Interactive Monte Carlo Denoising using Affinity of Neural Features

MUSTAFA İŞIK, Technical University of Munich, Germany
KRISHNA MULLIA, Adobe, California, USA
MATTHEW FISHER, Adobe, California, USA
JONATHAN EISENMANN, Adobe, California, USA
MICHAËL GHARBI, Adobe, California, USA

High-quality denoising of Monte Carlo low-sample renderings remains a critical challenge for practical interactive ray tracing. We present a new learning-based denoiser that achieves state-of-the-art quality and runs at interactive rates. Our model processes individual path-traced samples with a lightweight neural network to extract per-pixel feature vectors. The rest of our pipeline operates in pixel space. We define a novel pairwise affinity over the features in a pixel neighborhood, from which we assemble dilated spatial kernels to filter the noisy radiance. Our denoiser is temporally stable thanks to two mechanisms. First, we keep a running average of the noisy radiance and intermediate features, using a per-pixel recursive filter with learned weights. Second, we use a small temporal kernel based on the pairwise affinity between features of consecutive frames. Our experiments show our new affinities lead to higher quality outputs than techniques with comparable computational costs, and better high-frequency details than kernel-predicting approaches. Our model matches or outperforms state-of-the-art offline denoisers in the low-sample count regime (2–8 samples per pixel), and runs at interactive frame rates at 1080p resolution.

1 INTRODUCTION

Rendering noise-free path-traced images at interactive frame rates remains an elusive objective despite the advent of hardware raytracing accelerators and GPU raytracing APIs [Khronos 2020; Microsoft 2021; Parker et al. 2013]. The fastest modern GPUs can only afford to trace a handful of samples per pixel in real time for typical display resolutions, leading to severe noise in the renderings.

State-of-the-art Monte Carlo denoisers use large kernel-predicting neural networks [Bako et al. 2017; Gharbi et al. 2019; Kettunen et al. 2019; Vogels et al. 2018; Xu et al. 2019]. Their computational cost is rarely an issue for off-line rendering, since raytracing dominates the overall latency, but it is impractical for interactive applications. The fastest denoisers typically use hand-designed filters [Mara et al. 2017] or much more compact neural networks [Chaitanya et al. 2017; Meng et al. 2020] that sacrifice quality for performance. Denoising artifacts are often amplified in video animation where care must be taken to create temporally stable results that do not contain high-frequency artifacts between adjacent frames. The shortcomings of interactive denoisers are particularly salient around soft shadows, complex global illumination, glossy reflections and refractive materials — precisely where raytracing has the most appeal.

ACM Reference Format:

Fig. 1. Our denoiser targets images from interactive path-tracers with low sample per pixel (spp) budget (a). It runs at interactive rates and produces temporally-stable, high-quality results (c), with better details and fewer artifacts than state-of-the-art off-line denoisers (b), and significantly outperforms interactive denoisers (not shown). Timings are recorded using PyTorch implementations on a GeForce RTX 2080 Ti graphics card. See Table 1 for our optimized implementation that runs interactively. The reference (d) was rendered with a much higher computational budget. The resolution is 1024 × 1024.


Additional Key Words and Phrases: Monte Carlo denoising, deep learning

Authors’ addresses: Mustafa Işık, Technical University of Munich, Munich, Germany, m.isik@tum.de; Krishna Mullia, Adobe, San Jose, California, USA, mulliala@adobe.com; Matthew Fisher, Adobe, San Francisco, California, USA, matfish@adobe.com; Jonathan Eisenmann, Adobe, San Francisco, California, USA, j@adobe.com; Michaël Gharbi, Adobe, San Francisco, California, USA, mgharbi@adobe.com.

© 2021 Association for Computing Machinery.
This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in ACM Transactions on Graphics, https://doi.org/10.1145/3450626.3459793.
We present a new denoiser for low-sample, interactive ray-tracing applications that directly operates on the path-traced samples. Our model is a lightweight neural network (~94k parameters) that summarizes rich per-sample information into low-dimensional per-pixel feature vectors. We define a novel pairwise affinity over these features, which we use to weight the contributions of neighboring per-pixel radiance values in a local weighted average filtering step. These new affinity-based kernels lead to better denoising performance compared to kernel-predicting techniques [Bako et al. 2017; Gharbi et al. 2019; Munkberg and Hasselgren 2020; Vogels et al. 2018], especially when reconstructing fine details, or in high-frequency light transport configurations.

We evaluate our denoiser on a diverse set of static renderings and fly-through animations. We show it produces much cleaner outputs and lower numerical error than state-of-the-art interactive denoisers with comparable runtime cost. For low-sample count renderings, our model even matches or outperforms the state-of-the-art off-line denoisers, for a fraction of the computational cost (Figure 1). In short, our contributions are the following:

- A state-of-the-art denoiser for low-sample count Monte Carlo rendering that runs at interactive rates and outperforms the quality of even powerful off-line baselines in the low-sample count regime.
- A novel filtering algorithm that uses pairwise affinity of per-pixel deep features learned from the raw path-tracing samples to learn iteratively-applied 2D dilated kernels.
- A new temporal aggregation mechanism which uses the same pairwise affinity to significantly improve the temporal stability of Monte Carlo denoising.

2 RELATED WORK

With the advent of powerful hardware accelerators, ray-tracing has become viable for real-time rendering. But, because today’s GPUs can only trace a few samples at interactive rates, fast high-quality denoisers remain critical. We discuss the most recent related work, and refer the reader to Zwicker et al. [2015] for an overview of denoising techniques.


Machine-Learning based image-space methods. Recent work has used neural networks for denoising, avoiding the need to manually tune filter parameters for each scene. Kalantari et al. [2015] use a fully-connected neural network and predict the parameters of a cross-bilateral filter. Bako et al. [2017] showed that a convolutional network that predicts per-pixel filtering kernels is more robust and easier to train than models that directly output the denoised radiance. Vogels et al. [2018] extended this technique with multiscale filtering [Delbracio et al. 2014], temporal aggregation and adaptive sampling. Xu et al. [2019] use a generative adversarial training procedure to improve image quality. Kuznetsov et al. [2018] predict sampling density maps for adaptive sampling. Our denoiser also uses kernels, but instead of predicting the full-rank kernels, our network outputs feature vectors for each pixel from which we define pairwise affinities. This makes the learning problem simpler, which leads to a more parsimonious model, suitable for interactive use.

Sample-based image-space methods. These methods strive to fully exploit per-sample information to maximize denoising quality. Sen and Darabi [2012] use per-sample cross-bilateral filters to filter the noise caused by random parameters in the Monte Carlo integration. Gharbi et al. [2019] use a neural network for sample-based denoising. They predict splatting kernels for each sample, and take special care in ensuring their network is invariant to permutation of the samples within a pixel. Their method scales poorly for interactive rendering settings because the network evaluation cost grows linearly with the number of samples. Munkberg and Hasselgren [2020] mitigate this issue by partitioning and averaging the samples into a fixed number of layers, filtering the layers independently, and compositing them front to back. Our method also processes individual samples but applies kernels to per-pixel averages not samples, which significantly reduces the runtime cost.

Interactive denoising. Denoisers designed for interactive applications focus on efficiency and temporal stability. Early techniques used hierarchical wavelet filters and edge-stopping functions based on auxiliary render buffers [Dammertz et al. 2010] or edge-aware guided filters [Bauszat et al. 2015, 2011; He et al. 2010]. Schied et al. [2017] used a fixed-weight recursive temporal smoothing filter for temporal stability, which Schied et al. [2018] improved by using adaptive temporal accumulation weights. We also use adaptive temporal accumulation in our temporal smoothing strategy, with weights predicted by our network, but complement it with a temporal kernel computed with our new affinity. Koskela et al. [2019] brings regression-based denoising to the real-time domain. Chaitanya et al. [2017] achieve high-quality, temporally-stable results at interactive frame rates using a recurrent U-net [Ronneberger et al. 2015]. Hasselgren et al. [2020] extend the work of Kuznetsov et al. [2018] to the temporal domain and obtain better temporal stability than Chaitanya et al. [2017] by using motion vectors to warp the previous prediction. Meng et al. [2020] splat the noisy radiance into a bilateral grid. Slicing in the grid along a learned guidance map gives the denoised output. They also use a recursive filter with fixed weight for temporal stability. We discuss the similarities between the neural bilateral grid and our affinities in Section 3.7. Our adaptive temporal accumulation improves denoising performance, in particular around occlusions and specular reflections. Our affinity-based temporal kernel further improves temporal stability, removing low-frequency noise and enlarging the filter’s footprint in time.
3 DENOISING WITH LEARNED PAIRWISE AFFINITY

The key novelty in our approach is a pairwise affinity measure on learned neural features, from which we derive spatio-temporal filtering kernels to denoise the frames of an animation. We work with samples from a pathtracer [Kajiya 1986] with render buffer features \( r_{xy,t} \) (§ 3.1), where \((x,y)\) are the pixel coordinates, \(s \in \{1,\ldots,S\}\) are the sample indices within a pixel, and \(t \in \{1,\ldots,T\}\) are the frame indices. In interactive applications the number of samples per pixel is typically low \((S < 8)\), making temporal stability challenging. Our model addresses this via two mechanisms: first, we use recursive filters with adaptive, per-pixel parameters to maintain running averages of neural embeddings and noisy radiance; second, we apply our denoising kernels to pairs of frames.

We visualize our method in Figure 2. We start by independently extracting per-sample embeddings \( e_{xy,t} \) from each sample’s render buffer data using a small pointwise network (§ 3.2). This is the only operation on the raw samples. We use a temporal recursive filter over the embeddings with adaptive learned parameters to propagate information between frames and then average the sample embeddings over all samples in each pixel. From these per-pixel embeddings, we predict a vector of parameters for each pixel using a convolutional network (§ 3.3); convolutions enable information-sharing between neighboring pixels. We use the per-pixel parameters to construct filtering kernels at each pixel (§ 3.4), which we apply to the noisy radiance to obtain the final, denoised output (§ 3.5.2). We decompose the kernels into a sequence of \( K \) = 3 dilated 2D kernels [Dammertz et al. 2010; Holschneider et al. 1990; Yu and Koltun 2016] with increasing dilation, applied sequentially (§ 3.5.1), followed by a temporal kernel (§ 3.5.2). Dilation enables larger spatial footprints, which helps remove low-frequency noise, but has a minimal impact on performance relative to using large dense kernels. The temporal kernel is built from the affinity between features of the current and previous frames; it smooths the output over time.

3.1 Input path-traced sample features

Previous work showed that working with samples rather than pixel averages can improve denoising [Gharbi et al. 2019; Hachisuka et al. 2008; Lehtinen et al. 2011, 2012], but can incur a significant computational overhead (see Section 5.4). Similarly to [Munkberg and Hasselgren 2020], we adopt a hybrid strategy: we use per-sample information to compute filtering weights, but the filters operate on the integrated (box-filtered) pixel radiance \( L_{xy,t} \), not the samples.

During rendering we store, for each sample, a vector of 18 render buffer features \( r_{xy,t} \), which we describe next. First, we store the radiance, split into diffuse and specular components [Bako et al. 2017], both tonemapped with \( x \mapsto \log(1 + x) \) to compress their dynamic range. This tonemapping is only used for the sample input features passed to the network; our denoising kernels operate on the linear radiance. We treat materials with roughness below a fixed threshold (0.1 linear roughness in PBRT-v3) as specular. Second, we use the normal \((3)\), and depth \((1)\) as geometric features. And third, we characterize materials by their roughness \((1)\), albedo \((3)\), and 4 binary variables: emissive — indicates whether the path sampled hits emissive surface, metallic — differentiates between dielectric and conductors, transmissive — distinguishes between reflections and refractions, and specular-bounce — which is ‘true’ if first vertex on the camera path is a specular interaction. Except the radiance, all features are computed at the first non specular interaction.

3.2 Mapping samples to per-pixel features

Given our interactivity constraints, we keep per-sample processing to a minimum. We independently embed the samples using a shallow fully-connected network, then reduce the embeddings to per-pixel summary statistics by averaging over the sample dimension:

\[
e_{xy,t} = \frac{1}{S} \sum_{s=1}^{S} FC(r_{xy,t}). \tag{1}
\]

This fully-connected network uses leaky ReLU activations. It has 3 layers with 32 channels each. In Section 5.5.2, we show this embedding strategy outperforms the simple first and second-order statistics of raw render buffers used in previous work [Bako et al. 2017; Vogels et al. 2018] for low-spp renderings (2–8 spp).

3.3 Spatio-temporal feature propagation

We next process the per-pixel embeddings with a lightweight U-net [Ronneberger et al. 2015]. This network takes as input the embeddings of both the current and previous frames and produces, for each pixel, a set of feature vectors \( f_{xy,t}^{k} \in \mathbb{R}^{d} \), and scalars \( a_{xy,t}^{k} \) and \( c_{xy,t}^{k} \), for \( k = 1, \ldots, K \). We use these parameters and features to compute \( K \) dilated spatial kernels, which we apply sequentially to...
denoise the frame (§ 3.4). Specifically, we compute the spatial kernels by calculating distances between affinity features \( f_{xyt}^k \), scaled by the bandwidth parameters \( a_{xyt}^k \), and \( c_{xyt} \) is the kernel’s central weight. The network also outputs two additional scalars for our temporal filtering mechanisms: \( b_{xyt} \) is a bandwidth parameter modulating the feature affinity between successive frames in a temporal kernel; and \( \lambda_{xyt} \) is the parameter of an exponential moving average filter that accumulates the pixel embeddings and noisy radiance temporally (Eq. (3) and (5)). Our implementation uses \( K = 3 \). Formally, the U-net’s output is given by:

\[
\left( f_{xyt}^k, a_{xyt}^k, c_{xyt} \right), b_{xyt}, \lambda_{xyt} = \text{UNet} \left( \mathbf{e}_{xyt}, \mathbf{W}_t \mathbf{e}_{xy, t-1} \right).
\]

(2)

\( \mathbf{W}_t \) is a warping operator that reprojects frame \( t-1 \) to frame \( t \) with nearest neighbor interpolation using the geometric flow at the primary intersection point computed by the path tracer [Schied et al. 2017]. And, \( \mathbf{e}_{xy, t-1} \) is a temporal accumulation of the pixel embeddings defined by:

\[
\begin{align*}
\mathbf{e}_{xy0} &= \mathbf{e}_{xy0}, \\
\mathbf{e}_{xyt} &= (1 - \lambda_{xyt}) \mathbf{e}_{xyt} + \lambda_{xyt} \mathbf{W}_t \mathbf{e}_{xy, t-1}.
\end{align*}
\]

(3)

This temporal accumulation of the embeddings helps make temporally consistent predictions, compared to simply passing the previous frame’s embeddings (i.e., replacing \( \mathbf{e}_{xy, t-1} \) with \( \mathbf{e}_{xy, t-1} \) in Equation (2)). Figure 5 shows the per-pixel blending weights can selectively mask out \( \lambda_{xyt} = 0 \) the warped embedding when they are inaccurate due, e.g., to occlusions/disocclusions, or moving reflections not captured in the geometric velocity vectors. We use a sigmoid activation to ensure \( \lambda_{xyt} \in [0, 1] \). We discuss alternative temporal accumulation strategies in Section 5.5.4. Our U-net [Ronneberger et al. 2015] has 5 scales. Its structure is given by:

\[
\begin{array}{c}
c64\times64 \rightarrow c64\times64 \rightarrow c64\times64 \rightarrow c80\times80 \\
c96×96 \rightarrow c80\times80 \rightarrow c64\times64 \rightarrow c64\times64 \rightarrow c32\times32,
\end{array}
\]

where \( c \) is a convolution with \( k \) channels and \( 3 \times 3 \) kernels, \( d \) is a \( 2 \times 2 \) maxpooling operator, \( u \) is \( 2 \times 2 \) bilinear upsampling. Every convolution except the last has a leaky ReLU activation. We use concatenation in the skip connections.

3.4 Spatial kernels from pairwise affinity

We define our spatial filtering kernels as an affinity over the features \( f_{xyt}^k \) produced by the U-net:

\[
\begin{align*}
\mathbf{w}_{xynt}^k &= \begin{cases} 
c_{xyt}^k \\
\exp \left( -c_{xyt}^k \| f_{xyt}^k - f_{yut}^k \|^2 \right) 
\end{cases} \text{ if } x = u \text{ and } y = v, \\
\lambda_{xyt} \exp \left( -c_{xyt}^k \| f_{xyt}^k - f_{yut}^k \|^2 \right) \text{ otherwise.}
\end{align*}
\]

(4)

In the network, we use a sigmoid activation to ensure the center weight \( c_{xyt}^k \) is in \([0, 1]\) and a squaring function to ensure \( d_{xyt}^k \geq 0 \). The latter variable acts as a bandwidth parameter that adaptively tunes a pixel’s sensitivity to feature differences with its neighbors. With higher values, small differences quickly make the weight go to zero, reducing the kernel’s effective footprint to a single pixel. As \( a_{xyt}^k \rightarrow 0 \), the filter becomes progressively insensitive to features discrepancy, and turns into a box filter. We visualize a few kernels and their bandwidth parameter in Figure 4.

Our network predicts central weights adaptively for each pixel. Setting \( c_{xyt}^k = 1 \), the center pixel contributes fully to the output. With this weight set to \( c_{xyt}^k = 0 \), the network can suppress the bright outliers that often appear in low-sample renderings when high-energy, low-probability paths are sampled. Figure 3 illustrates this property. Wu et al. [2013] propose a similar strategy to adapt the central weight of non-local mean filters per pixel, but use ad-hoc closed-form expressions rather than these parameters. Previous Monte Carlo denoising methods [Kalantari et al. 2015; Meng et al. 2020] usually suppress outliers in a pre- or post-processing step.

3.5 Temporally-stable kernel-based denoising

Prior to filtering, we accumulate the noisy radiance \( L_{xyt} \) over time. This has a denoising effect in areas where the motion vector-based warping provides useful correspondences — and improves overall temporal stability:

\[
\begin{align*}
L_{xy0} &= L_{xy0}, \\
L_{xyt} &= (1 - \lambda_{xyt}) L_{xyt} + \lambda_{xyt} \mathbf{W}_t L_{xyt}.
\end{align*}
\]

(5)
We use the same temporal blending weights \( \lambda_{x,y,t} \) as for the accumulated embeddings in Equation (3). We use this temporal accumulation as the input for our kernel-filtering step. We obtain the denoised image by sequentially filtering the noisy frame \( L_{x,y,t} \) with \( K = 3 \) dilated spatial filters, and combining the result with the previous denoised image using a temporal kernel.

### 3.5.1 Spatial filtering with a sequence of dilated filters
We initialize \( L_{x,y,t}^{(0)} = L_{x,y,t} \) and apply the first \( K - 1 \) kernels from Equation (4) sequentially using the following expression:

\[
L_{x,y,t}^{(k)} = \sum_{u,v} w_{x,y,u,v}^{k} l_{x,y,u,v}^{(k-1)}.
\]

The sums run over a square neighborhood with dilation \( 2^{k-1} \) (see Figure 2 for an illustration), and \( \epsilon = 10^{-15} \) is a small constant to avoid numerical instability. Every time the radius is filtered, the noise statistics change, so we use different affinities at each step (i.e., different \( f_{x,y,t}, c_{x,y,t}^{0} \) and \( a_{x,y,t}^{0} \)). Our implementation uses kernels with \( 13 \times 13 \) taps, so the kernel with the largest dilation factor \( 2^{K-1} = 4 \) has an effective footprint of \((13 - 1) \times 4 + 1 = 49 \) pixels in each dimension. Composing the sequential filtering steps gives an overall pixel footprint of \( 85 \times 85 \).

### 3.5.2 Temporal filtering
In our final filtering step, we apply a temporal kernel that measures the affinity between the features of the current and previous frame, after warping, similar to Equation (4):

\[
\omega_{x,y,u,v,t} = \exp \left( -\beta_{x,y,t} \| f_{x,y,t}^{K} - W_{t} f_{x,y,u,v,t-1}^{K} \|_{2}^{2} \right).
\]

Here again, we ensure the bandwidth parameter \( \beta_{x,y,t} \) is non-negative using a squaring function. This temporal filter uses the same features \( f_{x,y,t}^{K} \) as the last spatial kernel (Equation (4) with \( k = K \)). Note that \( w_{x,y,u,v}^{k} \) are the spatial kernels and \( \omega_{x,y,u,v,t} \) is the temporal kernel.

We obtain our final denoised image using this temporal kernel and the last spatial kernel as follows:

\[
O_{x,y,t} = \frac{\sum_{u,v} W_{y} W_{x} a_{x,y,u,v}^{K} \omega_{x,y,u,v,t} + \sum_{u',v'} \epsilon \omega_{x,y,u,v'} W_{t} O_{x,y,u',v',t-1}}{\epsilon + \sum_{u,v} W_{x} W_{y} + \sum_{u',v'} \omega_{x,y,u',v'}}.
\]

The \( u, v \) run over a square neighborhood with dilation \( 2^{K-1} \), and \( u', v' \) run over a smaller neighborhood, with dilation 1. We also use a \( 13 \times 13 \) window for the temporal kernel. Unlike Equation (6), the \( K \)-th spatial kernel is jointly normalized with the temporal filter. We use \( \omega_{x,y,u,v,t} = 0 \) at \( t = 0 \).

### 3.6 Comparison to kernel-predicting networks
As shown by Munkberg and Hasselgren [2020]; Vogels et al. [2018], kernel-predicting methods [Bako et al. 2017] require deeper and larger networks to fully benefit from larger kernels. This is because the number and complexity of pairwise interactions between pixels under the kernel increases with kernel size. In contrast, our method does not require such an increase in network "capacity" since we predict per-pixel features with a closed-form affinity, rather than full-rank kernels. In Section 5.5.6, we show our affinity-based kernels are significantly better at reconstructing high-frequency details than gather kernels [Bako et al. 2017]. Additionally, since we compute the kernels \( a \) \textit{posteriori} from the learned pixel affinities, we can dynamically change the kernel size at runtime, without retraining. Kernel-predicting networks are limited to the fixed kernel size they were trained with. This property can be used as a dynamic quality–performance trade-off control (§ 5.4).

We analyze the effect of kernel size in Section 5.5.3.

### 3.7 Relation to the neural bilateral grid
Meng et al. [2020] (NBG) use a neural bilateral grid [Gharbi et al. 2017] for denoising, thus approximating a bilateral filter. They use a 3D grid: the first 2 dimensions are the screen-space coordinates, and the third is a learned scalar parameter, which would correspond to the range filter in a traditional bilateral filter [Tomasi and Manduchi 1998]. This is similar to using feature vectors \( \{ f_{x,y,t}^{K} \} \) with dimension \( d = 1 \) in our affinities (we use \( d = 8 \)). However, we show this dimension parameter is critical in Section 5.5.1. Setting \( d = 1 \) leads to oversmoothing (Figure 9) and severely reduces quality (Table 4).
4 DATASET AND TRAINING PROCEDURE

We train our denoiser to reconstruct clean, temporally stable animations in which a camera flies through an otherwise static scene. Each sequence is 8-frame long and rendered at 256 × 256 resolution. For model selection, we also render a validation with 821 scenes, using the same random generator. Our training objective and procedure are detailed in Section 4.2 and 4.3, respectively.

4.1 Dataset generation

We adapted the scene generator from Gharbi et al. [2019]. Our goal is to cover a wide variety of light transport scenarios, so that our denoiser generalizes to new scenes, by randomizing the scene content and parameters.

Geometry. We use 6 rooms from Tungsten [Bitterli 2016] as background. For each data sample, we choose a room at random and add 50 objects, randomly chosen from ShapeNet [Chang et al. 2015], applying a random rigid transformation to each object. We take the camera position into account for object placement in order to minimize the chance of a completely occluded rendering. The randomness in our generator can still lead to some completely black renderings which we remove in a post-processing pass.

Lights and materials. Next, each material is randomized by selecting one of the PBRT-v3 materials (e.g., substrate, matte, plastic, disney). To increase the diversity of appearance, we randomly select texture maps from the Describable Textures Dataset [Cimpoi et al. 2014]. We randomly scale each texture for variety in the frequency characteristics. In addition to randomizing the energy of each light source in the background scene, we also insert 2 point light sources with random positions and energies in order to maximize the diversity of the lighting and shadow patterns.

Camera and rendering. Each scene is rendered using a perspective pinhole camera model in PBRT-v3 [Pharr et al. 2016], with a modified path tracer integrator. We randomized the camera’s field of view to vary the frequency content of our renderings. As pointed out by [Gharbi et al. 2019], PBRT scales ray differentials (to reduce aliasing) based on the sample count. This causes discrepancies in appearance between the low-sample input and ground truth, so we use a constant scaling factor instead. We render the noisy inputs using 8 samples per pixels, storing the samples individually so we can randomize the number of samples at the training time. We render the ground truth at 2048 spp, changing the random seed so it is not correlated with the input.

Animation and motion vectors. The animations in our training set are simple translations or rotations of the camera. We randomize the speed of the camera and render 8 consecutive frames per scene. We altered PBRT-v3 to generate per-sample motion vectors. We test whether the normal and world position in the current and previous frame agree after warping using the motion vectors, and discard motion vectors that fail this test.

4.2 Losses

We train our denoiser to reconstruct clean, temporally stable animations by minimizing a reconstruction loss, a temporal stability loss, and a regularization on the affinity parameters.

\[
\mathcal{L} = \mathcal{L}_{\text{recons}} + 0.25 \cdot \mathcal{L}_{\text{temporal}} + 10^{-3} \cdot \mathcal{L}_{\text{reg}}.
\]

We use the Symmetric Mean Absolute Percentage Error over the linear radiance (SMAPE) [Vogels et al. 2018] as our reconstruction loss:

\[
\mathcal{L}_{\text{recons}} = \text{SMAPE}(O, O^\star),
\]

where \( O^\star \) is the ground truth image. SMAPE is defined for two tensors \( A \) and \( B \) as:

\[
\text{SMAPE}(A, B) = \frac{1}{3} \frac{1}{1 + \|A_{xyt} - B_{xyt}\|_1} \left( \frac{\|A_{xyt}\|_1}{1 + \|B_{xyt}\|_1} + 3 \right).
\]

The expectation \( \mathbb{E} \) is taken over pixels and frames, \( \| \|_1 \) is the L1 norm, and the factor 3 accounts for color channels.

To penalize inconsistency between successive frames, we use the temporal loss:

\[
\mathcal{L}_{\text{temporal}} = \text{SMAPE}(\partial_t O, \partial_t O^\star),
\]

with \( \partial_t \) the finite difference operator along the time dimension [Chaitanya et al. 2017].

Finally, our regularization is an L2 penalty over the kernels’ bandwidth parameters:

\[
\mathcal{L}_{\text{reg}} = \sum_k \| \partial_k x y t \|_2^2 + \| \partial_k x y t \|_2^2.
\]

We found that without this regularizer, the scale ambiguity between these parameters and the magnitude of the features \( \partial_k x y t \) made the training numerically unstable.

4.3 Training details

Our denoiser is implemented in PyTorch [Paszke et al. 2019] and trained to convergence using the ADAM optimizer [Kingma and Ba 2014] with learning rate set to \( 10^{-4} \) and batch size 4; all other optimizer settings are kept at their default values. We augment our dataset with horizontal and vertical flips, 90 degrees rotation and random cropping of size 128 × 128. Training typically converges after 750 epochs, which takes roughly 5 days for our 8-frames animation dataset on a GeForce RTX 2080 Ti GPU and an Intel Xeon W-2235 CPU with 12 cores. Training a single-frame variant of our model (i.e., with no temporal filtering) takes about 2 days. We multiply the learning rate by 0.75 every 150 epochs. For the single-frame variant, we decrease the learning rate once to \( 5 \times 10^{-5} \) when the validation loss plateaus, around epoch 600. During training, we select the number of samples \( S \in \{1, 2, 4\} \) in a round-robin fashion at the beginning of each epoch (we use \( S \in \{2, 4, 8\} \) for the single-image variant). We keep the model with the best validation loss at 4 spp (resp. 8 spp for the single-image task). We use 13 × 13 kernels during training but, because our kernels are computed from pairwise affinities, we can change the kernel size at test time.

5 RESULTS

We evaluate our denoiser on a test dataset of 26 publicly available scenes [Bitterli 2016; Pharr et al. 2016] converted and rendered with PBRT-v3 [Pharr et al. 2016]. We compare to state-of-the-art solutions for both interactive and off-line denoising on 26 static renderings and 6 videos. The videos are 120 frames long fly through animations,
We compare to three state-of-the-art off-line denoisers, KPCN [Bako et al. 2020], which takes HDR radiance, normals and albedo as input, and produces HDR outputs. Unless otherwise specified we use the same input features for our method, NDLE, and SBMC. Since NDLE uses the U-Net architecture of our method in its guide net and include a slower but higher-quality variant, NBG-large, which is a high-quality but slower model; and two interactive methods, NBG [Meng et al. 2020]; Schied et al. [2017]. Without learnable blending weights, the model oversmoothes specular highlights (b).

Fig. 6. Benefit of learning $\lambda_{x,y,t}$. Learning per-pixel blending weights in our temporal accumulation helps discard inconsistent reprojections in the warps, thus recovering better details in occluded areas and sharp specular reflections (c). We show an ablation of our model that fixes $\lambda_{x,y,t} = 0.8$ for all pixels, as is done in Koskela et al. [2019]; Meng et al. [2020]; Schied et al. [2017]. Without learnable blending weights, the model oversmoothes specular highlights (b).

rendered at 1920 × 1024 resolution. We render the ground truth images at 4096 spp, and the videos at 2048 spp. We use bidirectional path tracing [Lafortune and Willems 1993; Veach and Guibas 1995] for some scenes with complex light transport. The noisy inputs always use uni-directional path tracing.

Our algorithm outperforms previous work in both the single frame (§ 5.2) and animation setups (§ 5.3). It has better temporal stability and runs at interactive framerates (§ 5.4). We analyze the impact of our design choices in an ablation study (§ 5.5). Full resolution static and video results can be found in the supplemental material, together with additional quantitative metrics.

5.1 Baselines and metrics

We compare to three state-of-the-art off-line denoisers, KPCN [Bako et al. 2017], SBMC [Gharbi et al. 2019], and NDLE [Munkberg and Hasselgren 2020]; and two interactive methods, NBG [Meng et al. 2020] and OptiX (ONND), adapted from Chaitanya et al. [2017].

We implemented the two variants described by Munkberg and Hasselgren [2020]: NDLE is a high-quality but slower model; and NDLE-ms is an interactive, multi-scale variant. Unlike the original, our implementation of NDLE-ms is not optimized, so we exclude it from our speed comparison. We reimplemented NBG in PyTorch following the open-source implementation provided by the authors, and include a slower but higher-quality variant, NBG-large, which uses the U-Net architecture of our method in its guide net. We followed Bako et al. [2017] for our implementation of KPCN. We adapted the open-source code from Gharbi et al. [2019] for SBMC, adjusting the number of filters in the fully-connected and convolutional layers so that the total number of parameters roughly match NDLE (~ 18 million). We used the official OptiX 7.2 SDK for ONND, which takes HDR radiance, normals and albedo as input, and produces HDR outputs. Unless otherwise specified we use the same input features for our method, NDLE, and SBMC. Since NDLE uses

17×17 kernels, we adopted the same kernel size for SBMC and KPCN, rather than their original 21 × 21 size. More details on the training and architecture of the baselines can be found in supplemental.

5.1.1 Evaluation metrics. For both single images and videos, we compare denoisers using the PSNR on the tonemapped radiance:

$$\text{PSNR}(O, O^*) = -10 \log_{10} \left( \frac{1}{3} \mathbb{E}_{x,y,t} \| \tau O_{x,y,t} - \tau O^*_{x,y,t} \|_2^2 \right).$$  \hspace{1cm} (14)$$

The expectation $\mathbb{E}$ is over pixels and frames, the factor 3 accounts for color channels, and $\tau$ is a tonemapping and gamma correction operator [Reinhard et al. 2002], defined pointwise as

$$\tau : x \in \mathbb{R}^+ \mapsto \left( \frac{x}{1 + x} \right)^{17} .$$  \hspace{1cm} (15)$$

For animations, we also compute the Temporal Relative Mean Absolute Error (TRMAE), which we define as:

$$\text{TRMAE}(O, O^*) = \frac{1}{3} \mathbb{E}_{x,y,t} \frac{\| d\tau O_{x,y,t} - d\tau O^*_{x,y,t} \|_1}{\| d\tau O^*_{x,y,t} \|_1 + \epsilon} .$$  \hspace{1cm} (16)$$

where $\epsilon = 0.01$ prevents the numerical instability. We find this metric to correlate well with perceptual evaluation as a temporal stability measure in our evaluations. We refer the reader to the supplemental material to see other metrics applied on the temporal gradients.

5.2 Single-frame comparison

We first evaluate our denoiser on static images, configuring all methods to process frames independently, one at a time. We disable our temporal accumulation (Eq. (3) and (5)) and temporal filtering (Eq. (8)) mechanisms, and call this single-frame variant ours-single. Our algorithm consistently outperforms recent interactive denoisers. It matches or outperforms off-line denoisers, for a fraction of the cost. Table 2 summarizes our evaluation, and Figure 14 shows a few examples. Ours-K1 uses a single 17 × 17 kernel like the kernel-predicting baselines. In Section 5.5.6, we discuss the advantage of using several, iteratively applied kernels instead of using a single large one. Full resolution images can be found in the supplemental.

5.3 Video denoising

Table 3 shows a quantitative comparison of our full model, with temporal filtering activated, against prior methods for video denoising. As in the single-frame experiment, our method achieves higher PSNR and lower TRMAE than prior methods. Ours-wpred, ours-heur, ours-ntk and ours-nar are ablations discussed in Section 5.5.4 and Section 5.5.5.

5.4 Implementation and performance

We optimized our model for inference, using cuDNN 8.0 to implement the fully-connected sample embedding network and the U-net, and exploiting half-precision floating-point operations for speed. We implemented other modules using custom CUDA kernels. Because cuDNN cannot fuse convolution-bias-activation operations unless the activation is ReLU, we replaced leaky ReLU with standard ReLU in our pretrained model then fine-tuned for 150 epochs. We
applied the same procedure to all models analyzed in Section 5.5.4 and Section 5.5.5. Table 1 shows the runtime cost of each component of the optimized implementation of our full, temporally-stable model. We analyze the effect of the kernel size on the final quality and performance of this optimized implementation in Figure 7, observing that 11 × 11 or 13 × 13 kernels give the best compromise.

For fair comparison with unoptimized baselines, we benchmarked our single-frame variant (ours-single) and the baselines directly in PyTorch implementations. We report throughputs in megapixels per second (log-scale). Top-right is better. PSNR results are averaged over 26 static test images. Table 1 shows a performance breakdown of our optimized implementation.

5.5 Model ablations

In this section we examine: the effect of changing the kernel sizes at test time, how the affinity features’ dimensionality impacts quality, alternative strategies for temporal stability, and the advantage of sample embeddings over mean/variance reduction of the input.

5.5.1 Dimension of the affinity features. The dimension of our affinity features directly affects the complexity of pairwise relationships our model can capture between pixels under the kernel. A low dimension d can only represent the simplest relationships (e.g., "identical color"). Table 4 summarizes our results. Using 1-dimensional

Table 1. Performance breakdown. Timings of our cuDNN implementation for 4spp inputs with 1920 × 1024 resolution using a GeForce RTX 2080 Ti graphics card. We use 13 × 13 spatial and temporal kernels.

<table>
<thead>
<tr>
<th>Components</th>
<th>FC</th>
<th>U-Net</th>
<th>Kernels</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (ms)</td>
<td>5.76</td>
<td>21.07</td>
<td>22.72</td>
<td>3.19</td>
<td>52.74</td>
</tr>
</tbody>
</table>

Table 2. Quantitative evaluation on static single-frame denoising. When processing a single image, our method is competitive with state-of-the-art off-line denoisers that have a significantly higher computational cost (first group), and it consistently outperforms interactive denoisers (second group). Starred method (*) were designed to account for temporal stability. We highlight the first and second best result in each column. Higher PSNR (resp. lower SMAPE) is better.

<table>
<thead>
<tr>
<th>method</th>
<th>2spp</th>
<th>4spp</th>
<th>8spp</th>
<th>16spp</th>
<th>32spp</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPCN</td>
<td>27.95</td>
<td>30.86</td>
<td>32.79</td>
<td>33.90</td>
<td>34.54</td>
</tr>
<tr>
<td>SBMC</td>
<td>28.97</td>
<td>32.33</td>
<td>33.97</td>
<td>34.57</td>
<td>34.44</td>
</tr>
<tr>
<td>NDLE</td>
<td>30.08</td>
<td>32.20</td>
<td>33.63</td>
<td>34.45</td>
<td>34.73</td>
</tr>
<tr>
<td>NDLE-ms</td>
<td>29.47</td>
<td>31.81</td>
<td>33.14</td>
<td>33.87</td>
<td>34.12</td>
</tr>
<tr>
<td>ONND</td>
<td>29.40</td>
<td>30.62</td>
<td>31.79</td>
<td>32.88</td>
<td>33.91</td>
</tr>
<tr>
<td>NBG</td>
<td>26.04</td>
<td>27.58</td>
<td>28.52</td>
<td>29.11</td>
<td>29.42</td>
</tr>
<tr>
<td>NBG*</td>
<td>28.58</td>
<td>30.09</td>
<td>31.20</td>
<td>31.94</td>
<td>32.58</td>
</tr>
<tr>
<td>ours-K1</td>
<td>29.13</td>
<td>31.87</td>
<td>33.28</td>
<td>34.11</td>
<td>34.42</td>
</tr>
<tr>
<td>ours-single</td>
<td>30.93</td>
<td>32.67</td>
<td>33.66</td>
<td>34.34</td>
<td>34.81</td>
</tr>
</tbody>
</table>

Table 3. Quantitative evaluation on video denoising. The first three groups are: single-frame off-line denoisers, interactive denoisers, and ablation (§5.2, §5.5.4 and §5.5.5) on our technique. Starred method (*) were not designed to handle videos, in particular they make no attempt at temporal stability, hence the relatively higher TRMAE values. We highlight the first and second best result in each column. Higher PSNR (resp. lower SMAPE) is better.

<table>
<thead>
<tr>
<th>method</th>
<th>2spp</th>
<th>4spp</th>
<th>8spp</th>
<th>16spp</th>
<th>32spp</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPCN</td>
<td>30.98</td>
<td>33.05</td>
<td>34.37</td>
<td>0.764</td>
<td>0.618</td>
</tr>
<tr>
<td>KPCN*</td>
<td>31.02</td>
<td>32.91</td>
<td>33.96</td>
<td>0.806</td>
<td>0.662</td>
</tr>
<tr>
<td>ONND</td>
<td>30.66</td>
<td>31.74</td>
<td>32.74</td>
<td>1.988</td>
<td>1.693</td>
</tr>
<tr>
<td>NBG</td>
<td>26.67</td>
<td>28.48</td>
<td>29.56</td>
<td>2.058</td>
<td>1.695</td>
</tr>
<tr>
<td>NBG-large</td>
<td>29.31</td>
<td>30.67</td>
<td>31.60</td>
<td>3.020</td>
<td>1.695</td>
</tr>
<tr>
<td>ours-K1*</td>
<td>30.48</td>
<td>32.64</td>
<td>33.97</td>
<td>0.803</td>
<td>0.650</td>
</tr>
<tr>
<td>ours-single*</td>
<td>32.17</td>
<td>33.67</td>
<td>34.54</td>
<td>0.640</td>
<td>0.556</td>
</tr>
<tr>
<td>ours-wpred</td>
<td>32.18</td>
<td>33.48</td>
<td>34.33</td>
<td>0.513</td>
<td>0.482</td>
</tr>
<tr>
<td>ours-heur</td>
<td>32.05</td>
<td>32.89</td>
<td>33.33</td>
<td>0.511</td>
<td>0.485</td>
</tr>
<tr>
<td>ours-ntk</td>
<td>32.55</td>
<td>33.79</td>
<td>34.44</td>
<td>0.502</td>
<td>0.474</td>
</tr>
<tr>
<td>ours-nar</td>
<td>32.37</td>
<td>33.76</td>
<td>34.56</td>
<td>0.508</td>
<td>0.473</td>
</tr>
<tr>
<td>ours</td>
<td>32.34</td>
<td>33.93</td>
<td>34.81</td>
<td>0.517</td>
<td>0.467</td>
</tr>
</tbody>
</table>

Fig. 7. Quality of various kernel sizes vs. performance (optimized). On the left, we set the spatial and temporal kernel size to the same value from 7 × 7 to 19 × 19. We run our denoiser on a GeForce RTX 2080 Ti, and average the timings over 120 4spp frames of the Bedroom scene with resolution 1920 × 1024. On the right, we fix the spatial kernel size to 13 × 13, and vary the temporal kernel size from 1 × 1 to 19 × 19. PSNR (higher is better) and TRMAE (lower is better) values are averaged over 6 test videos.

Fig. 8. PSNR vs. throughput (unoptimized). With an unoptimized implementation, our method already achieves an appealing runtime/quality profile. Except ONND, for which we use the official release API, all the timings reported here use unoptimized PyTorch implementations. We report throughput in megapixels per second (log-scale). Top-right is better. PSNR results are averaged over 26 static test images. Table 1 shows a performance breakdown of our optimized implementation.
Interactive Monte Carlo Denoising using Affinity of Neural Features

Fig. 9. Impact of the affinity feature dimension $d$ on quality. On the top row, we show how dimensionality affects the reconstruction of the fine details in the albedo. Using fewer features in the learned affinity, our denoiser with $d = 1$ cannot reconstruct the texture of the wooden wall and leaves high-frequency artifacts (c). When $d = 2$, our method cannot effectively distinguish the wooden block from the wall behind it, the color of the wooden block smears onto the wall (d). As we increase the dimension of the affinity features to $d = 4$ (e) and $d = 8$ (f), our reconstruction of high-frequency albedo improves significantly. The bottom row shows a similar behavior for high frequency geometric details. Using $d = 1$ leaves jaggy artifacts (c) whereas with $d = 2$, most of the details on the red carpet are blurred (d). As the dimension increases to $d = 4$ (e) and $d = 8$ (f), our model reconstructs most of the details. For these examples, we did not observe further improvements for $d$ larger than 8.

Table 4. Ablation on the affinity features dimension $d$. Increasing the dimension of the affinity features $f_{كته}^*$ gives more discriminative power to our kernels, improving quality. This evaluation is done using ours-single model on static images. See Figure 9 for qualitative differences.

<table>
<thead>
<tr>
<th>feature size</th>
<th>2spp</th>
<th>4spp</th>
<th>8spp</th>
<th>16spp</th>
<th>32spp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 1$</td>
<td>26.65</td>
<td>27.77</td>
<td>28.25</td>
<td>28.44</td>
<td>28.55</td>
</tr>
<tr>
<td>$d = 2$</td>
<td>30.23</td>
<td>31.98</td>
<td>32.97</td>
<td>33.61</td>
<td>34.04</td>
</tr>
<tr>
<td>$d = 4$</td>
<td>30.73</td>
<td>32.43</td>
<td>33.40</td>
<td>34.06</td>
<td>34.53</td>
</tr>
<tr>
<td>$d = 8$ (ours)</td>
<td>30.93</td>
<td>32.68</td>
<td>33.67</td>
<td>34.34</td>
<td>34.81</td>
</tr>
<tr>
<td>$d = 12$</td>
<td>30.97</td>
<td>32.62</td>
<td>33.64</td>
<td>34.35</td>
<td>34.86</td>
</tr>
<tr>
<td>$d = 16$</td>
<td>30.11</td>
<td>32.63</td>
<td>33.76</td>
<td>34.44</td>
<td>34.81</td>
</tr>
</tbody>
</table>

Fig. 10. Samples embeddings vs. summary statistics. Mapping the raw sample feature buffers (b, one map shown) to learned embeddings prior to averaging outperforms simple statistics, especially when using few samples. The simple statistics ablation leaves high-frequency artifacts (c) when the auxiliary buffers are noisy due to undersampled geometry (b, top) or undersampled albedo (b, bottom). Our method (d) with sample encoding distinguishes the noise from the data, producing clean results.
means of temporal filtering mechanisms, which effectively lets the method use a larger spatial footprint.

5.5.4 Temporal accumulation. We show the importance of temporally accumulating features (Eq. (3)) and radiance (Eq. (5)), by disabling both mechanisms, but keeping the temporal kernel. In this ablation, ours-wpred, we replace the warped previous sample embeddings in the U-net’s inputs (\( W_\ell \bar{e}_{x,y,t−1} \)) with the warped previous prediction. This strategy is inspired by Hasselgren et al. [2020]. In a second ablation, ours-heur, we replace our learned per-pixel temporal blending weight with a constant \( \lambda_{xy} = 0.8 \) [Koskela et al. 2019; Meng et al. 2020; Schied et al. 2017]. This ablation cannot properly reconstruct view-dependent effects (Fig. 6). Table 3 shows both ablations reduce quality (up to ~ 1.5dB PSNR), suggesting that our temporal accumulation scheme with adaptive weights is key.

5.5.5 Temporal filtering. In ours-ntk, we disable the temporal kernel (§3.5.2) but keep the temporal accumulation mechanism. Unlike our full model, this ablation cannot completely remove low-frequency noise (Fig. 11). Table 3 shows the temporal kernel of our full method is more beneficial at larger sampler counts (> 2spp), i.e., the model benefits less from the previous denoised frame when noise is severe.

Using both a temporal kernel and temporal radiance accumulation (Eq. (5)) might seem redundant. In ours-nar, we disable radiance accumulation to show this is not the case. The pixel embeddings are still accumulated temporally, so the U-Net can still produce reliable features, benefiting from an effectively higher sample count. Radiance accumulation is more effective at higher sample counts (Table 3).

5.5.6 Affinity-based kernels vs. gather kernels. To better understand their properties, we run an ablation where our affinity-based kernels are replaced with full-rank gather kernels [Bako et al. 2017] (labeled gather). Furthermore, we run this comparison for \( K = 1 \) and \( K = 3 \) kernels to illustrate the advantages of our sequential dilated filtering approach. Except for the last convolution, whose channel count depends on the kernel size, the ablation’s architecture is identical to our full model. This analysis is summarized in Table 5.

In the single kernel case (\( K = 1 \) with no dilation in Section 3.3), gather kernels numerically outperform our affinities. However, both approaches create block artifacts (images in supplemental) because the kernels are too small to be effective, suggesting a larger spatial context is needed. Iteratively filtering with \( K = 3 \) dilated kernels, yields a much larger spatial support, from which our affinity-based kernels benefit significantly, outperforming gather kernels. The improvements are much more modest for gather kernels.

The advantage of our affinities becomes clearer with large kernels, where modeling complex pairwise interactions between pixels is challenging. Overall, affinity-based kernels are significantly better at reconstructing high-frequency details (Figure 12). Conversely, visual inspection shows that gather kernels are better at removing low-frequency noise at low sample counts, when the kernel size is small (the top-left region of the Table 5 is dominated by gather). Still, together with our multi-scale dilation and temporal filtering strategies, our affinity-based kernels can filter low-frequency noise better.

Finally, the memory access pattern of gather kernels make them more efficient to apply to the noisy image. However, they require a final convolution layer with significantly more parameters (quadratic in the kernel size) than our affinities (constant size), so that, overall, both approaches have roughly the same execution time. Gather kernels require more memory as the kernel size increases. This tilts the runtime balance in favor of our affinity-based kernels. The supplemental material shows a numerical comparison on memory and runtime performance, and additional visual comparisons.

5.6 Limitations

We visualize some of the limitations of our denoiser in Figure 13. First, our training dataset does not contain renderings with distributed effects, such as motion blur and depth of field which limits the capability of our method to generalize to such scenes. Second, we have difficulty with light transport scenarios where highly contributing paths are sampled with very low probabilities during the rendering process. In future work, it is natural to explore connecting the denoiser with a neural importance sampling network [Müller et al. 2019]. Third, we generate motion vectors only at the first intersection point for each pixel during the rendering process, which...
We have presented a novel method for denoising Monte Carlo renderings at interactive speeds with quality on-par with off-line denoisers. We use an efficient network to aggregate relevant per-sample features into temporally-stable per-pixel features. Pairwise affinity between these features are used to predict dilated 2D kernels that are iteratively applied to the input radiance to produce the final denoised result. We show our model can spatially adjust the kernels to effectively smooth out noise and preserve fine details. We further demonstrate how to incorporate the spatially-warped content from previous frames to produce a temporally consistent result.

ACKNOWLEDGMENTS

We thank Sebastian Weiß for his valuable feedback, Jacob Munkberg and Josef Stumpfegger for their help with the previous work, and the anonymous Siggraph reviewers for their precious suggestions.

REFERENCES


Fig. 14. **Qualitative comparison on single-frame denoising.** Our method (a) and (f) produces noticeably cleaner outputs than previous interactive techniques (d,e), especially around specular materials. It often resolves fine details better than state-of-the-art offline methods (c).