information beyond the Nyquist frequency of the given layer. Previous works [Hasselgren et al. 2020; Munkberg and Hasselgren 2020; Vogels et al. 2018] address this problem by using learnable composition (Equation 4), as discussed in Section 3.3. Here, we propose learnable upsampling kernels, which can better preserve edges, are not prone to ringing and blurring, and are convenient to predict using neural networks.

Our upsampling and compositing is performed in a coarse-to-fine order. A finer layer is the sum of the corresponding denoised layer and an upsampled coarser layer:

$$\tilde{L}^4_{xyt} = \hat{L}^4_{xyt}, \tag{6}$$

$$\tilde{L}^l_{xyt} = \hat{L}^l_{xyt} + \sum_{uv} 4\, \tilde{L}^{l+1}_{uvt}\, \sigma_{xy}(\kappa^{l+1}_{uvxyt}) \quad \text{for} \quad l = 3, 2, 1, 0. \tag{7}$$

where κ^{l+1}_{uvxyt} is a 4×4 learned upsampling kernel, $\hat{L}$ are the denoised layers, and $\tilde{L}$ are the composed and layers. σ_{xy} denotes softmax normalization performed. Our filter uses 5 layers, making $l = 4$ our coarsest layer. We ablate this choice in Section 6.3. The result of the operation is $\tilde{L}^0$. Figure 3 shows the geometry of our kernel application step. The prediction of the kernel κ^{l+1}_{uvxyt} will be explained in Section 5.1.

Making the upsampling kernels learnable allows our method to adopt edge-preserving strategies. These strategies are crucial considering our method's effective footprint; at the lowest resolution layer, one pixel corresponds to 16×16 full-resolution pixels, and after applying the 5×5 denoising kernels and successive 4×4 upsampling kernels, our method reaches an effective footprint of 110×110 pixels. The edge-preserving properties of our filter let it mitigate low-frequency noise and transport radiance over large distances. Figure 5 and rightmost column of Figure 6 demonstrate the edge-preserving capabilities of our upsampler.

5 IMPLEMENTATION

Our work mainly focuses on the filtering stage, so we adapt most of our overall pipeline from previous methods [Hasselgren et al. 2020; Işık et al. 2021]. However, we note some changes regarding linear radiance and temporal processing. We propose two weight predictor networks to drive our filter; first, we build a 15 million parameter convolutional network (OursSmall), similar to previous work [Işık et al. 2021; Munkberg and Hasselgren 2020], that we optimise to run in real-time using TensorRT [2] and XLA [Sabne 2020]. Second, we build a 30 million parameter network (OursLarge) from ConvNext [Liu et al. 2022a] blocks in the Restormer [Zamir et al. 2022] configuration, optimised for interactive to offline performance. In the following section, we provide an overview of our pipeline, with Supplementary, Sec. A including all the details of our weight predictor networks and feature engineering.

5.1 Pipeline

Figure 4 presents major components of our denosing pipeline. Its input is per-sample linear radiance $\boldsymbol{L}$ and a vector $\boldsymbol{r}$ including base colour, normal in camera coordinates, and depth. We find that additional features degrade test-time performance. First, following Işık et al. [2021] and Munkberg and Hasselgren [2020] we use a small fully-connected network, g_θ, to encode per-sample data and then we average encodings for each pixel:

$$\boldsymbol{e}_{xyt} = \frac{1}{S}\sum_{s=1}^{S} g_\theta(\boldsymbol{r}_{xyst}), \tag{8}$$

where s denotes the sample index, and S stands for the total number of samples per pixel. Similarly, we average the sample radiance values $\boldsymbol{L}_{xyst}$ for each pixel in the current frame t:

$$\boldsymbol{L}_{xyt} = \frac{1}{S}\sum_{s=1}^{S} \boldsymbol{L}_{xyst}. \tag{9}$$

We compute all features at the primary ray intersection. Consequently, samples visible through specular reflections and transmissions do not contribute to these features making their image more difficult to denoise. Robustly handling such cases is an active, orthogonal area of research that we touch on in Section 7.

We compute the accumulated radiance $\bar{\boldsymbol{L}}$ and embeddings $\bar{\boldsymbol{e}}$ over time (refer to Figure 4). Neural temporal blending weights λ_{xyt} control such accumulation:

$$\bar{\boldsymbol{L}}_{xyt} = (1-\lambda_{xyt})\boldsymbol{L}_{xyt} + \lambda_{xyt}\mathcal{W}_t\bar{\boldsymbol{L}}_{xy,t-1}, \tag{10}$$

$$\bar{\boldsymbol{e}}_{xyt} = (1-\lambda_{xyt})\boldsymbol{e}_{xyt} + \lambda_{xyt}\mathcal{W}_t\bar{\boldsymbol{e}}_{xy,t-1}, \tag{11}$$

where $\mathcal{W}_t$ is the warping operator, which warps and bilinearly interpolates frame $t-1$ to frame t using backwards motion vectors calculated at the centre of each pixel in frame t. We restrict λ_{xyt} to the range $(0,1)$ using sigmoid mapping.

Next, we predict parameters for our denoising filter using a weight predictor network f_θ (refer to Figure 4). As input, we provide the radiance and embeddings from the current frame, and the accumulated radiance and embeddings warped from the previous frame:

$$\lambda_{xyt}, w^l_{xyt}, K^l_{uvxyt}, \kappa^l_{uvxyt}, T_{uvxyt}, \mu_{xyt} = f_\theta(\boldsymbol{L}_{xyt}, \boldsymbol{e}_{xyt}, \mathcal{W}_t\bar{\boldsymbol{L}}_{xy,t-1}, \mathcal{W}_t\bar{\boldsymbol{e}}_{xy,t-1}). \tag{12}$$

Then we apply our denoising filter as described in Equations 5, 2, and 7. Finally, we apply a secondary temporal loop to improve temporal stability:

$$\boldsymbol{O}_{xyt} = (1-\mu_{xyt})\tilde{\boldsymbol{L}}^0_{xyt} + \mu_{xyt}\sum_{uv}(\mathcal{W}_t\boldsymbol{O}_{t-1})_{uv}\cdot\sigma_{xy}(T_{uvxyt}), \tag{13}$$

where μ_{xyt} is a secondary set of blending weights, similar to λ_{xyt}, and T_{uvxyt} is a 5×5 temporal kernel, applied similarly to K^0_{uvxyt}. We adopt this mechanism from NTASD [Hasselgren et al. 2020], as we find it is necessary to achieve good temporal stability; every denoising filter we tested (see Section 6) produced intense flickering without this secondary temporal loop.

5.2 Training procedure

Spatiotemporal processing Many video super-resolution methods rely on recurrent convolutional neural networks as their weight predictors [Liu et al. 2022b]. Inspired by their success, we adopt one of their key components; we use Backpropagation Through Time (BPTT) [Hochreiter and Schmidhuber 1997], taking staggered training iterations over every two consecutive frames. Denoising does not offer delayed rewards; our objective is to output the highest possible quality frame in every iteration. Therefore, the greedy two-iteration BPTT is well suited for training the temporal element of our denoiser.

Previous methods [Hasselgren et al. 2020; Işık et al. 2021] process entire sequences in each training iteration. Backpropagating through such iterations is extremely memory-consuming, limiting these approaches to 8 frame sequences of 128×128 patches. In contrast, we train our model on 64 frame sequences of 256×256 patches, exposing our model to substantially more spatiotemporal information.

Dataset Inspired by Hypersim [Roberts et al. 2021], we leverage Evermotion's Archinteriors and Archexteriors collections to build our production-quality training dataset that exceeds the quality and diversity of datasets used in previous works. Refer to the supplementary Supplementary, Sec. B for more details on our training dataset.

Loss function Our loss function employs perceptual component as proposed by Thomas et al. [2022], complemented with SMAPE following Munkberg and Hasselgren [2020]. Both components contribute also to the temporal loss that additionally involves the warped previous frame, as we detail in the supplementary Supplementary, Sec. C.

Training We implement our pipeline in Tensorflow 2 [Abadi et al. 2016]. We use the Adam [Kingma and Ba 2014] optimiser with a batch size of 8. We begin training at a learning rate of 10^{-4} and exponentially decay the learning rate, halving it every 11 epochs. We train on the full 256×256 patches included in our dataset, augmented with flips and rotations. Training typically converges after 50 epochs, taking 2-3 days on a single NVIDIA A40 GPU for

[2]https://developer.nvidia.com/tensorrt

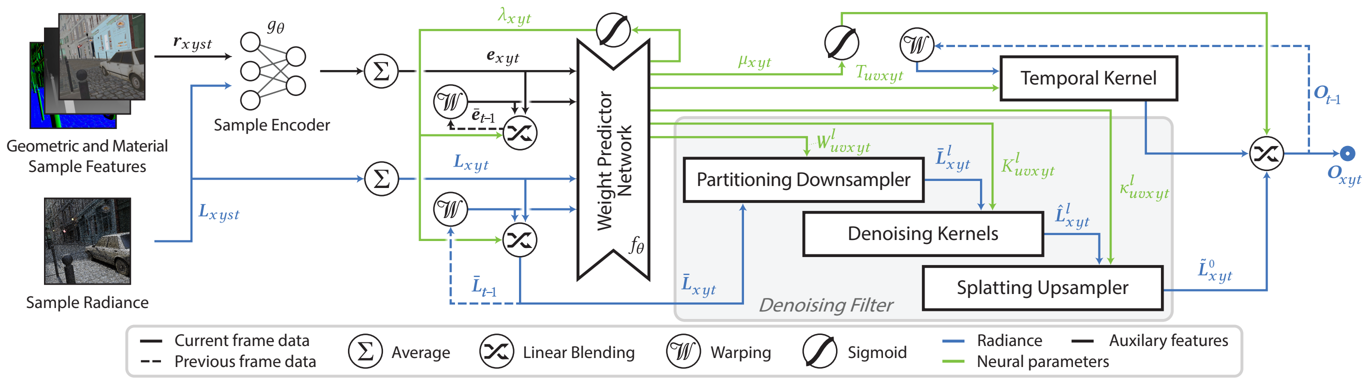

Fig. 4. Processing diagram for our denoiser. Sample radiance and geometric/material features are encoded and averaged over each pixel. The per-pixel radiance and features for the current frame are then combined with those of the previous frame (after motion compensation, Eqs. 10 and 11). The kernels of the per-pixel denoising filters and other parameters are predicted by the *weight predictor network* (Eq. 12). To apply denoising kernels, the radiance is partitioned into a 5-layer pyramid (4.1), then convolved with the kernels (Eq. 2), and finally the layers are combined using adaptive splatting upsampler (4.2). For better temporal stability, the final radiance values are filtered using predicted temporal kernels (Eq. 13).

our small weight predictor, and 4-5 days on 4 A40s for our large weight predictor.

6 EVALUATION

In this section, we compare our pipeline performance to state-of-the-art techniques and run several ablations of our pyramidal denoising.

Our test scenes include *Kitchen*, *Dining-room*, *Bedroom* [Bitterli 2016], *Bistro* [Nvidia 2017], and *Zero-Day* [Winkelmann 2019] (all excluded from the training dataset). As our method and some compared methods rely on temporal information, we allow a 16-frame warm-up phase at the start of each sequence. We tonemap the output frames based on the ACES guidelines [Hill 2022] to match professional animations. We use the Falcor [Kallweit et al. 2022] renderer to render our input samples and 6144 samples per pixel for reference frames. We compute our reference frames as the median of three uncorrelated 2048 spp estimates to suppress fireflies.

6.1 Baseline comparisons

We compare our full pipeline to state-of-the-art denoisers. We render our training dataset using Falcor. We compare against AFGSA [Yu et al. 2021], Intel Open Image Denoise (OIDN) [Intel 2022] version 1.4.3, and Nvidia OptiX AI-accelerated Denoiser (ONND) [Nvidia 2022b] version 7.6, with temporal [Hasselgren et al. 2020] and kernel-based [Bako et al. 2017] extensions enabled. OIDN and AFGSA are high-quality offline denoisers with inference times on the order over 500 milliseconds, while ONND is an optimised interactive denoiser taking roughly 100 milliseconds. We average our per-sample albedo, normal and depth data as each competitor requires per-pixel inputs. We retrain AFGSA and OIDN and use the pre-trained, proprietary ONND included in OptiX. While version 5 of ONND was retrainable, it was based on early work by Chaitanya et al. [2017], surpassed by more recent works [Bako et al. 2017; Xu et al. 2019; Yu et al. 2021], and is unsupported on recent GPUs.

In our comparison, we consider commonly used image quality metrics such as per-pixel PSNR, structure-oriented SSIM and MS-SSIM [Wang et al. 2003], perception-informed FovVideoVDP [3] [Mantiuk et al. 2021], Flip [Andersson et al. 2020], and tPSNR [Banitalebi-Dehkordi et al. 2016]. Commonly used PSNR and SSIM measure error per pixel or in a small sliding window. Meanwhile, MS-SSIM and FovVideoVDP apply filter banks, capturing distortions over large areas. Moreover, FovVideoVDP is a video quality metric that captures temporal distortions, including popping and flickering artefacts perceivable by human observers.

Table 1 presents the obtained results where the OursLarge weight predictor (refer to Section 5) consistently outperforms all competing solutions for all considered metrics. This observation holds for all numbers of input samples per pixel. OursSmall takes 32 milliseconds to denoise a full HD frame (1920×1080), which is three times faster than ONND; OursLarge similar as OIDN requires around 500 milliseconds, while AFGSA 5.5 seconds.

Figure 7 shows a visual comparison of denoising performance for selected frames. We refer the reader to our supplementary material for full-resolution images and videos of our animated sequences.

Table 1. Comparison of the baseline methods. Denoising quality is measured using a number of well-established full-reference metrics. We denote in bold the best quality score and with a gray background the second-best score for a given number of samples per pixel (spp). ↑/↓ indicate that higher/lower scores are better.

	AFGSA		OIDN		OursLarge		ONND		OursSmall	
method	8spp	32spp	8spp	32spp	8spp	32spp	2spp	4spp	2spp	4spp
PSNR↑	29.33	31.58	29.22	30.90	**30.99**	**32.41**	25.39	26.51	**28.54**	**29.70**
SSIM↑	0.877	0.905	0.884	0.900	**0.904**	**0.919**	0.813	0.836	**0.878**	**0.892**
MS-SSIM↑	0.960	0.974	0.963	0.973	**0.976**	**0.980**	0.920	0.940	**0.960**	**0.969**
FoVVDP↑	7.867	8.156	7.878	7.986	**8.456**	**8.505**	7.153	7.554	**7.806**	**8.146**
FliP↓	0.102	0.078	0.113	0.094	**0.085**	**0.076**	0.144	0.126	**0.109**	**0.095**
tPSNR↑	27.00	29.16	27.10	28.60	**29.01**	**30.49**	23.87	24.80	**26.55**	**27.58**

[3] FovVideoVDP v1.1.3, 37.84 ppd, 200 nit peak, 0.5979 nit black, non-foveated

Table 2. Pyramidal filter ablations. The interpretation of columns follows Figure 5. Refer to the figure caption for more details.

method	OursSmall	w/o Upsampler	4 layers	3 layers
PSNR↑	**29.70**	29.47	29.47	29.23
SSIM↑	**0.892**	0.890	0.891	0.888
MS-SSIM↑	**0.969**	0.967	0.968	0.965
FoVVDP↑	**8.146**	8.089	8.098	8.006
FliP↓	**0.095**	0.101	0.101	0.107
tPSNR↑	27.58	27.53	**27.59**	27.47

Table 3. Replacements of our denoising filter. We evaluate alternatives to our proposed denoising filter: weighted sum [Munkberg and Hasselgren 2020], scale-composition [Vogels et al. 2018], and affinity kernels [Işık et al. 2021]. We leave the rest of our Small and Large pipelines, driven by OursSmall and OursLarge respectively, unchanged. The best and second-best scores are marked separately for the Small and Large weight predictors.

	Ours		Weighted sum		Scale-comp.		Affinity	
method	Small	Large	Small	Large	Small	Large	Small	Large
PSNR↑	29.70	**29.98**	29.33	29.37	29.40	29.63	**29.72**	29.03
SSIM↑	**0.892**	**0.895**	0.889	0.891	0.888	0.894	0.889	0.883
MS-SSIM↑	**0.969**	**0.971**	0.967	0.968	0.967	0.969	0.967	0.966
FoVVDP↑	**8.146**	**8.213**	8.075	8.092	8.099	8.130	8.092	8.095
FliP↓	**0.095**	**0.094**	0.103	0.105	0.103	0.105	0.096	0.098
tPSNR↑	27.58	**28.00**	27.42	27.68	27.47	27.98	**27.65**	26.98

6.2 Choice of denoising filter

We compare our denoising filter against several filters proposed in previous works. We swap our denoising filter in our pipeline with implementations of other filters, keeping the rest of our pipeline the same for a fair comparison. We test each filter 4 spp inputs using the OursSmall weight predictor and the OursLarge weight predictor to analyse each filter's scalability. Table 3 shows several metrics averaged for our test scenes.

Although metrics mostly show small differences, our denoising filter scales well with respect to weight predictor size. As only our filter shows the best overall performance with both weight predictors, it is a reliable choice for general Monte Carlo noise filtering.

6.3 Pyramidal filter ablations

To analyse each component's role, we perform ablation studies. We use the OursSmall weight predictor network with 4 spp inputs. We test our denoising filter with 3 and 4 layers instead of 5, we substitute our learnable upsampling stage with bilinear upsampling. Table 2 shows several metrics averaged for our test scenes, and Figure 5 shows side-by-side qualitative comparisons.

Our learnable partitioning downsampler is a crucial component of our denoising filter. The learnable upsampling stage, while having a small impact on metrics, improves the denoising of low frequencies and enhances sharp details. Dropping to 3 or 4 layers also has little effect on metrics but limits the low-frequency denoising capabilities of our filter. The computational cost savings in these cases are negligible; therefore, we favour our full scheme.

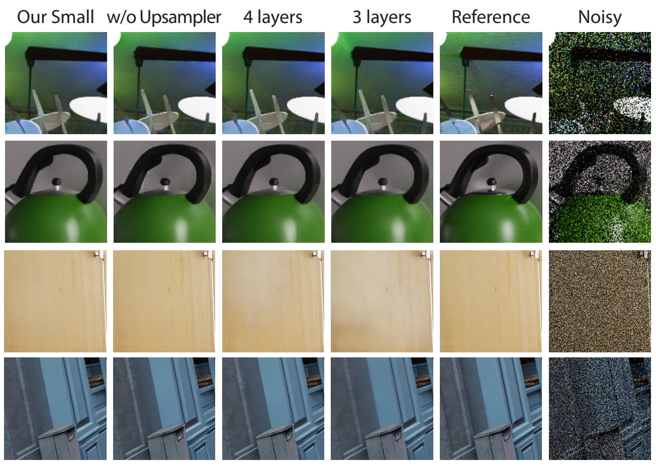

Fig. 5. Pyramidal filter ablations. The OursSmall weight predictor is considered and the role of: (1) learnable upsampling (Section 4.2) that is replaced by a simple bilinear upsampling (the column "w/o Upsampler") and (2) the number of filter layers as shown in Figure 6 (3rd and 4th columns) are considered. Our complete filter uses 5 layers. The last two columns show the reference image and the noisy input. *Bistro* © 2023 Amazon Lumberyard, *Kitchen* © 2023 Jay-Artist, *Bedroom* © 2023 SlykDrako

Important visual consequences of dropping learnable upsampling can be observed in the 1st and 2nd row of Figure 5, where bilinear upsampling leads to ringing and overshoot artifacts. In general, reconstructed edges are more blurry in the second column. Reducing the number of layers might lead to excessive washing out of noisy low-contrast patterns and residual low-frequency noise as can be seen in the 3rd and 4th row. The 5th layer is rarely utilised but does occasionally help resolve artefacts as reflected by metrics.

7 LIMITATIONS AND FUTURE WORK

Our filter is affected by over-blurring and synthetic structural artefacts common to neural networks. When our filter cannot reconstruct a clean, detailed image of the available samples, it inevitably has to tradeoff between these artefacts. While perceptual loss functions can somewhat control the learned strategy, our network occasionally produces dissatisfactory results.

Relying on primary ray intersections diminishes our denoiser's performance in scenes with specular objects. However, path-based denoising [Cho et al. 2021; Lin et al. 2021] and temporally reliable motion vectors [Thomas et al. 2022] are orthogonal research areas with good potential to resolve these issues.

Finally, we do not consider motion blur, depth of field, or other distributed effects—however, the depth-wise layer decomposition proposed by Munkberg and Hasselgren [2020] is orthogonal to our work and could potentially alleviate this limitation.

8 CONCLUSION

In this work, we proposed a novel pyramidal filter that, combined with a lightweight network, achieves close to real-time performance while offering denoising quality comparable to previous offline solutions. Combining the same pyramidal filter with a large network, we obtain substantial improvements over previous work. Visually comparing our denoised frames reveals better reproduction of high-frequency details and a significant reduction of low-frequency noise and artefacts.

Some of the improvements can be attributed to our training strategies that adapt backpropagation through time, handling longer frame sequences and larger patch sizes. Nevertheless, embedding previously considered state-of-the-art filters into our training pipelines and network architectures still diminishes performance.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

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

Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. 2022. Self-Supervised Post-Correction for Monte Carlo Denoising. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 18, 8 pages. https://doi.org/10.1145/3528233.3530730

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph.* 36, 4 (2017), 97–1.

Amin Banitalebi-Dehkordi, Maryam Azimi, Mahsa T. Pourazad, and Panos Nasiopoulos. 2016. Visual saliency aided High Dynamic Range (HDR) video quality metrics. In *2016 IEEE International Conference on Communications Workshops (ICC)*. 486–491. https://doi.org/10.1109/iccw.2016.7503834

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Peter J Burt and Edward H Adelson. 1987. The Laplacian pyramid as a compact image code. In *Readings in computer vision*. Elsevier, 671–679. https://doi.org/10.1016/b978-0-08-051581-6.50065-9

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

In-Young Cho, Yuchi Huo, and Sung-Eui Yoon. 2021. Weakly-Supervised Contrastive Learning in Path Manifold for Monte Carlo Image Reconstruction. *ACM Trans. Graph.* 40, 4, Article 38 (jul 2021), 14 pages. https://doi.org/10.1145/3450626.3459876

Mauricio Delbracio, Pablo Musé, Antoni Buades, Julien Chauvier, Nicholas Phelps, and Jean-Michel Morel. 2014. Boosting Monte Carlo Rendering by Ray Histogram Fusion. *ACM Trans. Graph.* 33, 1, Article 8 (feb 2014), 15 pages. https://doi.org/10.1145/2532708

Hangming Fan, Rui Wang, Yuchi Huo, and Hujun Bao. 2021. Real-time Monte Carlo Denoising with Weight Sharing Kernel Prediction Network. *Computer Graphics Forum* 40, 4 (2021), 15–27. https://doi.org/10.1111/cgf.14338

Siyuan Fu, Yifan Lu, Xiao Hua Zhang, and Ning Xie. 2021. Monte Carlo Denoising with a Sparse Auxiliary Feature Encoder. In *SIGGRAPH Asia 2021 Posters* (Tokyo, Japan) *(SA '21 Posters)*. Association for Computing Machinery, New York, NY, USA, Article 10, 2 pages. https://doi.org/10.1145/3476124.3488631

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. 2019. Sample-Based Monte Carlo Denoising Using a Kernel-Splatting Network. *ACM Trans. Graph.* 38, 4, Article 125 (jul 2019), 12 pages. https://doi.org/10.1145/3306346.3322954

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

J. Hasselgren, J. Munkberg, M. Salvi, A. Patney, and A. Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. *Computer Graphics Forum* 39, 2 (2020), 147–155. https://doi.org/10.1111/cgf.13919

Stephen Hill. 2022. ACES Tone Mapping Operator. https://github.com/TheRealMJP/BakingLab/blob/master/BakingLab/ACES.hlsl.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

Yuchi Huo and Sung-eui Yoon. 2021. A survey on deep learning-based Monte Carlo denoising. *Computational Visual Media* 7, 2 (01 Jun 2021), 169–185. https://doi.org/10.1007/s41095-021-0209-9

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

Intel. 2022. Intel Open Image Denoise. https://www.openimagedenoise.org/.

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty. 2022. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise Multi-Order Feature Regression for Real-Time Path-Tracing Reconstruction. *ACM Trans. Graph.* 38, 5, Article 138 (jun 2019), 14 pages. https://doi.org/10.1145/3269978

P Kozlowski and T Cheblokov. 2021. ReLAX: A denoiser tailored to work with the ReSTIR algorithm. GPU Technology Conference.

Weiheng Lin, Beibei Wang, Jian Yang, Lu Wang, and Ling-Qi Yan. 2021. Path-based Monte Carlo Denoising Using a Three-Scale Neural Network. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 369–381.

Hongying Liu, Zhubo Ruan, Peng Zhao, Chao Dong, Fanhua Shang, Yuanyuan Liu, Linlin Yang, and Radu Timofte. 2022b. Video super-resolution based on deep learning: a comprehensive survey. *Artificial Intelligence Review* (2022), 1–55.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022a. A ConvNet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11966–11976. https://doi.org/10.1109/cvpr52688.2022.01167

Yifan Lu, Siyuan Fu, Xiao Hua Zhang, and Ning Xie. 2021. Denoising Monte Carlo renderings via a multi-scale featured dual-residual GAN. *The Visual Computer* 37, 9 (01 Sep 2021), 2513–2525. https://doi.org/10.1007/s00371-021-02204-4

YiFan Lu, Ning Xie, and Heng Tao Shen. 2020. DMCR-GAN: Adversarial Denoising for Monte Carlo Renderings with Residual Attention Networks and Hierarchical Features Modulation of Auxiliary Buffers. In *SIGGRAPH Asia 2020 Technical Communications* (Virtual Event, Republic of Korea) *(SA '20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/3410700.3425426

Rafał K. Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. 2021. FovVideoVDP: A Visible Difference Predictor for Wide Field-of-View Video. *ACM Trans. Graph.* 40, 4, Article 49 (jul 2021), 19 pages. https://doi.org/10.1145/3450626.3459831

Xiaoxu Meng, Quan Zheng, Amitabh Varshney, Gurprit Singh, and Matthias Zwicker. 2020. Real-time Monte Carlo Denoising with the Neural Bilateral Grid.. In *EGSR (DL)*. 13–24.

Jacob Munkberg and Jon Hasselgren. 2020. Neural Denoising with Layer Embeddings. *Computer Graphics Forum* 39, 4 (2020), 1–12. https://doi.org/10.1111/cgf.14049

Nvidia. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

Nvidia. 2022a. NVIDIA Real-Time Denoisers. https://developer.nvidia.com/rtx/ray-tracing/rt-denoisers.

Nvidia. 2022b. OptiX AI-Accelerated Denoiser. https://developer.nvidia.com/optix-denoiser.

Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. 2021. Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 10892–10902. https://doi.org/10.1109/iccv48922.2021.01073

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Amit Sabne. 2020. Xla: Compiling machine learning for peak performance. (2020).

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In *Proceedings of High Performance Graphics* (Los Angeles, California) *(HPG '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/3105762.3105770

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Manu Mathew Thomas, Gabor Liktor, Christoph Peters, Sungye Kim, Karthik Vaidyanathan, and Angus G. Forbes. 2022. Temporally Stable Real-Time Joint Neural Denoising and Supersampling. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 3, Article 21 (jul 2022), 22 pages. https://doi.org/10.1145/3543870

C. Tomasi and R. Manduchi. 1998. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 839–846. https://doi.org/10.1109/iccv.1998.710815

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Trans. Graph.* 37, 4, Article 124 (jul 2018), 15 pages. https://doi.org/10.1145/3197517.3201388

Z. Wang, E.P. Simoncelli, and A.C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. 1398–1402 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292216

Xinyue Wei, Haozhi Huang, Yujin Shi, Hongliang Yuan, Li Shen, and Jue Wang. 2021. End-to-End Adaptive Monte Carlo Denoising and Super-Resolution. *arXiv preprint arXiv:2108.06915* (2021).

Mike Winkelmann. 2019. Zero-Day, Open Research Content Archive (ORCA). https://developer.nvidia.com/orca/beeple-zero-day

Bing Xu, Junfei Zhang, Rui Wang, Kun Xu, Yong-Liang Yang, Chuan Li, and Rui Tang. 2019. Adversarial Monte Carlo denoising with conditioned auxiliary feature modulation. *ACM Trans. Graph.* 38, 6 (2019), 224–1.

Jiaqi Yu, Yongwei Nie, Chengjiang Long, Wenju Xu, Qing Zhang, and Guiqing Li. 2021. Monte Carlo Denoising via Auxiliary Feature Guided Self-Attention. *ACM Trans. Graph.* 40, 6, Article 273 (dec 2021), 13 pages. https://doi.org/10.1145/3478513.3480565

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming–Hsuan Yang. 2022. Restormer: Efficient Transformer for High-Resolution Image Restoration. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5718–5729. https://doi.org/10.1109/cvpr52688.2022.00564

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

Dmitry Zhdan. 2021. *ReBLUR: A Hierarchical Recurrent Denoiser.* Apress, Berkeley, CA, 823–844. https://doi.org/10.1007/978-1-4842-7185-8_49

Shaokun Zheng, Fengshi Zheng, Kun Xu, and Ling-Qi Yan. 2021. Ensemble denoising for Monte Carlo renderings. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–17.

M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum* 34, 2 (2015), 667–681. https://doi.org/10.1111/cgf.12592

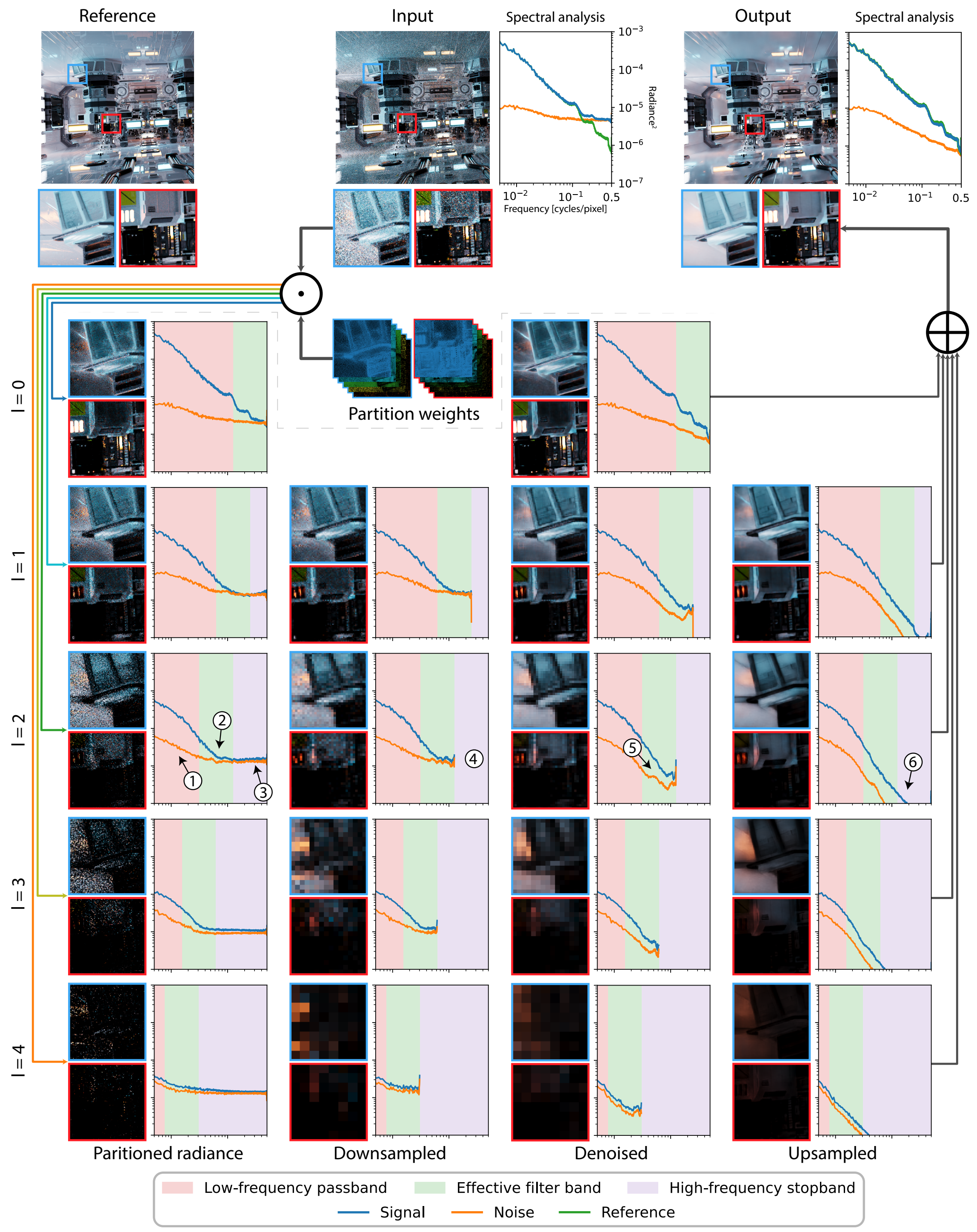

Fig. 6. Fourier analysis of our filter. The plots show the power spectra of the entire image at consecutive processing stages. The blue line represents the signal and the orange line represents noise (calculated by subtracting the reference image). The range of values on both axes is identical in all the plots. Note that the rightmost column does not show $\tilde{L}^l$, but each layer upsampled separately. The predicted partitions contain insignificant low-frequency noise below the denoising kernel's effective band in each layer ①, a mix of useful radiance and noise in the effective band ②, and primarily white noise in the higher frequencies ③, later removed by downsampling ④. The noise filtering stage reduces the noise in the effective band ⑤, and the upsampling stage reconstructs the small amount of useful high-frequency radiance lost during downsampling ⑥. *Zero-Day* © 2023 Mike Winkelmann

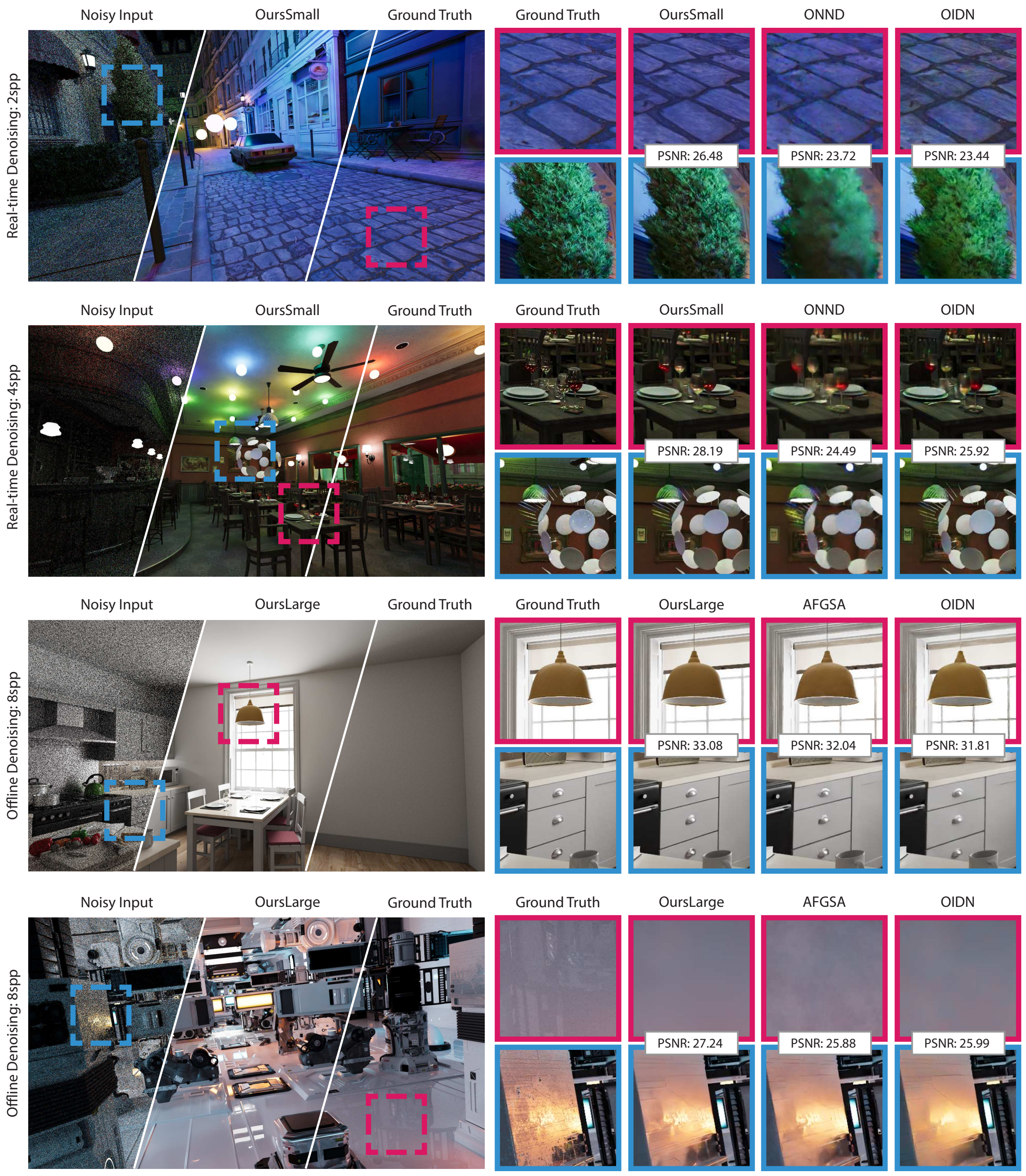

Fig. 7. Our denoiser compared to state-of-the-art methods on four scenes: *Bistro1*, *Bistro3*, *Kitchen*, and *Zero-Day*. Our method was able to remove noise effectively, as seen in the pink inset of the *Bistro1* scene. Other methods (ONND and OIDN) retained some of high and low-frequency noise. Additionally, our method was able to preserve the contrast of the tree in the blue inset of the *Bistro1* scene. In the *Bistro3* scene, our method was able to preserve high frequencies of caustics (as seen in the pink inset) and provided better appearance of thin objects, without leading to extensive blurring (as in the case of ONND) or aliasing and partial disappearing of the cutlery (as in the case of OIDN). In the *Kitchen* scene, our method was able to preserve geometrical details of the window frame while other methods failed in this task. Additionally, the lampshade denoised by AFGSA was highly contaminated by artifacts and OIDN changed its color. In the *Zero-Day* scene, all methods struggled with retrieving reflections in glossy surfaces (as seen in the pink inset). Our method produced a smooth surface, which was plausible and pleasing for the eye, while AFGSA and OIDN introduced low-frequency noise. The blue insets also shows that our method preserves more details in the reflections than other methods. *Bistro* © 2023 Amazon Lumberyard, *Kitchen* © 2023 Jay-Artist, *Zero-Day* © 2023 Mike Winkelmann