

Stochastic Gaussian Splat Rendering Denoising using Lightweight Autoencoders

Maj Bregar¹, Uroš Šmajdek¹, Matija Marolt¹, Ciril Bohak¹, Žiga Lesar^{1,2}

¹Univerza v Ljubljani, Fakulteta za računalništvo in informatiko

²Univerza v Novi Gorici

E-mail: mb95331@student.uni-lj.si, {uros.smajdek, matija.marolt, ciril.bohak}@fri.uni-lj.si, ziga.lesar@ung.si

Abstract

Stochastic visibility sampling provides an efficient alternative to depth-sorted alpha blending for Gaussian splat rendering, but produces noisy frames with visible temporal flickering. We present a lightweight recurrent U-Net-style autoencoder for denoising such stochastic renders. The model propagates hidden states across frames to exploit temporal information and reconstruct stable outputs. We evaluate non-recurrent, recurrent, and confidence-augmented variants on seen and unseen scenes. The results show that temporal recurrence improves reconstruction quality and temporal consistency, while the tested confidence-map setup does not improve performance.

1 Introduction

Rendering Gaussian splats typically requires depth sorting followed by alpha blending, which can be costly for large dynamic views. Stochastic visibility sampling avoids per-view sorting by relying on depth testing and sampling fragments according to their opacity. Each fragment is treated as a binary visibility sample, and averaging repeated samples yields a Monte Carlo estimate of the alpha-composited image.

This replaces alpha blending with simple stochastic visibility tests, but individual frames contain substantial sampling noise. Since the selected fragments vary randomly between frames, the resulting RGB and depth images exhibit pixel-level noise and temporal flickering, making them unsuitable for direct interactive display.

This noise is difficult to isolate from RGB values alone. Unlike sparse high-energy Monte Carlo noise, stochastic splat rendering produces dense sampling noise with plausible splat colors, so noisy pixels always contain valid colors. Available auxiliary information is also limited, since Gaussian splat scenes do not directly provide explicit surfaces, surface normals, UV parameterizations, material textures, or object-level geometry.

We address this problem as an end-to-end learned reconstruction task. Our method uses a lightweight recurrent autoencoder that takes a noisy 720p render as input and predicts a temporally stable denoised image suitable for real-time visualization.

2 Related work

A substantial body of work addresses image denoising using deep neural networks. In this work, we focus on lightweight approaches capable of denoising sequences of noisy frames in real time.

Chaitanya *et al.* [1] propose a recurrent denoising autoencoder for low-sample Monte Carlo image sequences. Their method uses a U-Net-style autoencoder with recurrent blocks that propagate hidden states between frames, improving temporal stability. The network also supports auxiliary per-pixel buffers such as depth and normals. Our model follows this general design, using a recurrent autoencoder structure with renderer-derived auxiliary inputs.

Meng *et al.* [2] introduce a real-time neural denoiser for 1 spp Monte Carlo renderings based on a differentiable bilateral grid. Their network learns a guide image for splatting and slicing noisy samples in bilateral space, combining end-to-end learning with efficient edge-aware filtering. This provides a lightweight alternative to direct autoencoder-based reconstruction.

Fan *et al.* [3] propose a real-time kernel-prediction denoiser for very low-sample Monte Carlo path-traced images. Instead of predicting full per-pixel kernels directly, their method predicts a compact kernel-map encoding and reconstructs filtering kernels with an efficient decoder. This represents another efficient approach to real-time learned denoising.

In contrast to these Monte Carlo denoising methods, we focus on stochastic Gaussian splat rendering, where the noise is dense, temporally unstable, and supported by limited auxiliary information.

3 Method

3.1 Denoising Model Architecture

We use a recurrent U-Net-style autoencoder with five encoder and decoder stages, as shown in Figure 1. The network receives the renderer-constructed input tensor and predicts a denoised RGB image. We selected the U-Net architecture for it's

Each encoder stage applies convolutional feature extraction followed by a recurrent convolutional update using the hidden state from the previous frame. Hidden states are maintained independently at each encoder scale, allowing temporal information to propagate at multiple

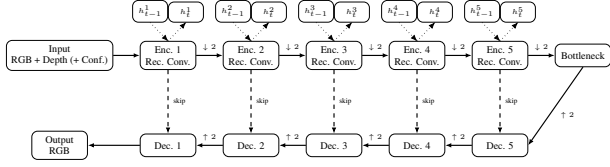


Figure 1: Recurrent U-Net-style denoising autoencoder. Each encoder stage updates a scale-specific hidden state from the previous frame. Decoder stages use skip connections to preserve high-frequency spatial detail.

spatial resolutions. Starting from a base width of 24 channels, the feature width increases by a factor of $\frac{4}{3}$ at each encoder stage. This keeps the model lightweight by growing the channel count more slowly than the common doubling strategy.

The decoder upsamples the bottleneck representation and concatenates the corresponding recurrent encoder features through skip connections. The network predicts a residual RGB correction from the full noisy input tensor and adds it to the noisy RGB input:

$$\hat{I} = \sigma(f(X) + I), \quad (1)$$

where, X denotes the full renderer-derived input tensor, while I denotes its noisy RGB component.

3.2 Training Objective

The recurrent model is trained on short sequences generated by the stochastic renderer. Each training sample contains $N = 7$ consecutive frames, and a random 128×128 crop is extracted during dataset loading. Hidden states are initialized to zero at the beginning of each sequence and propagated through time using truncated backpropagation.

The training objective combines spatial reconstruction, gradient consistency, and temporal consistency. The spatial term is an L_1 loss between the prediction $\hat{I}_t$ and the target image T_t at sequence time t :

$$L_{\text{spatial}}(t) = \left| \hat{I}_t - T_t \right|. \quad (2)$$

The gradient term represents the L_1 loss after applying a fixed Laplacian-of-Gaussian filter G :

$$L_{\text{gradient}}(t) = \left| G(\hat{I}_t) - G(T_t) \right|. \quad (3)$$

We enforce temporal consistency by matching frame-to-frame differences between predictions and targets:

$$L_{\text{temporal}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left| (\hat{I}_t - \hat{I}_{t-1}) - (T_t - T_{t-1}) \right|. \quad (4)$$

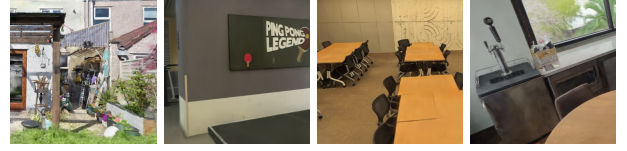
We accumulate spatial and gradient losses over 7-frame sequences using fixed Gaussian weights that assign larger contributions to more recent frames. The complete sequence loss is:

$$L_{\text{sequence}} = \lambda_s L_s + \lambda_g L_g + \lambda_t L_{\text{temporal}}, \quad (5)$$

where L_s and L_g denote the weighted sequence averages of the spatial and gradient losses, and λ_s , λ_g , and λ_t are the corresponding loss weights.

3.3 Training Setup

Training used sampled frames from three enclosed Gaussian splat scenes: a backyard garden [4], an office space ¹,



(a) Garden (b) Gym (c) Office (d) Kitchen

Figure 2: Ground-truth renders of all scenes.

and an indoor gymnasium [5]. These scenes provide broad spatial coverage, making random patch extraction more reliable. Figure 2 shows the training scenes and a held-out Kitchen [6] scene used for evaluation.

For each sampled camera view, the frame was rendered twice. The stochastic splat renderer produced the noisy RGB and depth inputs, while depth-sorted alpha blending produced the ground-truth target. Camera paths were generated automatically by rotating around each scene and inserting short pauses, allowing the model to distinguish changes caused by camera motion from stochastic sampling noise.

The training dataset contained 516 frames from Garden, 408 from Gymnasium, and 410 from Office. Training examples were constructed from 30-frame sequences by extracting 7-frame subsequences, and every fifth generated sequence was used for validation and checkpoint selection. All rendering, training, and inference used 720p images.

Table 1: Training hyperparameters.

Parameter	Value
Render Resolution	1280 × 720
Epochs	200
Optimizer	AdamW
Scheduler	Cosine annealing
Learning rate	10^{-3} to $5 \cdot 10^{-6}$
Base channels	24
Loss weights ($\lambda_s, \lambda_g, \lambda_t$)	(0.8, 0.1, 0.1)
Sequence length	7
Patch size	128 × 128

3.4 Confidence Map Variant

We additionally evaluated a recurrent model with a per-pixel temporal confidence map as an auxiliary input, shown in Figure 3. The confidence map estimates how much reliable history supports each current pixel.

Each frame is treated as a new observation. Each non-background pixel is reprojected into the previous frame, where the accumulated color, depth, and confidence are sampled. Confidence increases when consecutive observations agree in depth, screen-space position, and color, but drops when a new observation disagrees, such as after disocclusion or motion. The carried confidence also controls how strongly the accumulated color history is blended with the current noisy color.

This variant was intended to expose temporal reliability explicitly to the network, helping it distinguish stable accumulated information from camera motion and stochastic sampling noise.

¹<https://supersplat.scene/d152f2df>



Figure 3: Visualization of confidence map. Dark pixels represent lower confidence due to stochastic sampling noise and camera motion.

Table 2: Evaluation metrics for different models, training datasets, and evaluation datasets.

Model	Train set	Eval set	RMSE	SSIM
U-Net	Garden	Kitchen	0.037	0.871
Recurrent	Garden	Kitchen	0.029	0.895
Recurrent	Garden + Gym + Office	Kitchen	0.025	0.901
Recurrent + Conf.	Garden	Kitchen	0.033	0.878
Recurrent + Conf.	Garden + Gym + Office	Kitchen	0.067	0.824
U-Net	Garden	Garden	0.067	0.624
Recurrent	Garden	Garden	0.054	0.704
Recurrent	Garden + Gym + Office	Garden	0.054	0.696
Recurrent + Conf.	Garden	Garden	0.056	0.694
Recurrent + Conf.	Garden + Gym + Office	Garden	0.082	0.653

3.5 Evaluation Setup

We evaluated three U-Net-style denoising models: a non-recurrent five-stage U-Net (denoted *U-Net*), a recurrent model using noisy RGB and depth inputs (denoted *Recurrent*), and a recurrent model with an additional confidence-map input (denoted *Recurrent + Confidence*).

The recurrent models were trained using two training dataset configurations. The first, denoted Garden, contained only frames from the Garden scene, while the second contained frames from all three training scenes. The non-recurrent U-Net was trained only on the Garden dataset.

Each model was evaluated on separate 200-frame evaluation sequences. The Garden evaluation sequence measured performance on a seen training scene, while the Kitchen sequence was excluded from training and was used to evaluate cross-scene generalization.

We use root mean square error (RMSE) and structural similarity index measure (SSIM) as the primary evaluation metrics. RMSE measures pixel-wise reconstruction accuracy and penalizes larger errors more strongly, while SSIM measures structural similarity between the denoised output and the target rendering. Together, these metrics capture both numerical reconstruction error and perceptual image quality.

4 Results

The quantitative evaluation results are shown in Table 2.

The recurrent models outperform the non-recurrent U-Net on both evaluation sequences, showing the benefit of temporal information. The multi-scene-trained model im-

Table 3: Inference-only runtime for the evaluated denoising configurations. FPS denotes average processed frames per second, and frame time is reported in milliseconds.

Metric	Rec.	Rec. + Conf.	Rec. + Conf. FP16	PyTorch	Compiled PyTorch
Average FPS	21.1	20.7	26.1	45.9	138.3
Frame time (ms)	47.3	48.2	38.2	21.7	7.2

proves performance on the unseen Kitchen evaluation sequence, achieving lower RMSE and higher SSIM than the Garden-only recurrent model. This suggests that adding scenes improves generalization. However, on the Garden evaluation sequence, the Garden-only model remains slightly better, indicating that scene-specific training can still be advantageous when desiring maximal quality.

The confidence-map input does not improve performance. In all cases, confidence-augmented models produce higher RMSE and lower SSIM than the corresponding recurrent models without confidence input. Although it was designed to represent per-pixel reliability, this signal may not directly correspond to reconstruction usefulness. The recurrent model may already infer similar information from recurrent hidden states, making the explicit confidence channel redundant or difficult to exploit.

4.1 Qualitative Results

Figure 4 shows a qualitative comparison on the Kitchen evaluation sequence. The non-recurrent U-Net produces substantial blur and noticeable color inaccuracies. The recurrent model without confidence produces sharper reconstructions and better preserves scene structure.

The confidence-map variant does not show a clear visual improvement over the basic recurrent model, matching the quantitative results in Table 2.

All trained models still exhibit noticeable blurring, especially in high-frequency regions. This suggests that the current loss function does not sufficiently penalize missing fine detail.

4.2 Runtime Performance and Model Size

The final recurrent model with 24 base channels contains 849,513 trainable parameters.

Because denoising inference dominates the runtime of the rendering pipeline, Table 3 reports inference-only performance for several model and backend configurations. Rendering and other pipeline stages are excluded from these measurements.

The main inference pipeline uses the WebGPU² ONNX Runtime³ backend. For comparison, we also evaluated native PyTorch execution in both eager and compiled modes. All tests were performed on a machine with an Intel i5-13600K CPU and an AMD RX 7900 XTX GPU.

Runtime measurements show that the inference backend is the main practical bottleneck. Browser-based ONNX inference provides broad compatibility, but is substantially slower than native PyTorch execution. The compiled PyTorch result suggests that the model itself is lightweight

²<https://webgpu.org/>

³<https://onnxruntime.ai/>

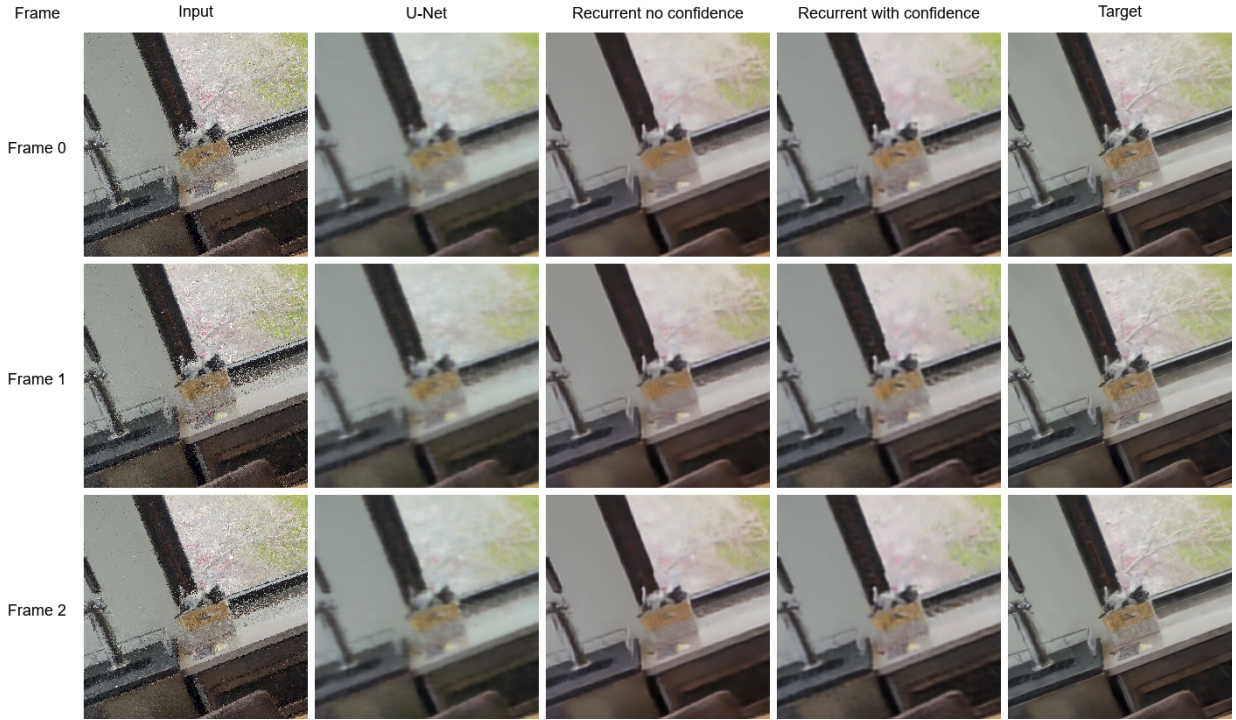


Figure 4: Qualitative comparison on the Kitchen evaluation sequence. The non-recurrent U-Net model was trained only on Garden dataset, while the Recurrent and Recurrent + Confidence models were trained on the full three-scene dataset. Rows show consecutive frames, while columns compare the noisy input, model outputs, and reference image for the same frame.

enough for real-time use, while browser deployment would require further ONNX or WebGPU-specific model design and optimizations.

5 Conclusion

We presented a lightweight recurrent U-Net-style denoiser for stochastic Gaussian splat rendering. The model reconstructs stable RGB frames from noisy stochastic renders using auxiliary depth information and recurrent hidden states.

The results show that temporal recurrence improves reconstruction quality and temporal consistency compared to frame-independent denoising. This could be additionally supported by evaluation with time-aware metrics such as: temporal warping error or temporal SSIM. Adding an explicit confidence map did not improve performance. Multi-scene training improved performance on the unseen Kitchen scene, while Garden-only training remained slightly better on the seen Garden scene.

Future work should focus on reducing residual blur, improving cross-scene generalization with larger datasets, and optimizing the inference backend for real-time browser deployment. It would be interesting to check if such approach could be integrated into real-time web-based rendering systems such as [7, 8].

6 Acknowledgment

The research was conducted within the projects Cell visualization of unified microscopic data and procedurally generated sub-cellular structures (J2-50221) and Geospatial

Information Technologies for Resilient and Sustainable Society (GeoAI) (GC-0006), both funded by the Slovenian Research and Innovation Agency (ARIS).

References

- [1] C. R. A. Chaitanya et al. “Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder”. In: *ACM Transactions on Graphics* 36.4 (2017), pp. 1–12. DOI: 10.1145/3072959.3073601.
- [2] X. Meng et al. “Real-time Monte Carlo Denoising with the Neural Bilateral Grid”. In: *Eurographics Symposium on Rendering - DL-only Track*. Ed. by C. Dachsbacher and M. Pharr. The Eurographics Association, 2020. ISBN: 978-3-03868-117-5. DOI: 10.2312/sr.20201133.
- [3] H. Fan et al. “Real-time Monte Carlo Denoising with Weight Sharing Kernel Prediction Network”. In: *Computer Graphics Forum* 40.4 (2021), pp. 15–27. DOI: 10.1111/cgf.14338.
- [4] M. Ostromecki. *Garden*. <https://superspl.at/scene/52ef6279>. Gaussian splat scene. Licensed under CC BY 4.0: <https://creativecommons.org/licenses/by/4.0/>. 2026.
- [5] Ethan. *Table Tennis Room*. <https://superspl.at/scene/0d9e613b>. Gaussian splat scene. Licensed under CC BY 4.0: <https://creativecommons.org/licenses/by/4.0/>. 2026.
- [6] A. Small. *CoWorking Kitchen*. <https://superspl.at/scene/4294afa9>. Gaussian splat scene. Licensed under CC BY 4.0: <https://creativecommons.org/licenses/by/4.0/>. 2026.
- [7] C. Bohak et al. “RenderCore – a new WebGPU-based rendering engine for ROOT-EVE”. In: *CHEP 2023*. Newport News (VA), 2023, str. 1–8. DOI: 10.48550/ARXIV.2312.11729. arXiv: 2312.11729. URL: <https://doi.org/10.48550/arXiv.2312.11729>.
- [8] A. Kurtagić et al. “A data-oriented point-based rendering engine on WebGPU”. In: *CESCG : Central European Seminar on Computer Graphics*, 27 - 30 April 2026, Smolenice Castle, [Smolenice, Slovakia]. 2026, pp. 1–8.