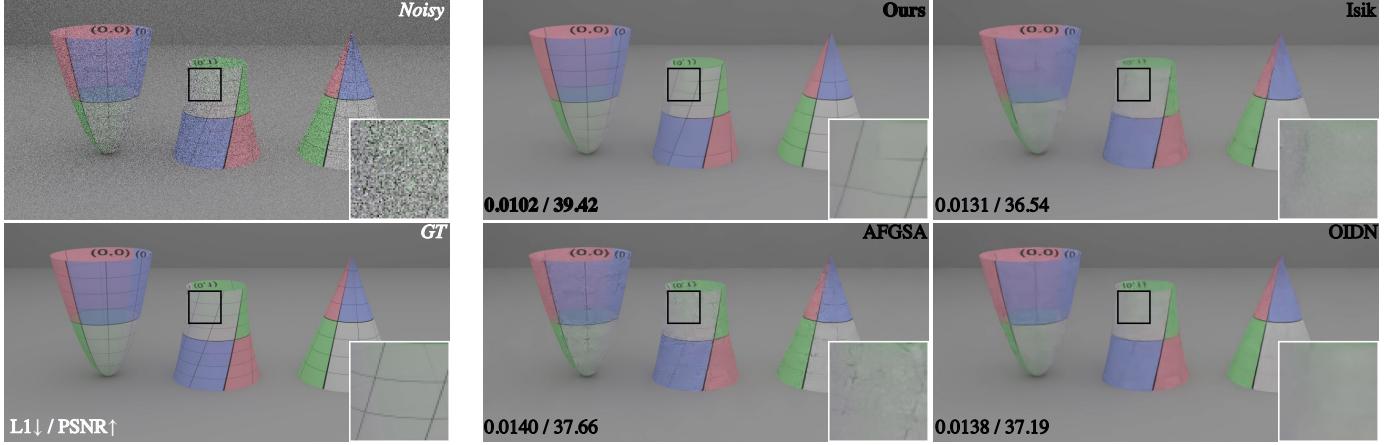


Denoising Monte Carlo Renders With Diffusion Models



We present a denoiser based on a pixel-space diffusion model. Because our method has a strong prior of what a real image looks like, it can generalize better on out of distribution images. In the example above, notice how competing methods produce unwanted artifacts like splotchy/blurry regions and missing/broken-up lines on the surface texture. Thus, while other methods produce competitive error metrics, the poor qualitative behavior is not always captured by those numbers. Our method will consistently produce an image that looks like a real image (e.g. minimal unwanted artifacts), while adhering to the conditioning. Results shown on a 4spp test image.

Vaibhav Vavilala, Rahul Vasanth, and David Forsyth

Abstract—Physically-based renderings contain Monte-Carlo noise, with variance that increases as the number of rays per pixel decreases. This noise, while zero-mean for good modern renderers, can have heavy tails (most notably, for scenes containing specular or refractive objects). Learned methods for restoring low fidelity renders are highly developed, because suppressing render noise means one can save compute and use fast renders with few rays per pixel. We demonstrate that a diffusion model can denoise low fidelity renders successfully. Furthermore, our method can be conditioned on a variety of natural render information, and this conditioning helps performance. Quantitative experiments show that our method is competitive with SOTA across a range of sampling rates, but current metrics slightly favor competitor methods. Qualitative examination of the reconstructions suggests that the metrics themselves may not be reliable. The image prior applied by a diffusion method strongly favors reconstructions that are “like” real images – so have straight shadow boundaries, curved specularities, no “fireflies” and the like – and metrics do not account for this. We show numerous examples where methods preferred by current metrics produce qualitatively weaker reconstructions than ours.

Index Terms—Diffusion model, Monte Carlo denoising, Image restoration

1 INTRODUCTION

The image produced by a physically based renderer is the value of a random variable. Typically, the mean of this random variable is the (unknown) true result; but light transport effects mean that the variance – the result of using “too few rays” – is complicated and heavy tailed. Analogous effects appear in very low light photography. Increasing N – the number of rays per pixel – quickly results in diminishing returns, because the variance goes down as $1/N$, and so there is a significant literature that aims to suppress render noise. This paper shows that a method based on diffusion is

quantitatively competitive with the state of the art (SOTA) while producing images that differ strongly in qualitative aspects.

Obvious solutions to render noise fail. Real-world scenes in the film industry are too large to fit into GPU RAM and must be rendered on a CPU. Often even hundreds to thousands of rays per pixel may fail to reach an artist’s desired level of quality. Problems are caused by the presence of desirable but complex light transport phenomena such as indirect specular, large numbers of light sources, subsurface scattering, and volumetric effects.

The concept of relying on pretrained foundation models has been extensively explored for image restoration [1], [2]. Diffusion models have successfully removed JPEG noise

• V. Vavilala is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 61801.
E-mail: vv16@illinois.edu

and film grain; images can be recolored; and local objects can be inpainted. Even the success of large language models relies on a foundation model trained for next token prediction, then finetuned for particular tasks (for example, chatbots like ChatGPT relying on GPT-4 as the backbone). **Until now, the concept of foundation models has not been applied to denoising Monte Carlo (MC) renders.**

There are multiple reasons for this. For one, the area has matured - high quality denoisers already exist, both learning and non-learning, some geared towards high quality and others towards speed. Another reason is that there are known workarounds when existing denoisers create problems. Artists can sample scenes for longer (requiring additional compute and slowing artist iteration) and/or manually touch-up unwanted artifacts in finishing software like Photoshop or Nuke.

Further, it is not obvious that image foundation models can handle high dynamic range gracefully, since they are trained on images with per-pixel radiance from $[0 - 1]$. Ray traced images in contrast store the per-pixel colors in linear space, which may exceed 1 if rendering a bright or shiny object. Subsequent processing such as gamma correction, tonemapping, compositing, and color grading bring the final values to $[0 - 1]$, suitable for standard dynamic range (SDR) display. Denoisers must process inputs in a manner respecting high dynamic range to be useful in practice. **Across the billions of images foundation models are trained on, a variety of post-processing will be reflected** in the datasets such that the result sits in the range $[0 - 1]$. We believe that **this wide array of tonemapping, geometry, textures, and lighting is an asset if the foundation model can be utilized effectively**. As we show, images denoised with foundation models can often look better than existing methods, even when current error metrics like PSNR and SMAPE do not necessarily demonstrate the improvement.

Another potential pitfall is that these large scale training sets will include images with unwanted lossy compression artifacts, film grain, image rescaling, motion blur, and other phenomena associated with the capture and storage of imagery. It's not obvious whether an image can be effectively denoised without unwanted artifacts or hallucinations leaking into the result. In practice, we don't notice this to be a problem when evaluating on unseen test images.

Expensive data collection is another concern - it is extremely compute-intensive to generate 3D scenes and render them with a path tracer to an acceptable level of quality. A single image may require tens to hundreds of CPU hours to converge, and tens of thousands of frames are required to build a sufficient training set. While high quality rendering software exists (such as RenderMan used in VFX houses, PBRT and MITSUBA for research [3], [4], [5]), high quality datasets are not freely available. The research community would benefit from large open source renders of practical scenes with feature buffers at low sample counts, and accompanying high sample target renders.

A problem we aim to address in this work is that existing denoisers frequently struggle to generalize to scenes with phenomena different from the training set (for example different hair colors/styles, volumetric effects, and novel textures). That may result in users having to collect more training data to retrain an existing denoiser (a computa-

tionally demanding effort). Alternatively, denoiser limitations may result in additional artist effort to detect and fix problems manually - or simply accepting a lower-quality product.

Advancing the SOTA for denoising may alleviate some of these problems. This paper is firmly in the high quality regime (as opposed to focusing on speed), as we are utilizing a multi-pass diffusion model and evaluating against 1-pass neural networks. We however present suggestions for making our model run comparably fast with existing methods.

Our contributions are:

- 1) We are the first to use large-scale image generation foundation models to denoise MC renders, and we demonstrate that conditioning on render buffers provides essential information to the diffusion model. To do so, we apply the ControlNet architecture to a pixel-based (rather than latent variable based) diffusion model.
- 2) Quantitative evidence places our approach close to SOTA; qualitative evidence suggests it is generally better.

2 RELATED WORK

Early denoisers relied on linear regression models and hand-designed filters that run quickly. Zwicker et. al. provide an excellent survey on classical approaches [8], while Huo et. al survey relatively recent deep learning methods on denoising Monte Carlo renderings [9]. Recent advancements in Monte Carlo denoising reflect the progression of learning-based approaches. We focus our literature review on recent denoising works and controlled image synthesis.

2.1 Real-time and Interactive Denoisers

Chaitanya et. al lower temporal noise in animations and are the first to use a U-net [10], [11]. Thomas et. al incorporate a U-net, using a low-precision feature extractor and multiple high-precision filtering stages to perform both supersampling and denoising [12]. Fan et. al make improvements to efficiency, building on the hierarchical approach utilized by Thomas et. al by predicting a kernel for a single channel and incorporate temporal accumulation [13]. Lin et. al employ a path-based approach [14].

Meng et. al train a network to splat samples onto multi-scale, hierarchical bilateral grids, then denoise by slicing the grid [15]. Lee et. al incorporate kernel prediction and temporally accumulated and 1spp images for real time denoising [16]. Munkberg et al. build on the approach used by Gharbi et. al., but splat radiance and sample embeddings onto multiple layers. Filter kernels are applied to layers which are composited, improving performance [17]. Munkberg et. al eschew Laplacian pyramids used by Vogels et. al and improve temporal stability compared to Chaitanya et. al. by maintaining a per-sample approach [17], [18], [19]. Lee et. al incorporate kernel prediction and temporally accumulated and 1spp images for real time denoising [20].

Similar to Munkberg et. al, Isik et. al use per-sample information when computing filter weights, though the filters operate on pixel-wise averages. Their network predicts

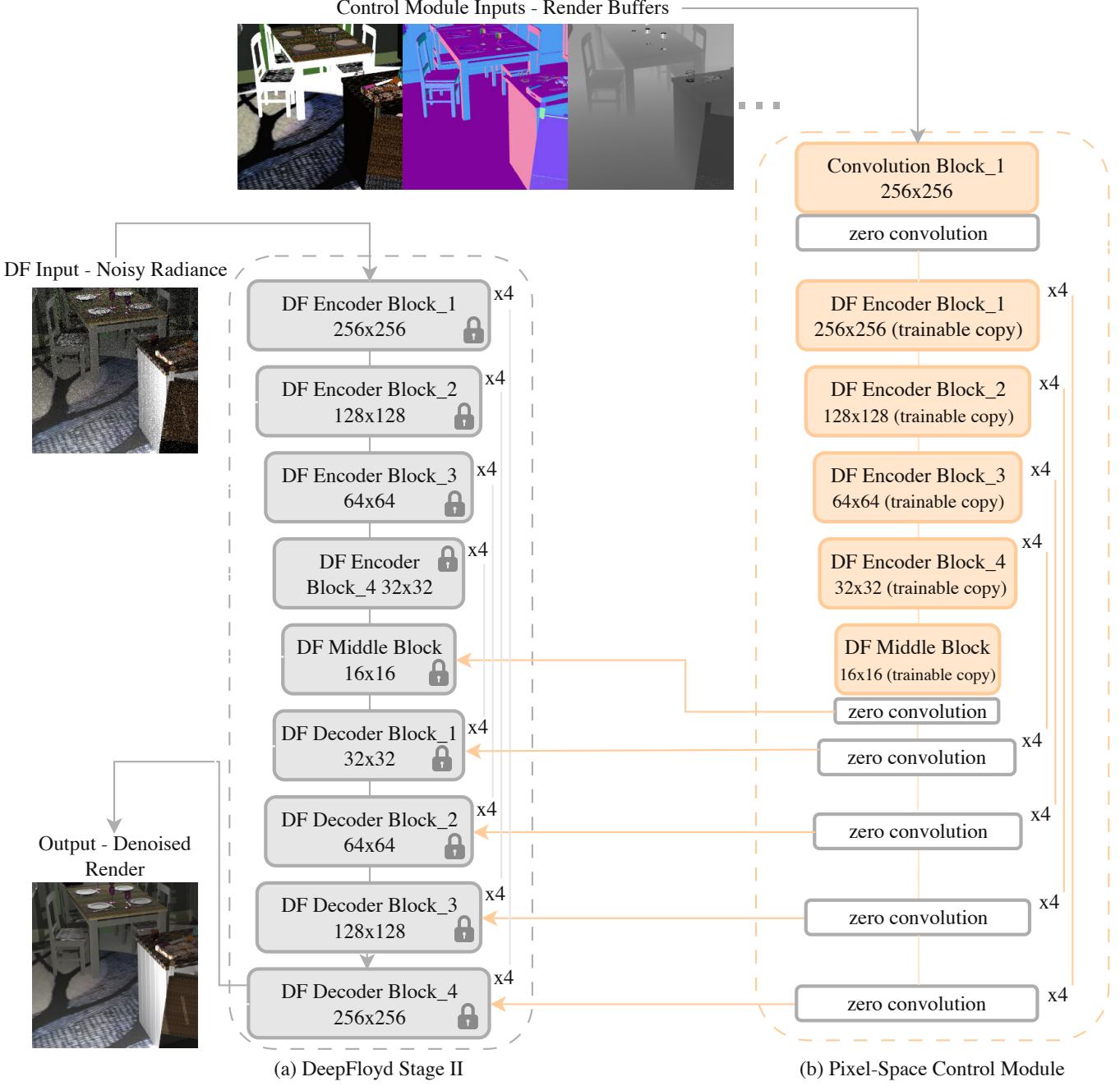


Fig. 1: Overview of our method. We leverage a pretrained pixel-space synthesizer, DeepFloyd Stage II [6], as our base synthesizer that is fixed during training. It accepts the noisy radiance as well as a forward-diffused copy of the noisy radiance (see methodology section 3 for details). We introduce a trainable Control Module, analogous to ControlNet [7], initialized with the encoder and middle blocks. It accepts all the auxiliary feature buffers from the renderer like albedo, normals, and depth, in addition to the noisy radiance. The outputs of the Control Module are added to the DF decoder blocks at varying spatial resolutions. We utilize zero convolutions to ease early stages of training. Time and prompt encoding not shown for brevity. Our prompt is the empty string during training and inference.



Fig. 2: Example images from our procedurally-generated dataset.

dense features, which are then incorporated in a pairwise-affinity metric that results in per-pixel dilated 2D blur kernels applied iteratively to the low sample radiance. An optional temporal kernel can blend the previous frame’s denoised output with the current frame [17], [21].

2.2 Offline Denoisers

Kalantari et al. utilize a multi-layer perceptron at a per-pixel level to optimize cross-bilateral filter parameters [22].

Rather than hand-crafted kernels, Bako et al. use a convolutional neural network to predict filtering kernel weights adaptively at a pixel level, then apply the kernels to the noisy image input. Specular and diffuse components are separately processed. The kernel prediction convolutional network (KPCN) and kernel-prediction approaches are popular as they train quickly and are more robust than predicting colors directly [23].

Vogels et al. utilize hierarchical pyramid kernels, approximating the behavior of large kernels with a small kernel multi-resolution cascaded filtering strategy. This work extends KPCN to animated sequences, incorporating an adaptive sampling approach with temporal aggregation [19]. Instead of inferring weights per pixel, Gharbi et al. use radiance samples, demonstrating their utility for denoising by predicting splatting kernels for each sample [18]. Balint et al. further develop the pyramidal filtering approach with improvements to upsampling, weight predictor networks, and learnable partitioning [24].

Xu et. al show that generative adversarial networks can be used for denoising, eliminating in-between layers used in Bako et. al, but similarly train separate networks to process specular and diffuse components [23], [25]. Yu et. al develop this adversarial approach by processing all components together and incorporate a modified self-attention: auxiliary

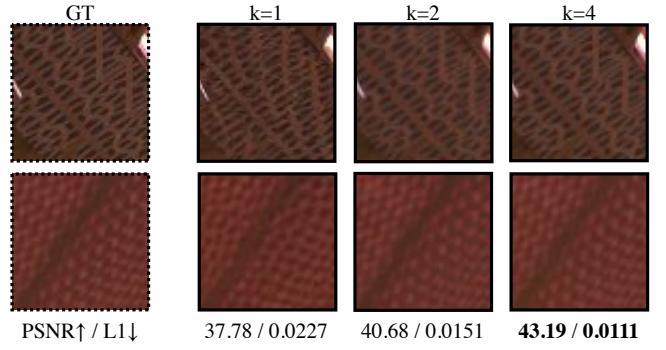


Fig. 3: Best viewed online in color. Diffusion methods based on a latent variable image representation will not work for render denoising, because the VAE decoding creates significant problems. Existing ControlNet [7] architectures use a latent representation of the image. The limited size of the VAE dictionary limits the accuracy of very precise pixel space tasks. Upsampling the image helps, but does not remove this effect. In each row, 64x64 cropped training images are shown, highlighting texture. We take the GT training image, upsample by k , feed it through a VAE, decode, downsample by k and show the result. We use the default VAE from Stable Diffusion 2.1². Error metrics shown underneath are averaged over 64 random training images. The VAE introduces unacceptable changes to texture. In the first row, the black patches are sharply defined in GT, but blurred after VAE decoding. In the second, the VAE shifts the color of the texture. These changes are partially due to the 8x spatial downsampling factor of VAEs, converting a $(H,W,3)$ image into a $(H/8, W/8, 4)$ dimensional latent code. Thus, existing control mechanisms relying on latent-space diffusion models like ControlNet are not suitable for pixel-space tasks like MC denoising. Even with 4x upscaling, which defeats much of the efficiency gain of the VAE, the PSNR is comparable with existing SOTA denoisers, which effectively caps the quality that can be achieved. Thus, in this work we introduce spatial controls to pixel-space diffusion models.

² <https://huggingface.co/stabilityai/stable-diffusion-2-1>

feature guided self-attention (AFGSA) [26]. This work effectively implements a global blur kernel since each pixel attends to the features of all pixels via the cross-attention mechanism. Back et. al apply a post-processing network incorporating a self-supervised loss to KPCN, AFGSA, and Xu et. al [25] to improve denoising quality [27].

Our work deviates from much of the recent kernel-based approaches in that we predict the colors directly.

2.3 Diffusion Models

Stable Diffusion is a foundational model that implements latent diffusion [28], [29], [30], whereby a VAE compresses the spatial resolution of the input image, the diffusion model is run in latent space, and the resulting latent code is decoded into the output image. The synthesis is commonly conditioned by text input via cross attention. Additional controls can be introduced, most notably ControlNet [7]. ControlNet has been shown to facilitate improved control

over latent diffusion models by conditioning image synthesis on arbitrary spatial information like edges, depth, and segmentation.

Imagen is another successful large-scale image synthesis model that operates in pixel space [31], with image quality comparable to Stable Diffusion. Imagen relies on three diffusion models: the first synthesizes at 64x64 resolution; the second upsamples to 256 resolution; and the third upsamples to 1024. Thus, super resolution replaces the need for a VAE, easing compute requirements. DeepFloyd is an open source implementation of Imagen [6].

Wang et. al pass pixel-level features to a pretrained Stable Diffusion model for restoration [32]. Yang et. al introduce a pixel-aware cross attention module to latent diffusion models for realistic image super-resolution [33]. Similarly, Instruct-Imagen utilizes cross-attention to condition a pixel space diffusion model on spatial layout [34]. In contrast, we use concatenation and addition to condition our diffusion model, like ControlNet, but in pixel space.

3 METHOD OVERVIEW

3.1 Data

Rendering We generate approx. 5650 random scenes using a procedure similar to [21]. Each scene contains randomly arranged ShapeNet objects, textures, and lighting configurations [35]. For each scene, we create a random camera move over the course of 8 frames and render with PBRT-V3 at 256 resolution [4]. We render with the path-tracing integrator and allow several bounces to capture indirect illumination. For each frame, we render the ground truth GT_{raw}^1 at 4096 spp, and render 16 additional frames with feature buffers at 4spp (all with different random seeds to avoid noise correlations). Thus we can train our denoiser with $spp \in \{4, 16, 64\}$. Our auxiliary feature buffers F include normals, albedo, depth, direct specular, indirect specular, direct diffuse, indirect diffuse, roughness, emissive, metallic, and transmission. Variance for each channel is also stored.

Data sanitization Unfortunately, our target renders were still a bit noisy at such a high spp due to indirect illumination noise. Noise2Noise [36] establishes that neural networks can learn to denoise even when ground truth is noisy if the training objective (often an L1 or L2 loss) attempts to recover the mean of the distribution. This assumption allows several prior methods to succeed even when ground truth is noisy. However, methods relying on a discriminator such as AFGSA [26] are not compatible with noisy ground truth because noise will leak into the generated distribution. Thus we adopt the following procedure to post-process the ground truth images. We render an additional GT image GT_{raw}^2 at 4096 spp for each scene, but use a different random seed than GT_{raw}^1 . We then train a network with only SMAPE loss that accepts a GT render and predicts the other one from the pair that we rendered. This data-cleaning network C_θ is optimized with:

$$L_{\text{SMAPE}} = \frac{1}{N} \sum_i \text{SMAPE}(C_\theta(F_i, GT_{i,\text{raw}}^1), GT_{i,\text{raw}}^2) \quad (1)$$

Where the loss is averaged over $i = 1, 2, \dots, N$ samples in a minibatch and $GT_{i,\text{raw}}^1$ and $GT_{i,\text{raw}}^2$ are interchanged at random during training. SMAPE is defined as:

$$\text{SMAPE}(A, B) = \frac{\|A - B\|_1}{\|A\|_1 + \|B\|_1 + \epsilon} \quad (2)$$

We adopt the architecture of Isik [21] for C_θ , conditioning the input on features but excluding any temporal loss terms. The fully-trained data cleaning network is then run on all the ground truth images we rendered (we just pick one out of each pair of raw GT frames), generating clean GT renders:

$$GT_{i,\text{clean}} = C_\theta(F_i, GT_{i,\text{raw}}^1). \quad (3)$$

All evaluated methods use GT_{clean} as the target during training, and have access to the same training, validation, and test splits.

Range-compression Large-scale pretrained diffusion models are trained on images with various degradations and tonemappings applied, ultimately with pixel values ranging from 0 to 1. Our dataset however has useful radiance values well beyond that - we clamp the radiance buffers to 6. The practice of range-compressing in log space is common across all MC denoising works we evaluated. In our case, we observe that gamma-correcting tonemappers produce more natural-looking results consistent with datasets diffusion models were trained on. Thus we conduct initial experiments with the following tonemapper:

$$V_{\text{out}} = AV_{\text{in}}^\gamma \quad (4)$$

We set $A = 0.47, \gamma = 1/2.4$, fully capturing the 0 – 6 range. However, early results showed that log tonemapping produces better results for our model:

$$V_{\text{out}} = A * \log(1 + V_{\text{in}}) \quad (5)$$

We set $A = 0.51$. In our evaluation, we show that our diffusion model can effectively denoise images in a LDR space, and we reverse-tonemap the synthesized result back to HDR space for error metric calculation. All figures in this manuscript are tonemapped with gamma (eqn. 4), as the details appear more clear.

Additional test data We hold out 3% of the data for validation and 5% for testing. For additional test data, we render a few PBRT-v3 test scenes that portray practical real-world scenarios¹ though we do not apply post-processing to the ground truth as we did for the training images.

For 4 spp models, we randomly sample one low-res stack of buffers out of 16 available; we sample 4 buffers for the 16 spp models; and use them all for the 64 spp models. The aux buffers are averaged over the selections.

3.2 Architecture

We use DeepFloyd [6] (which is based on Google’s Imagen [31]) as our base architecture, and build a Control Mechanism around it in a manner similar to ControlNet [7]. Our Control Module, shown in Fig. 1, consists of a few convolutional layers that accept arbitrary numbers of channels (39 aux buffers in our case), the DeepFloyd encoder layers, and the DeepFloyd middle layer. The Control Module generates feature maps at resolutions (256, 128,...16) which are summed with the original DeepFloyd encoder outputs and

1. <https://www.pbrt.org/scenes-v3>

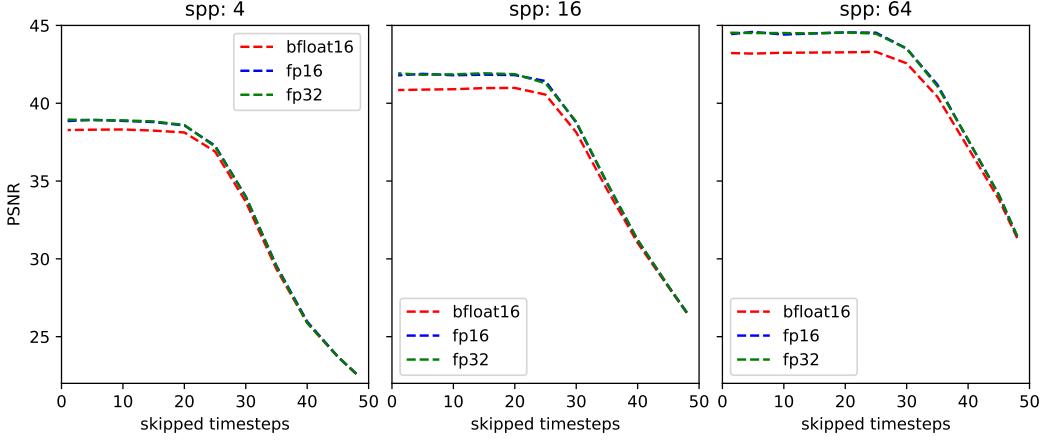


Fig. 4: Evaluation of our method on test data. There is no loss in quality when running inference at FP16 instead of FP32, saving nearly 40% of the cost ($6.17 \rightarrow 3.80$ seconds for a 256×256 res image and 50 DDIM steps on a single A40 GPU). We can save an additional approx. 45% of compute time by skipping some of the early steps of DDIM sampling (whereby at inference time, we add gaussian noise to the low-spp render via equation 6).

Method	PSNR ↑			L1 ↓			SMAPE ↓		
	4spp	16spp	64spp	4spp	16spp	64spp	4spp	16spp	64spp
OIDN [37]	36.24	39.81	42.54	0.0643	0.0538	0.0491	0.0718	0.0466	0.0339
AFGSA [26]	38.69	42.07	45.09	0.0279	0.0184	0.0142	0.0538	0.0365	0.0267
Isik [21]	38.97	42.14	45.22	0.0525	0.0446	0.0426	0.0439	0.0309	0.0226
Ours	38.68	41.62	44.24	0.0251	0.0171	0.0123	0.0602	0.0438	0.0353
DeepFloyd-II [6]	26.67	27.34	30.84	0.1192	0.0959	0.0584	0.2314	0.2117	0.1384

TABLE 1: **Quantitatively, our method is competitive with SOTA.** We compare error metrics (L1, SMAPE, PSNR) across different spp settings (4, 16, 64) for various methods on our test set, consisting of 226 sequences of 8 frames each at 256 res. We remark that Isik benefits from its ability to gather temporal information; all other methods are single-frame. All evaluation metrics only take into account single-frame performance. Our method outcompetes all others in L1, which is computed in HDR $[0 - 6]$ space, and is in-line with other methods in terms of PSNR. In the **final row**, we disable our Control Module and denoise with an off-the-shelf DeepFloyd Stage II model, presenting the best numbers after ablating on the number of skipped denoising time steps and the *aug_level* parameter. Applying pure DeepFloyd to low-ray estimates is not successful. **Fourth row** Thus, conditioning our method with render buffer information via our Control Module makes a significant difference to performance.

passed to the DeepFloyd decoder. DeepFloyd modules are not trainable, only the Control Module is, consistent with the original ControlNet. Zero convolutions are initialized to ensure the system produces realistic images from the very beginning. We only use the DeepFloyd Stage II module for this work, which is intended to perform super resolution; we show it can perform image denoising here. We use the smaller IF-II-M model for efficiency, which has 450M parameters. Our base module accepts two 3 channel inputs that are concatenated: a time-dependent noisy radiance buffer $q(\mathbf{z}_t | \mathbf{x})$, which undergoes the diffusion forward noising process, and that same noisy radiance \mathbf{x} without forward noising (because the *aug_level* parameter is fixed to 0 during training and inference). The Control Module accepts the feature buffers F_i . The forward diffusion process is defined as:

$$q(\mathbf{z}_t | \mathbf{x}) = \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{x}, \sigma_t^2 \mathbf{I}), \\ q(\mathbf{z}_t | \mathbf{z}_s) = \mathcal{N}(\mathbf{z}_t; (\alpha_t / \alpha_s) \mathbf{z}_s, \sigma_{t|s}^2 \mathbf{I}) \quad (6)$$

where $0 \leq s < t \leq 1$, $\sigma_{t|s}^2 = (1 - e^{\lambda_t - \lambda_s}) \sigma_t^2$, and α_t, σ_t

define a differentiable noise schedule whose log signal-to-noise-ratio, i.e., $\lambda_t = \log [\alpha_t^2 / \sigma_t^2]$, decreases with t until $q(\mathbf{z}_1) \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$. For generation, the diffusion model is trained to reverse this forward process. We refer the reader to [31] for additional details.

Losses We use the usual losses documented in [6], [31], i.e., mean squared error and variational lower bound. We let the text prompt be the empty string during training and inference.

Training & Inference We train with AdamW optimizer, FP32 precision, batch size 8, 256 res. While the model is trained at 256 res, it can run inference at varying spatial resolutions as it relies on convolutional and transformer layers. We use the SMART50 DDIM inference schedule. We experimentally set the *aug_level* parameter to 0 during training and inference. This parameter is intended to augment the Stage II input, which typically comes from Stage I. In our case we supply the noisy radiance \mathbf{x} (which is already at the target resolution). We train for 5 days with learning rate 0.00002 and an additional 2 days halving the learning rate, corresponding to approx. 170 total epochs. All experiments are conducted on one NVIDIA A40 GPU. We train

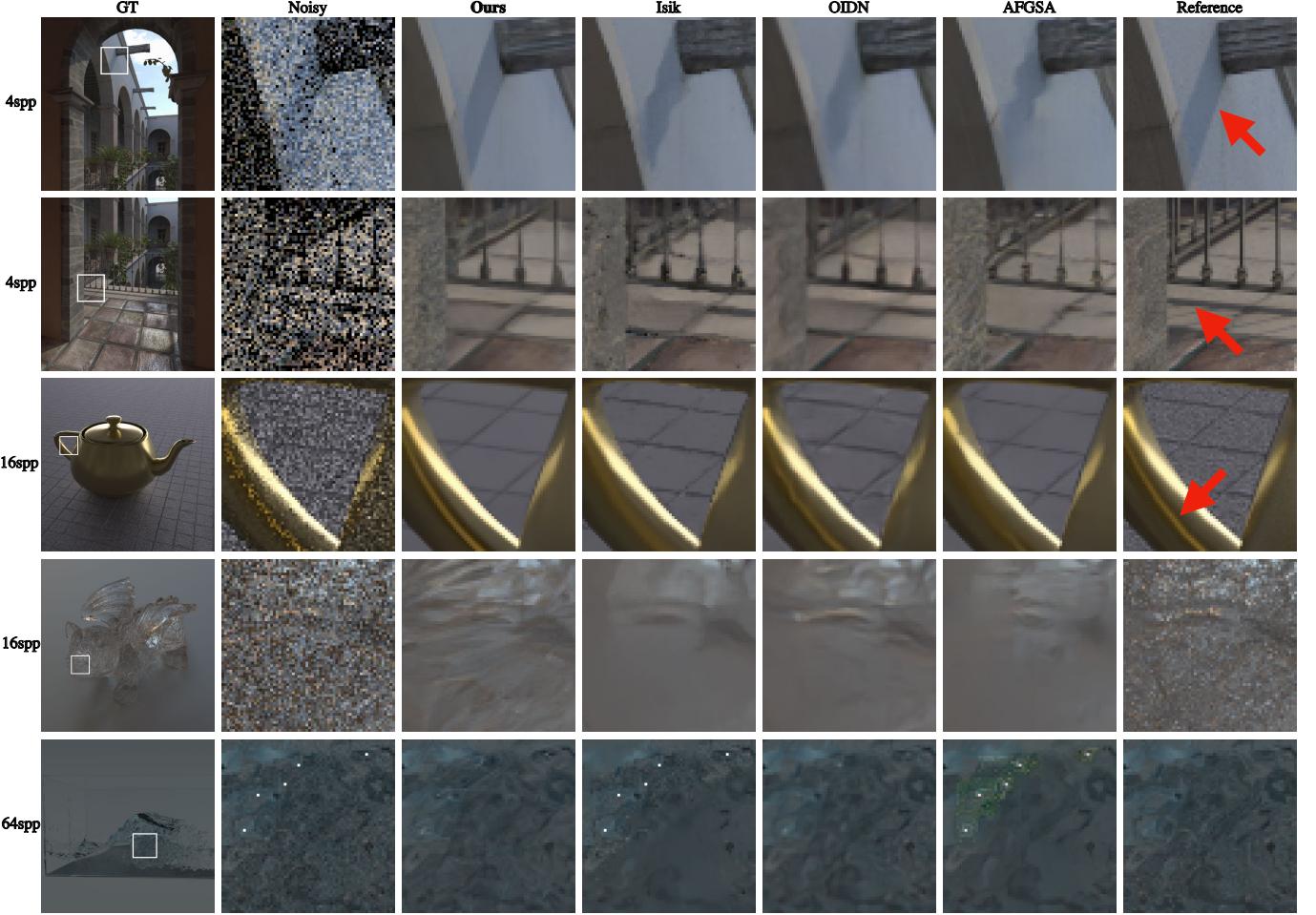


Fig. 5: Qualitatively, our reconstructions look like real images when others tend not to, because DeepFloyd has a very strong notion of what an image looks like (it has seen a huge dataset) and because the conditioning buffers offer strong guidelines (e.g. normals, albedo, and depth). We believe that current metrics cannot evaluate the extent to which an image is “like a real image” and so undervalue our method. In the final column, red arrows point to areas of interest. **First row.** Notice the straight edge on the shadow (ours; real images tend to have straight shadow edges) compared with blurred or blotchy edges (others). **Second row.** In undersampled regions, our method fills in the shadow underneath a railing (real images do not have incomplete shadows); other methods render a blurred or incomplete shadow. **Third row.** Notice the clean sharp highlight on the teacup handle, and smooth highlight boundaries (ours; real images have clean sharp highlight boundaries) compared with absent highlights and blotchy boundaries (others). Notice also aliasing effects in the background, most prominent for OIDN, and absent from our reconstruction. **Fourth row.** All methods have problems with this specular dragon. Competing methods overblur the dragon’s mouth, whereas ours hallucinates plausible details. **Fifth row.** Fireflies, also known as spike noise, are single very bright pixels, which do not occur in real images. It is rare that fireflies appear at 64 spp in our training set, so AFGSA and Isik fail to remove them. OIDN succeeds in removing them, likely because we use their pretrained model trained on a large dataset.

an independent network for each sampling rate (4/16/64) though we would expect one appropriately-trained network to be robust to a sweep of spp’s in practice. For fairness, we train independent models for competing methods as well.

4 EVALUATION

We focus our comparative evaluation on 3 prior methods known for quality - Isik, AFGSA, and OIDN [21], [26], [37]. Isik and OIDN rely on CNN UNets, and AFGSA uses a transformer with adversarial loss. We evaluate the best available OIDN pretrained model (OIDN-VERSION = 2.1.0) and supply it at test time with noisy radiance, albedo, and

normals. We retrain Isik and AFGSA with default hyperparameters, all feature buffers, and best-performing model on the held-out validation set used for testing.

We evaluate each method using three standard metrics: L1 and SMAPE loss (applied in HDR space) and PSNR (in LDR space). For PSNR, we tonemap GT and prediction via the following tonemapper, consistent with existing evaluation methods:

$$V_{out} = \frac{V_{in}}{1 + V_{in}} \quad (7)$$

Quantitative evaluation We test each method on 226 sequences of 8 frames each at 256 res in Table 1. Across spp

$\in \{4, 16, 64\}$, our method is the top-performer as measured by L1. We are comparable with other methods as measured by PSNR and SMAPE. Our higher SMAPE metric indicates we make slightly more errors in dark regions than brighter regions, as it is a relative metric. For our method, we use 50 DDIM steps at FP32 precision. However, in Fig. 4, we ablate a few test-time hyperparameters. First, FP16 precision matches FP32 numbers, resulting in approx. 40% savings. Second, we show that we can skip nearly half the DDIM steps with no loss in PSNR, resulting in additional savings. This behavior is consistent across all spp’s we tested.

Finally, in the last row of Table 1, we examine the effects of removing the Control Module and denoising with an off-the-shelf DeepFloyd Stage II model, finding that the results are much worse. We test several skipped time step and *aug_level* values, and report the best numbers. We skip 20, 30, and 40 time steps for 4, 16, and 64 spp respectively, and set *aug_level* = 0.5.

Qualitative evaluation The key finding from our method is that images can look more plausible and realistic without necessarily giving the best error metrics. We demonstrate this in Fig. 5. In many cases, competing methods produce numbers comparable or slightly better than ours. But upon closer inspection, the details rendered from our diffusion model look more realistic, because our model has a strong prior of what a real image looks like. Thus shadows, specular highlights, and undersampled edges look reasonable. Other methods overblur undersampled regions, fail to remove fireflies even when trained to do so, and hallucinate splotchy patches when pushed outside the training distribution. Our method is not explicitly engineered to remove fireflies or hallucinate nice-looking specularities and shadows; it only knows it should synthesize a realistic image that follows the conditioning.

We note that Isik [21] benefits from training with temporal coherence, boosting its performance on our primary test set shown in Table 1. However, our qualitative evaluation of real world scenes in Fig. 5 is single-frame only.

5 CONCLUSION

Pretrained large-scale image models enable MC denoising that is quantitatively near SOTA and qualitatively more realistic. One-step models like GANs and distilled diffusion models may yield significant efficiency gains with acceptable quality tradeoffs [38], [39].

One area of future work is what conditioning to use; another is what effects to capture in a training set. Our training set is relatively straightforward, though we test on images with challenging effects. A diverse training set that aggressively oversamples difficult effects might produce better results.

A final area of future work is video. As video generators become available [40], [41], our approach might usefully be extended to temporally-coherent denoising.

REFERENCES

- [1] C. Saharia, W. Chan, H. Chang, C. A. Lee, J. Ho, T. Salimans, D. J. Fleet, and M. Norouzi, “Palette: Image-to-image diffusion models,” 2022.
- [2] X. Li, Y. Ren, X. Jin, C. Lan, X. Wang, W. Zeng, X. Wang, and Z. Chen, “Diffusion models for image restoration and enhancement—a comprehensive survey,” *arXiv preprint arXiv:2308.09388*, 2023.
- [3] P. Christensen, J. Fong, J. Shade, W. Wooten, B. Schubert, A. Kensler, S. Friedman, C. Kilpatrick, C. Ramshaw, M. Bannister *et al.*, “Renderman: An advanced path-tracing architecture for movie rendering,” *ACM Transactions on Graphics (TOG)*, vol. 37, no. 3, pp. 1–21, 2018.
- [4] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. MIT Press, 2023.
- [5] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, “Mitsuba 2: A retargetable forward and inverse renderer,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–17, 2019.
- [6] S. AI, “Deepfloydif.”
- [7] L. Zhang, A. Rao, and M. Agrawala, “Adding conditional control to text-to-image diffusion models,” 2023.
- [8] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon, “Recent advances in adaptive sampling and reconstruction for monte carlo rendering,” *Computer Graphics Forum*, vol. 34, no. 2, pp. 667–681, 2015. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12592>
- [9] Y. Huo and S. eui Yoon, “A survey on deep learning-based monte carlo denoising,” 2021. [Online]. Available: <https://doi.org/10.1007/s41095-021-0209-9>
- [10] C. R. A. Chaitanya, A. Kaplanyan, C. Schied, M. Salvi, A. E. Lefohn, D. Nowrouzezahrai, and T. Aila, “Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder,” *ACM Transactions on Graphics (TOG)*, vol. 36, pp. 1 – 12, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:3350221>
- [11] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.
- [12] M. M. Thomas, G. Liktor, C. Peters, S. ye Kim, K. Vaidyanathan, and A. G. Forbes, “Temporally stable real-time joint neural denoising and supersampling,” *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 5, pp. 1 – 22, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:251107201>
- [13] H. Fan, R. Wang, Y. Huo, and H. Bao, “Real-time monte carlo denoising with weight sharing kernel prediction network,” *Computer Graphics Forum*, vol. 40, no. 4, p. 15–27, Jul. 2021. [Online]. Available: <http://dx.doi.org/10.1111/cgf.14338>
- [14] W. Lin, B. Wang, J. Yang, L. Wang, and L.-Q. Yan, “Path-based monte carlo denoising using a three-scale neural network,” *Computer Graphics Forum*, vol. 40, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:221764375>
- [15] X. Meng, Q. Zheng, A. Varshney, G. Singh, and M. Zwicker, “Real-time monte carlo denoising with the neural bilateral grid,” in *Eurographics Symposium on Rendering*, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:220284605>
- [16] J. Lee, S. Lee, M. Yoon, and B. C. Song, “Real-time monte carlo denoising with adaptive fusion network,” *IEEE Access*, vol. 12, pp. 29 154–29 165, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:267950688>
- [17] J. Munkberg and J. Hasselgren, “Neural denoising with layer embeddings,” *Computer Graphics Forum*, vol. 39, no. 4, pp. 1–12, 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14049>
- [18] M. Gharbi, T.-M. Li, M. Aittala, J. Lehtinen, and F. Durand, “Sample-based monte carlo denoising using a kernel-splatting network,” *ACM Trans. Graph.*, vol. 38, no. 4, jul 2019. [Online]. Available: <https://doi.org/10.1145/3306346.3322954>
- [19] T. Vogels, F. Rousselle, B. Mcwilliams, G. Röthlin, A. Harvill, D. Adler, M. Meyer, and J. Novák, “Denoising with kernel prediction and asymmetric loss functions,” *ACM Trans. Graph.*, vol. 37, no. 4, jul 2018. [Online]. Available: <https://doi.org/10.1145/3197517.3201388>
- [20] J. Lee, S. Lee, M. Yoon, and B. C. Song, “Real-time monte carlo denoising with adaptive fusion network,” *IEEE Access*, vol. 12, pp. 29 154–29 165, 2024.
- [21] M. Isik, K. Mullia, M. Fisher, J. Eisenmann, and M. Gharbi, “Interactive monte carlo denoising using affinity of neural features,” *ACM Trans. Graph.*, vol. 40, no. 4, jul 2021. [Online]. Available: <https://doi.org/10.1145/3450626.3459793>

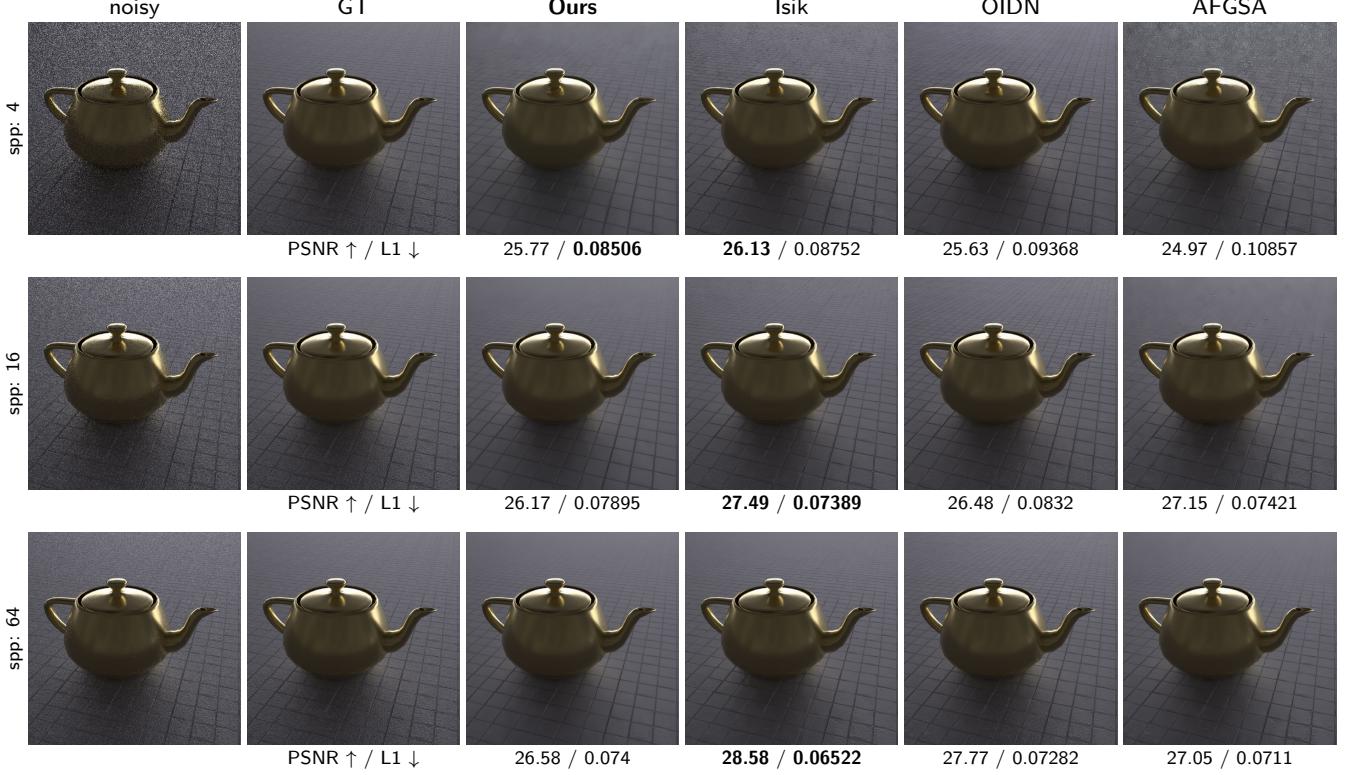


Fig. 6: Additional samples from our test set. Across low, medium, and high sample counts, our procedure synthesizes images with fewer unwanted artifacts and more natural-looking texture than existing work, which does not utilize a large-scale foundation model.

- [22] N. K. Kalantari, S. Bako, and P. Sen, "A machine learning approach for filtering monte carlo noise," *ACM Trans. Graph.*, vol. 34, no. 4, jul 2015. [Online]. Available: <https://doi.org/10.1145/2766977>
- [23] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. DeRose, and F. Rousselle, "Kernel-predicting convolutional networks for denoising monte carlo renderings," *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017)*, vol. 36, no. 4, July 2017.
- [24] M. Balint, K. Wolski, K. Myszkowski, H.-P. Seidel, and R. Mantiuk, "Neural partitioning pyramids for denoising monte carlo renderings," in *ACM SIGGRAPH 2023 Conference Proceedings*, ser. SIGGRAPH '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3588432.3591562>
- [25] B. Xu, J. Zhang, R. Wang, K. Xu, Y.-L. Yang, C. Li, and R. Tang, "Adversarial monte carlo denoising with conditioned auxiliary feature modulation," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2019)*, vol. 38, no. 6, pp. 224:1–224:12, 2019.
- [26] J. Yu, Y. Nie, C. Long, W. Xu, Q. Zhang, and G. Li, "Monte carlo denoising via auxiliary feature guided self-attention," *ACM Trans. Graph.*, vol. 40, no. 6, dec 2021. [Online]. Available: <https://doi.org/10.1145/3478513.3480565>
- [27] J. Back, B.-S. Hua, T. Hachisuka, and B. Moon, "Self-supervised post-correction for monte carlo denoising," in *ACM SIGGRAPH 2022 Conference Proceedings*, ser. SIGGRAPH '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: <https://doi.org/10.1145/3528233.3530730>
- [28] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.
- [29] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.
- [30] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," 2020.
- [31] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," 2022.
- [32] J. Wang, Z. Yue, S. Zhou, K. C. Chan, and C. C. Loy, "Exploiting diffusion prior for real-world image super-resolution," in *arXiv preprint arXiv:2305.07015*, 2023.
- [33] T. Yang, X. X. Peiran Ren, and L. Zhang, "Pixel-aware stable diffusion for realistic image super-resolution and personalized stylization," in *arXiv:2308.14469*, 2023.
- [34] H. Hu, K. C. Chan, Y.-C. Su, W. Chen, Y. Li, K. Sohn, Y. Zhao, X. Ben, B. Gong, W. Cohen *et al.*, "Instruct-imagen: Image generation with multi-modal instruction," *arXiv preprint arXiv:2401.01952*, 2024.
- [35] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.
- [36] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2noise: Learning image restoration without clean data," 2018.
- [37] A. T. Áfra, "Intel® Open Image Denoise," 2024, <https://www.openimagedenoise.org>.
- [38] A. Sauer, T. Karras, S. Laine, A. Geiger, and T. Aila, "Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis," in *International conference on machine learning*. PMLR, 2023, pp. 30105–30118.
- [39] A. Sauer, F. Boesel, T. Dockhorn, A. Blattmann, P. Esser, and R. Rombach, "Fast high-resolution image synthesis with latent adversarial diffusion distillation," *arXiv preprint arXiv:2403.12015*, 2024.
- [40] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh, "Video generation models as world simulators," 2024. [Online]. Available: <https://openai.com/research/video-generation-models-as-world-simulators>
- [41] A. Blattmann, T. Dockhorn, S. Kulal, D. Mendelevitch, M. Kilian, D. Lorenz, Y. Levi, Z. English, V. Voleti, A. Letts, V. Jampani, and R. Rombach, "Stable video diffusion: Scaling latent video diffusion models to large datasets," 2023.

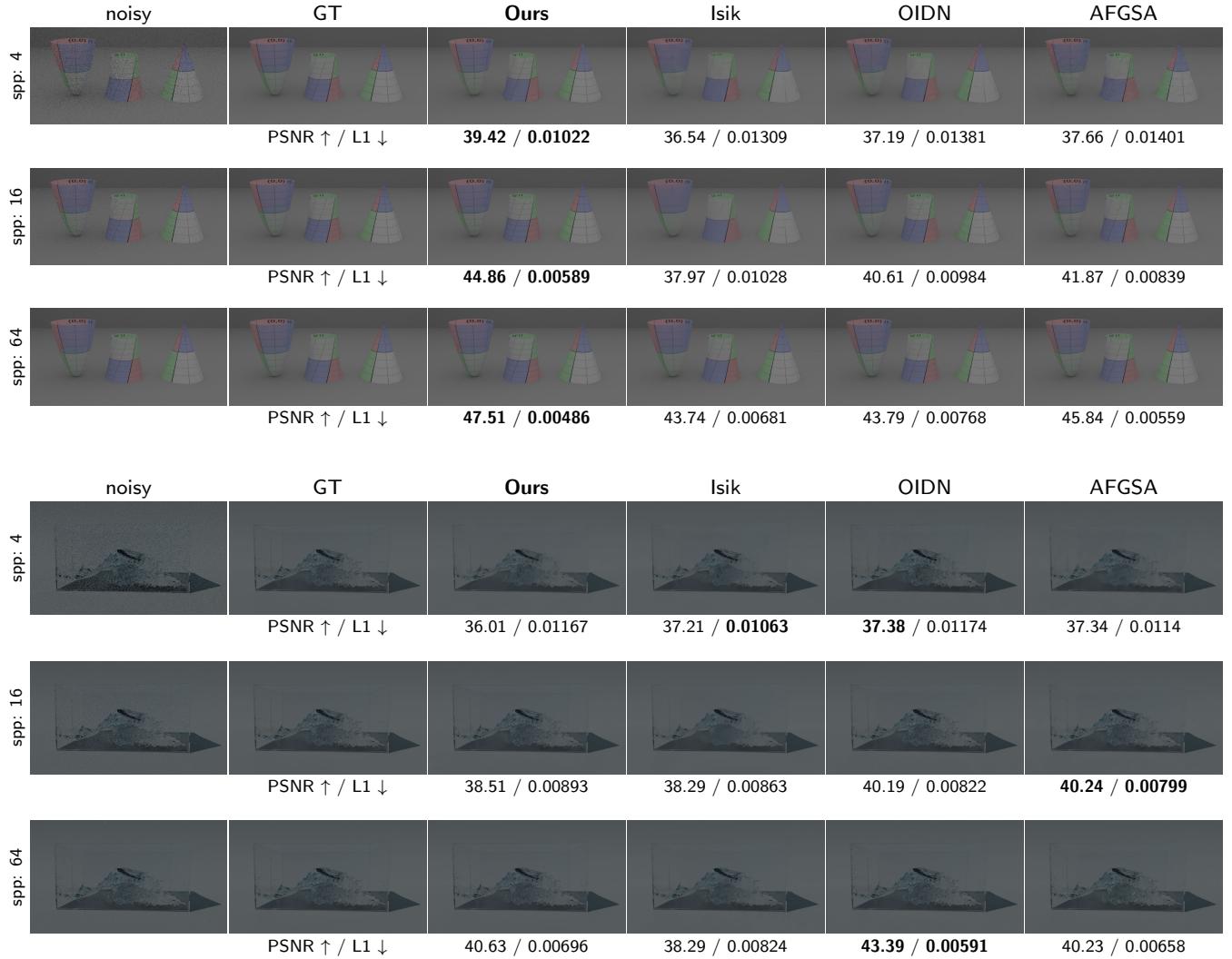


Fig. 7: Additional samples from our test set.

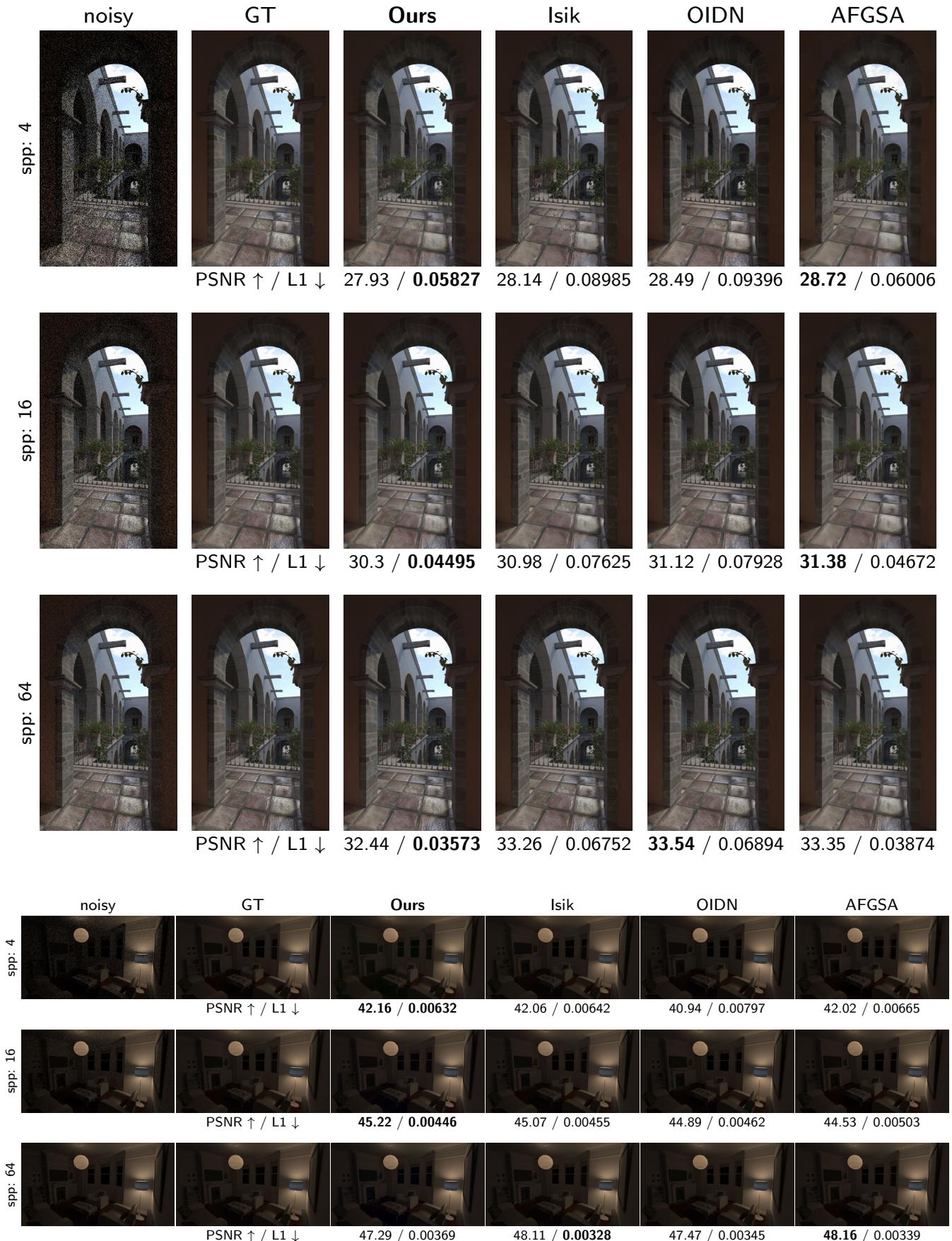


Fig. 8: Additional samples from our test set.

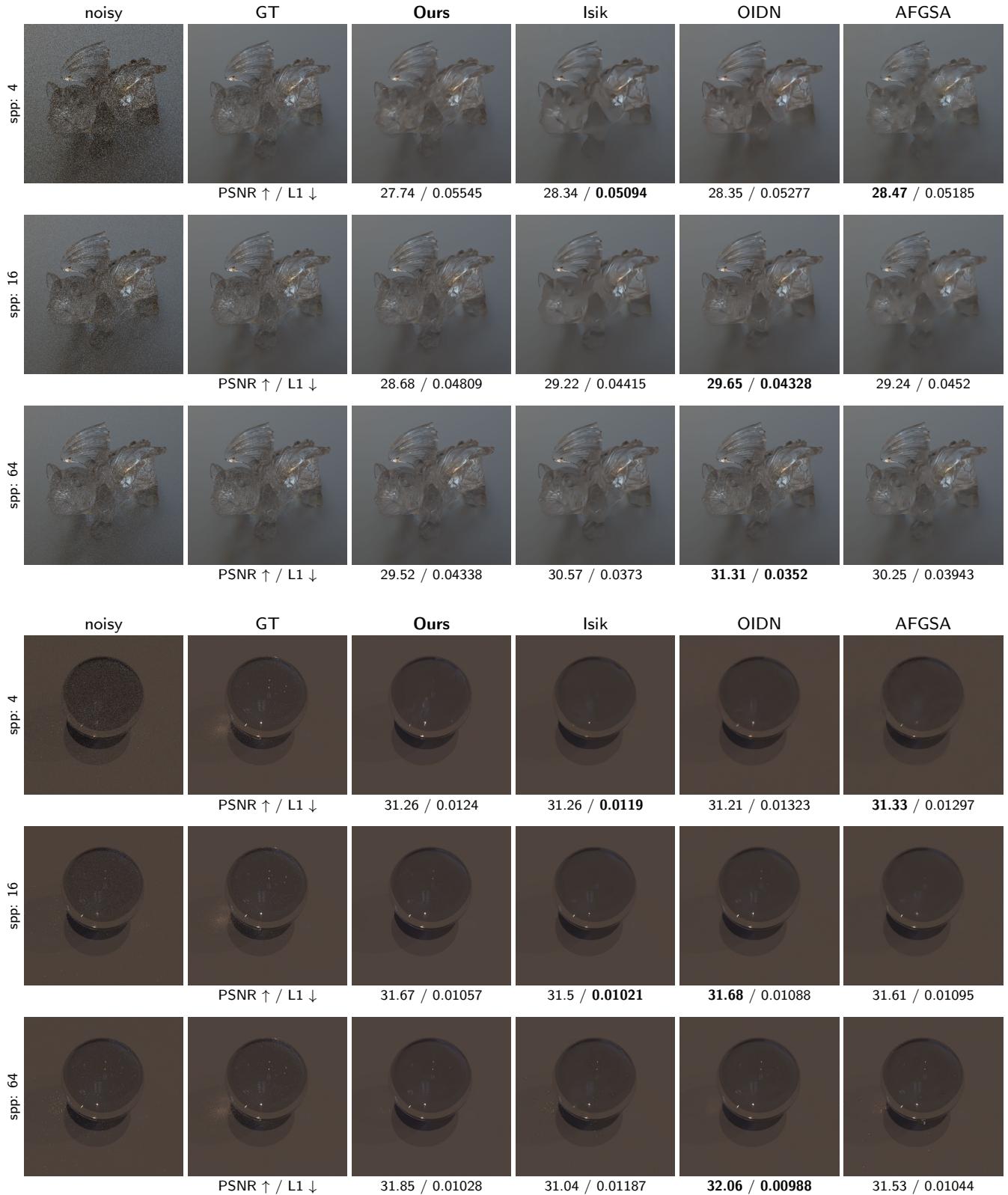


Fig. 9: Additional samples from our test set.

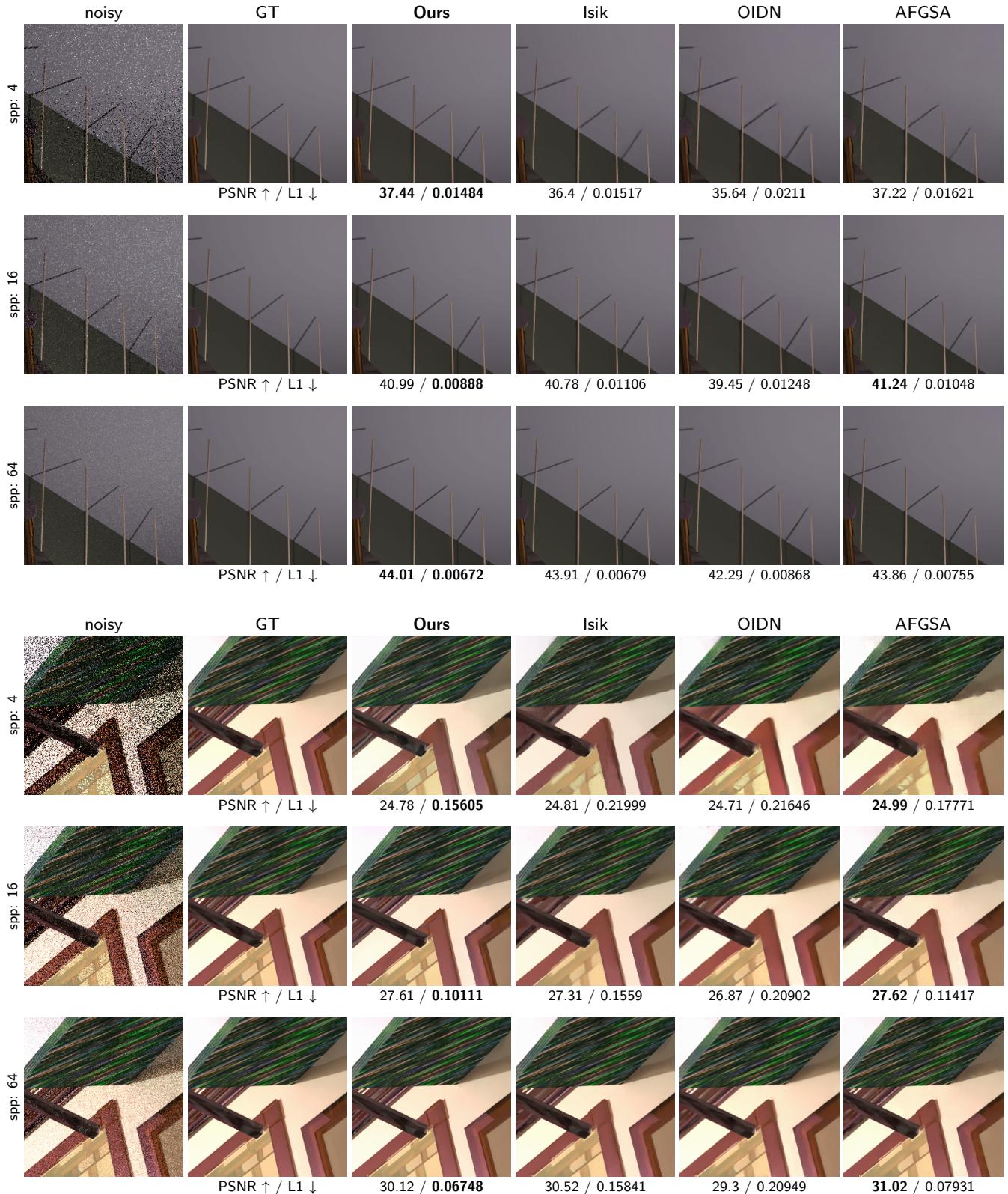


Fig. 10: Additional samples from our test set.



Fig. 11: Additional samples from our test set.