MVSplat: Efficient 3D Gaussian Splatting from Sparse Multi-View Images

Supplementary Material

Yuedong Chen[1]✉, Haofei Xu[2,3], Chuanxia Zheng[4], Bohan Zhuang[1], Marc Pollefeys[2,5], Andreas Geiger[3], Tat-Jen Cham[6], and Jianfei Cai[1,6]

[1]Monash University [2]ETH Zurich [3]University of Tübingen, Tübingen AI Center
[4]VGG, University of Oxford [5]Microsoft [6]Nanyang Technological University
donydchen.github.io/mvsplat

A More Experimental Analysis

All experiments in this section follow the same settings as in Sec. 4.3 unless otherwise specified, which are trained on RealEstate10K [9] and reported by averaging over the full test set.

Using cost volume in pixelSplat. In the main paper, we have demonstrated the importance of our cost volume design for learning feed-forward Gaussian models. We note that such a concept is general and is not specifically designed for a specific architecture. To verify this, we replace pixelSplat's probability density-based depth prediction module with our cost volume-based approach while keeping other components intact. The results shown in Tab. A again demonstrate the importance of the cost volume by significantly outperforming the original pixelSplat, indicating the general applicability of our proposed method.

Setup	PSNR↑	SSIM↑	LPIPS↓
pixelSplat (w/ probability density depth) [1]	25.89	0.858	0.142
pixelSplat (w/ our MVSplat cost volume depth)	**26.63**	**0.875**	**0.122**

Table A: Using cost volume in pixelSplat. Our cost volume-based depth prediction approach can also be used in the pixelSplat [1] model by replacing its probability density-based depth branch and its performance can be significantly boosted, which demonstrates the general applicability of our method.

Ablations on the backbone Transformer. Differing from pixelSplat [1] and GPNR [6] that are based on the Epipolar Transformer, we adopt Swin Transformer [3] in our backbone (*i.e.*, Multi-view feature extraction as in Sec. 3.1). To compare the Transformer architectures, we conduct ablation experiment by replacing our Swin Transformer with the Epipolar Transformer in Tab. B. Since the Swin Transformer does not need to sample points on the epipolar line, where the sampling process is computationally expensive, our model is more efficient than the Epipolar Transformer counterpart. Besides, there is no clear difference

Setup	Time (s)	PSNR↑	SSIM↑	LPIPS↓
MVSplat (w/ Epipolar Transformer [1])	0.055	26.09	0.865	0.133
MVSplat (w/ Swin Transformer)	**0.038**	26.12	0.864	0.133

Table B: Comparisons of the backbone Transformer. Our Swin Transformer [3]-based architecture is more efficient than the Epipolar Transformer counterpart in pixelSplat [1] since the expensive epipolar sampling process is avoided. Besides, there is no clear difference observed in their rendering qualities.

Setup	Render Time (s)	PSNR↑	SSIM↑	LPIPS↓
MVSplat (1 Gaussian per pixel)	0.0015	26.39	0.869	0.128
MVSplat (3 Gaussians per pixel)	0.0023	26.54	0.872	0.127

Table C: Comparisons of Gaussians' number per pixel. Increasing the number of Gaussians improves the performance but slows down the rendering speed.

observed in their rendering qualities, demonstrating the superiority of our Swin Transformer-based design.

Ablations on the Gaussian numbers per pixel. Unlike pixelSplat that predicts three Gaussians per image pixel, our MVSplat by default only predicts one per pixel. However, MVSplat can also benefit from increasing the number of Gaussians. As reported in Tab. C, our default model can be further boosted by predicting more Gaussians, but it also impacts the rendering speed. We choose to predict one Gaussian per pixel to balance performance and rendering speed.

Ablations on backbone initialization. In our implementations, we initialize our backbone (*i.e.*, Multi-view feature extraction as in Sec. 3.1) with the publicly available UniMatch [8] pretrained weight[1], where its training data has no overlap with any datasets used in our experiments. However, as reported in Tab. D, our model can also be trained with random initialization ("w/ random init") and still outperforms previous state-of-the-art method pixelSplat [1], whose backbone is initialized with the weights pretrained on ImageNet with a DINO objective. To compensate for the lack of a proper initialization, our random initialized model needs more training iterations (450K) to reach the performance of the UniMatch initialized model. The results further demonstrate our model's high efficiency and effectiveness, where strong performance can still be obtained without relying on pretraining on large-scale datasets.

Validation curves of the ablations. To better perceive the performance difference of different ablations, we provide the validation curves throughout the whole training phase in Fig. A). The cost volume plays a fundamental role in our full model, and interestingly, the model without cross-view attention suffers from overfitting after certain training iterations.

[1] `https://github.com/autonomousvision/unimatch`

Method	PSNR↑	SSIM↑	LPIPS↓
pixelSplat w/ DINO init (300K) [1]	25.89	0.858	0.142
MVSplat w/ UniMatch init (300K)	26.39	0.869	0.128
MVSplat w/ random init (300K)	26.01	0.863	0.133
MVSplat w/ random init (450K)	26.29	0.868	0.128

Table D: Comparisons of backbone initialization. By default we train our models for 300K iterations with the publicly available UniMatch pretrained weights as initialization. However, our model can also be trained with random initialization ("w/ random init") and still outperforms previous state-of-the-art method pixelSplat. The random initialized model needs more training iterations (450K) to reach the performance of the UniMatch initialized model.

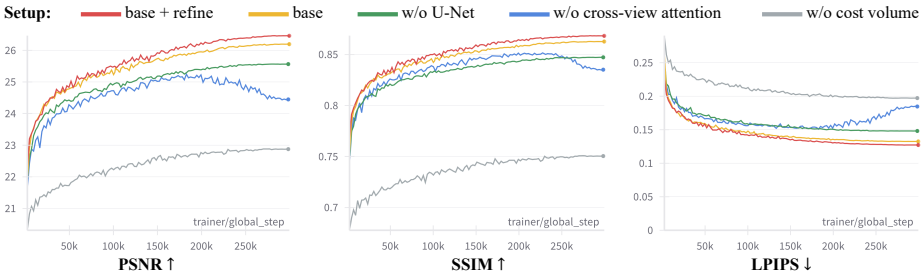

Fig. A: Validation curves of the ablations. The setup of each model is illustrated on the top, which refers to the same one as in Tab. 3 of the main paper. The cost volume plays a fundamental role in our full model, and interestingly, the model without cross-view attention suffers from over-fitting after certain training iterations.

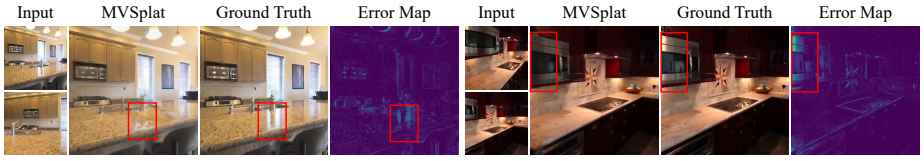

Fig. B: Failure cases. Our MVSplat might be less effective on the non-Lambertian and reflective surfaces.

Limitation. Our MVSplat might be less effective on non-Lambertian and reflective surfaces, as shown in Fig. B. Integrating the rendering with additional BRDF properties and training the model with more diverse datasets might be helpful for addressing this issue in the future.

Potential negative societal impacts. Our model may produce unreliable outcomes, particularly when applied to complex real-world scenes. Therefore, it is imperative to exercise caution when implementing our model in safety-critical

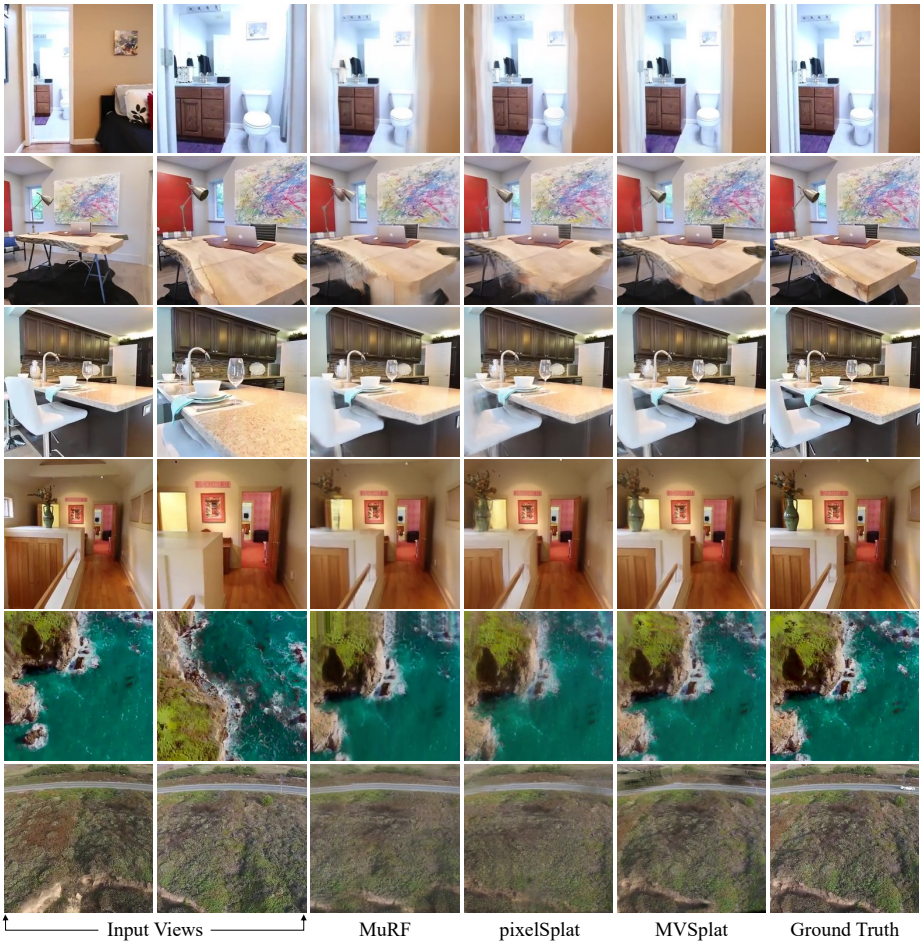

Fig. C: More comparisons with the state of the art. These are the extended visual results of Fig. 3. Scenes of the first four rows come from the RealEstate10K, whilst scenes of the last two rows come from ACID. Our MVSplat performs the best in all cases.

situations, *e.g.*, when augmenting data to train models for autonomous vehicles with synthetic data rendered from our model.

B More Visual Comparisons

In this section, we provide more qualitative comparisons of our MVSplat with state-of-the-art methods on the RealEstate10K and ACID in Fig. C. We also provide more comparisons with our main comparison model pixelSplat [1] regarding geometry reconstruction (see Fig. D) and cross-dataset generalizations

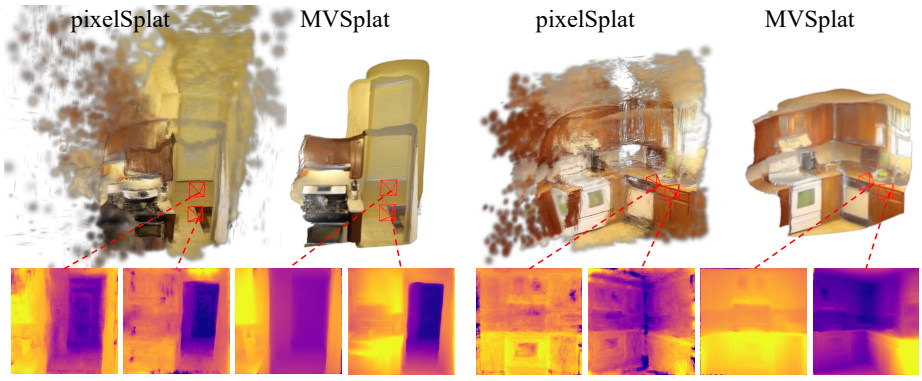

Fig. D: More comparisons of 3D Gaussians (top) and depth maps (bottom). These are the extended visual results of Fig. 4. Extra depth-regularized fine-tuning is *not* applied to either model. 3D Gaussians and depth maps predicted by our MVSplat are both of higher quality than those predicted by pixelSplat.

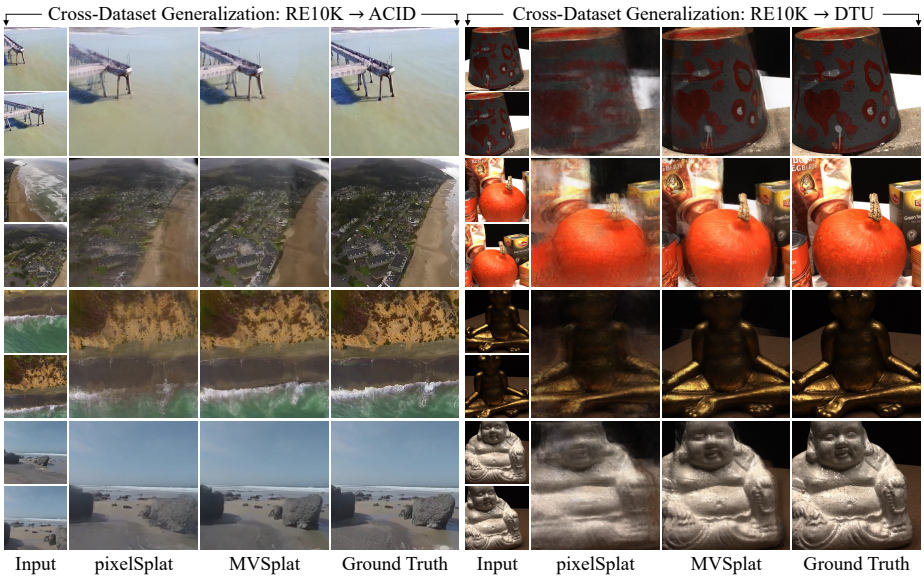

Fig. E: More comparisons of cross-dataset generalization. These are the extended visual results of Fig. 5. By training on indoor scenes (RealEstate10K), our MVSplat generalizes much better than pixelSplat to outdoor scenes (ACID) and object-centric scenes (DTU), showing the superior of our cost volume-based architecture.

(see Fig. E). Besides, readers are referred to the project page for the rendered videos and 3D Gaussians models (provided in ".ply" format).

C More Implementation Details

Network architectures. Our shallow ResNet-like CNN is composed of 6 residual blocks [2], In the last 4 blocks, the feature is down-sampled to half after every 2 consecutive blocks by setting the convolution stride to 2, resulting in overall $4\times$ down sampling. The following Transformer consists of 6 stacked Transformer blocks, each of which contains one self-attention layer followed by one cross-attention layer. Swin Transformer's [3] local window attention is used and the features are split into 2×2 in all our experiments. For the cost volume refinement, we adopt the 2D U-Net implementations from [4]. We keep the channel dimension unchanged throughout the U-Net as 128, and apply 2 times of $2\times$ down-sampling, with an additional self-attention layer at the $4\times$ down-sampled level. Inspired by existing multi-view based models [5,7], we also flatten the features before applying self-attention, allowing information to propagate among different cost volumes. The following depth refinement U-Net also shares a similar configuration, except that we apply 4 times of $2\times$ down-sampling and add attentions at the $16\times$ down-sampled level.

More training details. As aforementioned, we initialize the backbone of MVSplat in all experiments with the UniMatch [8] pretrained weight. Our default model is trained on a single A100 GPU. The batch size is set to 14, where each batch contains one training scene, including two input views and four target views. Similar to pixelSplat [1], the frame distance between two input views is gradually increased as the training progressed. For both RealEstate10K and ACID, we empirically set the near and far depth plane to 1 and 100, respectively, while for DTU, we set them to 2.125 and 4.525 as provided by the dataset.

References

1. Charatan, D., Li, S., Tagliasacchi, A., Sitzmann, V.: pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In: CVPR (2024)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
3. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021)
4. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022)
5. Shi, Y., Wang, P., Ye, J., Long, M., Li, K., Yang, X.: Mvdream: Multi-view diffusion for 3d generation. ICLR (2024)
6. Suhail, M., Esteves, C., Sigal, L., Makadia, A.: Generalizable patch-based neural rendering. In: ECCV (2022)
7. Tang, J., Chen, Z., Chen, X., Wang, T., Zeng, G., Liu, Z.: Lgm: Large multi-view gaussian model for high-resolution 3d content creation. arXiv (2024)
8. Xu, H., Zhang, J., Cai, J., Rezatofighi, H., Yu, F., Tao, D., Geiger, A.: Unifying flow, stereo and depth estimation. PAMI (2023)
9. Zhou, T., Tucker, R., Flynn, J., Fyffe, G., Snavely, N.: Stereo magnification: learning view synthesis using multiplane images. TOG p. 65 (2018)