Denoising Deep Monte Carlo Renderings

Delio Vicini[1,2] David Adler[1] Jan Novák[2] Fabrice Rousselle[2] Brent Burley[1]

[1]Walt Disney Animation Studios
[2]Disney Research

(a) KAKAMORA *(b) Noisy (c) Denoised (d) Composited*

Figure 1: *A production scene (a) rendered as a deep noisy image (b), denoised using our method (c), and composited (d) with a deep volume. Our algorithm removes a significant amount of the input Monte Carlo noise, while preserving the depth separation of the deep image. © Disney*

Abstract

We present a novel algorithm to denoise deep Monte Carlo renderings, in which pixels contain multiple color values, each for a different range of depths. Deep images are a more expressive representation of the scene than conventional flat images. However, since each depth bin receives only a fraction of the flat pixel's samples, denoising the bins is harder due to the less accurate mean and variance estimates. Furthermore, deep images lack a regular structure in depth—the number of depth bins and their depth ranges vary across pixels. This prevents a straightforward application of patch-based distance metrics frequently used to improve the robustness of existing denoising filters. We address these constraints by combining a flat image-space Non-Local Means filter operating on pixel colors with a deep *cross-bilateral filter operating on auxiliary features (albedo, normal, etc.). Our approach significantly reduces noise in deep images while preserving their structure. To our best knowledge, our algorithm is the first to enable efficient deep-compositing workflows with denoised Monte Carlo renderings. We demonstrate the performance of our filter on a range of scenes highlighting the challenges and advantages of denoising deep images.*

CCS Concepts

•***Computing methodologies*** → ***Rendering; Image processing;***

1. Introduction

Deep compositing has recently become a commonly used technique for generating complex production images from multiple sources [Sey14]. Unlike "flat" images used in traditional compositing, which require manual rotoscoping if the layers are not distinctly separated into a front to back order, *deep images* [KBSH13] may be composited in arbitrary order. To achieve correct depth interleaving, each *deep pixel* contains multiple values representing contributions from objects at different camera depths. Having multiple depth values available to the compositor enables additional capabilities such as inserting atmosphere [HHHF12] or surfaces, isolating color corrections and lighting adjustments to particular depths, or even repositioning elements without re-rendering. See Figure 2 for a deep-compositing example.

Flat or partially-deep compositing can introduce artifacts as shown in Figure 3. Similar problems would occur when depth or world position cues color adjustments. A common solution is to partition the scene into distinct sets of objects and multiple flat render passes. But high depth complexity (Figure 3a) could require many render passes (especially if camera motion affects depth relationships), which is inefficient and complicates the compositing process. These problems are solved by deep compositing.

Figure 2: *A deep-compositing example: (top to bottom) original image, denoised deep image, deep bins visualized as 3D points, final deep composite with depth-based color and lighting adjustments, and inserted atmosphere and volume elements. © Disney*

With the advent of physically-based rendering, images—both flat and deep—produced by modern renderers are often plagued with residual noise. While sophisticated sampling strategies for simulating light transport can reduce the noise, avoiding it completely inside the renderer is nearly impossible due to the slow convergence rate of Monte Carlo integration. Removing the noise *a-posteriori* is thus often the only viable solution, yet an appealing one as recent denoising algorithms produce high-quality, clean outputs at a negligible cost compared to increasing sample counts. However, despite the wide adoption of deep images across the movie industry, none of the recent denoising algorithms can preserve depth decompositions as all available denoisers operate on flat data only. Artists are thus left with an unfortunate set of choices: render surfaces and volumes in a single render and give up compositing flexibility, use surface "hold-outs" in the volume render and vice-versa and face matte line artifacts, or forego denoising and face potentially prohibitive render times.

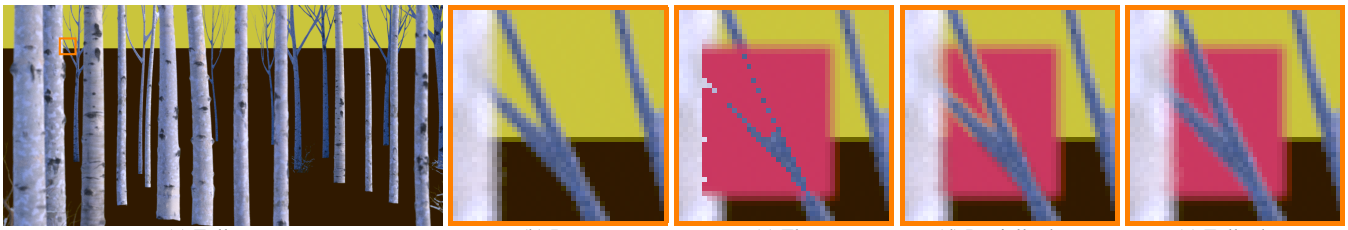
The seemingly obvious option of avoiding the need to denoise deep images is to run the denoiser *after* compositing on flattened images. Unfortunately, this suffers from several practical problems:

- Denoising after compositing can introduce significant bias depending on the level of noise in the input (see Figure 4). Such non-predictive behavior would effectively force the compositing artists to work on high-quality renders at all times, preventing them from fast iterations when rerendering is involved.
- Artists would be forced to implement compositing operations on noisy data, making certain operations (e.g. rotoscoping) significantly more challenging.
- Post-compositing denoising may blur added noise-free elements, such as adjustment masks (like the rectangle in Figure 4), background image plates, or text.
- Blur operations during compositing induce correlation between pixels, which can prevent the denoiser from removing fireflies if run after compositing.
- Variance estimates and auxiliary feature buffers (surface normal, albedo, etc.) would have to be tracked through individual compositing operations. Even if the associated theoretical challenges were solved, such tracking would still require substantial engineering and computational and memory overhead.

Hence it is important to denoise rendered images *before* compositing and preserve their depth decomposition. In contrast to the two image-space dimensions, the depth dimension is unstructured and there are typically no correspondences between depth bins of neighboring pixels. Applying existing patch-based approaches is thus challenging as treating depth as merely an extra dimension and computing distances between 3D neighborhoods, as done in temporal filtering, is not possible. Furthermore, denoising deep images requires balancing the conflicting interests of versatility and denoising robustness: A finer binning may allow for a wider range of compositing operations, but hinders per-bin statistics and increases memory requirements. A coarser binning provides more accurate estimates of color mean and variance in each bin, but reduces artistic flexibility.

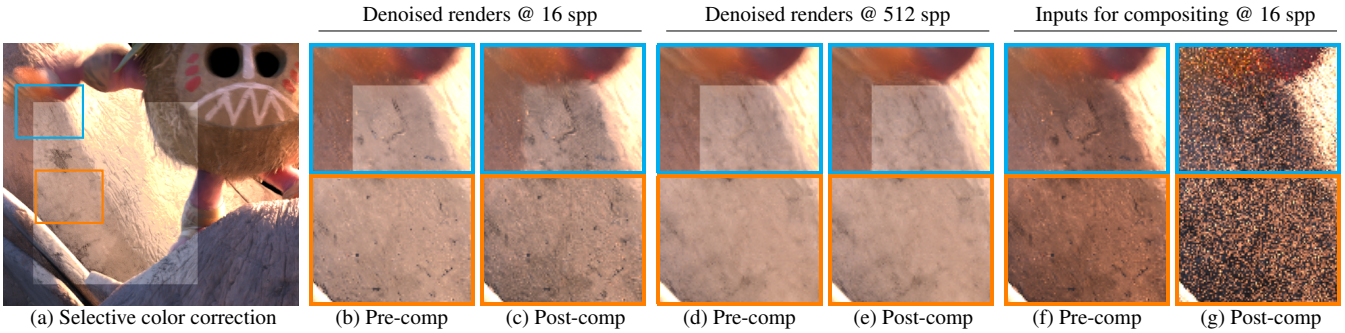
In this paper, we present a denoising framework that respects and leverages deep features in order to denoise deep Monte Carlo renderings. Our goal is to preserve the full expressiveness of the deep image structure through denoising, while matching the robustness of filters operating on flattened data—where denoising is better behaved thanks to the higher signal-to-noise ratio. To this end, we combine an image-space Non-Local Means filtering of pixel colors with a deep cross-bilateral filter operating on auxiliary features, such as albedo and normal.

To demonstrate the robustness of our filter, we compare its denoised deep output to nearly noise-free, reference deep renders. We also analyze potential regressions of denoising deep images: we compare to a standard denoiser run on flattened images while avoiding any compositing operations; these can potentially invali-



(a) Full scene (b) Input (c) Flat (d) Partially deep (e) Fully deep

Figure 3: *Why deep compositing is needed: In this render (a), (b), the trees are blurred by depth of field and rendered in front of distant background objects. The composite inserts a red object between trees and the background, cued by depth. A flat composite from a single render (c) incorrectly covers part of the trees with the inserted object since the render blends the tree and background depths into a single value. A partially-deep render has deep alpha and depth but uses the same flat color for all depths. While this does not require rendering and denoising deep colors, it causes compositing artifacts (d): the background "leaks" through the inserted object because the render blends the tree and background colors into a single value. A fully-deep render and composite (e) produces the correct result without artifacts. © Disney*



(a) Selective color correction (b) Pre-comp (c) Post-comp (d) Pre-comp (e) Post-comp (f) Pre-comp (g) Post-comp

Figure 4: *Denoising before versus after compositing: we applied a non-linear color correction to a rectangular region in a flat render of the* KAKAMORA *scene in a compositing program and compared two denoising workflows: denoising before compositing (pre-comp), and denoising after compositing (post-comp) at 16 and 512 samples/pixel. Denoising after compositing may produce undesired artifacts whose magnitude depends on the sampling rate, e.g. blurring boundaries of elements produced in compositing and brightness changes like the darkening in (c). Such unpredictable behavior is unacceptable in production environments. Furthermore, denoising after compositing forces the artists to work with images that suffer from noise (g). Denoising before compositing does not suffer from these drawbacks but requires preserving the depth decomposition of rendered images, which is the main focus of this paper. © Disney*

date the variance and feature buffers. We demonstrate that our deep denoiser yields comparable performance to a flat-image denoiser despite operating on unstructured data with a lower signal-to-noise ratio. Lastly, we discuss limitations and suggest alternative solutions that would not only preserve the depth decomposition but also yield higher denoising quality.

2. Related Work

Deep Images. The concept of retaining multiple color values resolved along the line of sight is also known as *multi-layer z-buffers* [Max96], *layered depth images* [SGHS98], and *deep G-buffers* [MMNL16], and was applied to re-rendering scenes from novel viewpoints [Max96, SGHS98] and fast computation of indirect illumination [NSS10, MMNL16]. To reduce the memory storage, Duan and Li [DL03] proposed to separately compress color and depth information and analyzed the performance of numerous coding tools.

Pivotal work by Lokovic and Veach introduced Deep Shadow Maps [LV00] which store a multiple-sample transmittance function per pixel in order to achieve shadowing through volumes as well as motion-blurred or translucent surfaces. Almost immediately, Deep Shadow Maps gained wide usage in production rendering for shadowing but also for transferring arbitrary deep attributes between render passes. Hillman later introduced deep compositing [HWW10] which was then standardized through additions to the OpenEXR 2.0 image library [KBSH13] receiving widespread support from rendering and compositing software.

To keep deep images to manageable sizes, Lokovic and Veach included the idea of sample compression where samples are collapsed if a subsequent pixel sample falls within an error tolerance of the extrapolated transmittance from the previous samples. Even so, files sizes could often be prohibitive in images with high depth complexity such as hair or fur rendering. Subsequent work further approximated the transmittance and reduced the complexity using parallel planes [KN01], per-pixel k-means clustering [MKBR04], or using a fixed number of depth samples per pixel [YK08] suitable for GPU rendering. To further reduce file size and also improve the quality of blending separately rendered hard surfaces, Egstad et al. [EDL15] collapsed samples belonging to the same geometry and material, but retained the spatial layout of the samples within each pixel using a sub-pixel mask.

Denoising MC Renderings. Over the last decade, denoising of Monte Carlo renderings has been a very active field of research with a plethora of publications too large to be reviewed here; for an overview see the state-of-the-art report by Zwicker et al. [ZJL*15]. Since we aim to denoise high-end production images, we only discuss prior art relevant to denoising of arbitrary combinations of light-transport effects or capable of preserving the information encoded in deep images.

Early work by Rushmeier and Ward [RW94] introduced the idea of using image-space non-linear filters to denoise Monte Carlo renderings, and McCool [McC99] later proposed to guide the filter using noise-free auxiliary buffers encoding the scene information (surface orientation, albedo and depth). While those techniques were still designed around relatively simple scenes, later work by Overbeck et al. [ODR09] demonstrated that complex rendering scenarios could also be handled. The focus then gradually shifted towards increasing the robustness of the denoising process, notably by better handling noise in auxiliary buffers [SD12, LWC12, RMZ13, MIGMM17], leveraging regression models of increasingly high order [BEM11, MCY14, BRM*16, MMMG16], or leveraging additional statistics (e.g. histograms and covariance matrices) [DMB*14, BB17]. Our work builds upon a regression framework used in state-of-the-art image-space denoisers, but extends it to operate on the unstructured data of deep images.

Several recent works proposed to leverage neural networks to infer locally optimal parameters for regression models [KBS15], reconstruct a noise-free image using predicted kernels [BVM*17, MBC*17], or produce the image directly [CKS*17]. While deep learning will undoubtedly offset denoising performance in the future, the acquisition of sufficiently large training sets (there are currently none with deep images), the increased memory requirements due to the deep structure, and their relatively poor generalization currently permits deployment only in big production houses.

The vast majority of image-space methods are *pixel*-based, i.e. they operate on "flat" images, where all samples are first aggregated in a single per-pixel average. While efficient in storage and oblivious to scene complexity, flat images prevent artists from employing deep-compositing techniques. A subset of denoising approaches are *sample*-based, i.e. they operate on individual samples, and therefore theoretically allow producing inputs suitable for deep-compositing pipelines. In practice though, none of these techniques is sufficiently general for production use cases. Light field reconstructions techniques [LAC*11, LALD12] handle only a subset of light transport cases, generic methods [HJW*08, SD12] scale poorly to the high sampling rates needed for high-end production, and the recent work of Bauszat et al. [BEJM15] handles only indirect illumination and depth of field relying solely on geometric buffers to guide the filter—this makes it prone to inconsistencies with high sampling rates.

In our work, we directly denoise deep images, where individual samples are not aggregated in a single pixel value, but rather splatted into multiple depth bins per pixel. This approach allows combining the benefits of pixel-based and sample-based methods while retaining the generality and sample-rate scalability of the former, and the flexibility of the later, thereby enabling for the first time the use of denoising in deep compositing workflows.

3. Denoising Flat Images with NL-Means

State-of-the-art denoising algorithms all rely on a combination of noisy color and feature information to guide the denoising process. Our work builds on the joint NL-Means filter used in previous works [RMZ13, MJL*13, KBS15, ZRJ*15], which we describe in this section. Our generalization to deep data follows in Section 4.

As highlighted previously [BRM*16, MMMG16], a joint NL-Means filter solves a zero-order regression by computing a denoised pixel value as a weighted average of its neighborhood,

$$\hat{O}_p = \frac{1}{C_p} \sum_{q \in \mathcal{N}_p} w(p,q)\, O_q, \tag{1}$$

where $\hat{O}_p$ is the filtered value of pixel p, O_q is the noisy input value of a neighbor pixel q in a square neighborhood $\mathcal{N}_p$ centered on p, $w(p,q)$ weights the contribution of q to the filtered value of p, and C_p is a normalization factor that forces weights sum up to 1.

The joint filter weight is computed as $w = \min(w_O, w_F)$, where w_O is an NL-Means weight computed on the data itself, and w_F is a cross-bilateral weight computed on a set of auxiliary features (the surface albedo, normal and depth in our case).

NL-Means weight. w_O is computed on the noisy input as

$$w_O(p,q) = \exp^{-D_O(O_{\mathcal{P}_p}, O_{\mathcal{P}_q})}, \tag{2}$$

where $D_O(O_{\mathcal{P}_p}, O_{\mathcal{P}_q})$ measures the distance between square patches centered on p and q, according to the noisy input values. This distance is computed as the average distance between pairs of pixels in the patches $\mathcal{P}_p$ and $\mathcal{P}_q$,

$$D_O(O_{\mathcal{P}_p}, O_{\mathcal{P}_q}) = \max\left(0, \frac{1}{|\mathcal{P}|} \sum_{n \in \mathcal{P}_0} d_O(p+n, q+n)\right), \tag{3}$$

where $\mathcal{P}_0$ enumerates the offsets to the pixels within a patch, and

$$d_O(p,q) = \frac{1}{|O|} \sum_{i=1}^{|O|} \frac{\|O_{i,p} - O_{i,q}\|^2 - (\mathrm{Var}_{i,p} + \min(\mathrm{Var}_{i,p}, \mathrm{Var}_{i,q}))}{\varepsilon + k_O^2 \cdot (\mathrm{Var}_{i,p} + \mathrm{Var}_{i,q})}$$

is the normalized square distance between pixels p and q, $|O|$ is the number of channels in the noisy image, $O_{i,p}$ and $\mathrm{Var}_{i,p}$ represent the value and the variance of the i-th channel of pixel p, k_O^2 is a user-defined constant that controls the aggressiveness of the filter, and ε is a constant preventing divisions by zero; we use $\varepsilon = 10^{-10}$.

To further improve the filter output, one can apply the patch-wise reconstruction proposed by Buades et al. [BCM05], which performs collaborative filtering by combining neighboring patches instead of pixels.

Cross-bilateral weight. w_F is computed on auxiliary features as

$$w_F(p,q) = \exp^{-D_F(F_p, F_q)}, \tag{4}$$

with the distance between two pixels computed using the auxiliary feature f that maximizes the dissimilarity between p and q, i.e.

$$D_F(F_p, F_q) = \max_{f \in F} d_f(p,q). \tag{5}$$

where F is the set of all auxiliary features.

The feature distance is computed $d_f(p,q)$ as

$$d_f(p,q) = \frac{1}{|f|}\sum_{j=1}^{|f|}\frac{\|f_{j,p}-f_{j,q}\|^2-(\mathrm{Var}_{j,p}+\min(\mathrm{Var}_{j,p},\mathrm{Var}_{j,q}))}{k_f^2\cdot\max(\tau,\mathrm{Var}_{j,p},\|\mathrm{Grad}_{j,p}\|^2)},$$

where $|f|$ is the number of channels of feature f, $f_{j,p}$ and $\mathrm{Var}_{j,p}$ represent the value and the variance of the j-th channel of pixel p, and $\|\mathrm{Grad}_{j,p}\|^2$ is the squared magnitude of the corresponding gradient clipped to the maximum of a user-defined threshold τ and variance $\mathrm{Var}_{j,p}$.

In addition to k_O, k_f and τ, the joint NL-Means filter has two parameters W and w, such that the size of the neighborhood $\mathcal{N}_p$ is $W\times W$ and the size of the patch $\mathcal{P}_p$ is $w\times w$.

4. Denoising Deep Images

Our framework for denoising deep images builds on top of the generalization of joint NL-Means filter described in Section 3. We now describe how we extended this *flat* filter to a *deep* filter, and how it is used in our pipeline to denoise deep Monte Carlo renderings. The deep filter operates on images where the depth samples in each pixel are clustered into discrete bins.

4.1. Deep Joint NL-Means Filter

Our deep joint NL-Means filter retains the dual nature of the flat filter, that is, we still compute a set of NL-Means weights according to the input data, and a set of cross-bilateral weights according to a set of auxiliary features. The key conceptual difference is that we now aim at denoising depth bins, as opposed to pixels. Consequently, we rewrite Equation 1 as

$$\hat{O}_{p_b} = \frac{1}{C_{p_b}}\sum_{q\in\mathcal{N}_p}\sum_{q_d\in B_q} w(p_b,q_d)O_{q_d}, \tag{6}$$

where $\hat{O}_{p_b}$ is the denoised color of bin b in pixel p, O_{q_d} is the input color of bin d in pixel q located in the square neighborhood $\mathcal{N}_p$ centered on p, B_q lists all the bins of pixel q, $w(p_b,q_d)$ weights the contribution of bin q_d to the filtered value of bin p_b, and C_{p_b} is a normalization factor ensuring that the weights of all neighboring bins sum up to 1. The different terms are illustrated in Figure 5. Similar as in the flat filter, the weight $w(p_b,q_d)$ is the minimum of a data weight $w_O(p_b,q_d)$ and a feature weight $w_F(p_b,q_d)$.

Given the unstructured nature of deep image data, NL-Means weights cannot be computed directly on the bin values. Instead, our deep filter uses *alpha-weighted* NL-Means weights computed on the flattened deep data

$$w_O(p_b,q_d) = w_O(p,q)\cdot w_\alpha(q_d), \tag{7}$$

where $w_\alpha(q_d)$ is the *effective* alpha of the neighbor bin that measures its relative contribution to pixel q. This effective alpha equals the number of samples in a bin divided by the total number of samples in a given pixel. For the n-th bin, the effective alpha $w_\alpha(p_n)$ is computed recursively using the cumulative effective alpha of the previous $n-1$ bins and the stored n-th bin alpha $\alpha(p_n)$,

$$w_\alpha(p_n) = \left(1-\sum_{i=1}^{n-1} w_\alpha(p_i)\right)\alpha(p_n). \tag{8}$$

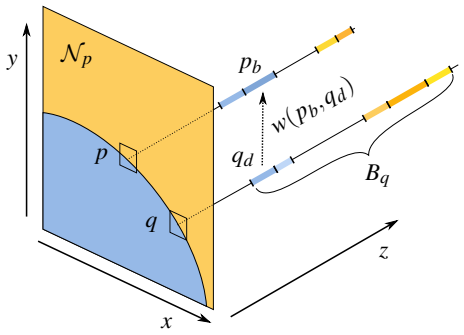


Figure 5: *Illustration of the terms used in Equation 6. This exemplary scene depicts the deep image structure for a blue object in front of a yellow background object. The color of bin q_d in pixel q is splatted into bin q_b of pixel p using the weight $w(p_b,q_d)$.*

Here, the n-th bin alpha $\alpha(p_n)$ is an alpha value corresponding to an *over compositing* alpha weight as specified by the OpenEXR 2.0 deep image file format [KBSH13].

Our proposed alpha-weighted deep NL-Means filter is a generalization of the flat NL-Means filter, which ensures that *flattening the filtered* deep data yields an identical result to *filtering the flattened* data. Mathematically, this can be shown by rewriting the filtered flat pixel value $\hat{O}_p$ as

$$\hat{O}_p = \frac{1}{C_p}\sum_{q\in\mathcal{N}_p} w_O(p,q)\,O_q \tag{9}$$

$$= \frac{1}{C_p}\sum_{q\in\mathcal{N}_p} w_O(p,q)\left[\sum_{q_d\in B_q} w_\alpha(q_d)q_d\right] \tag{10}$$

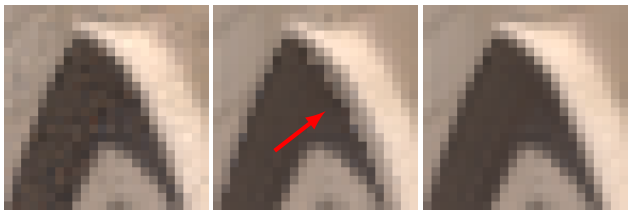
$$= \frac{1}{C_p}\sum_{q\in\mathcal{N}_p}\sum_{q_d\in B_q} w_O(p,q)\cdot w_\alpha(q_d)q_d \tag{11}$$

$$= \frac{1}{C_p}\sum_{q\in\mathcal{N}_p}\sum_{q_d\in B_q} w_O(p_b,q_d)q_d = \hat{O}_{p_b} \tag{12}$$

where the second equality used the definition of the effective alpha weights to expand the pixel value O_q and the last equality used the definition of $w_O(p_b,q_d)$. When filtering a deep image using the alpha-weighted NL-means weights $w_O(p_b,q_d)$, all bins in pixel p are assigned the same color $\hat{O}_{p_b}$ and thus the flattened filtered deep image is identical to a filtered flat image. The results are however identical *only with respect to the NL-Means weights*, but in practice we also consider the cross-bilateral weights. These are computed differently in the deep and flat denoisers, which explains why the resulting outputs differ as can be seen in Figure 8.

Figure 6 illustrates how our alpha-weighted NL-Means filter resolves the artifacts caused by directly applying the standard NL-Means weights to the deep data.

Extending the cross-bilateral weights to the deep data is relatively straightforward, apart for the computation of the auxiliary feature gradient, which again is ill-defined because of the unstructured nature of the data. As before, we use a hybrid approach computing the feature distances according to the bin values, but the



(a) Input (b) No alpha weights (c) Alpha weights

Figure 6: *Denoised results both without and with the effective alpha weights. Without using the effective alpha weights, artifacts can occur where bins have a very small alpha (e.g. near edges). We disabled feature weights in this figure to make the artifact more visible. The artifact is less pronounced when using features since features reduce the contribution of these very different bins.*

feature gradient according to the flattened data,

$$d_f(p_b,q_d)=\frac{1}{|f|}\sum_{i=j}^{|f|}\frac{\|f_{j,p_b}-f_{j,q_d}\|-(\mathrm{Var}_{j,p_b}+\min(\mathrm{Var}_{j,p_b},\mathrm{Var}_{j,q_d})}{k_f^2\cdot\max(\tau,\mathrm{Var}_{j,p_b},\|\mathrm{Grad}_{j,p}\|^2)}.$$

The final cross-bilateral weight is computed using Equations (4) and (5) replacing p and q by p_b and q_d, respectively.

4.2. Deep Denoising Framework

Given the deep NL-Means filter defined in Section 4.1, we can now setup our denoising framework. As input, we take a single deep multichannel EXR, with the following set of information per bin: the mean and variance of the color, albedo, normal and depth samples, as well as the alpha value. The input data is actually split into half buffers as proposed by Rousselle et al. [RKZ12], each of which stores one half of the samples, which allows us to estimate the mean squared error (MSE) of the filtered output.

Auxiliary Feature Prefiltering. Despite the binning procedure (see Appendix A for details) reducing noise in feature buffers that stems from variation in depth, the features may still suffer from noise. For example, consider a textured quad sliding within its plane. Multiple texture values will project into the same pixel (and bin) and since the time domain is sampled stochastically, the albedo feature will suffer from noise. Similarly to previous works [RMZ13, KBS15, ZRJ*15, MMMG16] we therefore prefilter the albedo and normal using our joint deep filter. For the albedo, the alpha-weighted NL-Means weights are computed using the flattened albedo bins with the depth values acting as an auxiliary feature. The normal buffer is processed analogously. We use the following parameters in this preprocessing step: $k_O=1.50$, $k_f=0.05$, $\tau=0.001$, $W=5$ and $w=3$.

Note that we do not prefilter the alpha values in our framework, as these are quite sensitive. This ensures consistent deep compositing operations, at the cost of some residual noise artifacts.

Color Filtering. Given the pre-filtered albedo and normal values, we now turn to denoising the color values of the bins. For this, we use a filter bank of three candidate deep filters with the following parameters: 1) $k_O=0.45$, $k_f=0.7$; 2) $k_O=0.6$, $k_f=2.0$; 3) $k_O=10^4$, $k_f=2.0$. All three scales use $\tau=0.001$, $W=9$ and $w=3$.

We then linearly combine these three candidate filters on a per-pixel basis according to their estimated MSE. Note that we compute the statistics on the flattened deep data, to increase their robustness. For the MSE estimation, we used the approach proposed by Bitterli et al. [BRM*16],

$$\mathrm{MSE}[F]\approx\frac{1}{2}\left((F_0-C_1)^2+(F_1-C_0)^2\right)-2\,\mathrm{Var}[C]-\mathrm{Var}[F] \tag{13}$$

where F denotes the candidate filter output, F_1 and F_2 are the filtered half buffers and C is the noisy input, which is split in half buffers C_0 and C_1. Similar to previous work [RMZ13], we first filter the noisy MSE estimates using a flat NL-means filter and then combine the candidate filters using a selection map computed on the filtered MSE estimates.

5. Implementation Details

We modified the open-source Mitsuba renderer [Jak10] to output all the required features and half buffer data as deep images. For the RUNNING MAN and KAKAMORA scenes, we used Disney's Hyperion renderer.

5.1. Rendering Deep Images

We generate deep images by first running a depth-only rendering prepass in which we determine the depth binning of each pixel and an alpha value for each bin. We then render the deep image using the same random seed and assign samples to the nearest bin in absolute depth distance. Using a depth prepass allows us to pre-allocate a fixed-size frame buffer and avoids the need to capture and sort all samples which could otherwise be prohibitive, especially given the large number of render output channels typically produced.

Ideally, both prepass and main render pass would use the same number of samples per pixel, such that the depth distributions of the prepass and the main render pass match. In practice this is not necessarily the case, as we may want to use adaptive sampling in the main render pass and the prepass does not have the adaptive sampling information (e.g. per-pixel variance estimates of the color samples). This discrepancy can potentially increase the depth variance in some of the bins.

To determine the bin layout from the prepass, we sort the depth samples for each pixel and collapse consecutive samples that fall within a depth threshold (see Appendix A for pseudocode). The depth threshold for the first bin in each pixel is determined by projecting the screen size of the pixel to each sample's scene depth, thus making the depth quantization comparable to the pixel quantization. To constrain depth complexity, we scale the threshold at subsequent bins by a factor of b^E where b is the bin number and E is a user parameter. We find a default value of $E=2$ to work well in a variety of scenes, providing good separation between layers while keeping the number of bins low even in scenes with high depth complexity. In the KAKAMORA scene, for example, there are an average of 1.66 bins per pixel.

5.2. Decomposition and Irradiance Factorization

During rendering, we decompose the input into a pair of diffuse and specular components, based on the roughness of the sampled BSDF lobe, such that summing these two components yields back the full color. As shown by Zimmer et al. [ZRJ*15], this type of decomposition allows tailoring the filter behavior to the distinct noise characteristic and structure of each component, leading to higher quality denoised results. For the diffuse component (but not the specular one), we also make use of the effective irradiance proposed by Zimmer et al. We compute it directly on the deep data, by dividing the noisy diffuse color of each bin by its *noisy* albedo (offset by a small constant to prevent division by zero). Computing the effective irradiance directly on the deep data is crucial, as the flattened albedo is not necessarily representative of each bin in a pixel. The denoised diffuse component is obtained by multiplying back the denoised irradiance of each bin by its *denoised* albedo, ensuring better preservation of the fine albedo details than when directly filtering the diffuse color. The final result is then simply the sum of the denoised diffuse and specular components.

5.3. Variance Prefiltering

For the auxiliary features, we directly make use of the sample mean variance. However, for the color values, we instead use the two-buffer variance computed across our two half buffers, as proposed by Rousselle et al. [RMZ13]. Using the two-buffer variance is needed for renderings produced with correlated samples, where the sample mean variance may overestimate the actual estimator variance. In practice, most rendering systems use low-discrepancy or stratified random samples, which are highly correlated [PJH16]. Additionally, using two buffers allows us to robustly compute the variance of the effective diffuse irradiance, which would otherwise be challenging given that the diffuse color and albedo are not only both noisy, but also correlated. Since the two-buffer variance estimate is itself very noisy, we prefilter it using a Gaussian kernel with $\sigma = 0.5$, and then use the maximum of the noisy and blurred two-buffer variances to guide our deep filter.

6. Results

We implemented our filter on the CPU using a mixture of Python and C++ code. Partially multithreaded filtering performance is around 12 minutes per megapixel, where the prefiltering of the features takes around one minute and scale selection around 10 seconds. Performance was measured on an Intel i7-4930K processor, running 6 cores (12 threads) at 3.4Ghz.

In Figure 7, we demonstrate how our deep filter preserves the structure of the input data. We insert a clipping plane in each deep image and visualize bins that are in front of the plane (RUNNING MAN, SIBENIK, KAKAMORA) and behind the plane (SPONZA). The reference solutions use the same set of bins as the noisy input, but estimate radiance with much higher SPP to suppress the noise. Due to the stochastic nature of computing primary visibility, some bins receive a very low number of samples (see pixels around the fuzzy border in the SPONZA insets). Our deep denoiser filters these bins robustly by finding similar bins in other pixels producing nearly noise-free results even in these difficult scenarios.

Figure 1 illustrates a deep compositing result where a deep render of a smoke is inserted into a deep render of the KAKAMORA scene. The deep compositing pipeline allows the artist to change the placement of the smoke very quickly without having to rerender holdouts whenever the position of the smoke is adjusted.

In Figure 8, we assess the quality of our deep filter with respect to prior work. Given that there are currently no alternative deep-image denoisers, we opted to perform the comparison against flattened data, that is, we compare the result of flattening the output of our deep denoiser to the result of denoising the flattened input. We denoise raw renders without any compositing operations, since the flat denoiser inputs are otherwise generally ill-defined. For instance, problematic cases would involve updating the per-pixel variances following a non-linear color correction, or handling composited material for which the auxiliary features (variance, surface normal, etc.) are not defined. Avoiding any compositing for this particular comparison conveniently sidesteps these difficulties. We use the RDFC filter [RMZ13] as the baseline, which builds on a similar joint NL-Means scheme and yields results that are close to the current state of the art in regression-analysis denoising [BRM*16]. We implemented RDFC in the same codebase as our deep filter, and run it on the same decomposed input (including irradiance factorization); our implementation requires about two minutes per megapixel. The comparison demonstrates that our deep filter offers visually similar results to RDFC. The richer input allows our filter to better preserve some details, such as the character's fur in the KAKAMORA scene. However, since filtering unstructured bins is inherently more difficult than filtering the regular structure of pixels, occasional regressions in quality may appear and the overall rMSE is thus slightly higher. In particular, our deep filter tends to leave more residual noise, partially because our auxiliary feature prefiltering uses conservative settings to prevent excessive blurring across bins within the target pixel.

7. Limitations and Future Work

Our filter has a number of limitations that could be addressed in future work.

Pixel-based weights. Our deep joint NL-Means filter uses a mixture of pixel-based color weights and bin-based feature weights. This hybrid approach is crucial, since the per-bin color information is simply not sufficiently reliable, but it hinders our deep filter. In particular, our color weights cannot accommodate conflicting constraints, such as the need of filtering a noisy motion blurred object in front of a noise-free static background, even though the deep nature of our filter in theory affords it. This limitation is however mitigated by the bin-based feature weights. An interesting avenue for future research would be to operate on individual samples and leverage advanced machine learning approaches, e.g. deep neural network, to avoid the need for storing all samples.

Computational overhead. Working on sparse, binned data incurs a significant overhead: we have multiple bins to process per pixel, and the neighborhood of each bin must be explicitly assembled on a per-bin basis. Our filter cost is therefore, in contrast with pixel-based image-space filters, not independent of scene complexity, although it is independent of the sampling rate.

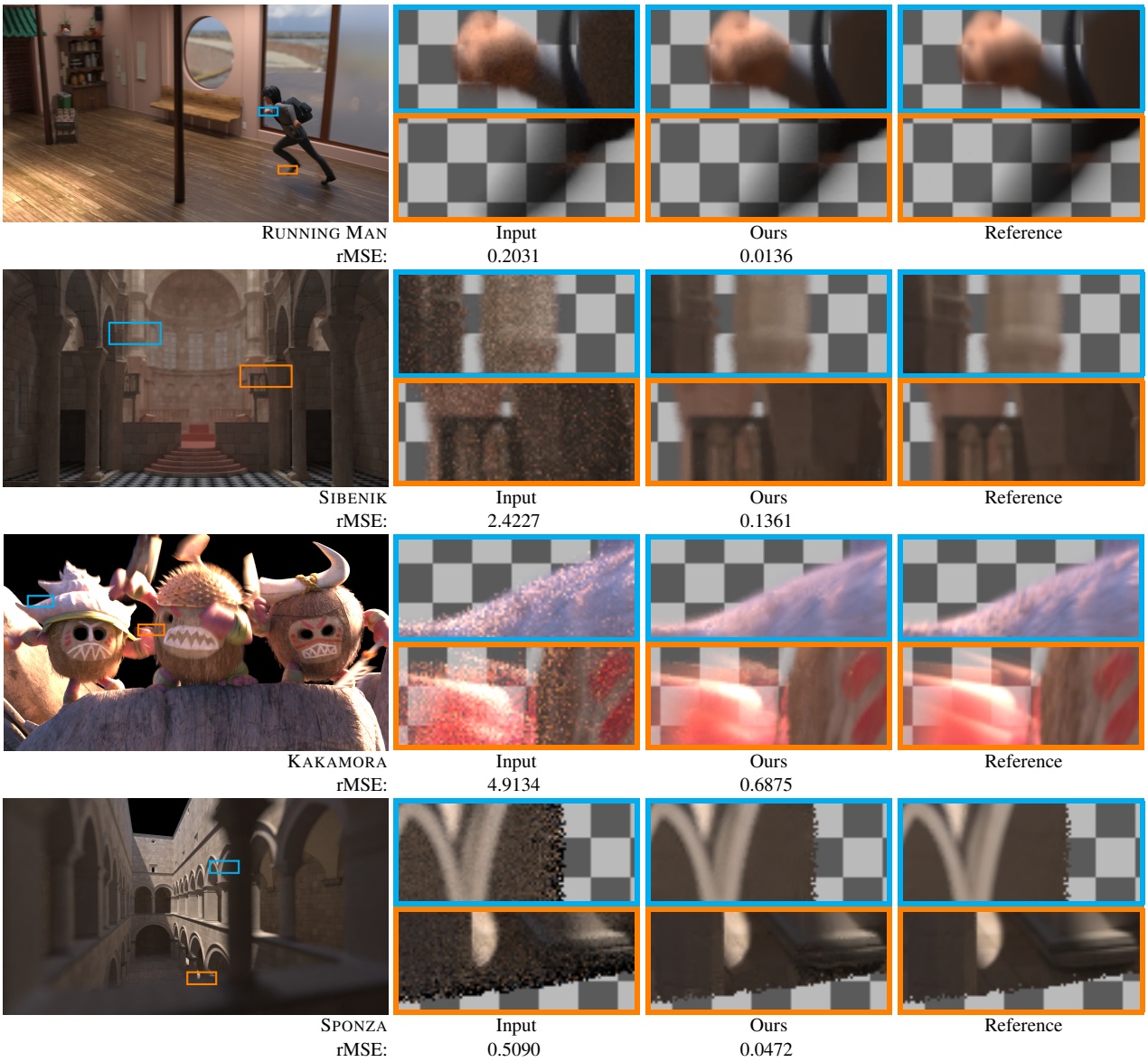


Figure 7: *Comparisons of deep images showing insets with noisy inputs (128 SPP), outputs of our deep denoiser, and references, all clipped at a certain depth. The references use the same depth bins as the noisy input but higher SPP. The relative mean squared error (rMSE) is computed on the flattened images to quantify filtering quality. All error values are scaled by 100. © Disney*

Noisy alpha. To ensure sharp compositing boundaries, we currently do not denoise the alpha value of each bin. While the alpha value noise is typically of small magnitude, it is sometimes very noticeable in the denoised output, e.g. with strong depth of field. To address this issue, a simple solution is to oversample the alpha value by tracing additional camera rays. Those rays have a relatively small overhead since we only record the surface depth and terminate at the first intersection, whereas production rendering cost is dominated by indirect bounces and shading computation. In Figure 9, we illustrate how alpha supersampling resolves the residual noise artifacts, but a robust alpha-denoising solution would nonetheless be preferable. None of the results in our paper make use of alpha supersampling.

Noisy features. Our depth-based binning process ensures that each bin depth value is relatively noise-free, however the albedo and normal values still suffer from noise. Robust prefiltering of albedo and normal however is challenging for deep data, and our approach of using conservative settings is clearly suboptimal.

8. Conclusions

We presented a denoising algorithm for deep images, which achieves similar quality to a recent image-space filter while preserving the deep data structure. Despite the relatively straightforward design of our deep filter and its limitations, our results demonstrate the feasibility of denoising deep Monte Carlo renderings. The

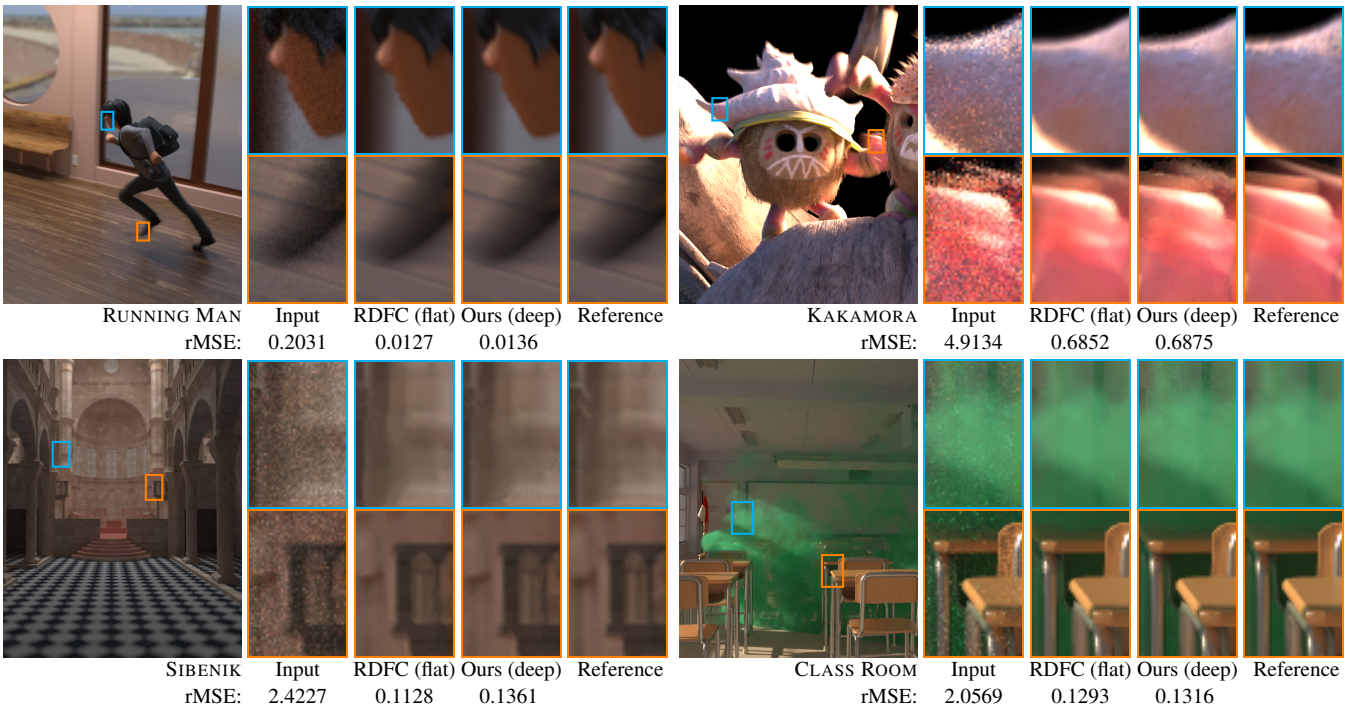


Figure 8: *Comparisons of flattened noisy inputs (128 SPP,* CLASS ROOM *using 256 SPP), flattened inputs denoised with RDFC, and deep inputs that are denoised with our filter and then flattened. The relative mean squared error (rMSE) is computed on the flattened images to quantify filtering quality. All error values are scaled by 100. Our denoiser performs comparably to RDFC but allows preserving the deep structure of deep images. These comparisons are on raw data, i.e. without any compositing, to avoid the difficulties of tracking variance and feature buffers through compositing operations. © Disney*

approach presented here represents the first step towards integrating denoising into modern deep-compositing workflows, which could be further extended in future work, e.g. by improving the binning strategy, reducing the computational overhead of processing deep data, improving the robustness of feature prefiltering, using the deep data to drive a more finely-tuned adaptive sampling scheme, using volumetric deep samples to improve compositing for volumes, or extending this type of denoising to handle rendered light fields. Most importantly, however, our paper introduces a new perspective on the denoising problem and shall stimulate future publications to consider the more challenging scenario of denoising deep images to ensure compatibility with production environments that rely on deep compositing.

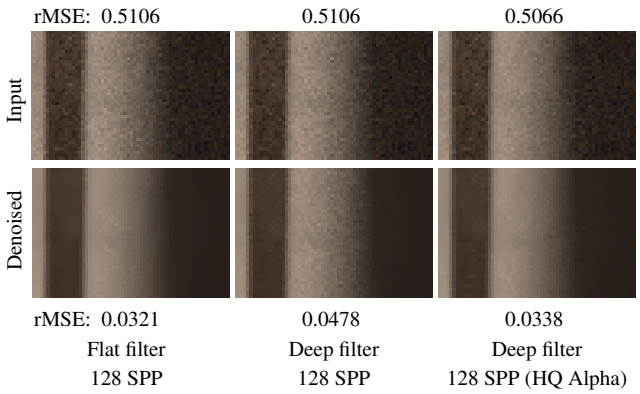


Figure 9: *Denoising results on the* SPONZA *scene using both low- and high-quality alpha values. Even moderate alpha noise can cause noticeable noise in the denoised color result. Using higher quality alpha (1024 SPP) prevents this residual noise.*

Acknowledgments

We thank the anonymous reviewers for their critical feedback. We thank Marios Papas for useful discussions and proofreading. We also would like to thank Greg Nichols and Ralf Habel for their work on the sample clustering algorithm.

We thank BlendSwap user NovaZeeke for the CLASS ROOM scene, Marko Dabrovic for the SPONZA and SIBENIK scenes and Alex Nijmeh and other artists at Walt Disney Animation Studios who worked on the KAKAMORA and RUNNING MAN scenes.

References

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

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

[BEJM15] BAUSZAT P., EISEMANN M., JOHN S., MAGNOR M.: Sample-based manifold filtering for interactive global illumination and depth of field. *Comp. Graph. Forum 34*, 1 (2015), 265–276. 4

[BEM11] BAUSZAT P., EISEMANN M., MAGNOR M.: Guided image filtering for interactive high-quality global illumination. *Comp. Graph. Forum 30*, 4 (2011), 1361–1368. 4

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J. A., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. *Comp. Graph. Forum (Proc. EGSR) 35*, 4 (June 2016), 107–117. 4, 6, 7

[BVM*17] BAKO S., VOGELS T., MCWILLIAMS B., MEYER M., NOVÁK J., HARVILL A., SEN P., DEROSE T., ROUSSELLE F.: Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph. (Proc. SIGGRAPH) 36*, 4 (2017), 97:1–97:14. 4

[CKS*17] CHAITANYA C. R. A., KAPLANYAN A., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZAHRAI D., AILA T.: Interactive reconstruction of noisy Monte Carlo image sequences using a recurrent autoencoder. *ACM Trans. Graph. (Proc. SIGGRAPH) 36*, 4 (2017). 4

[DL03] DUAN J., LI J.: Compression of the layered depth image. *IEEE Transactions on Image Processing 12*, 3 (March 2003), 365–372. 3

[DMB*14] DELBRACIO M., MUSÉ P., BUADES A., CHAUVIER J., PHELPS N., MOREL J.-M.: Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 8:1–8:15. 4

[EDL15] EGSTAD J., DAVIS M., LACEWELL D.: Improved deep image compositing using subpixel masks. In *Proceedings of Digital Production Symposium* (2015), pp. 21–27. 3

[HHHF12] HANIKA J., HILLMAN P., HILL M., FASCIONE L.: Camera space volumetric shadows. In *Proceedings of Digital Production Symposium* (2012), pp. 7–14. 1

[HJW*08] HACHISUKA T., JAROSZ W., WEISTROFFER R. P., DALE K., HUMPHREYS G., ZWICKER M., JENSEN H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 33:1–33:10. 4

[HWW10] HILLMAN P., WINQUIST E., WELFORD M.: Compositing "avatar". SIGGRAPH 2010 Talks, 2010. 3

[Jak10] JAKOB W.: Mitsuba renderer, 2010. 6

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph. (Proc. SIGGRAPH) 34*, 4 (July 2015), 122:1–122:12. 4, 6

[KBSH13] KAINZ F., BOGART R., STANCZYK P., HILLMAN P.: Technical introduction to OpenEXR, 2013. 1, 3, 5

[KN01] KIM T.-Y., NEUMANN U.: Opacity shadow maps. In *Proc. of Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer, pp. 177–182. 3

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4 (July 2011), 55:1–55:12. 4

[LALD12] LEHTINEN J., AILA T., LAINE S., DURAND F.: Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4 (July 2012), 51:1–51:10. 4

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Annual Conference Series (Proc. SIGGRAPH)* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 385–392. 3

[LWC12] LI T.-M., WU Y.-T., CHUANG Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6 (Nov. 2012), 194:1–194:9. 4

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Proc. of Eurographics Workshop on Rendering Techniques* (London, UK, 1996), Springer, pp. 165–174. 3

[MBC*17] MILDENHALL B., BARRON J. T., CHEN J., SHARLET D., NG R., CARROLL R.: Burst Denoising with Kernel Prediction Networks. *ArXiv e-prints* (Dec. 2017). 4

[McC99] MCCOOL M. D.: Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph. 18*, 2 (Apr. 1999), 171–194. 4

[MCY14] MOON B., CARR N., YOON S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (Sept. 2014), 170:1–170:14. 4

[MIGMM17] MOON B., IGLESIAS-GUITIAN J. A., MCDONAGH S., MITCHELL K.: Noise reduction on g-buffers for Monte Carlo filtering. *Comp. Graph. Forum* (2017), 600–612. 4

[MJL*13] MOON B., JUN J. Y., LEE J., KIM K., HACHISUKA T., YOON S.-E.: Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Comp. Graph. Forum 32*, 1 (2013), 139–151. 4

[MKBR04] MERTENS T., KAUTZ J., BEKAERT P., REETH F. V.: A self-shadow algorithm for dynamic hair using density clustering. In *Rendering Techniques (Proc. EG Symposium on Rendering)* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 173–178. 3

[MMMG16] MOON B., MCDONAGH S., MITCHELL K., GROSS M.: Adaptive polynomial rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* (2016). 4, 6

[MMNL16] MARA M., MCGUIRE M., NOWROUZEZAHRAI D., LUEBKE D.: Deep g-buffers for stable global illumination approximation. In *HPG* (June 2016). 3

[NSS10] NIESSNER M., SCHÄFER H., STAMMINGER M.: Fast indirect illumination using layered depth images. *The Visual Computer 26*, 6-8 (2010), 679–686. 3

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5 (2009), 140–1. 4

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation (3rd ed.)*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Oct. 2016. 7

[RKZ12] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6 (Nov. 2012), 195:1–195:11. 6

[RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *Comp. Graph. Forum (Proc. of Pacific Graphics) 32*, 7 (2013), 121–130. 4, 6, 7

[RW94] RUSHMEIER H. E., WARD G. J.: Energy preserving non-linear filters. In *Annual Conference Series (Proc. SIGGRAPH)* (New York, NY, USA, 1994), ACM, pp. 131–138. 4

[SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph. 31*, 3 (June 2012), 18:1–18:15. 4

[Sey14] SEYMOUR M.: The art of deep compositing, 2014. 1

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *Annual Conference Series (Proc. SIGGRAPH)* (New York, NY, USA, 1998), ACM, pp. 231–242. 3

[YK08] YUKSEL C., KEYSER J.: Deep Opacity Maps. *Comp. Graph. Forum* (2008). 3

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Comp. Graph. Forum (Proc. Eurographics) 34*, 2 (May 2015), 667–681. 4

[ZRJ*15] ZIMMER H., ROUSSELLE F., JAKOB W., WANG O., ADLER D., JAROSZ W., SORKINE-HORNUNG O., SORKINE-HORNUNG A.: Path-space motion estimation and decomposition for robust animation filtering. *Comp. Graph. Forum (Proc. EGSR) 34*, 4 (June 2015). 4, 6, 7

```
CLUSTERSAMPLES(depth[], ppuScale)
 1  if depth.size() == 0
 2      return
 3  depth = SORT(depth)
 4  deltas = []
 5
 6  // Compute relative depth deltas metric
 7  for k = 0 to depth.size() - 1
 8      deltas[k] = (depth[k + 1] - depth[k]) * (ppuScale / depth[k])
 9  deltas[depth.size() - 1] = 0
10
11  // Cluster samples into bins
12  bins = []
13  k = 0
14  while k < depth.size()
15      // Start a new a bin for sample k
16      bins.add(depth[k])
17      threshold = (bins.size() + 1)^E
18
19      // Advance k until sum > threshold
20      sum = deltas[k]
21      k++
22      while k < depth.size()
23          sum += deltas[k];
24          if sum > threshold
25              break
26          k++
27  return bins
```

Figure 10: *Pseudocode for the deep sample clustering heuristic.*

Appendix A: Depth Sample Clustering

Our depth sample clustering heuristic is outlined the pseudocode in Figure 10. This algorithm takes the samples from the depth prepass as an input and outputs an array of depth bins in each pixel.

First we sort the sampled depth values in each pixel.

Next we compute the relative depth *deltas* metric between subsequent pairs of depth samples: the absolute depth difference is scaled by the perspective projection of *ppuScale*, which measures the size of a 1-unit quad one unit away from the camera. The metric thus indicates how large a cube spanning the two depth samples would appear on the image plane. This metric makes our clustering more aggressive farther from the camera and makes it independent of scene scale.

Finally, we iteratively cluster samples into the same bin until the sum of relative depth differences is greater than *threshold*, at which point we start a new bin. The threshold grows larger as we allocate more bins, penalizing creating an excessive number of bins. The constant E controls how aggressively depth samples are merged. In our implementation we use a value of $E = 2$, which seems to work well in a variety of scenes