

Neural Denoising for Deep-Z Monte Carlo Renderings

Xianyao Zhang^{†1,2} , Gerhard R othlin², Shilin Zhu³, Tun Ozan Aydın², Farnood Salehi²,
Markus Gross^{1,2} and Marios Papas^{†2} 

¹ETH Z urich, Switzerland ²Disney Research | Studios, Switzerland ³Pixar Animation Studios, USA

Abstract

We present a kernel-predicting neural denoising method for path-traced deep-Z images that facilitates their usage in animation and visual effects production. Deep-Z images provide enhanced flexibility during compositing as they contain color, opacity, and other rendered data at multiple depth-resolved bins within each pixel. However, they are subject to noise, and rendering until convergence is prohibitively expensive. The current state of the art in deep-Z denoising yields objectionable artifacts, and current neural denoising methods are incapable of handling the variable number of depth bins in deep-Z images. Our method extends kernel-predicting convolutional neural networks to address the challenges stemming from denoising deep-Z images. We propose a hybrid reconstruction architecture that combines the depth-resolved reconstruction at each bin with the flattened reconstruction at the pixel level. Moreover, we propose depth-aware neighbor indexing of the depth-resolved inputs to the convolution and denoising kernel application operators, which reduces artifacts caused by depth misalignment present in deep-Z images. We evaluate our method on a production-quality deep-Z dataset, demonstrating significant improvements in denoising quality and performance compared to the current state-of-the-art deep-Z denoiser. By addressing the significant challenge of the cost associated with rendering path-traced deep-Z images, we believe that our approach will pave the way for broader adoption of deep-Z workflows in future productions.

CCS Concepts

• *Computing methodologies* → *Ray tracing; Image processing;*

1. Introduction

Compositing is an essential part of animated film and visual effects production. Through compositing operations, such as holdout, merge, color correction, and position adjustments, different parts of a frame can be post-processed independently to achieve the desired look. Compared to adjusting the underlying scene and re-rendering the frame, these effects can be achieved with significantly lower costs via compositing.

While traditional compositing workflows are based on *flat* images, recent years have seen the prevalence of *deep compositing* workflows using *deep-Z* images since their standardization in OpenEXR [Ope13]. Each pixel in a deep-Z image can contain an arbitrary number of *bins*, each of which stores the color value and opacity (alpha) at the corresponding depth, along with optional auxiliary channels. The sub-pixel depth-resolved information allows overlapping parts in a frame to be stored separately during a single rendering pass, facilitating downstream compositing operations. More specifically, deep-Z images can ameliorate artifacts when performing compositing operations that selectively affect a

part of the visible depth range, as illustrated in Figure 1. Operating on deep-Z images with depth-resolved color and other information extends compositing capabilities and further reduces the need for re-rendering. As such, deep compositing workflows are commonly employed during the production of animated features and visual effects [Sey14].

However, path-traced deep-Z images generated by production renderers [Ren22, Arn23, VR23] suffer from the same problem as flat images—noise. Noise can significantly interfere with the interpretability of the rendered content and increase the difficulty of achieving the desired effect. Due to the lack of high-quality and efficient denoisers that output deep-Z images, deep compositing has to rely on either fully converged deep-Z renderings or “quasi-deep” images which combine denoised flat color and converged deep-Z depth and opacity.

While it is possible for an artist to perform deep compositing with noisy data, flatten the result, and then apply state-of-the-art flat image denoisers *after* compositing, such a workflow is undesirable [VAN*19]. Denoising flat images after deep compositing with noisy data limits the types of possible compositing operations and introduces discrepancies with the final result. Compositing with the noisy image can alter the noise characteristics and lead to denoising

[†] xianyao.zhang@inf.ethz.ch, marios.papas@disneyresearch.com

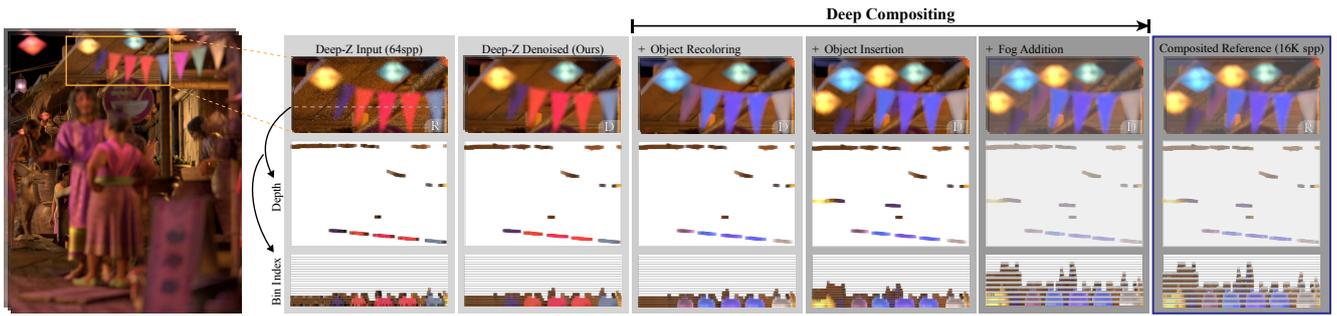


Figure 1: Deep compositing pipelines allow editing operations such as recoloring a part of the depth range and inserting objects at the desired depth. They rely on deep-Z images, which store depth-resolved information at multiple bins within each pixel. Performing these operations on deep-Z instead of flattened images avoids artifacts on anti-aliased edges and semi-transparent objects. In this illustrative example, we start from a noisy deep-Z rendering, denoise it with our proposed denoiser and then perform a series of deep compositing operations such as recoloring the out-of-focus flags in the foreground, adding objects in between the foreground and the background and simulating fog within the scene. The two rightmost columns compare the deep compositing result on our denoised 64spp image with the outcome of the same editing operations on a rendered reference with 16K samples per pixel. For each stage, we visualize the color for each bin within a marked scanline at the corresponding depth (middle row) or bin index (bottom row). © Disney

artifacts by introducing inter-pixel correlations unseen during the denoiser’s training, *e.g.*, through smearing, selective blurring, and even plain resizing. In addition, the denoiser might introduce blur to composited noise-free elements, such as background plate images and text. Finally, high-quality denoisers rely on auxiliary features such as surface normals, albedo, and statistics (*e.g.*, color variance) that are very difficult, if not impossible, to reasonably track through each compositing operation. For these reasons, a high-quality deep denoiser producing denoised deep-Z images which can directly be used in deep compositing pipelines is important in practice.

Despite the ample practical reasons for denoising deep-Z renderings, it is fundamentally more challenging than denoising flat renderings, and the deep-Z format causes practical issues for a direct extension of the prevalent neural flat denoisers based on convolutional neural networks (CNNs) [VRM*18, XZW*19, ZMV*21, BWM*23]. A deep-Z denoiser, whose input and output are both deep-Z images, aims to produce an accurate noise-free reconstruction of the color values in *all bins*. In contrast, a flat denoiser focuses on reconstructing only one value per pixel. A deep-Z denoiser should additionally accurately reconstruct per-bin alpha and depth values, which are also subject to noise during path tracing. Moreover, a bin in a noisy deep-Z image aggregates only a fraction of the paths traced for the associated pixel and is likely to exhibit a higher amount of noise than the pixel value which aggregates all path samples. The deep-Z format, unlike the 2-D image format, is essentially semi-structured since each deep pixel, depending on the complexity of the depth information, can have different numbers of bins at arbitrary depth positions. This can be problematic for convolutional architectures and kernel-predicting denoising because the neighborhood of each bin is defined by the bin index, which does not necessarily align with the neighborhood in depth space, where neighboring bins useful for denoising are more likely to be found. Such misalignment creates artifacts when denoising with CNNs applied on the spatial-bin dimensions, which rely on assumptions about translation invariance along all three dimensions.

In this work, we introduce a CNN-based neural deep-Z denoiser that tackles the aforementioned issues and achieves high denoising quality for deep-Z renderings. Our method uses a hybrid flat-deep-Z architecture to improve the result on flat regions and uses depth as a prior for aligning bin neighborhoods. On a dataset consisting of production-quality deep-Z renderings, we evaluate our method by comparing against a state-of-the-art non-neural deep-Z denoiser [VAN*19] and a flat kernel-predicting neural denoiser [VRM*18] trained on a flattened version of our dataset. Our main contributions are listed below:

- The first neural deep-Z image denoiser that produces state-of-the-art results while being significantly more efficient than previous non-neural methods.
- A hybrid flat-deep-Z network architecture with flattened pixel context and learned combination of flat and deep reconstruction.
- Light-weight depth-aware neighbor indexing of the input of convolutions and denoising kernels that addresses depth misalignment in deep-Z data.

Next, we review the background knowledge on deep-Z renderings and discuss relevant previous work in the areas of denoising path-traced flat and deep-Z images.

2. Background and related work

Deep-Z images. We follow the OpenEXR standard [Kai13b] for defining deep-Z images. A deep-Z image can contain an arbitrary number of bins per pixel. Each bin is associated with a depth range and an opacity (alpha) value, and it can contain an arbitrary number of channels.

Let $\mathbf{p} \in \mathbb{Z}_{>0}^3$ with components (x, y, b) be a three dimensional index vector of a bin within a deep-Z image, where (x, y) correspond to spatial indices within the image plane and b corresponds to the index along the bin dimension. We extract the stored data of each bin with the help of functions such as $z_f(\mathbf{p})$ and $z_b(\mathbf{p})$ which return the depth of the bin bounds (front and back of the bin) in scene

units, $\mathbf{c}(\mathbf{p})$ for the color, $\alpha(\mathbf{p})$ for the *over* compositing alpha, $\mathbf{n}(\mathbf{p})$ for the surface normal, $\rho(\mathbf{p})$ for the surface albedo. Note that all these quantities, with the exception of depth information, are pre-multiplied by alpha as defined in the OpenEXR format [Kai13b]. Our current rendered deep-Z dataset only contains bins with a single depth value ($z_f(\mathbf{p}) = z_b(\mathbf{p})$) but it is also possible for a bin \mathbf{p} to cover a depth range with $z_f(\mathbf{p}) < z_b(\mathbf{p})$.

The bin layout is defined by the number of bins of all deep pixels and is computed by the renderer, either in a pre-pass or progressively during rendering. The bin layout of our path-traced production dataset is fixed in a pre-pass [VAN*19], which also determines the location of each bin. All noise levels of the same scene, including the reference, share the same bin layout.

The ability of deep-Z images to capture and preserve a richer set of data provides enhanced flexibility and versatility in final compositing [Ren18]. With depth values, we can get accurate representation of spatial relationships like occlusions between different objects, which helps creating depth-based effects in the composited image [Ope23, Fou23]. In addition, storing multiple color and alpha samples per pixel allows more control over object transparency and more accurate anti-aliasing when blending multiple objects at different depths. These depth-aware images facilitate matte extraction and object separation without relying on complex techniques traditionally used on flat images to reduce edge artifacts [PD84], significantly simplifying operations like re-positioning or replacing foreground and/or background objects [HSDC11]. In film production, compositing artists prefer the greater flexibility of manipulating individual objects and channels, making fine-grained adjustments, and controlling precisely over the final composited look to achieve visually compelling images [Ren22, Ren12], which deep-Z images are designed for.

Denoising of path-traced renderings. As an effective post-process that can reduce the render time by orders of magnitude, denoising techniques for rendered content have been an active area of research [ZJL*15, HY21]. Most denoising approaches focus on the flat image format and reconstruct a denoised pixel as a weighted average of noisy input pixel values in the pixel's spatial or temporal neighborhood. Recent years saw the rise of deep learning denoising approaches of flat renderings, which advance the state of the art in big strides by leveraging large datasets [KBS15, CKS*17, BVM*17, VRM*18]. Below, we discuss relevant previous denoising methods on flat and deep-Z images.

Flat image denoising. Rousselle et al. [RMZ13] propose using geometry buffers, such as normal vector, albedo, and depth, in the pixel similarity measure when computing the filter kernels for each pixel because these auxiliary features typically contain less noise than the color channels but reflect important discontinuities that should be preserved. Bako et al. [BVM*17] propose using convolutional neural networks (CNNs) to predict per-pixel filtering kernels from the noisy image and auxiliary feature buffers. Combined with the diffuse and specular decomposition, their proposed denoiser (KPCN) can improve upon previous non-neural denoisers in quality significantly. Building on KPCN, Vogels et al. [VRM*18] propose reusing temporal information to further enhance the denoising quality, and Zhang et al. [ZMV*21] propose to learn a

decomposition of the noisy input into easier-to-denoise components before applying kernel-predicting denoising to achieve even more faithful reconstructions. Starting from single samples instead of aggregated sample statistics at each pixel, the layer denoising method [MH20] learns to group individual samples into layers that are separately denoised, improving efficiency when processing the large number of raw samples compared to previous sample-based methods [GLA*19] and achieves higher quality compared to pixel-based methods at low sample count. These denoising techniques can use either pixels or samples as input, but they all aim at producing high-quality *flat* images, and do not preserve depth separation within pixels like our deep-Z denoiser.

Deformable convolutions. Deformable convolutions [DQX*17] are an extension of the classic convolution, where the regular grid of a convolution kernel is replaced with a learnable offset field. This makes them unstructured, which is advantageous for object detection and similar tasks where it is important to attend to the object being detected rather than a conventional 2-D grid [DQX*17]. Deformable convolutions could also be used in deep-Z denoising to adapt the bin neighborhood, but at triple the computation cost. Additionally, we argue that, for our use case, depth is already a strong prior for finding bins that belong to the same object or have similar color information. This allows us to rely on a simple heuristic and avoid the additional parameters needed by deformable convolutions.

Point cloud denoising. Irregularly spaced geometric data can be processed with point cloud and graph neural networks (GNNs). Recently proposed models include PointNet [QSMG17], PointNet++ [QYSG17], GCNConv [KW17], GraphConv [MRF*19], GATConv [VCC*18], and many others [PyG23]. Point cloud processing or GNNs are also possible techniques for solving the task, but they pose unique challenges, such as large spatial density variation and complexity from millions of points from a single deep-Z image. These techniques are designed for completely unstructured data and potentially require significant adaptation and architectural redesign to fit our specific denoising problem. Given the proven success of CNNs in denoising structured flat images in production [DAN19, ZZR*23], we opted to use a CNN denoiser for the deep-Z images. Our work leverages the structure of deep-Z images, particularly the 2-D uniform pixel grid, and introduces depth-aware neighborhoods to effectively denoise deep-Z images, even with irregular depth structures.

Deep-Z image denoising. Denoising of deep-Z images is still relatively less studied, and deep learning methods solving this problem do not exist, to the best of our knowledge. The current state of the art in denoising deep-Z renderings [VAN*19] extend the hand-crafted heuristics of flat feature-guided non-local means denoising technique [RMZ13] to deep-Z renderings. Their filter kernels are derived for each bin based on the similarity of features between bin pairs and the color information between the associated pixels. The combination of pixel-level and bin-level information mitigates artifacts introduced by the excessive noise in per-bin color estimates and inspires our neural hybrid reconstruction, which achieves much superior denoising quality after training on sizable datasets.

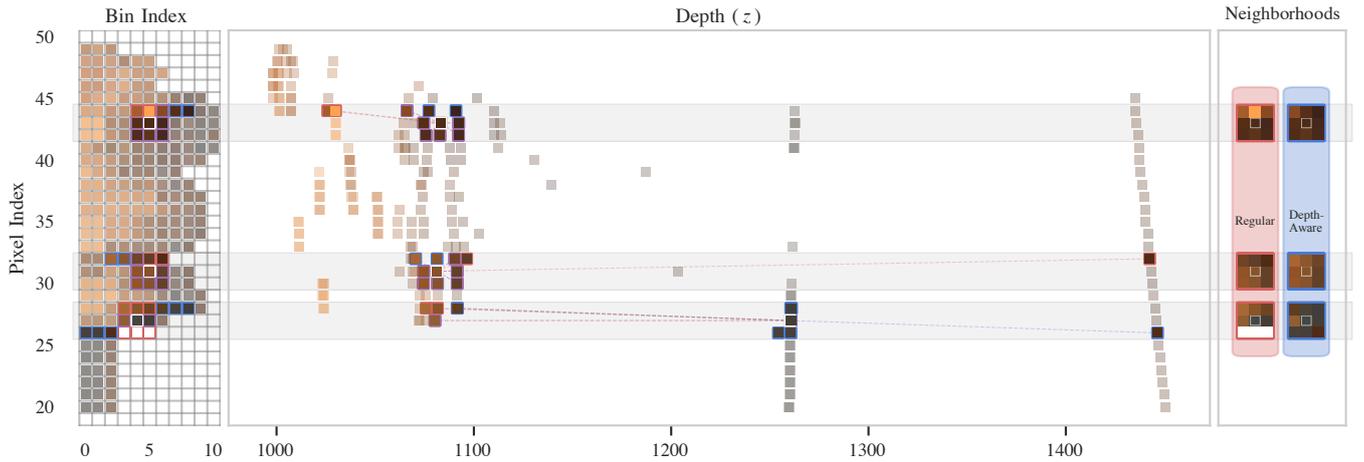


Figure 2: Visualization of the bin structure within a scanline from a rendered clean deep-Z image. On the left, we show the color of non-empty bins, and in the middle, we remap the x-axis to the scene depth, revealing the correlation between color and depth. We compare 3×3 neighborhoods for three highlighted center bins (white), showing regular (red) and depth-aware (blue) indexing. We highlight common neighbors in purple. Regularly indexed neighborhoods (red) contain bins with unrelated colors from the foreground (top example at $[43,5]$), distant background (middle example at $[31,5]$), and empty bins (bottom example at $[27,4]$). On the right, are the neighborhoods of all center bins, highlighting the disparity in color values between regularly indexed neighborhoods (red) and our depth-aware method.

3. Methodology

In this section, we lay out the details of our method. In order to handle irregularly spaced depth bins, we augment the convolution and kernel application operators to use depth-aware neighborhoods (Section 3.1). These operators are used in a U-Net architecture that processes the input in a multi-scale fashion. The scales are denoised independently and combined through combiner modules (Section 3.2). Finally, we introduce a bin–pixel hybrid denoising approach that jointly learns a flat denoiser alongside the deep-Z denoiser and combines the result of both as the final output (Section 3.3).

3.1. Depth-aware neighborhoods

Neural flat denoisers utilize a CNN to identify useful neighboring pixels, and predict per-pixel filter kernels for final reconstruction [BVM*17, VRM*18]. These 2-D kernel-predicting CNNs are not directly applicable for denoising deep-Z images due to the additional bin dimension. Still, the idea of reusing information from neighboring pixels, which motivates the use of kernel-predicting CNNs, is valid for our task, and we propose a 3-D CNN architecture that extends it along with *depth-aware indexing* of bin neighborhoods to combat misalignment in deep-Z data. Applying convolutions and denoising kernels on depth-aware bin neighborhoods is more effective at finding relevant bins in neighboring pixels.

We define the input to our 3-D kernels as a subset of neighboring bins selected based on depth information. Conventional $k_W \times k_W \times k_B$ convolutions (where $k_W = 2r_W + 1$ and $k_B = 2r_B + 1$) process feature maps with a regular cuboid grid pattern, centered around the bin $\mathbf{p} = (x, y, b)$. For our kernels, we keep the regular square pattern in the x and y dimensions. For each neighboring pixel (x', y') with $|x' - x| \leq r_W$ and $|y' - y| \leq r_W$, we first find the bin $b_{x', y'}$ closest (in

depth) to the center bin \mathbf{p} . We then add $2r_B + 1$ bins centered around $b_{x', y'}$ (i.e., bins b_c located at the pixel (x', y') with $|b_c - b_{x', y'}| \leq r_B$) to the sampling grid at location (x', y') . Repeating this for all $k_W \times k_W$ neighbors (including the center pixel) gives us a sampling grid of size $k_W \times k_W \times k_B$. We provide a visual aid in Figure 3. For brevity, we defer the formal definition to Appendix A.

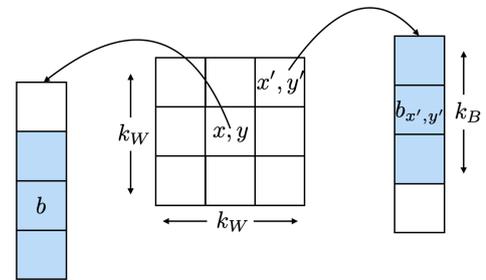


Figure 3: Illustration of depth-aware neighborhood construction around bin $\mathbf{p} = (x, y, b)$.

The difference between the depth-aware and regular neighborhoods can be seen in the 2-D illustration of Figure 2 using a scanline of a deep-Z image. For such a region containing both foreground and background, depth-aware indexing ensures the direct neighbor for a bin is close in depth, overcoming bin misalignment caused by the changing number of bins in the foreground and background within each pixel. The bottom example in Figure 2 further demonstrates the edge case when neighboring bins are empty and padded with zeros, where depth-aware neighborhoods cover close-by neighbors.

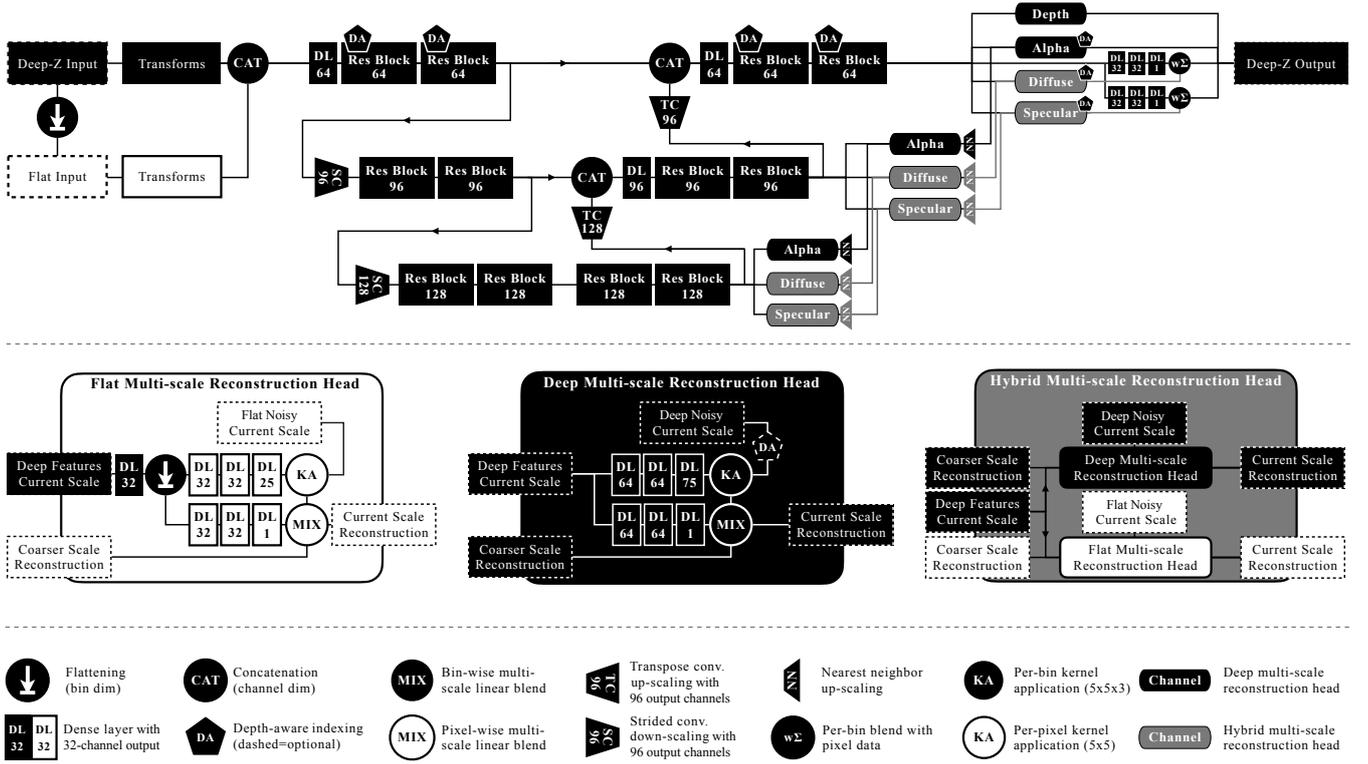


Figure 4: Network architecture of our deep-Z denoiser. Our model includes a U-Net and multi-scale kernel-based reconstruction heads (Section 3.2). Raw input data goes through some transforms (Table 2) before being passed to the U-Net. The convolution and denoising kernels on depth-aware neighborhoods (Section 3.1) are utilized at the finest scale, except for reconstructing the depth channels (front and back). Diffuse and specular channels utilize hybrid reconstruction with additional flat pixel data. The internals of our three types of reconstruction heads are shown in the middle row, and the bottom row includes legends.

3.2. Multi-scale neural deep-Z denoiser

We describe our multi-scale neural deep-Z denoiser architecture below, which simultaneously reconstructs the deep data components at multiple scales and consecutively combines them into a single deep-Z image.

Multi-scale denoising. In order to expand the receptive field of the denoiser, and to reuse bins that are farther away in the spatial (x and y) dimensions, we propose a multi-scale deep-Z denoiser architecture using a U-Net and a multi-scale reconstruction module, similar to the denoiser module described by Zhang et al. [ZMV*21] with a straightforward extension to process the bin dimension. As shown in Figure 4, the U-Net takes the full-resolution input noisy images and auxiliary features, which can be potentially transformed, and processes them via encoder and decoder residual blocks at different scales. The encoder and decoder parts at each scale are connected with a residual connection. Down-scaling and up-scaling between scales are performed by $2 \times 2 \times 1$ strided convolutions and strided transposed convolutions, respectively, and thus they only change the spatial resolution (by $2 \times$), which is typically significantly larger than the bin count.

As a direct extension of the multi-scale reconstruction from a previous flat denoiser [VRM*18], our multi-scale reconstruction

module with S scales contains S kernel prediction modules and $(S - 1)$ multi-scale combiners. At each scale s , the kernel prediction module predicts the reconstruction kernels to be applied on the down-sampled noisy input image from the U-Net feature maps at scale s . After the per-scale reconstruction, the multi-scale combiner at scale s predicts mixing weights w_s between the current scale s and the coarser scale $s + 1$ and combines them by applying the frequency decomposition [VRM*18] to each bin index b :

$$\hat{O}_{S,b} = O_{S,b}$$

$$\hat{O}_{s,b} = O_{s,b} - w_{s,b}U(D(O_{s,b})) + w_{s,b}U(\hat{O}_{s+1,b}), \quad s = S - 1, \dots, 1,$$

where $O_{s,b}$ and $\hat{O}_{s,b}$ denotes the denoised image at scale s and bin index b before and after multi-scale combining, respectively. $U(\cdot)$ and $D(\cdot)$ denote $2 \times 2 \times 1$ up- and down-sampling with nearest neighbor interpolation and average pooling, respectively. The arithmetic operations are element-wise.

Denoising multiple components and handling premultiplied alpha. In Monte Carlo denoising, particularly in final-quality scenarios, it is common to denoise different components of the light transport and combine them to create the final denoised image [BVM*17, VRM*18, XZW*19, ZMV*21]. We follow these approaches and denoise diffuse and specular components separately,

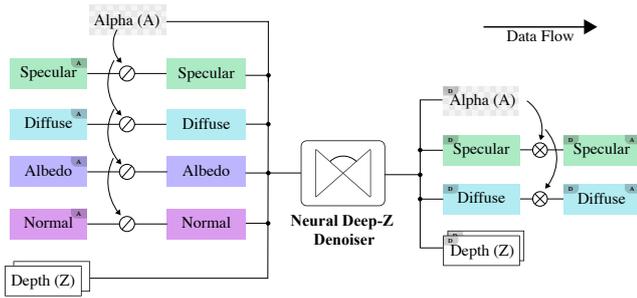


Figure 5: Pipeline of multi-component denoising. Our method denoises alpha, specular, diffuse, and depth channels, takes as input pre-multiplied channels and auxiliary features, and outputs alpha, depth, and pre-multiplied denoised diffuse and specular. The rectangles tagged with “A” denote alpha-premultiplied deep-Z channels, those marked with “D” denote denoised channels, and vice versa. Symbols \oslash and \otimes denote element-wise division and multiplication, respectively.

which are created by the renderer based on material properties at the first intersection. Finally, our method does not only denoise color channels but also the alpha and depth channels, as noise in those channels could potentially introduce visual artifacts during compositing even if the color channels are noise-free. Note that in our kernel-predicting architecture, each channel is reconstructed individually, with its own reconstruction heads.

In the standard deep-Z format, alpha is assumed to be pre-multiplied in the color channels and the auxiliary feature maps [Ope23], except for the depth channel. Differences in alpha thus lead to differences in all other channels, especially for auxiliary features, and can complicate the denoising procedure where similar bins need to be identified. We thus *unpremultiply* alpha before denoising, *i.e.*, dividing it out, on all pre-multiplied input channels, and multiply the denoised (unpremultiplied) diffuse and specular channels with the denoised alpha before summing them up as the final denoised color. Figure 5 summarizes our denoising workflow which denoises multiple components and handles alpha pre-multiplication.

3.3. Hybrid flat–deep-Z denoising

The neural deep-Z denoiser described above can produce relatively clean deep-Z images while preserving intra-pixel depth separation. However, its output often contains residual noise, as will be shown in Section 5.3.1. This can be partly attributed to the lower sample count and higher noise within input bins, compared to pixels. Also, compared with flat denoising, the additional dimension for deep-Z denoising means that it can be more difficult to gather information from the neighborhood that can provide context for denoising. For example, the bins within a pixel might have very different colors from the composited pixel itself. To achieve good denoised results both before and after flattening, we propose to combine the flat and deep-Z architectures and denote our final approach as a hybrid flat–deep-Z denoiser.

The first step towards the hybrid approach is to provide pixel context at the input level, as illustrated by the white boxes on the

left in Figure 4. In this step, each channel of the noisy deep-Z image is flattened using the noisy alpha channel, preprocessed, and concatenated to the preprocessed deep-Z image as additional feature channels for each bin. The additional pixel-wise information can improve the capability of per-bin denoising, *e.g.*, by providing aggregated context from all bins within the pixel and indicating whether the bin is at a depth discontinuity in the image.

The second step of our hybrid deep-Z denoiser is the hybrid reconstruction for the diffuse and specular channels shown on the right of Figure 4; alpha and depth channels only use deep-Z reconstruction because of the lower noise. Here we produce the final denoised deep-Z image by combining the output of the deep-Z and the flat reconstruction heads at the finest scale. As shown in the flat reconstruction head, by flattening the per-bin feature embedding from the U-Net (averaging across the bin dimension), we can obtain the per-pixel feature embedding at each scale. Taking these per-pixel feature maps as input, a flat multi-scale kernel-predicting reconstruction head can then denoise the flattened noisy image. Finally, the denoised flat image is combined with the denoised deep-Z result by another learned weighted average, whose weights are predicted from the embeddings from the finest scale of the U-Net for each bin. As will be shown in our results (Section 5.3.1), the hybrid flat–deep-Z reconstruction greatly improves the denoising result in flat regions, with minimal impact on the quality of the depth separation.

4. Implementation

Our neural deep-Z denoiser is implemented in TensorFlow 2 [AAB*15] and trained on imagery rendered with the Hyperion renderer [BAC*18]. In this section, we describe in more details the dataset, the neural network architecture and training setup.

4.1. Data

Our dataset contains a total of 516 pairs of noisy–reference deep-Z images from 100 production scenes at resolution 1920×804 . Each scene is rendered at between 4 and 7 noise levels, and the sample count ranges from 16 to 1024 samples per pixel (spp) for the noisy images. The training–validation–test split of our dataset is shown in Table 1.

Table 1: Dataset split. The validation set is used for adjusting the training hyper-parameters, and the test set for final evaluations.

	#Scenes	#Images	Purpose
Training	81	423	NN training
Validation	9	43	Hyper-param tuning
Testing	10	50	Final evaluation

Rendering deep-Z images. The bin layout is identical for each noisy–reference pair, since the bin layout and depth boundaries are created in a pre-pass [VAN*19]. In the pre-pass, paths are traced only until the first bounce to record depth values, and close-by depth values are averaged to determine the depth of a bin. As single-bounce ray tracing is inexpensive, the pre-pass for images

in our datasets uses hundreds to thousands of samples per pixel to ensure good coverage of potential depth values. After the pre-pass, full light paths are traced and each path’s color value is recorded in the bin closest to the path’s first-bounce depth. The recorded bin depth is the average depth of the color-pass samples that fall in this bin and the pre-pass samples are discarded. Notably, pixels in noisy images can receive fewer samples in the color pass than they did in the pre-pass. This results in empty bins that did not receive any color samples and have zeros in all channels, including depth. As depth values are needed for depth-aware neighbor indexing, we fill a missing bin’s depth by taking the median depth of non-empty bins in its regular neighborhood. We also ensure that all pixels remain tidy, *i.e.*, all bins are ordered by depth [Ope23]. The detailed depth filling algorithm can be found in Appendix B. Note that depth filling is *not* a requirement for our method, as renderers can be instructed to inherit the pre-pass depth value for empty bins.

Input channels and preprocessing. Our method’s input consists of the noisy color $\mathbf{c}(\mathbf{p})$, alpha $\alpha(\mathbf{p})$, depth (front $z_f(\mathbf{p})$ and back $z_b(\mathbf{p})$), diffuse $\mathbf{c}_D(\mathbf{p})$, specular $\mathbf{c}_S(\mathbf{p})$, normal vectors $\mathbf{n}(\mathbf{p})$ and albedo $\rho(\mathbf{p})$ of each bin in the noisy deep-Z image. As shown in the top-left corner of Figure 4, input preprocessing transforms are applied on the channels before they are concatenated as input to the neural network. Such input transforms can facilitate learning, *e.g.*, by mapping the raw features to a lower dynamic range. Table 2 summarizes the input transforms used in our method. First, alpha premultiplication is undone. Then, we apply the log transform $y = \log(x + 1)$ for color channels, and keep normals and albedo as is. For the depth channels, we normalize and use sine and cosine waves with 10 frequencies so that high-frequency changes along the depth dimension can be better identified. Finally, we apply three different transforms on alpha. “OverToAdd” turns the alpha values used in *over* compositing to those in *add*, representing the effective contribution of the bins to the flattened pixel value [VAN*19]. “AlphaToDensity” applies $y = \log(1 - \alpha)$ on bins with $\alpha < 1$, which roughly corresponds to the concept of “density” of a bin. “AlphaTypeMask” classifies the bins into three categories, $\alpha = 1$ (fully opaque), $\alpha = 0$ (empty) and $\alpha \in (0, 1)$ (partially transparent).

Table 2: Transforms and symbols applied to each input channel.

	Symbol	α -div	Transform
Color	$\mathbf{c}(\mathbf{p})$	✓	LogTransform
Diffuse	$\mathbf{c}_D(\mathbf{p})$	✓	LogTransform
Specular	$\mathbf{c}_S(\mathbf{p})$	✓	LogTransform
Normals	$\mathbf{n}(\mathbf{p})$	✓	Identity
Albedo	$\rho(\mathbf{p})$	✓	Identity
Front depth	$z_f(\mathbf{p})$	–	FourierFeatures
Back depth	$z_b(\mathbf{p})$	–	FourierFeatures
Alpha	$\alpha(\mathbf{p})$	–	Identity
		–	OverToAdd
		–	AlphaToDensity
		–	AlphaTypeMask

Bin merging for training. To process deep-Z images with the CNN architecture, we use empty bins as paddings so that all pixels have the same bin count. After denoising, we prune all the padding bins to preserve the original bin layout. However, this padding step can be inefficient when few pixels have a large number of bins, and the varying maximum bin count across batches can also cause memory consumption fluctuation during training. In fact, the images in our deep-Z dataset contain up to 53 bins in a pixel, and the max bin count of each image averages to 15.99, compared to the average bin count 3.35 across all pixels in all images, indicating that most of the computations could be spent on empty padding bins.

To facilitate training on modern GPUs and avoid wasting computation resources due to excessive padding, we use a bin merging method based on alpha values and proximity in depth, and process the training set to have a maximum of 8 bins per pixel. Details of the bin merging algorithm can be found in Appendix C. As shown in Section 5, though our denoisers are trained on images with reduced bin count, they can be directly applied to deep-Z images with the original bin layout and achieve high denoising quality.

4.2. Neural deep-Z denoiser architecture

As shown in Figure 4, we use 3 scales in our multi-scale neural deep-Z denoiser, both for the backbone U-Net and the final reconstruction. The encoder and the decoder parts of the U-Net each contain 2 residual blocks per scale, and each residual block contains 2 layers, either $3 \times 3 \times 3$ depth-aware convolutions (for the finest scale) or $3 \times 3 \times 3$ regular convolutions (for the coarser two scales). The convolution layers inside the residual blocks for both the encoder and decoder part at each scale, from fine to coarse, have 64, 96, and 128 output feature maps, respectively. All convolution layers use ReLU as the activation function, unless otherwise noted.

We denoise diffuse, specular, alpha, and front and back depth layers using kernel-predicting reconstruction. Each layer has its own (potentially multi-scale) kernel-predicting reconstruction module, so that the predicted denoising kernels can adapt to the content in different components. The two depth layers are denoised only at the finest scale because of their relatively low noise, using $5 \times 5 \times 3$ depth-aware kernels. 3-scale reconstruction is applied to the other layers (diffuse, specular, alpha), where the finest scale uses the same configuration as for the depth layers, and the coarser 2 scales use regular kernel prediction with the same kernel size. For per-scale kernel prediction, we use two dense layers of 64 channels prior to the kernel-predicting dense layer (softmax activation). The 3-scale flat reconstruction head uses 5×5 reconstruction kernels at each scale, predicted from two dense layers of 32 channels. Each of the multi-scale combiner for both the flat and deep-Z reconstruction heads uses 2 dense layers of 32 channels to predict per-pixel or per-bin mixing weights. The weights, with values between 0 and 1, are produced with tempered sigmoid activation function, $y = \sigma(x/5)$, where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. Experimentally, we find that stretching the sigmoid function helps stabilize the training. Finally, the flat–deep-Z mixture weights at the finest scale are also predicted using the stretched sigmoid activation. The per-layer reconstruction setup is listed in Table 3.

Table 3: Per-layer reconstruction modules. “553” under “Kernel” means $5 \times 5 \times 3$ denoising kernels, “DA” means using depth-aware kernels at the finest scale, and “Hybrid” means using the flat–deep-Z hybrid reconstruction, and the flat reconstruction head has 3 scales with 5×5 denoising kernels at each scale.

Layer	Symbol	Scales	Kernel	DA	Hybrid
Diffuse	$\mathbf{c}_D(\mathbf{p})$	3	553	✓	✓
Specular	$\mathbf{c}_S(\mathbf{p})$	3	553	✓	✓
Alpha	$\alpha(\mathbf{p})$	3	553	✓	–
Front depth	$z_f(\mathbf{p})$	1	553	–	–
Back depth	$z_b(\mathbf{p})$	1	553	–	–

4.3. Losses and training setup

We rely on SMAPE [Flo86] and its variants as the training losses for our neural deep-Z denoiser because of its proven ability as the loss function for training flat denoisers for renderings [VRM*18, ZMV*21]. Given a reference image $\hat{\mathbf{I}}$ and a denoised image $\hat{\mathbf{I}}$, defined on the same image grid (or bin layout), SMAPE between the images is computed as an average of the per-pixel (or per-bin) values:

$$\text{SMAPE}(\hat{\mathbf{I}}, \hat{\mathbf{I}}) = \sum_{p \in \mathcal{A}} \frac{|\hat{\mathbf{I}}(p) - \hat{\mathbf{I}}(p)|}{|\hat{\mathbf{I}}(p)| + |\hat{\mathbf{I}}(p)| + 10^{-2}}, \quad (1)$$

Where \mathcal{A} contains the indices of all pixels (or bins). For diffuse, specular and alpha layers, we supervise both at the bin level and at the flattened pixel level, *i.e.*, flattening both reference and denoised images with corresponding alpha maps before computing the loss. For layers with the hybrid reconstruction, we also directly supervise the flat denoiser’s output. The training losses for different layers are summarized in Table 4. The denoising of all layers are trained end-to-end with the total loss equal to the sum of all individual losses in the table.

Table 4: Per-layer SMAPE loss configuration. “Bin” and “Pixel” denote the use of bin-wise and pixel-wise (flattened) losses on the final output, and “H-Flat” is for direct supervision of the flat reconstruction in the hybrid architecture. The final column indicates the use of a log transform before loss computation of depth layers. All losses marked with “✓” have the same weight.

Layer	Symbol	Bin	Pixel	H-Flat	Transform
Diffuse	$\mathbf{c}_D(\mathbf{p})$	✓	✓	✓	–
Specular	$\mathbf{c}_S(\mathbf{p})$	✓	✓	✓	–
Alpha	$\alpha(\mathbf{p})$	✓	✓	–	–
Front depth	$z_f(\mathbf{p})$	✓	–	–	$\log(x+1)$
Back depth	$z_b(\mathbf{p})$	✓	–	–	$\log(x+1)$

The denoisers in this work, if not mentioned otherwise, are trained with the Adam optimizer [KB14] with an initial learning rate of 10^{-4} (other optimizer parameters set to default). Each batch

during training contains a single 128×128 patch of deep-Z image from our training set, and can be randomly flipped and rotated along the spatial dimensions for data augmentation purposes. Learning rate decay is applied 4 times during training, namely at 50%, 62.5%, 75%, 87.5% of the training duration of 786K total iterations, which takes about 2.5 days on a single NVIDIA RTX A6000 GPU.

5. Results

In this section, we report the evaluation results of our neural deep-Z denoiser on our test set with 10 scenes via quantitative metrics and visual comparisons. First, we show the effectiveness of our proposed deep-Z denoiser by comparing with the previous state-of-the-art deep-Z denoiser Vicini19 [VAN*19], which is a non-neural method based on feature-guided non-local means [RMZ13]. To assess the final denoising quality, we compare our method with a flat neural denoiser (denoted as Flat) trained on the flattened version of our training set, using similar hyperparameters. Finally, we justify the main decisions in our method through ablation studies, including the use of depth-aware bin neighborhoods and the hybrid flat–deep-Z denoising architecture, and discuss the method’s generalization capability by evaluating on an alternative test set. We use SMAPE (Equation (1)) and DSSIM ($1 - \text{SSIM}$ [WBSS04]) as error metrics, which are computed on flat images or flattened deep-Z images after denoising, unless otherwise noted. We include 5 noise levels in the evaluation, as indicated by sample count: 16spp, 32spp, 64spp, 128spp and 129+spp, with the last including images from 129 to 256 samples per pixel. Results on all scenes in our test set can be found in the supplemental viewer.

5.1. Effect of neural denoiser

The Vicini19 deep-Z denoiser relies on feature buffers and variance estimates to create filter kernels for each bin. Our implementation of Vicini19 uses 19×19 spatial kernel sizes and 7×7 patches around pixels. To denoise the bin color, Vicini19 computes two kernels, one using per-bin auxiliary feature similarity and the other using per-pixel non-local means; each bin’s contribution is weighted by its effective, or *add*, alpha (see Section 4.1). Note that Vicini19 requires half buffers and per-bin variance estimates, whereas our method only uses the noisy color channels and two auxiliary features, which is more practical considering the typically large size of deep-Z images.

In terms of timings, an unoptimized TensorFlow implementation of our method can denoise a typical deep-Z image at 1920×804 resolution in 5 minutes compared to more than 20 minutes for Vicini19 on an AMD Ryzen 7 5800X 8-Core CPU with 16 threads. On an NVIDIA RTX 3090 GPU, the denoising time of our method drops to around 7.5 seconds with the help of XLA.

Figure 6 summarizes the error metric comparison between our deep-Z denoiser and Vicini19, where metrics are computed on denoised flat images or flattened denoised deep-Z images. Both deep denoisers receive as input the original deep-Z image topology without any bin merging, despite our model being trained on merged data with up to 8 bins. In these comparisons, our method significantly outperforms Vicini19 at all noise levels and metrics.

When comparing our method with Vicini19 across different sample counts, our method always achieves lower error than Vicini19 even with half the samples in terms of SMAPE. The advantage is significantly higher with DSSIM as Vicini19 requires more than four times the sampling budget to match the quality of our method.

The comparisons mentioned above focus on per-pixel metrics only evaluated on the entirely flattened versions of the denoised deep-Z images. To quantitatively evaluate the depth-resolved quality of the deep denoising methods we perform an additional experiment in Figure 7. The resulting denoised deep-Z images are clipped either from the front or the back and flattened, before the per-pixel DSSIM error is computed with the correspondingly clipped reference. This evaluation can capture artifacts along the depth dimension such as blurring or leaking across depth boundaries that could otherwise be hidden by the entirely flattened comparisons. In both cases, as we move from left to right, more bins are retained. As shown in the plot, our method (blue) produces lower errors than Vicini19 (red) on depth-clipped images, demonstrating better preservation of the depth-resolved information even though our denoiser can blend in flattened reconstructed data at each bin.

Finally, columns 4 and 5 of Figure 8 show visual comparisons

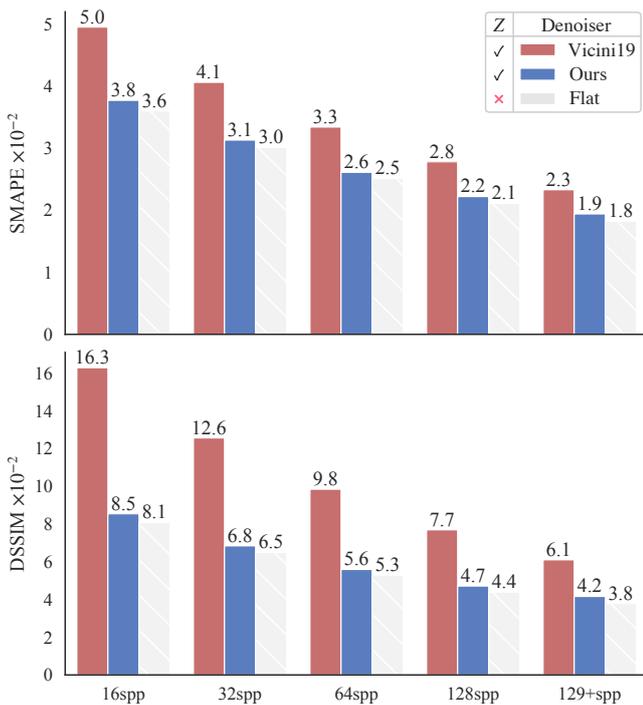


Figure 6: Quantitative comparison between the non-neural deep-Z denoiser (Vicini19) and our method (Ours) at different noise levels. The reported metrics for Vicini19 and Ours are computed on flattened versions of respective denoised deep-Z images. We observe significant improvement with our method across all metrics and noise levels compared to Vicini19. For context, we also provide the results of a flat neural denoiser (Flat) which we apply after flattening the noisy deep-Z image. This method destroys the depth structure in the deep-Z image and prohibits deep compositing.

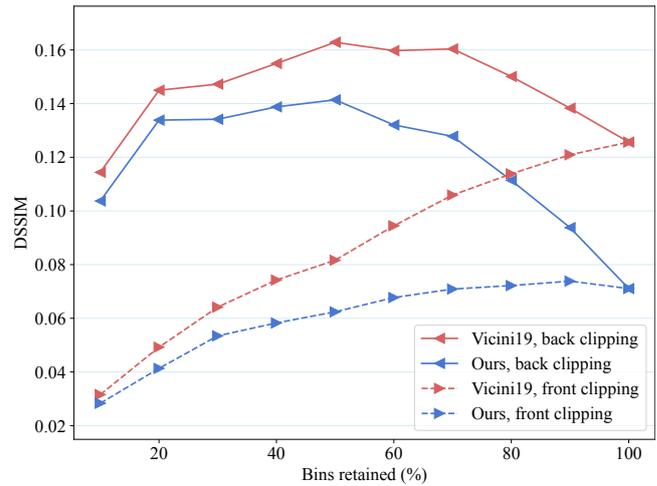


Figure 7: Depth-resolved denoising quality. In this comparison between Ours and Vicini19, we perform depth clipping on the denoised deep-Z data at 32spp. We then flatten the clipped deep-Z image and measure the DSSIM error with the corresponding reference. The horizontal axis indicates the percentage of bins kept after clipping, and the vertical axis shows the aggregated error over the entire test set. Here “back clipping” means removing the most distant bins in terms of depth, and “front clipping” means removing the closest ones, and the two curves for each method converge at 100% bins retained, which is the flattened full image.

between our method and Vicini19, on the full depth range and on depth slices. Compared to Vicini19, our method preserves the correct details (red inset of rows 1-3) and mitigates denoising artifacts (orange inset of rows 1-3). Our method achieves better quality in out-of-focus regions (row 4, orange inset), which require gathering information from relevant bins in neighboring pixels. Finally, in the depth-sliced result in the last row, where an out-of-focus foreground object was removed, our method does not produce artifacts in the noisy region behind the removed object.

In summary, our method achieves significantly higher reconstruction quality than Vicini19, requires fewer channels in the noisy input, and is much faster, especially when a GPU can be utilized.

5.2. Flat vs. deep-Z denoisers

Flat kernel-predicting denoisers represent the state of the art for denoising Monte Carlo renderings, and it is of interest to assess the final quality achieved by our neural deep-Z denoiser with a flat denoiser as a baseline. It is worth reminding that it is challenging for flat denoisers to support deep compositing workflows, such as the examples demonstrated in Figure 1, since they do not preserve depth separation, and also because denoising after deep compositing can change the intention of the compositing operations, e.g., by introducing artifacts on inserted clean objects.

Our flat denoiser baseline also uses a 3-scale U-Net and 3-scale kernel-based reconstruction with a multi-scale combiner, with layer settings similar to that of our deep-Z denoiser but in 2-D. It is

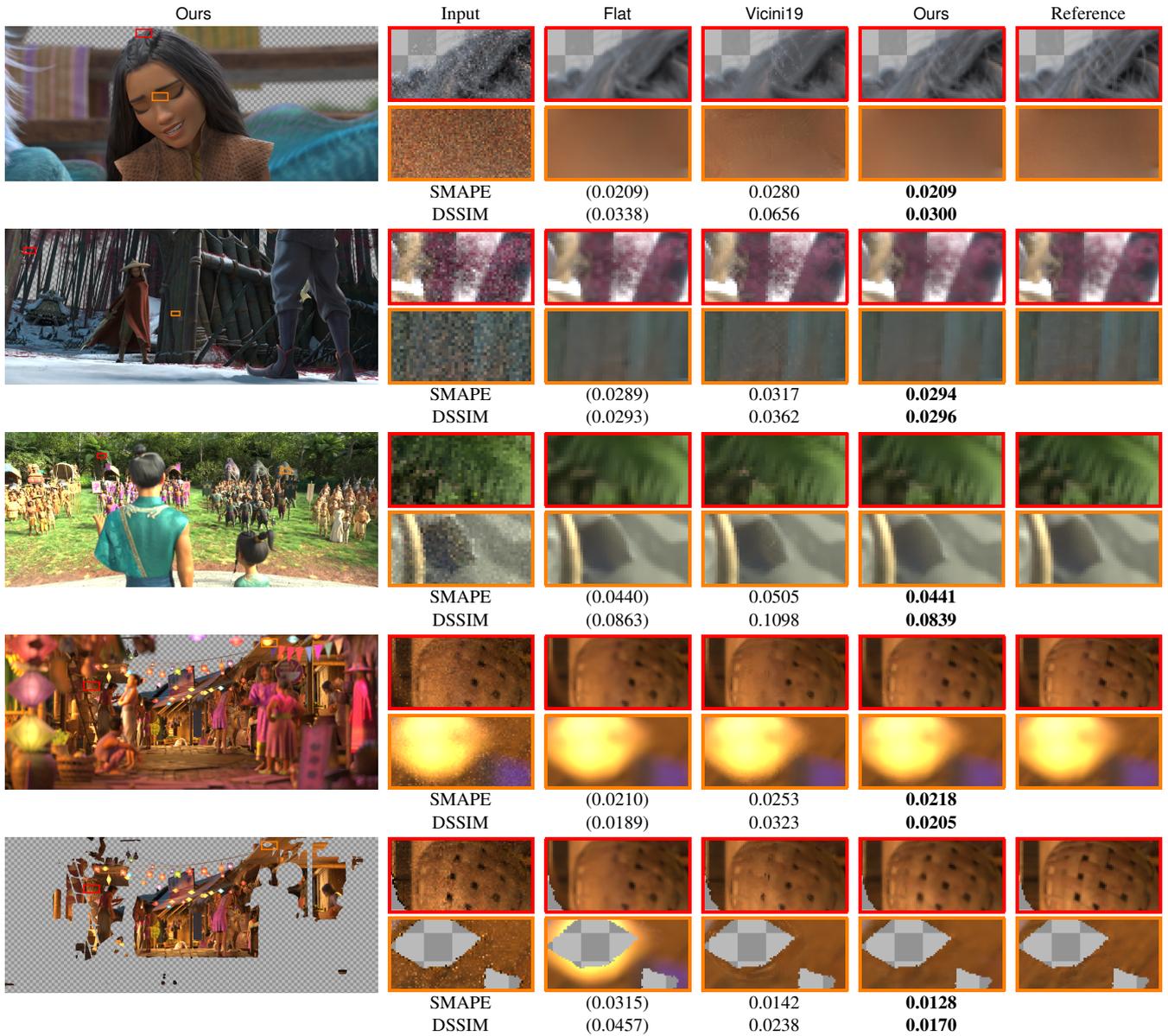


Figure 8: Comparing the denoising results of our method (Ours) with the previous non-neural deep-Z denoiser (Vicini19) on deep-Z images, both visually and through metrics computed on the flattened denoised images. We also show the results from a flat denoiser (Flat) that was applied to the flattened images. This is only for reference purposes and we mark the metrics of Flat with parentheses. As evidenced in the bottom row where a blurred foreground object was removed after denoising, the flat denoiser cannot be applied prior to deep compositing. © Disney

trained to denoise diffuse, specular, and alpha, but *not* depth. Appendix D lists the details of the architecture of this denoiser. It was trained on the flattened version of our training set using SMAPE on each denoised layer, with a similar training schedule, requiring approximately 1.5 days to reach the same number of training iterations.

In addition to the comparison between Vicini19 and our method, Figure 6 also compares the deep-Z denoisers (denoise before flattening) with this neural flat denoiser (denoise after flattening). De-

spite the fact that the latter produces a flat denoised image that prohibits deep compositing, it does provide context on the quality of our neural deep-Z denoiser. In terms of the average metrics, we observe that the quality of our deep denoiser is slightly worse than the neural flat denoiser (Figure 6). This small discrepancy measured on the entirely flattened denoised results can be further reduced by using a higher weight on the flattened component of the training loss of our method. This change, though, can hinder the quality of the

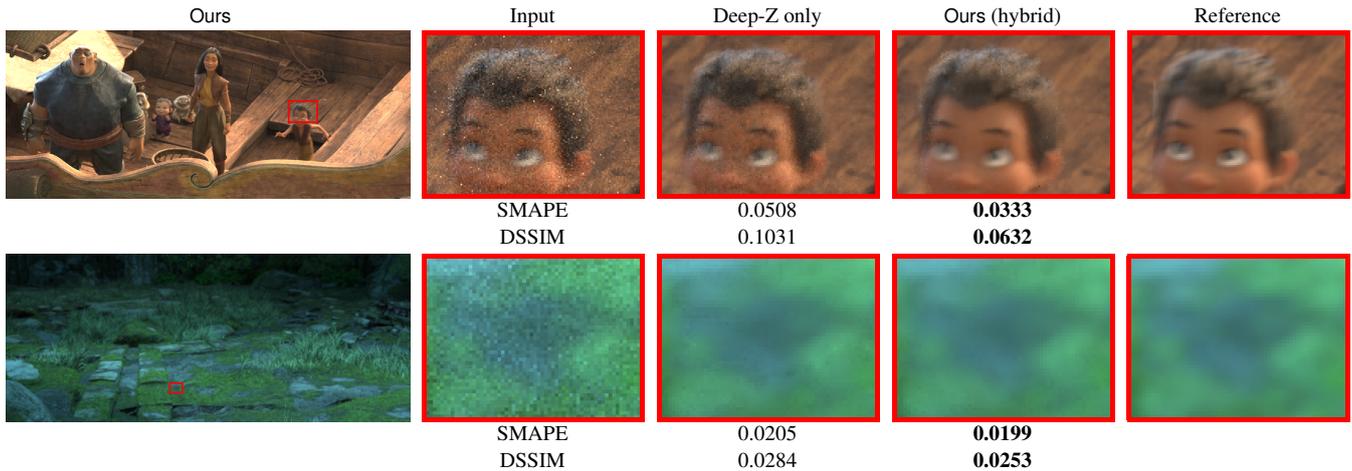


Figure 9: Comparing the hybrid flat–deep-Z denoising (Ours) with a deep-Z-only approach. The flat components help improve the reconstruction quality in regions with high noise or less depth structures. © Disney

depth separation and consequently produce artifacts during deep compositing.

We additionally include examples from the flat denoiser in our qualitative evaluation (Figure 8). Despite the overall lower error, the flat denoiser can over-blur regions with complex depth information (red inset for rows 1 and 3), which the two deep denoisers, and especially our method, preserve better. Finally, the last row of this figure demonstrates the degradation of depth separation when denoising flat data. Specifically, in this example, the input of the flat denoiser contains aggregated depth and color information at each pixel, which, in the case of out-of-focus boundaries, cannot be decomposed cleanly into the foreground and background. Both deep denoisers can prevent such color-leaking artifacts along depth.

5.3. Ablation studies

In this section, we validate the effects of the hybrid flat–deep-Z denoising and the depth-aware neighborhoods by removing the corresponding component from our architecture, and train and evaluate the resulting model under the same settings as mentioned above. We also apply our model on imagery from another renderer, in order to evaluate the generalization of the method.

5.3.1. Hybrid flat–deep-Z denoising

Table 5: Average error metrics on the test set for our hybrid method and a deep-Z only method.

Sample count (spp)	Ours		Deep-Z Only	
	SMAPE	DSSIM	SMAPE	DSSIM
16	0.0377	0.0854	0.0407	0.0963
32	0.0313	0.0684	0.0338	0.0756
64	0.0261	0.0559	0.0282	0.0602
128	0.0222	0.0471	0.0233	0.0484
129+	0.0194	0.0416	0.0197	0.0410

The hybrid flat–deep-Z architecture is designed to provide the

deep-Z denoiser with pixel context and to offer a shortcut for denoising flat regions by combining a flat reconstruction with each bin of the deep-Z output. The mean metrics on flattened images at each spp level are listed in Table 5 for the hybrid architecture compared to a purely deep architecture. It can be observed that at all except the highest spp levels, the hybrid architecture outperforms the deep-Z-only method for both metrics. The advantage of using a hybrid architecture is more prominent at lower sample counts, *i.e.*, with more noisy input. This can be attributed to bins in the noisier images being more susceptible to insufficient sampling, resulting in highly fluctuating and unreliable values in their bin neighborhood. The flat components in the hybrid architecture provide the critical context of the neighborhood, resulting in better reconstruction. As more samples accumulate in the bins, the bin neighborhood values become more stable, and the impact of the flattened context subsides. Figure 9 shows visual comparisons between the two methods. The character’s face in the first example is primarily flat, and the grass field in the second example is out of focus and has a relatively simple depth structure. When only reconstructing each bin, residual noise is noticeable in the flat or highly blurry regions, and the hybrid approach gives a much smoother reconstruction than the deep-Z-only approach. Also, as showcased previously in Figure 1 and Figure 8, the hybrid approach does preserve the depth separation, indicating that the learned combination can learn to selectively use the flat denoising result as needed. These comparisons confirm the benefit of the hybrid flat–deep-Z denoising approach.

5.3.2. Depth-aware neighborhoods

As described in Section 3.1, our depth-aware neighborhoods are designed to address artifacts caused by bin misalignment, as illustrated in Figure 2, and we provide visual results that demonstrate this in Figure 10. The non-depth-aware method uses a naive 3-D CNN with regular indexing in the input of convolution layers and denoising kernels for each bin. The first example showcases a scene with a very complex depth structure. The naive 3-D CNN approach struggles to find useful neighboring bins and produces results with artifacts and residual noise. Our method, augmented

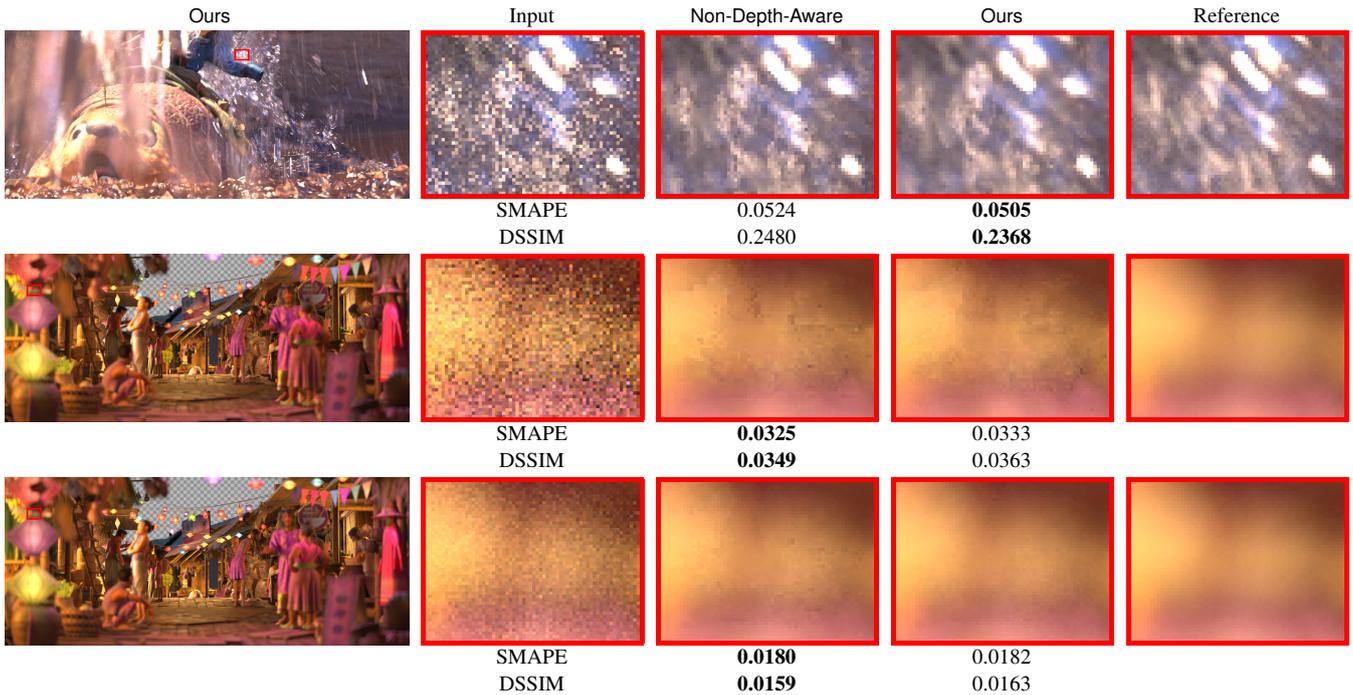


Figure 10: Comparing our method with a non-depth-aware variant. The first three examples show the artifacts from the regular convolution and kernel reconstruction operators. The second and third rows are different sampling levels of the same scene, showcasing the difficulty of mitigating such artifacts even with a significantly increased sample count (8x). © Disney

with the lightweight depth-aware indexing, is able to locate useful neighbors and blend them to create a more accurate reconstruction. The second example focuses on a region that is heavily out of focus but exhibits substantial bin misalignment due to the randomness in the depth values, as the blurred object covers a large depth range. The misalignment causes artifacts in the non-depth-aware approach, which are mitigated by using the depth-aware neighborhoods. In the third example, we increase the input sample count by a factor of 8 (from 32 to 256 spp), and we observe that these artifacts are still visible for the non-depth-aware approach.

Table 6: Average error metrics on the test set for our deep-Z denoiser with and without depth-aware neighborhood indexing. The two methods are very close in error metrics, and we refer the readers to the main text and figures for discussion on the effect of the depth-aware method.

Sample count (spp)	Ours		Non-Depth-Aware	
	SMAPE	DSSIM	SMAPE	DSSIM
16	0.0377	0.0854	0.0377	0.0850
32	0.0313	0.0684	0.0312	0.0682
64	0.0261	0.0559	0.0260	0.0561
128	0.0222	0.0471	0.0221	0.0474
129+	0.0194	0.0416	0.0193	0.0417

Though our method avoids reconstruction artifacts caused by bin misalignment, such cases only cover a minority of pixels in our test set, and on average, the methods with and without depth-aware

neighborhoods perform similarly in terms of the metrics, as listed in Table 6. Notably, depth-aware indexing operates with the assumption that, within a pixel, each surface is only represented by a single bin. As bins are created by closeness in depth, this assumption might be violated if a pixel contains slanted surfaces, *i.e.*, a single surface with a large depth range. There, depth-aware indexing might not be optimal, because the bins in the front of the center pixel will be closest to bins in the back of a neighboring pixel. In this case, there can be residual noise in the output, as shown in Figure 11. We argue that the problem is mainly caused by the renderer’s binning algorithm rather than the denoiser, and can be mitigated by more appropriately creating and merging bins. To validate this, we preprocess the datasets and merge bins that are close within 1% depth distance. A network trained on the new training dataset, denoted “Ours (Z-merged)” in Figure 11, does not suffer from the residual noise.

5.3.3. Generalization to new content

Our model is trained on a relatively small dataset, as described in Section 4.1, and all images in this dataset are from the same production, rendered with the same renderer. This is mainly caused by data availability restrictions, as some features were not supported by the renderer’s deep-Z mode, and manual modification was necessary for each scene; the large size of deep-Z images also limited the number of renderings. Though we did not observe overfitting during training, it is of interest to evaluate whether the current model can perform well on data from other productions, or even other renderers. Therefore, we apply the trained deep-Z denoiser on an

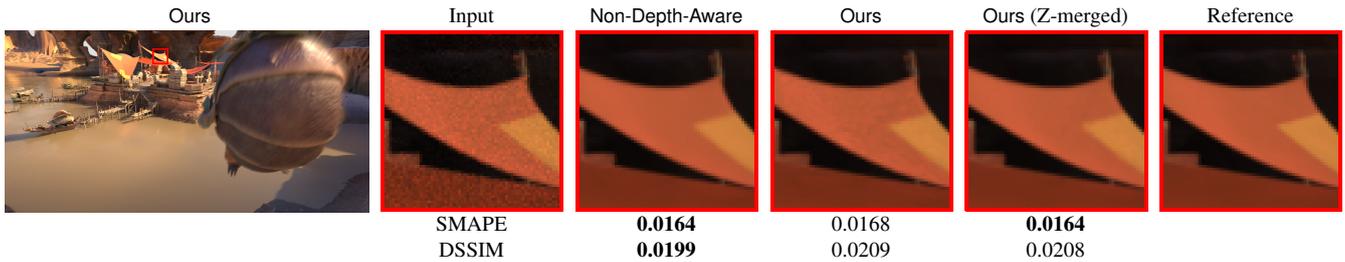
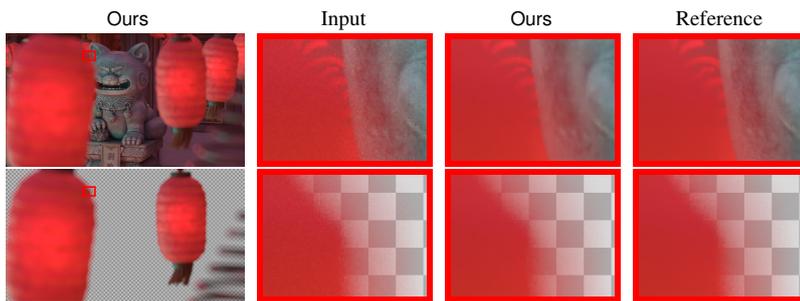
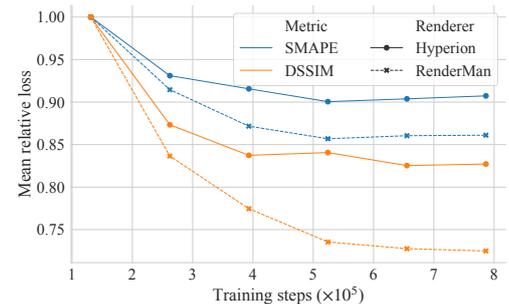


Figure 11: Failure case of depth-aware neighborhood when multiple bins are created for a single slanted surface. By merging bins based on depth more aggressively, the artifact can be mitigated, suggesting the potential benefit of more advanced binning and indexing methods. © Disney



(a) Visual example of deep-Z denoising on new data. The second row shows the result after removing the background. © Disney / Pixar



(b) The progress of mean relative losses (computed on flattened images) during training.

Figure 12: Generalization ability of our method. **(a)** Our model can denoise test images from a different production and renderer, despite not having trained on imagery from this production or renderer. **(b)** Losses on these new test images decrease during training in a similar manner as losses on our original training set.

additional test set from another production, rendered with RenderMan [CFS*18]. This test set contains 9 scenes rendered at 16–256 spp. As can be seen in Figure 12(a), our method can suppress the noise while preserving the separation between the foreground and background, despite not having been trained on data from RenderMan.

Furthermore, to examine possible overfitting, we evaluate the model with checkpoints at 6 different training iterations of the same model. Figure 12(b) shows the progress of mean relative losses on Hyperion (images from the same production as the training set, Section 4.1) and RenderMan test sets, with respect to losses at the earliest checkpoint. As can be noticed, losses on the RenderMan test set decrease with the same trend as those on Hyperion images. Also, the RenderMan relative losses are lower than the Hyperion ones, which can be explained by a lower residual noise level in the reference images.

6. Discussion and future work

As illustrated in previous sections, our hybrid neural deep-Z denoiser is capable of achieving high-quality reconstruction from rendered deep-Z images while preserving the depth separation within each pixel. In this section, we discuss the limitations of the current method and directions for potential future work.

Final denoising quality and data availability. Despite getting close to a flat denoiser’s results (Section 5.2), our method still can

be improved in terms of the quality of the final flattened image, e.g., through hyper-parameter tuning such as loss weighting. Moreover, as discussed in Section 5.3.3, challenges for creating a large number of deep-Z renderings have resulted in a relatively small dataset for training our method, which also leaves room for quality improvement. As an indicator of quality, we run Intel’s Open Image Denoise (OIDN) [Áfr23] on the flattened version of our test images, and compare it with our method and Flat. On average, OIDN achieves slightly better metrics than Flat, at around 1% lower SMAPE and 4% lower DSSIM, a distance smaller than that between our method and Flat. Such a difference is likely due to the much larger training set for OIDN. Two visual examples are shown in Figure 13. It can be seen that OIDN can better reconstruct textured details than models we trained, indicating possible quality improvement, especially for auxiliary feature usage, from larger training sets. However, in the second example, OIDN struggles in regions with hair, which could be atypical of its training set, again showing the importance of training data. Apart from gathering additional data, it would also be desirable to use flat denoisers trained on larger datasets to improve the quality of deep-Z denoisers for which training data is scarcer due to the cost of generating and storing deep data. Our current hybrid approach only uses a flat reconstruction module, but a fully flat, potentially pre-trained denoiser can allow the deep-Z part of the network to focus on details and bin separation.

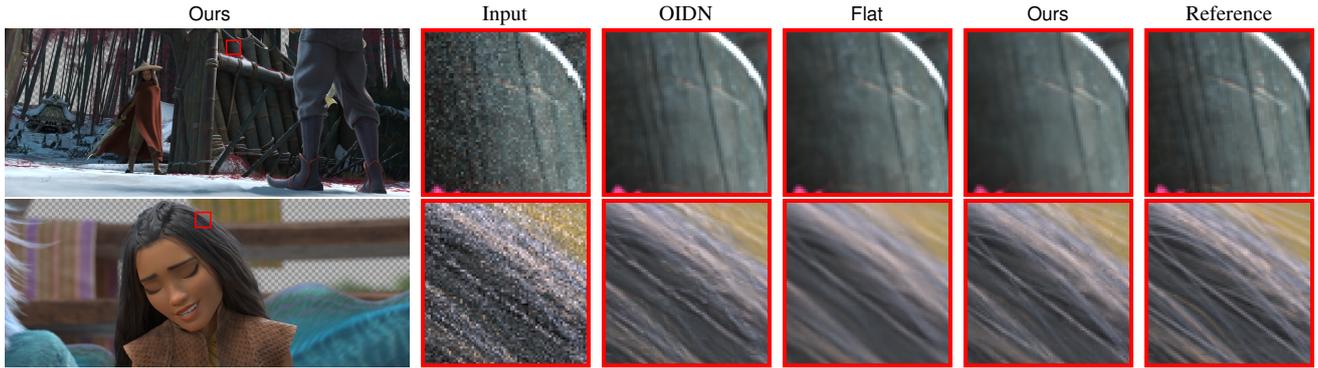


Figure 13: Comparing Intel’s Open Image Denoise (OIDN) with the flat denoiser trained on our dataset (Flat) and our proposed method (Ours). OIDN produces better reconstruction in textured areas, but creates artifacts with hair. © Disney

Artifacts from depth denoising. The output from our current approach is suitable for deep compositing operations because the depth separation is preserved and noise reduced, and this is achieved by joint denoising of color, alpha and depth channels. However, in rare cases, our method can produce incorrect denoised depth values. This can be observed in Figure 14 where the background is clipped away, but some background pixels still light up. Most problematic cases happen at input bins that are empty, as mentioned in Section 4.1. These artifacts can be cleaned up by reverting to the input (filler) depth for empty bins, but we believe better depth denoising, *e.g.*, by using more loss terms or through hard constraints on output value ranges, can help prevent such cases from the source.



Figure 14: Depth denoising artifacts (top) and result after using filler depth for empty bins (bottom). The results are shown for a 32spp input image with the background removed. © Disney

Identical bin layout through noise levels. In our dataset, all images from the same scene share the same bin layout that was created in a pre-pass. While it enables the direct application of bin-wise losses for training, the identical bin layout assumption does not hold for all renderers that support rendering deep-Z images. The bin layout can also be gradually constructed during rendering, where each sample can create a new bin or be included in an existing bin, and different sample counts can result in different numbers of bins. It is interesting to explore how to effectively train a deep-Z denoiser on noisy–reference pairs with different bin layouts, and we believe that it is possible to apply the depth-aware principle when finding

the corresponding bins between noisy and reference to compute losses.

Learned bin layout. Our method does not change the bin layout of the input image in terms of the per-pixel bin count. However, it is sometimes desirable to change the bin layout after rendering, that is, to increase or reduce the bin count for some pixels. In low-sample-count scenarios, pixels can have missing bins due to insufficient sampling, and on the other end, there can be redundant bins that could be merged for increased processing and storage efficiency. It is interesting to see whether it is possible to use depth-aware neural networks to construct new bin layouts that better suit downstream tasks.

Alternative neural architectures for semi-structured data. Our current method relies on an adapted convolutional neural network to perform deep-Z denoising. In terms of handling deep-Z images with a variable number of bins per pixel, other architecture options can be explored, including graph neural networks (Section 2), recurrent neural networks, or transformers. The former can represent a bin’s neighborhood as edges in a graph, and the latter two have been proven to work well with natural language processing tasks where sequences of different lengths are common.

Deep-Z and deep-objectID. Though deep-Z is the format of standard “deepEXR” defined by OpenEXR [Kai13a], alternative formulations of deep images exist, where each bin is not necessarily associated with a depth value. Deep-ObjectID is one of such formulations [CLC16,Hil18], which associates each bin with the ID of the first-intersection primitive. Deep-ObjectID images provide the possibility to perfectly separate each object in the scene. Our deep-Z denoising method is not directly applicable to deep-ObjectID images that do not contain depth information, and care needs to be taken when using object IDs, as they are typically arbitrary hash values of object name strings [Hil18] and the definition of an “object” can vary from scene to scene, resulting in possibly small numbers of available neighbors. Finally, combining deep-Z and deep-ObjectID can potentially lead to higher denoising quality than relying solely on either one.

Deep volumes. Volumetric effects, such as smoke, fog, clouds, or explosion, form one interesting application for deep compositing

because of their natural transparency [Duf17] and their high cost of rendering. Our current method was not tested on rendered volumetric effects due to data availability, but there is no fundamental limitation on applying our hybrid depth-aware deep-Z denoiser to such effects given sufficient training data.

Temporal deep-Z denoising. In this work, we address denoising single deep-Z images, and it is natural to consider denoising deep-Z image sequences for higher temporal stability and quality at the same sample count. In the case of flat image denoising, pixels from neighboring frames are aligned via motion vectors [VRM*18, IFME21], which provide pixel-to-pixel correspondence between frames. Such a step can also be used to align deep-Z images at the pixel level, but it could be more interesting to explore possibilities to align at the *bin* level, that is, to find out which bin(s) in the previous frame correspond to a bin in the current frame. Practical issues, such as memory consumption, may also arise for deep-Z denoisers processing frame sequences.

7. Conclusion

In this work, we introduced a neural kernel-predicting denoising method for deep-Z images which preserves their depth structure and enables deep compositing workflows. Our hybrid multi-scale deep-Z denoiser combines a flat and a deep-Z reconstruction result through learned weights, which ensures high quality in regions with and without depth structure. Depth-aware neighborhood indexing is used to align neighboring bins and more effectively gather information, which avoids artifacts caused by a naive 3-D extension of a kernel-predicting denoiser. Through evaluation on a production-quality dataset, we demonstrate that our method significantly outperforms the current state-of-the-art deep-Z denoiser, reaching the same quality at half or even a quarter of the sampling budget, while operating at a much-improved speed. Our method produces denoised images with quality close to that of a recent neural flat denoiser, while preserving depth separation and supporting deep compositing workflows. We believe that the quality and efficiency of our deep-Z denoiser addresses a major challenge for efficiently producing clean path-traced deep-Z images for deep compositing workflows and can be a significant addition to future production pipelines.

Acknowledgements

We thank David Adler for preparing the dataset described in Section 4.1, and Violaine Fayolle for refining Figures 1 and 5. We also thank Akshay Shah, Mark Meyer, Per Christensen, and Andre Mazzone for discussions throughout the project. Finally, we appreciate the anonymous reviewers' constructive comments that helped improve the quality of the paper greatly.

References

- [AAB*15] ABADI M., AGARWAL A., BARHAM P., ET AL.: TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: <http://tensorflow.org/>. 6
- [Áfr23] ÁFRA A. T.: Intel® Open Image Denoise, 2023. <https://www.openimagedenoise.org/>. 13
- [Arn23] ARNOLD: Deep exr - user guide, 2023. Accessed: 2023-09-30. URL: https://help.autodesk.com/view/ARNOL/ENU/?guid=arnold_user_guide_ac_output_aovs_ac_deep_exr_html.1
- [BAC*18] BURLEY B., ADLER D., CHIANG M. J.-Y., DRISKILL H., HABEL R., KELLY P., KUTZ P., LI Y. K., TEECE D.: The design and evolution of disney's hyperion renderer. *ACM Trans. Graph.* 37, 3 (July 2018). URL: <https://doi.org/10.1145/3182159>, doi:10.1145/3182159. 6
- [BVM*17] BAKO S., VOGELS T., MCWILLIAMS B., MEYER M., NOVÁK J., HARVILL A., SEN P., DEROSE T., ROUSSELLE F.: Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graphics (Proc. SIGGRAPH)* 36, 4 (July 2017), 97:1–97:14. 3, 4, 5
- [BWM*23] BALINT M., WOLSKI K., MYSZKOWSKI K., SEIDEL H.-P., MANTIUK R.: Neural partitioning pyramids for denoising monte carlo renderings. In *ACM SIGGRAPH 2023 Conference Proceedings* (New York, NY, USA, 2023), SIGGRAPH '23, Association for Computing Machinery. URL: <https://doi.org/10.1145/3588432.3591562>, doi:10.1145/3588432.3591562. 2
- [CFS*18] CHRISTENSEN P., FONG J., SHADE J., WOOTEN W., SCHUBERT B., KENSLER A., FRIEDMAN S., KILPATRICK C., RAMSHAW C., BANNISTER M.: Renderman: An advanced path-tracing architecture for movie rendering. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 30. 13
- [CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., SCHIED C., SALVI M., LEFOHN A., NOWROUZSAHRAI D., AILA T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graphics (Proc. SIGGRAPH)* 36, 4 (July 2017), 98:1–98:12. 3
- [CLC16] CORVAZIER C., LEGROS B., CHIKH R.: Openexr/ld isolate any object with a perfect antialiasing. In *ACM SIGGRAPH 2016 Posters* (New York, NY, USA, 2016), SIGGRAPH '16, Association for Computing Machinery. URL: <https://doi.org/10.1145/2945078.2945136>, doi:10.1145/2945078.2945136. 14
- [DAN19] DAHLBERG H., ADLER D., NEWLIN J.: Machine-learning denoising in feature film production. In *ACM SIGGRAPH 2019 Talks* (New York, NY, USA, 2019), SIGGRAPH '19, Association for Computing Machinery. URL: <https://doi.org/10.1145/3306307.3328150>, doi:10.1145/3306307.3328150. 3
- [DQX*17] DAI J., QI H., XIONG Y., LI Y., ZHANG G., HU H., WEI Y.: Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 764–773. 3
- [Duf17] DUFF T.: Deep compositing using lie algebras. *ACM Trans. Graph.* 36, 3 (jun 2017). URL: <https://doi.org/10.1145/3023386>, doi:10.1145/3023386. 15
- [Flo86] FLORES B. E.: A pragmatic view of accuracy measurement in forecasting. *Omega* 14, 2 (1986), 93–98. URL: <https://www.sciencedirect.com/science/article/pii/0305048386900137>, doi:https://doi.org/10.1016/0305-0483(86)90013-7. 8
- [Fou23] FOUNDRY: Deep compositing in nuke, 2023. Accessed: 2023-09-30. URL: https://learn.foundry.com/nuke/content/comp_environment/deep/deep_compositing.html.3
- [GLA*19] GHARBI M., LI T.-M., AITTALA M., LEHTINEN J., DURAND F.: Sample-based monte carlo denoising using a kernel-splatting network. *ACM Trans. Graph.* 38, 4 (2019), 125:1–125:12. 3
- [Hil18] HILLMAN P.: A scheme for storing object id manifests in openexr images. In *Proceedings of the 8th Annual Digital Production Symposium* (New York, NY, USA, 2018), DigiPro '18, Association for Computing Machinery. URL: <https://doi.org/10.1145/3233085.3233086>, doi:10.1145/3233085.3233086. 14
- [HSDC11] HECKENBERG D., SAAM J., DONCASTER C., COOPER C.: Deep compositing, 2011. URL: <https://>

- [//web.archive.org/web/20110713112546/http://www.johannessaam.com/deepImage.pdf](http://web.archive.org/web/20110713112546/http://www.johannessaam.com/deepImage.pdf). 3
- [HY21] HUO Y., YOON S.-E.: A survey on deep learning-based monte carlo denoising. *Computational visual media* 7 (2021), 169–185. 3
- [IFME21] IŞIK M., FISHER M., MULLIA K., EISENMANN J.: Interactive Monte Carlo Denoising using Affinity of Neural Features. *ACM Trans. Graph* 40 (2021). doi:10.1145/3450626.3459793. 15
- [Kai13a] KAINZ F.: Interpreting openexr deep pixels, 2013. Accessed: 2023-05-20. URL: <https://openexr.com/en/latest/InterpretingDeepPixels.html>. 14
- [Kai13b] KAINZ F.: Technical introduction to openexr, 2013. Accessed: 2023-05-20. URL: <https://openexr.com/en/latest/TechnicalIntroduction.html>. 2, 3
- [KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014). arXiv:1412.6980. 8
- [KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graphics (Proc. SIGGRAPH)* 34, 4 (July 2015), 122:1–122:12. 3
- [KW17] KIPF T. N., WELLING M.: Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)* (2017). 3
- [MH20] MUNKBERG J., HASSELGREN J.: Neural denoising with layer embeddings. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 1–12. 3
- [MRF*19] MORRIS C., RITZERT M., FEY M., HAMILTON W. L., LENSSEN J. E., RATTAN G., GROHE M.: Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence* (2019), vol. 33, pp. 4602–4609. 3
- [Ope13] OPENEXR: Openexr v2.0 release notes, 2013. Accessed: 2023-09-30. URL: <https://openexr.com/en/latest/news.html#april-9-2013-openexr-v2-0-released>. 1
- [Ope23] OPENEXR: Interpreting openexr deep pixels, 2023. Accessed: 2023-09-30. URL: <https://openexr.com/en/latest/InterpretingDeepPixels.html>. 3, 6, 7
- [PD84] PORTER T., DUFF T.: Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), pp. 253–259. 3
- [PyG23] PYG: Pytorch geometric library, 2023. Accessed: 2023-09-30. URL: <https://pytorch-geometric.readthedocs.io/en/stable/index.html>. 3
- [QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 652–660. 3
- [QYSG17] QI C. R., YI L., SU H., GUIBAS L. J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 30 (2017). 3
- [Ren12] RENDERMAN: Deep compositing, 2012. Accessed: 2023-09-30. URL: https://renderman.pixar.com/resources/RenderMan_20/deepCompositing.html. 3
- [Ren18] RENDERMAN: The grand tour: Compositing, 2018. URL: <https://renderman.pixar.com/the-grand-tour-compositing#deepCompositing>. 3
- [Ren22] RENDERMAN: Deepexr, 2022. Accessed: 2023-09-30. URL: <https://rmanwiki.pixar.com/display/REN25/DeepEXR>. 1, 3
- [RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *Computer Graphics Forum* 32, 7 (2013), 121–130. 3, 8
- [Sey14] SEYMOUR M.: The art of deep compositing, 2014. Accessed: 2023-10-06. URL: <https://www.fxguide.com/featured/the-art-of-deep-compositing/>. 1
- [VAN*19] VICINI D., ADLER D., NOVÁK J., ROUSSELLE F., BURLEY B.: Denoising deep monte carlo renderings. *Computer Graphics Forum* 38, 1 (2019), 316–327. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13533>, doi:<https://doi.org/10.1111/cgf.13533>. 1, 2, 3, 6, 7, 8, 17
- [VCC*18] VELIČKOVIĆ P., CUCURULL G., CASANOVA A., ROMERO A., LIÒ P., BENGIO Y.: Graph Attention Networks. *International Conference on Learning Representations* (2018). 3
- [VR23] V-RAY: Image format options, 2023. Accessed: 2023-09-30. URL: <https://docs.chaos.com/display/VMAYA/Image+Format+Options>.
- [VRM*18] VOGELS T., ROUSSELLE F., MCWILLIAMS B., RÖTHLIN G., HARVILL A., ADLER D., MEYER M., NOVÁK J.: Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)* 37, 4 (2018), 124:1–124:15. doi:10.1145/3197517.3201388. 2, 3, 4, 5, 8, 15
- [WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612. 8
- [XZW*19] XU B., ZHANG J., WANG R., XU K., YANG Y.-L., LI C., TANG R.: Adversarial monte carlo denoising with conditioned auxiliary feature. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2019)* 38, 6 (2019), 224:1–224:12. 2, 5
- [ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHY R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum (Proc. Eurographics)* 34, 2 (May 2015), 667–681. 3
- [ZMV*21] ZHANG X., MANZI M., VOGELS T., DAHLBERG H., GROSS M., PAPAS M.: Deep Compositional Denoising for High-quality Monte Carlo Rendering. *Computer Graphics Forum* 40, 4 (2021), 1–13. doi:10.1111/cgf.14337. 2, 3, 5, 8
- [ZZR*23] ZHU S., ZHANG X., RÖTHLIN G., PAPAS M., MEYER M.: Denoising production volumetric rendering. In *ACM SIGGRAPH 2023 Talks* (New York, NY, USA, 2023), SIGGRAPH '23, Association for Computing Machinery. URL: <https://doi.org/10.1145/3587421.3595432>, doi:10.1145/3587421.3595432. 3

Appendix A: Formal definition of depth-aware neighborhood

Below, we provide a formal description of our depth-aware neighborhood. We start by defining the input to our depth-aware 3-D kernels that operate on a subset of neighboring bins based on depth information. This 3-D neighborhood has a user-defined size of $k_W \times k_W \times k_B$ where $k_W = 2r_W + 1$ and $k_B = 2r_B + 1$ (e.g., our depth-aware convolutions use $k_W = k_B = 3$ and our depth-aware kernels use $k_W = 5$ and $k_B = 3$). The coordinate of the center bin $\mathbf{p} = (x, y, b)$ is located within pixel (x, y) .

We define a $k_W \times k_W$ spatial neighborhood $\mathcal{N}(x, y)$ that is centered around pixel (x, y) and includes the pixel itself. Within each neighboring pixel $(x', y') \in \mathcal{N}(x, y)$, we define the set of neighboring bins' coordinates as $(x', y', b') \in \mathcal{B}(x', y')$. We find the coordinates of the closest bin in depth for each neighboring pixel by computing

$$\mathbf{q}^*(x', y'; \mathbf{p}) = \underset{\mathbf{q} \in \mathcal{B}(x', y')}{\operatorname{argmin}} |z_c(\mathbf{p}) - z_c(\mathbf{q})|, \quad (2)$$

where $z_c(\cdot) := (z_f(\cdot) + z_b(\cdot))/2$ denotes the center depth for a bin.

Then we identify a set of bin coordinates from each pixel $(x', y', b') \in \mathcal{B}_{r_B}(\mathbf{q})$ with a bin index distance from $\mathbf{q} = (x', y', b)$ restricted by the user-defined size along the bin dimension $|b - b'| \leq$

r_B . We can now construct the input for our depth-aware 3-D kernels

$$\mathcal{Q}_{DA}(x, y) = \bigcup_{(x', y') \in \mathcal{N}(x, y)} \mathcal{B}_{r_B}(\mathbf{q}^*(x', y'; \mathbf{p})). \quad (3)$$

For comparison, we can also construct a neighborhood with the regular neighboring indexing as follows:

$$\mathcal{Q}_S(x, y) = \bigcup_{(x', y') \in \mathcal{N}(x, y)} \mathcal{B}_{r_B}(\mathbf{q}), \quad (4)$$

where in this case $\mathbf{q} = (x', y', b)$ shares the same bin index b with the central bin at \mathbf{p} . Compared to regular neighbor indexing, by using the depth information we essentially shift the bin dimension of all neighbors such that their central bin is as close as possible to the depth of bin $\mathbf{p} = (x, y, b)$.

Given the depth-aware neighbor-indexing method, our operators can be defined as receiving per-bin input values $I(\mathbf{q})$ within the neighborhood $\mathbf{q} \in \mathcal{Q}_{DA}(\mathbf{p})$, and a kernel $K(\mathbf{q}; \mathbf{p})$ of shape $k_W \times k_W \times k_B$, and computing a weighted sum $\hat{I}(\mathbf{p})$:

$$\hat{I}(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{Q}_{DA}(\mathbf{p})} K(\mathbf{q}; \mathbf{p}) I(\mathbf{q}). \quad (5)$$

Equation 5 describes both the application of depth-aware convolution kernels on feature maps inside the U-Net and the application of predicted depth-aware denoising kernels on noisy input channels, such as $\alpha(\mathbf{p})$ and $\mathbf{c}(\mathbf{p})$.

Appendix B: Filling depth for empty bins

Input bins that did not receive any samples during the color pass will be left empty, *i.e.*, with zeros in all channels. However, it is challenging to denoise them, as their location along the depth dimension is unknown. This is in contrast with black pixels in a flat image, where the pixel's position is known and only the color values are missing. We therefore fill these bins' depth values by reusing the median of the valid depths in its bin neighborhood and ensuring the tidiness within each pixel.

We list below the detailed steps of our depth filling algorithm, where b denotes an *empty* bin's index, and B denotes the total number of bins in a pixel. For brevity, we omit the dependence of B on the pixel location. Additionally, $d_{fb}(b) := z_b(b) - z_f(b)$ denotes the distance between the front and back depth values of bin b , *i.e.*, its depth range.

1. Set $z_f(b)$ to the global maximum depth.
2. Set $z_b(b)$ to 0.
3. If $b = 0$, set $z_f(b)$ to the median of the first bin depth in the 5×5 spatial neighborhood.
4. If $b = B - 1$, set $z_b(b)$ to the median of the last bin depth in the 5×5 spatial neighborhood.
5. If $b > 0$, set $z_f(b)$ to $\max_{b' < b} z_b(b')$.
6. If $b < B - 1$, set $z_b(b)$ to $\min_{b' > b} z_f(b')$.
7. Set $z_f(b)$ to the median eligible $z_f(b')$ in the $5 \times 5 \times 3$ neighborhood; bin b' being eligible means $z_f(b) \leq z_f(b') \leq z_b(b)$.
8. Set $d_{fb}(b)$ to the median eligible $d_{fb}(b')$ in the $5 \times 5 \times 3$ neighborhood; eligible means $d_{fb}(b') \geq 0$.
9. Set $z_b(b)$ to $z_f(b) + d_{fb}(b)$.

The tidiness is then ensured by interleaving front and back

depths within a pixel, *i.e.*, forming a list of $z_f(0)$, $z_b(0)$, $z_f(1)$, $z_b(1)$, \dots , $z_f(B-1)$, $z_b(B-1)$. Then we compute the running maximum from front to back, which will be used to update the depth values for *only the empty bins*. Afterwards, we compute the running minimum from back to front and update the empty bins' depth values again.

As discussed in the limitations, depth denoising for these bins still remains challenging after the filling step. An easy fix would be to directly use the filler depth as the output depth. Better techniques to ensure tidiness within pixels for the output deep-Z images can be an interesting path for future work.

Appendix C: Bin merging for training efficiency

To help us better balance the memory requirements during training and meet the GPU memory constraints, we propose a bin merging approach that reduces the maximum number of bins per pixel. Within each deep pixel, we utilize a heuristic for identifying merge candidates which aims for minimal degradation of how flattened color changes when removing part of the depth range. We approximate this with the help of the Z-A (depth-alpha) curve, which models accumulated opacity within a pixel with increasing depth as visualized in Figure 15. Based on the Z-A curve, we define the cost of merging two bins as the product of their depth and opacity ranges:

$$\text{cost}(\mathbf{p}, \mathbf{q}) = (z_b(\mathbf{q}) - z_f(\mathbf{p})) (\alpha_+(\mathbf{p}) + \alpha_+(\mathbf{q})), \quad (6)$$

where $\mathbf{p} = (x, y, b)$ and $\mathbf{q} = (x, y, b + 1)$ are two consecutive bins in the same pixel, $z_f(\cdot)$ and $z_b(\cdot)$ return the depth values at the front and back bin boundaries, and $\alpha_+(\cdot)$ returns the bin's effective alpha as described by Vicini et al. [VAN*19]. For each pixel with more bins than a user-defined maximum bin count threshold, we iteratively select the pair of bins with the lowest cost and merge them. The merged bin will span the depth range of the original bins and yield the same alpha-composited color at the back edge. More formally, equations for computing the depth (z'_f, z'_b), alpha (α'), and color (\mathbf{c}' , premultiplied) in the merged bin are:

$$\begin{aligned} z'_f(\mathbf{p}) &= z_f(\mathbf{p}), \\ z'_b(\mathbf{p}) &= z_b(\mathbf{q}), \\ \alpha'(\mathbf{p}) &= \alpha(\mathbf{p}) + (1 - \alpha(\mathbf{p}))\alpha(\mathbf{q}), \\ \mathbf{c}'(\mathbf{p}) &= \mathbf{c}(\mathbf{p}) + (1 - \alpha(\mathbf{p}))\mathbf{c}(\mathbf{q}). \end{aligned} \quad (7)$$

Examples of bin merging on rendered deep pixels are illustrated in Figure 15, where we demonstrate the ability of Equation 6 to reduce bin count in a manner that has little impact on the depth-resolved appearance of a deep pixel. Overall, we empirically found that using the proposed heuristic, we can consistently reduce the number of depth bins and thus overcome memory bottlenecks during training. We merge our training dataset in a pre-process, using the depth and alpha values from the reference deep-Z images that share the bin layout with the noisy inputs.

During training, we use a bin-merged version of the training set containing max 8 bins per pixel. Given the overall maximum original bin count of 53, such a bin merging pass greatly reduces the peak memory consumption and runtime requirement for training.

