Denoising Monte Carlo Renders with Diffusion Models

Vaibhav Vavilala
UIUC
vv16@illinois.edu

Rahul Vasanth
UIUC
rvasant2@illinois.edu

David Forsyth
UIUC
daf@illinois.edu

Abstract

Physically-based rendering lies at the core of a number of creative applications in gaming, cinema, and design. Unfortunately, these synthetic images often contain unwanted 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. 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. Qualitative examination of the reconstructions suggests that the image prior applied by a diffusion method strongly favors reconstructions that are "like" real images – so have straight shadow boundaries, curved specularities and no "fireflies."

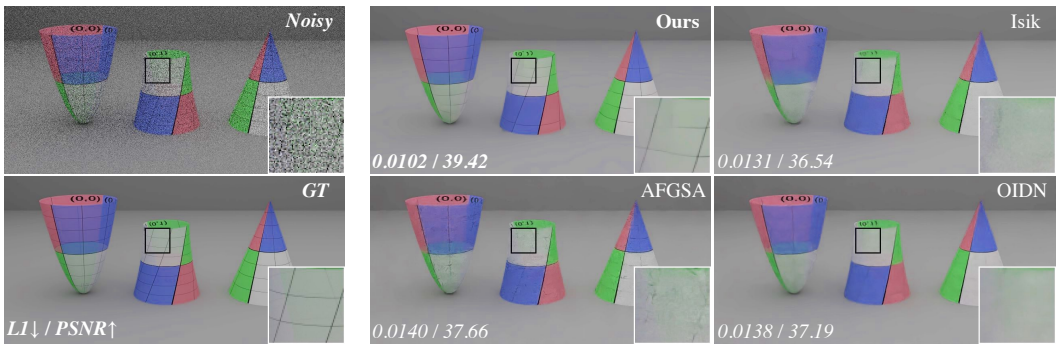

Figure 1: We present a denoiser based on a pixel-space diffusion model. Because our method has a strong prior of what a real image looks like, it can generalize better on out-of-distribution images. Notice how competing methods produce unwanted artifacts like splotchy/blurry regions unlike ours.

Introduction and Method

A number of learning-based methods have been developed to denoise Monte Carlo renders, with many widely deployed in industry [1, 4, 7]. Transformers, GANs, and efficient small models have all been investigated, but none take advantage of large-scale pretrained image models that have seen billions of images. Here, we present a method to denoise MC renderings via a pixel space diffusion model. We find that ControlNet [14] is not suitable for this task, because it relies on a latent-space diffusion model, whose 8x8 latent code patch size caps the quality of the model. Instead, we build our own Control Module on top of the open source DeepFloyd Stage II model [2], based on the architecture of [12]. The weights of the base diffusion model are frozen during training, but a conditional encoder

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

that accepts all the render buffers (noisy radiance, depth, normals, albedo, etc.) is trainable. Our encoder adds information to feature maps at varying spatial resolutions to the decoder block of the base diffusion model. **Dataset** We train and evaluate on our own procedurally-generated dataset of approx. 45k 256-res frames rendered from PBRT [10], as well as the open-source noisebase dataset [5]. These datasets consist of random arrangements of objects, camera pose, texture, and lighting, rendered with a path tracer at varying sampling rates and auxiliary buffers. We compare with existing SOTA works. We retrain Isik [7] and AFGSA [13] on our dataset, and use the pretrained benchmark OIDN [1]. Similar to other works, we apply log tonemapping to ensure high dynamic range is preserved.

Results

We present a qualitative evaluation in Fig 1. Notice that our approach synthesizes clean lines in the undersampled regions, whereas existing works, which do not have a notion of what a real image looks like, struggle to reconstruct these details. We then present a quantitative evaluation in Table 1. Across a range of standard benchmarks, our method outperforms existing work. Our method requires approx. 2.8 seconds to denoise a 256x256 image and 63 seconds to denoise an HD 1080x1920 frame (without skipping time steps) on an A40 GPU. We use mixed-precision during inference and the SUPER27 DDPM sampler in DeepFloyd. Even though diffusion is more expensive than single-pass methods, the cost of denoising remains much lower than the cost of rendering real-world scenes to convergence, which can be dozens of hours. Even scenes that fit into GPU memory can take dozens of minutes to render to convergence, which dwarfs the cost of denoising low-ray estimates. Future work may consider temporal coherence and few-step diffusion models.

Table 1: **Quantitatively, our method is competitive with SOTA**. We compare error metrics 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. All methods are run in single-frame mode and evaluation metrics do not take into account temporal performance. Our method outcompetes all others in L1, which is computed in HDR $[0-6]$ space, as well as FoVVDP. We are also competitive in other metrics, which evaluate both pixel-wise and perceptual quality. DINO [6, 9] and CLIP measure cosine similarity of the generated and GT global image feature vector. In the **fourth row**, we disable our Control Module and denoise with an off-the-shelf DeepFloyd Stage II model, presenting the best numbers after ablating the number of skipped denoising time steps and the aug_level parameter. Applying pure DeepFloyd to low-ray estimates is not successful. **Fifth row** Thus, conditioning our method with render buffer information via our Control Module makes a significant difference to performance.

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0279	38.672	0.1130	0.914	0.932	0.0494	8.699
Isik [7]	0.0499	38.828	0.0871	0.945	0.955	**0.0476**	8.835
OIDN [1]	0.0638	36.329	0.1192	0.915	0.904	0.0691	8.645
DeepFloyd-II [2]	0.1009	27.583	0.3860	0.742	0.804	0.1390	6.513
Ours	**0.0237**	**39.130**	**0.0748**	**0.948**	**0.965**	0.0487	**8.888**

4spp

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0184	42.076	0.0708	0.939	0.958	0.0350	9.222
Isik [7]	0.0449	42.044	0.0560	0.964	0.972	0.0346	9.283
OIDN [1]	0.0537	39.869	0.0858	0.937	0.924	0.0459	9.158
DeepFloyd-II [2]	0.0825	28.988	0.350	0.775	0.819	0.1076	7.150
Ours	**0.0156**	**42.343**	**0.0499**	**0.965**	**0.975**	**0.0338**	**9.328**

16spp

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0142	**45.091**	0.0467	0.956	0.972	**0.0260**	9.562
Isik [7]	0.0433	45.055	**0.0341**	**0.977**	**0.982**	**0.0260**	9.593
OIDN [1]	0.0488	42.664	0.0640	0.954	0.940	0.0329	9.482
DeepFloyd-II [2]	0.0696	29.777	0.317	0.786	0.830	0.0973	7.261
Ours	**0.0113**	44.953	0.0346	0.975	**0.982**	0.0261	**9.616**

64spp

References

[1] Attila T. Áfra. Intel® Open Image Denoise, 2024. https://www.openimagedenoise.org.

[2] Stability AI. Deepfloyd if. https://github.com/deep-floyd/IF.

[3] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):15:1–15:23, 2020.

[4] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017)*, 36(4), 2017.

[5] Martin Balint, Krzysztof Wolski, Karol Myszkowski, Hans-Peter Seidel, and Rafał Mantiuk. Neural partitioning pyramids for denoising monte carlo renderings. In *ACM SIGGRAPH 2023 Conference Proceedings*, New York, NY, USA, 2023. Association for Computing Machinery.

[6] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2023.

[7] Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. Interactive monte carlo denoising using affinity of neural features. *ACM Trans. Graph.*, 40(4), 2021.

[8] Rafał K. Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. Fovvideovdp: a visible difference predictor for wide field-of-view video. *ACM Trans. Graph.*, 40(4), 2021.

[9] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023.

[10] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[11] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[12] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[13] Jiaqi Yu, Yongwei Nie, Chengjiang Long, Wenju Xu, Qing Zhang, and Guiqing Li. Monte carlo denoising via auxiliary feature guided self-attention. *ACM Trans. Graph.*, 40(6), 2021.

[14] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023.

A Supplemental Material

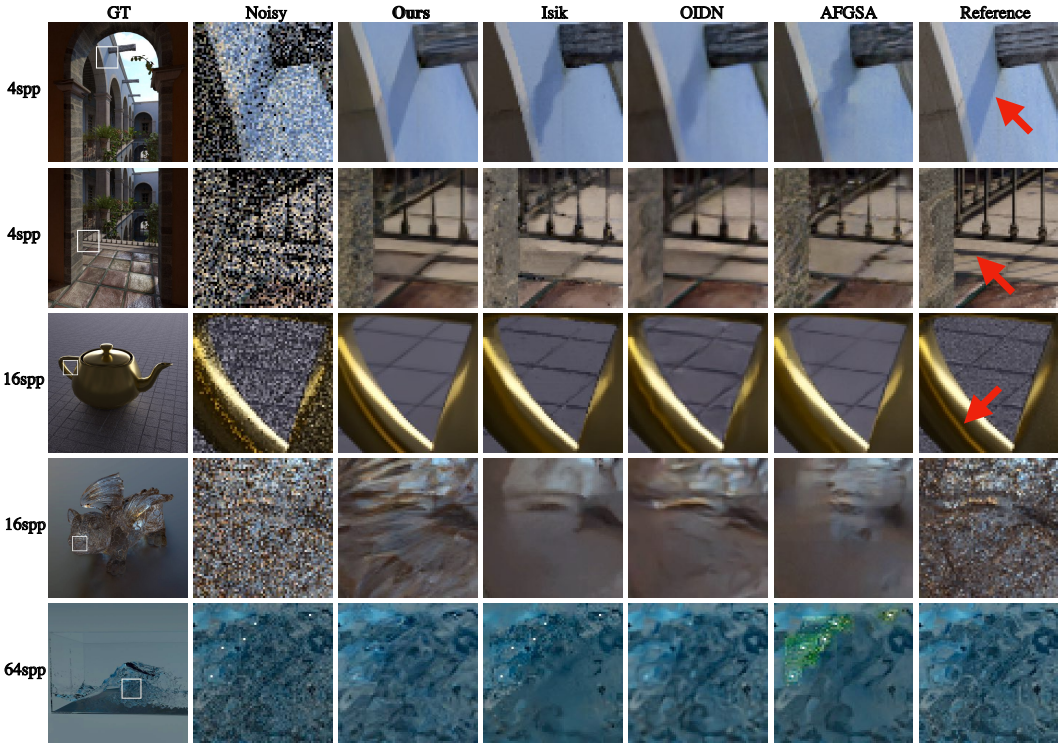

Figure 2: Qualitatively, our reconstructions look like real images, because DeepFloyd has a very strong notion of what an image looks like (it has seen a huge dataset) and because the conditioning buffers offer strong guidelines (e.g. normals, albedo, and depth). In the **final column**, red arrows point 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.

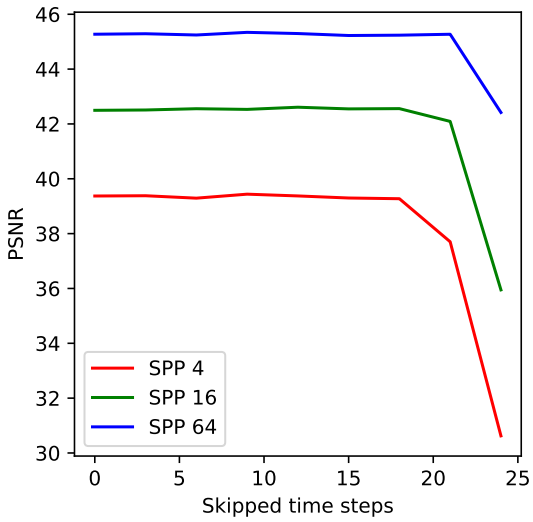

Figure 3: This experiment suggests that we can skip around 18 of the 27 denoising time steps with negligible loss in quality, making diffusion models even more practical. At inference time, we can add gaussian noise to the low-spp render instead of starting from pure noise. The gaussian noise strength needs to be sufficient to overcome the variance in the noisy render.

Table 2: Evaluation on the noisebase dataset (`https://github.com/balintio/noisebase`). **Quantitatively, our method is competitive with SOTA**. We compare error metrics across different spp settings (2, 4, 8, 32) for various methods on the noisebase test set, consisting of several scenes at full HD. Across all sampling rates, we are the best method for most metrics, which evaluate both pixel-wise and perceptual quality. DINO [6, 9] and CLIP measure cosine similarity of the generated and GT global image feature vector.

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.163	24.8	0.436	0.946	0.887	0.167	6.34
OIDN [1]	**0.0990**	**26.8**	0.421	0.974	0.924	0.148	**7.07**
Ours	0.114	26.5	**0.401**	**0.975**	**0.939**	**0.147**	6.94
2spp							

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0992	27.3	0.394	0.963	0.914	0.125	7.07
OIDN [1]	0.0769	28.2	0.392	0.980	0.936	0.123	**7.54**
Ours	**0.0748**	**28.5**	**0.368**	**0.982**	**0.950**	**0.114**	7.50
4spp							

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0834	28.8	0.361	0.975	0.938	0.104	7.61
OIDN [1]	0.0611	29.7	0.363	0.986	0.947	0.101	**8.00**
Ours	**0.0558**	**30.3**	**0.339**	**0.988**	**0.959**	**0.0901**	**8.00**
8spp							

Method	**L1** ↓	**PSNR** ↑	**LPIPS** ↓	**DINO** ↑ [9]	**CLIP** ↑ [11]	**FliP** ↓ [3]	**FoVVDP** ↑ [8]
AFGSA [13]	0.0616	31.7	0.310	0.988	0.970	0.0760	8.42
OIDN [1]	0.0387	**33.4**	0.318	0.994	0.965	0.0634	**8.93**
Ours	**0.0371**	**33.4**	**0.296**	**0.995**	**0.973**	**0.0589**	8.89
32spp							