

Figure 8. Spatially Efficient Self-Attention: While employing all queries as query in the self-attention mechanism, we leverage Farthest Point Sampling (FPS) to downsample certain 3D Gaussians. This process enables the extraction of their corresponding queries as keys and values within the self-attention operation.

A. Appendix

A.1. Model details

A.1.1. Spatial efficient self attention (SESA)

While our 3D-aware deformable attention mechanism is notably efficient, the computational cost and memory occupation mainly arises in the self-attention component, particularly when dealing with a large number of 3D Gaussians. However, updating each 3D Gaussian with information from all others is not always necessary because those neighbouring 3D Gaussians usually carry similar information.

To mitigate this issue, as depicted in Fig. 8 and drawing inspiration from [47], we introduce a technique aimed at reducing the size of the key and value components while maintaining the query component unaltered within the self-attention process. The core insights behind this approach is that while each 3D Gaussian requires updating, not every other 3D Gaussian needs to contribute to this update. We achieve this by selectively updating each query solely with a subset of corresponding queries linked to other 3D Gaussians.

More specifically, we leverage the Fast Point Sampling (FPS) algorithm commonly used in point cloud methodologies like PointNet [32] and PointNet++ [33]. We employ

the Gaussian centers μ to identify the most distantly located points and use these points to index the queries. By implementing this strategy, our model significantly reduces the model’s overall memory footprint while preserving essential information exchange among the Gaussians.

A.1.2. Analysis for the regression of the center of 3D Gaussians and the initialization for 3D Gaussians and queries

From camera space to world space In [40], the Gaussian centers are located in each input view’s camera space as shown in Eq. (6),

$$\mu_{cam} = \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} = \begin{bmatrix} u_1 d + \Delta_x \\ u_2 d + \Delta_y \\ d + \Delta_z \end{bmatrix} \quad (6)$$

The center coordinates $x_{cam}, y_{cam}, z_{cam}$ are parameterized by depth d and offsets $(\Delta_x, \Delta_y, \Delta_z)$. u_1, u_2 are the UV coordinates of the ray passing through the corresponding input image. This design represents each point with multiple Gaussians, potentially introducing ‘ghosting’ due to concatenation issues at various points caused by depth inaccuracies and tend to shortcut input views [50]. In our framework, we define unitary Gaussians in world space, project their centers to each input view for feature retrieval, as depicted in Fig. 3(a). The centers of Gaussians can be written as $\mu_{world} = [x_{world}, y_{world}, z_{world}]$.

3D Gaussian initialization However, during the initial training phases, discrepancies between the 3D Gaussian centers and ground truth often result in imprecise selection of image features at sampling points, presenting challenges for model convergence as shown in Sec. 4.2. To address this issue, we utilize a initialization that directly regress 3D Gaussian parameters from each pixel of the image features. We first train a coarse network that predicts 3D Gaussians for each pixel of each input view under the camera space of that view. After that, the 3D Gaussians are transformed and fused to get the 3D Gaussians under world space. The role of this network is to provide a coarse initialization of 3D Gaussians for the subsequent refinement. We use the UNet architecture, which has the same structure to the feature extractor.

Moreover, We employ a relative coordinate system, where the camera poses for all views are known. The initial input view is established as the world coordinates (with the camera pose represented by the identity matrix), and subsequently, all other views are transformed to align with the reference view. This approach allows us to represent all 3D data within this consistent relative coordinate system.

A.2. Experiment details

A.2.1. Datasets

We utilized a refined subset of the Objaverse LVIS dataset [6] for both training and validating our model. This subset was curated to exclude low-quality models, resulting in a dataset containing 36,044 high-quality objects. This open-category dataset encompasses a diverse range of objects commonly encountered in everyday scenarios. For training, we leveraged rendered images provided by zero-1-to-3 [25] for the random input setting. Each object in the dataset is associated with approximately 12 random views, accompanied by their respective camera poses. We partitioned 99% of the objects for training purposes, reserving the remaining 1% for validation. During training, we randomly selected a subset of views as input while using all 12 views for supervision. Each rendered image has a resolution of 512×512 , which we downsampled to 128×128 . For the fixed view setting, we render the images with fixed views as input and 32 more random views with elevation in $(-30, 30)$ degrees for supervision.

To evaluate our model’s performance in open-category settings, we conducted tests on the Google Scanned Objects (GSO) benchmark [7]. The GSO dataset comprises 1,030 3D objects categorized into 17 classes. For this evaluation, we utilized rendered images sourced from Free3D [61], which consist of 25 random views along with their corresponding camera poses. Notably, there are no restrictions on the elevation of the rendered views. We utilized the initial views as inputs and the remaining views for assessing our novel view synthesis task. Additionally, we observed that LGM [41] only support fixed-view inputs (e.g., front, left, back, and right). To address this, we evaluated a new rendered GSO dataset at 0 degrees elevation, testing it on 32 random views with elevations ranging from 0 to 30 degrees. To distinguish between the two test sets, we refer to them as GSO and GSO respectively in the following analysis.

A.2.2. Implementation details

We train our model on the setting of 4 views, each time we randomly select 4 views as input and all the views for supervision. For the initialization, we train the model with less views (i.e. 2 views) with resolution 128×128 and generate 16384 3D Gaussians as initialization of the fine stage. In the fine stage, We use 19600 3D Gaussians to represent the 3D object. For the 3D Gaussians from the initialization, we use the mask to remove the background points and padding the number of 3D Gaussians to 19600 by copying some of the remaining 3D Gaussians. The selected 3D Gaussians are then utilized to project queries onto image plane in the refine stage. In each deformable attention layer, we utilize 4 sampling points for each projected 3D Gaussian reference point to sample values on the image.

We use 4 decoder layers and the hidden dimension is

Table 4. Quantitative results for inputting 4 views on GSO dataset.

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Splatter Image	25.7660	0.8932	0.2575
LGM	15.1113	0.8440	0.1592
InstantMesh	17.3073	0.8525	0.1376
Our Model	26.3020	0.9255	0.0836

256. We use a mixed-precision training [31] with BF16 data type. We train our model with Adam [15] optimizer and the learning rate is 0.0001. We take 300K iteration with batch size 4. For the initialization, we train it on 8 3090 GPUs (24G) for 5 days and for the refinement stage, we train it on 8 A100 (80G) for 3 days.

A.3. More results

A.3.1. Input views with random camera poses

Previous methods (LGM and InstantMesh) usually rely on fixed views as input, as they align well with views generated from diffusion models like ImageDream [45]. In real-world scenarios, users are more inclined to provide random views as input. Tab. 4 displays the results when utilizing random 4 views as input on the GSO dataset. Notably, there is a performance drop observed in LGM and InstantMesh with random input views. For Splatter Image, although the PSNR does not reduced much, its SSIM and LPIPS reduced significantly. We provide more visualization in Fig. 15.

PSNR of Splatter Image in Tab. 4 is good but SSIM and LPIPS are not good enough, we further provide the visualization is in Fig. 9.

A.3.2. More example for ‘Ghosting’ problem

We gives more ‘ghosting’ visualization problem by visualize center of Gaussians from each view in different colors, as shown in Fig. 10. Gaussians from different views representing the same part of the object may lays on the different position in the 3D space and thus cause the ‘ghosting’ problem.

A.3.3. More visualization

Image-to-3D conversion represents a fundamental application in 3D generation. Following the methodology of LGM and InstantMesh [41, 52], we first leverage a multi-view diffusion model, ImageDream [45], to generate four pre-determined views. Subsequently, our model is employed for 3D Gaussian reconstruction. A comparative analysis with LGM and InstantMesh is detailed in Tab. 5. For this particular scenario, we utilize the fixed-view GSO test set with elevations ranging between 0 and 30 degrees. Given potential variations in camera poses among the generated multi-views, which may not align precisely with standard front, right, back, and left perspectives, we selectively re-

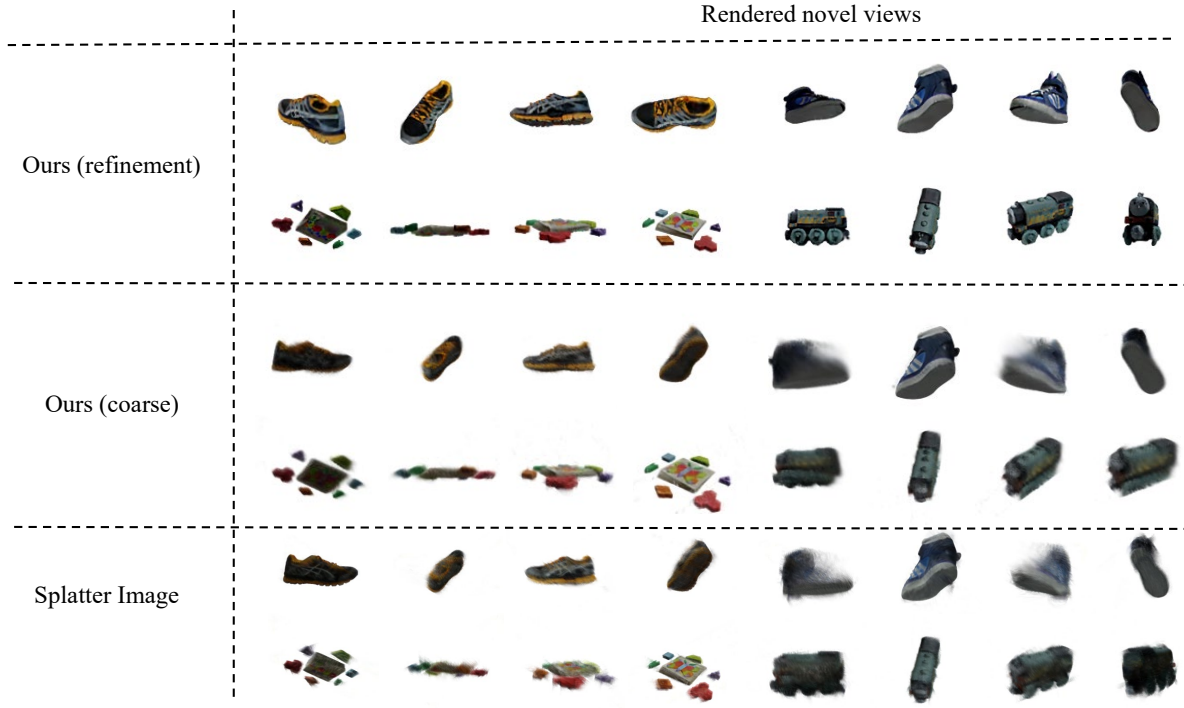


Figure 9. Visualization for Splatter Image with fixed view input and random view input.

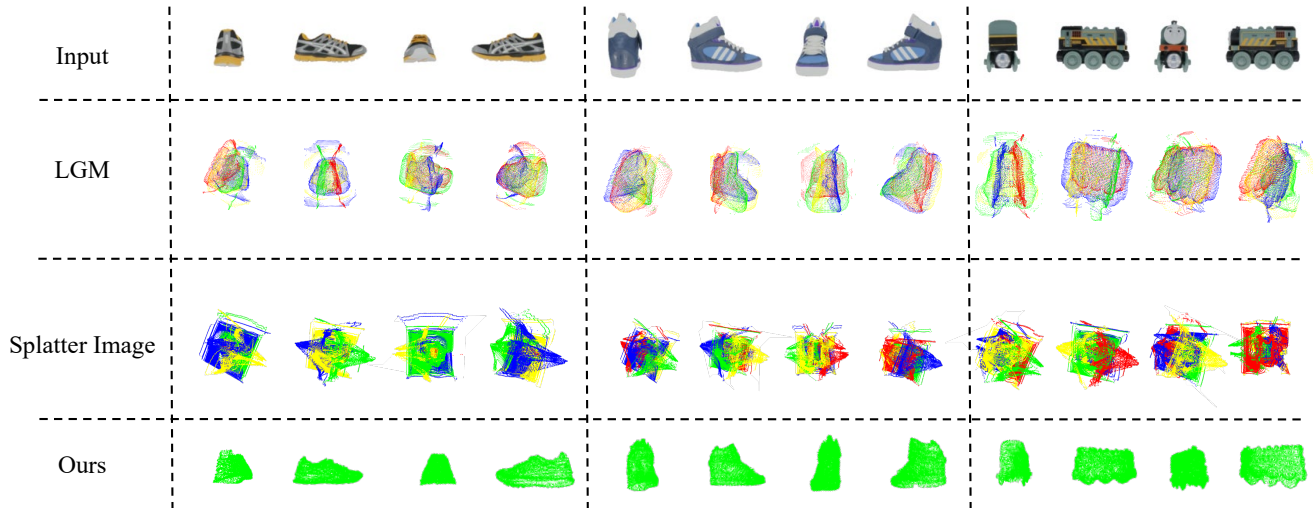


Figure 10. Point clouds of the center of Gaussians from each view. The Gaussians from different views are in different colors.

tain 266 objects that consistently yield accurate images under the provided camera poses.

As shown in Fig. 12, when given limited number of input, neither LGM nor InstantMesh gives the meaningful geometry.

Fig. 13 presents the quantitative results of novel views

rendered by recent models trained on 4 views. When provided with 4 random views as input, LGM [41] demonstrates a loss of geometry and encounters ‘ghosting’ problems stemming from its training on fixed views. In contrast, our approach produces a cohesive 3D Gaussian set that effectively captures object geometries.

Table 5. Quantitative results for single view reconstruction on GSO dataset.

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
LGM [41]	20.8139	0.8581	0.1508
InstantMesh [52]	19.4667	0.8379	0.1842
Our Model	22.3534	0.8567	0.1492

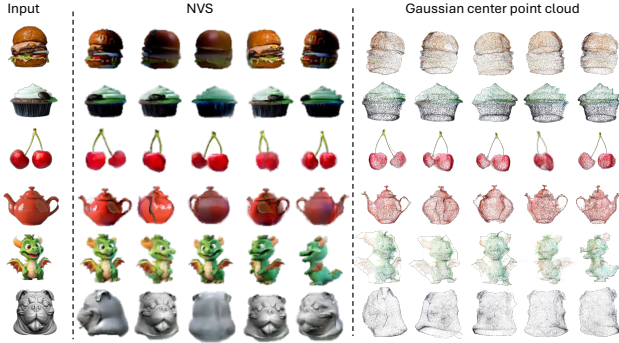


Figure 11. Quality for rendered novel views on in the wild data for inputting 1 view and using ImageDream to generate 4 views.

The figures illustrate that LGM encounters the issue of ‘ghosting’; for instance, there are multiple handles visible for the mushroom teapot. InstantMesh loses some details due to its utilization of a discrete triplane to represent continuous 3D space.

Fig. 14 shows the result of text-to-3D task. We have incorporated text-to-3D capabilities into our model. To assess quality, we employ MVDream [37] to create a single image from a text prompt. Subsequently, a diffusion model is utilized to generate multi-view images, which are then processed by our model to obtain a 3D representation.

The setting of random input view is obvious a more challenging task than the setting of fixed input view, thus our method also inevitably suffers from a performance drop but still perform better than other state-of-the-art methods. As for Splatter Image [40], it also meets a significant performance drop when random input views are used as its SSIM \uparrow decreased from 0.9151 to 0.8932 and LPIPS \downarrow increased from 0.1517 to 0.2575 despite its PSNR \uparrow has a slight increase. We visualize the results of the two settings to show the difference in Fig. 15.

We provide the visualization result with resolution 512 in Fig. 20.

A.3.4. Single image reconstruction

There are common points between our model and Triplane-Gaussian and Instant3D that we all use a unitary representation and use Transformer to regress. For Instant3D, it transforms image to Nerf, making longer rendering time. For Triplane Gaussian, which is a single view reconstruction

model with complex and costly triplane representation, representing compresses 3D space, leading to a lack of detailed information in the 3D structure and imposing a rigid grid alignment that limits flexibility [32, 41]. In the contrast, we use a more efficient way (deformable attention) to decode Gaussians. The comparison between Triplane-Gaussian and our methods is shown in Tab. 6. Triplane Gaussian requires 3D supervision and takes longer inference time while get worse performance comparing to our model. We test on the given light-weight checkpoint in the github on the single view situation. We also test TripoSR [43] on the single image reconstruction setting. As shown in Tab. 6, our model surpass the previous methods on both the performance and the inference speed. We provide the visualization results of our model on single image reconstruction task in Fig. 19

A.3.5. Comparison to masked LGM and Splatter Image

To better explain that the ‘ghosting’ problem is not caused by the background points from previous methods, we provide the results on removing background points of LGM and Splatter Image. LGM uses mask loss to make the most of the pixels contribute to the object itself, even for the background pixels, therefore, removing background use mask makes the results more sparse. It also removing some outliers and thus the rendering results is better as shown in Tab. 7. Splatter Image keep most of the pixels contribute to its original position, making most of the background points still located on a plane instead of the object. Therefore, removing background use mask does not influence the rendering result much but the rendering quality still reduced a little. Moreover, the ‘ghosting’ is not caused by the background points but the mis-alignment of 3D Gaussians from different views, removing the background use mask does not help solving the problem. We show the visualization in Fig. 16

A.3.6. Other number of view results

We present the results of training with varying numbers of views (2, 6, 8) and evaluate the corresponding results with the same number of views in Tab. 8.

Our model is positioned on the ‘sparse view’ setting, which indicates the number of views less then 10, so we only reports the performance of views from 2 to 8 in the main paper. With the increase of input views, information from similar views becomes redundant, so the gain for

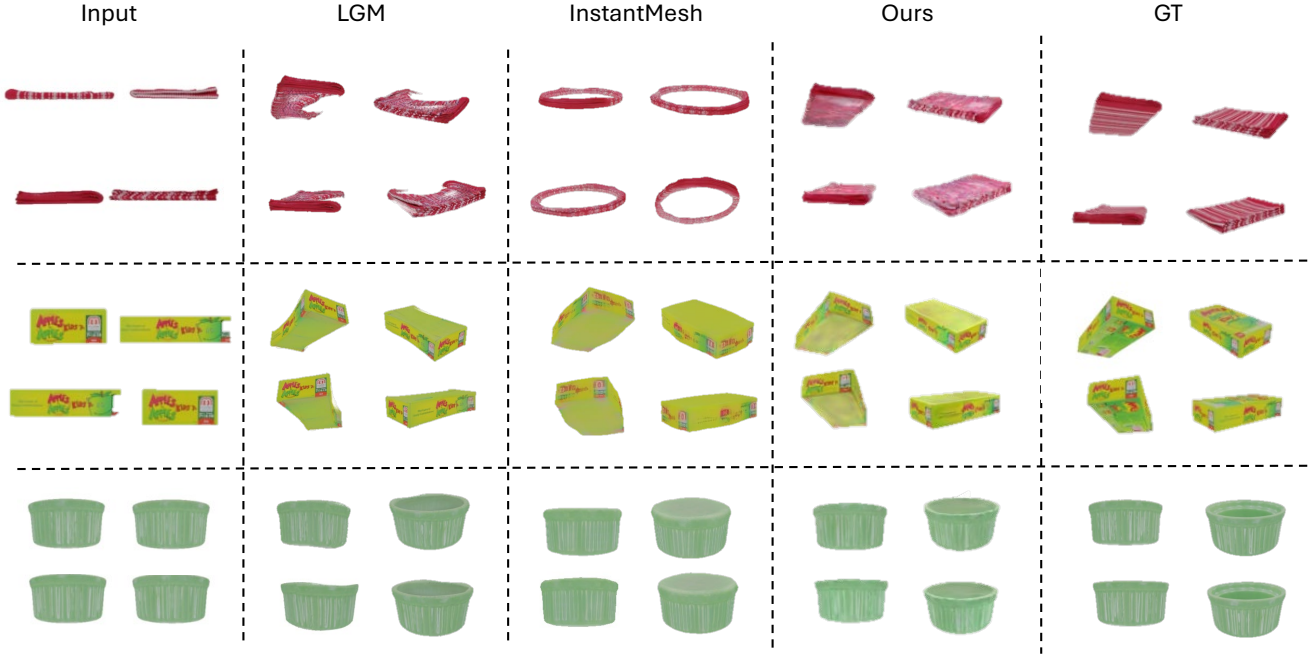


Figure 12. Quality for rendered novel views on GSO dataset for inputting 4 views with resolution 256 LGM large model.

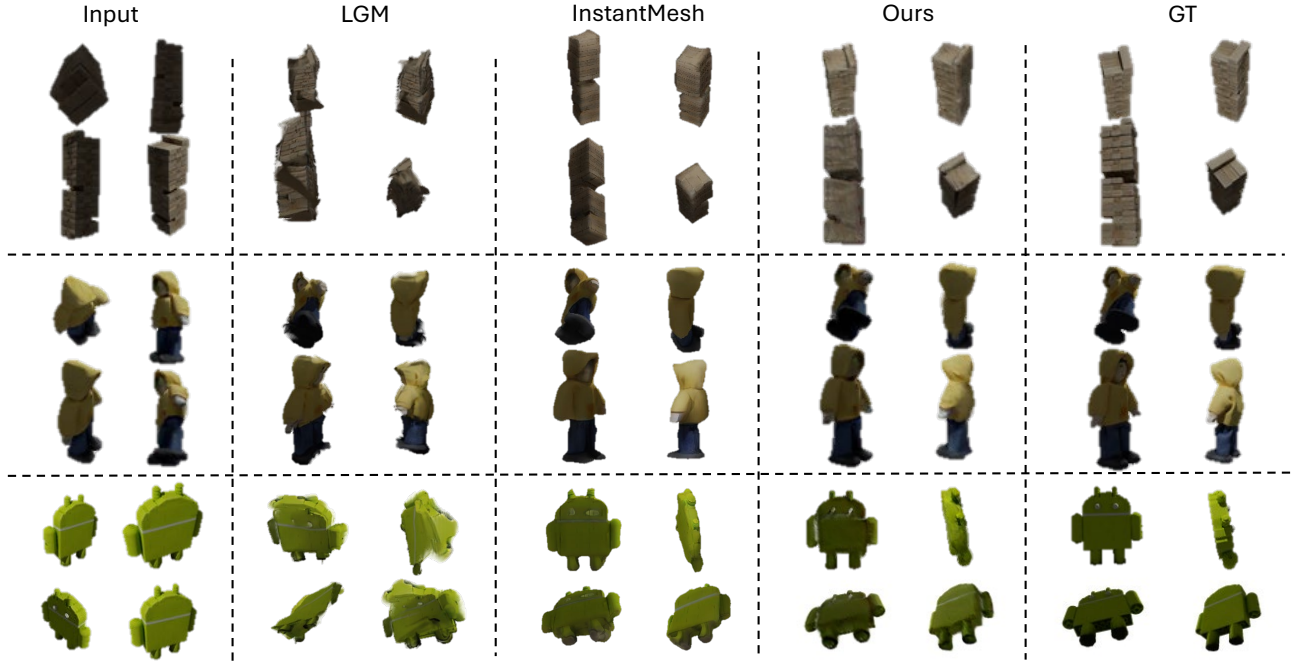


Figure 13. Quality for rendered novel views on GSO dataset for inputting 4 views with random camera poses.

1113 our model has become plateaued while other methods suffer
1114 from performance drop as they cannot handle too many
1115 input views due to the view inconsistent problem. As we
1116 keep increasing the number of input views larger than 8, our

method can still benefit from more input views (as shown in
Fig. 17) while others meet the CUDA-out-of-memory problem.

1117
1118
1119

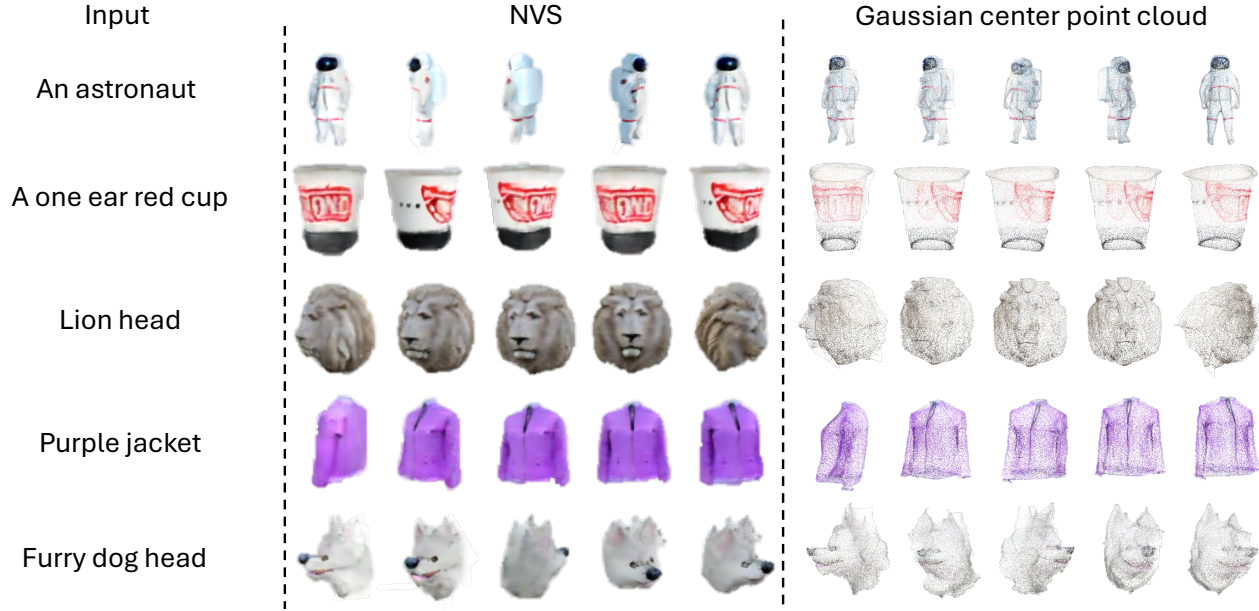


Figure 14. Quality for rendered novel views on inputting text and using MVDream to generate 4 views.

Table 6. Quantitative results trained on Objaverse LVIS and tested on GSO. 3D sup. means need 3D supervision.

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	3D sup.	Inference time	Rendering time
Triplane-Gaussian [63]	18.61	0.853	0.159	✓	1.906	0.0025
TripSR [43]	20.00	0.872	0.149	✗	3.291	22.7312
Ours	23.45	0.897	0.093	✗	0.476	0.0025

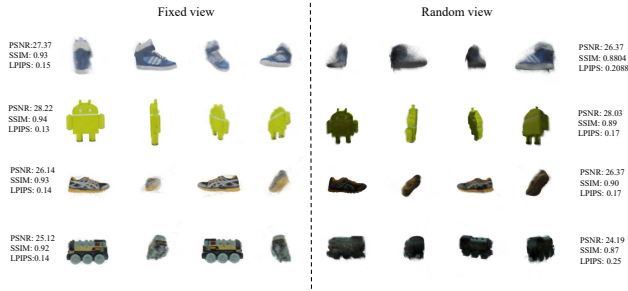


Figure 15. Visualization for Splatter Image with fixed view input and random view input.

Table 7. Comparison between masked and original pixel aligned methods

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
LGM	17.4810	0.7829	0.2180
LGM (masked)	21.6008	0.8608	0.1232
Splatter Image	25.6241	0.9151	0.1517
Splatter Image (masked)	25.0648	0.9147	0.1684

A.3.7. Comparison to MVSpLat and pixelSpLat

We present a comparative analysis involving MVSpLat [5] and pixelSpLat [2] on the GSO dataset by training it on Objaverse. Similar to LGM [41], both aforementioned methods follow a workflow that regress Gaussians from each views within the respective camera spaces and subsequently merge them in the global world space. Despite pixelSpLat’s integration of cross-view-aware features through an epipolar Transformer, accurately forecasting a dependable probabilistic depth distribution based solely on image features remains a formidable task [5]. This limitation often translates to pixelSpLat’s geometry reconstruction exhibiting comparatively lower quality and plagued by noticeable noisy artifacts [5]. Upon examination, we observed that even after isolating points within a visual cone and eliminating background Gaussians, the geometry fails to convey meaningful information, yielding unsatisfactory results.

In contrast, MVSpLat adopts a design that incorporates a cost volume storing cross-view feature similarities for all possible depth candidates. These similarities offer crucial geometric cues for 3D surface localization, leading to more substantial depth predictions. However, akin to Splatter Image, which assigns each pixel a Gaussian and thereby gen-

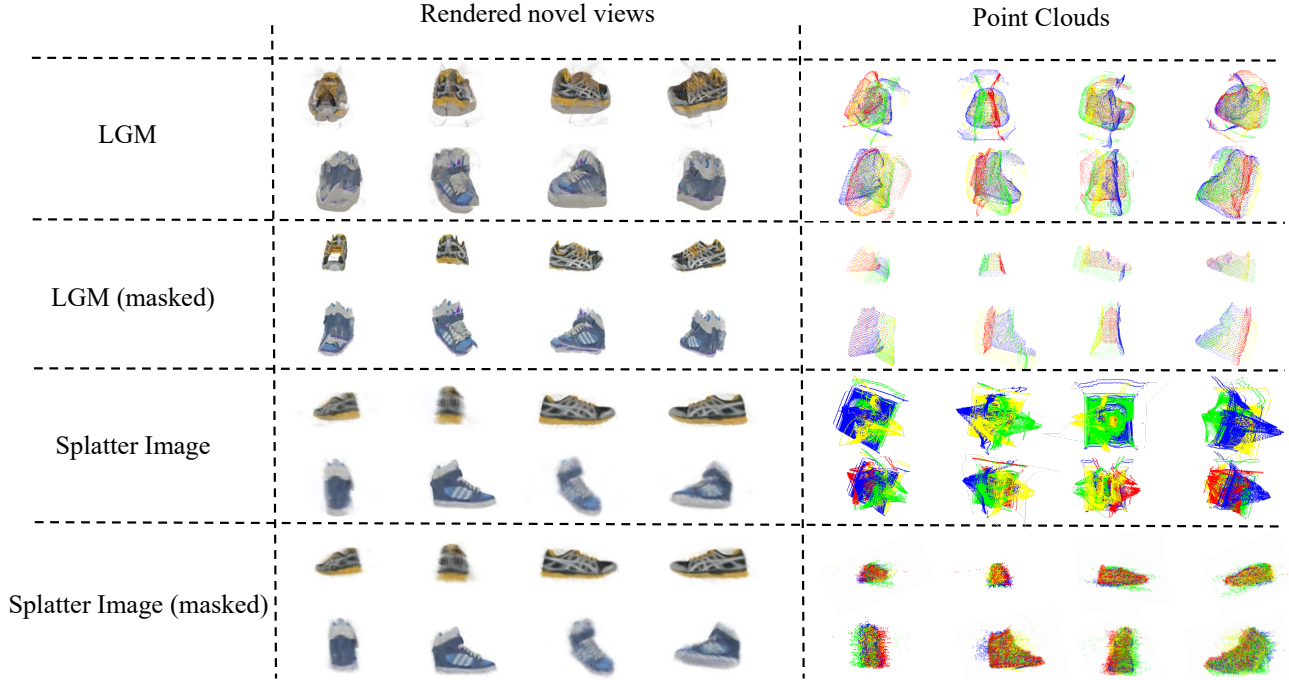


Figure 16. Removing the background use mask for Splatter Image and LGM

Table 8. Quantitative results of novel view synthesis training using 2, 6, and 8 input views, tested on the GSO dataset across 2, 6, and 8 views.

Method	PSNR \uparrow	2 views SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	6 views SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	8 views SSIM \uparrow	LPIPS \downarrow
Splatter Image	22.6390	0.8889	0.1569	26.1225	0.9178	0.1620	26.4588	0.9166	0.1714
Our Model	23.8384	0.8995	0.1254	28.1035	0.9489	0.0559	28.8262	0.9537	0.0492

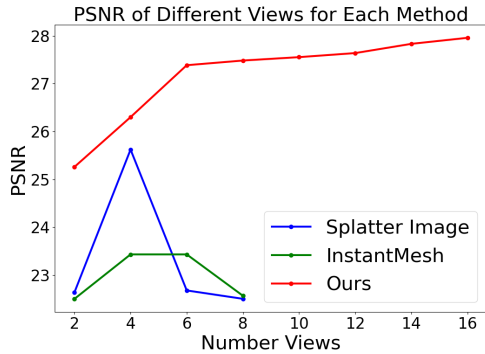


Figure 17. Visualization for Splatter Image with fixed view input and random view input.

erates a planar representation rather than the object itself, MVSpLat’s approach may obscure object details due to occlusion by background Gaussians from other viewpoints, resulting in suboptimal outcomes.

To address this issue, we selectively mask the position-

ing of Gaussians on background pixels, focusing solely on rendering Gaussians contributing to the object itself. This adjustment reveals significant ‘ghosting’ problems, as illustrated in Fig. 18. In the figure, we present the centers of Gaussians generated from different views in different color and the novel views are rendered from the Gaussians from all views. Furthermore, the elaborate incorporation of cross-view attention mechanisms and cost volumes in MVSpLat leads to extended inference times and heightened memory requirements as shown in Tab. 9.

A.4. Ablation study

Number of views for the initialization We add the ablation study on the number of images used during the the training of initialization. The results shown is that the number of images used does not influence the final result. The reason that we choose the number of views being 2 is that we want to support any number of input views. For example, if we choose the number of views being 8, we should at least provide 8 views so that the model can not support the number of views smaller than 8. And we tried to change

Table 9. Comparison with MVSpLat on the GSO dataset in the 4-view input setting.

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	Inference time	Rendering time
MVSpLat	23.06	0.90	0.13	0.112	0.0090
MVSpLat (masked)	24.10	0.91	0.12	0.112	0.0045
Ours	26.30	0.93	0.08	0.694	0.0019

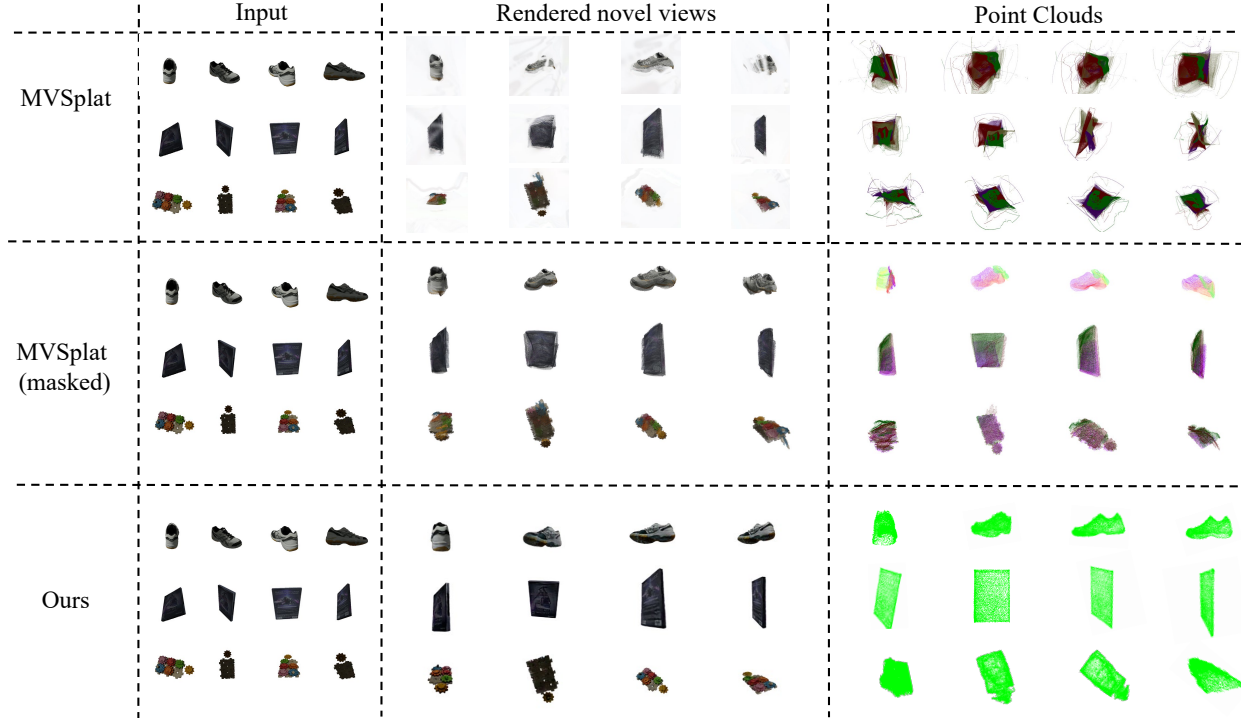


Figure 18. Visualization for MVSpLat and our method

the input views but the number of input views keeping 2 unchanged, the variance of PSNR for 10 different experiments is within 0.185.

Table 10. Ablation study results of different view and different number of views for the initialization (with 4 views in the refinement stage)

Number of views	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
1	30.2312	0.9608	0.0413
2	30.4245	0.9614	0.0422
3	30.3442	0.9618	0.0419
4	30.4521	0.9620	0.0412

Convergence for different regression target Upon investigation, we observe that prior techniques frequently predict depth rather than the centers of Gaussians. In our exploration, we conduct experiments focusing on regressing the

centers of 3D Gaussians while keeping other aspects constant. Through this analysis, we discover that regressing the positions of 3D Gaussians can introduce convergence obstacles. Table Tab. 11 illustrates the outcomes of these experiments on the Objaverse validation dataset after 100K steps.

More ablation studies Here we gives more ablation study mainly for hyperparameter selection. Due to computational costs, ablation models are trained at 100k iteration and test on Objaverse validation dataset.

Hyperparameter selection In Tab. 12, we opted for 4 decoder layers over 6, as the latter offers marginal improvement but demands significantly more computational resources. Our findings indicate that fine-tuning yields the better results.

Table 11. Ablation study on parameter selection.

Regression target	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Depth	24.3792	0.9012	0.1014
3D Gaussian centers (random initialize in visual cone)	19.2551	0.8343	0.1876
With initialization	25.5338	0.9126	0.0833

Table 12. Ablation study on parameter selection.

Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
2 decoder layers	24.5229	0.9195	0.1021
6 decoder layers	26.2442	0.9352	0.0778
Freeze encoder	25.3211	0.9264	0.1003
Default model	26.2313	0.9351	0.0788

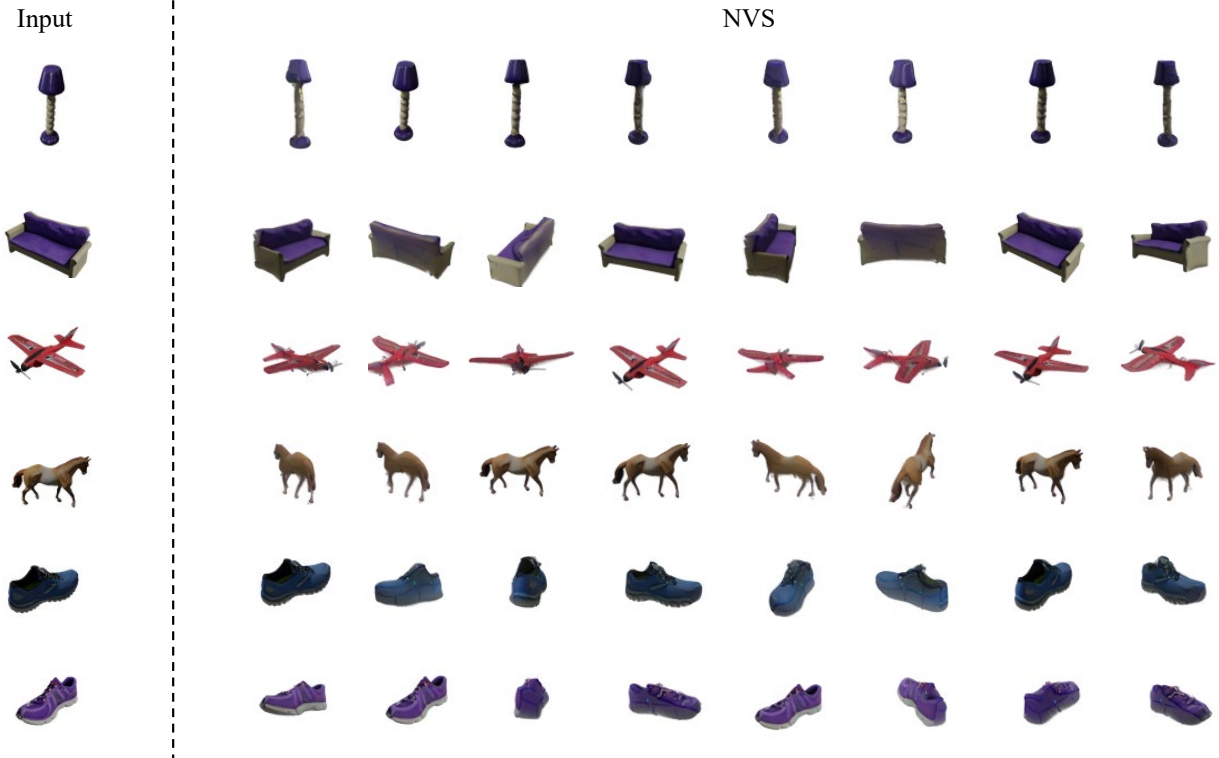


Figure 19. Single view 360 rendering visualization on GSO dataset

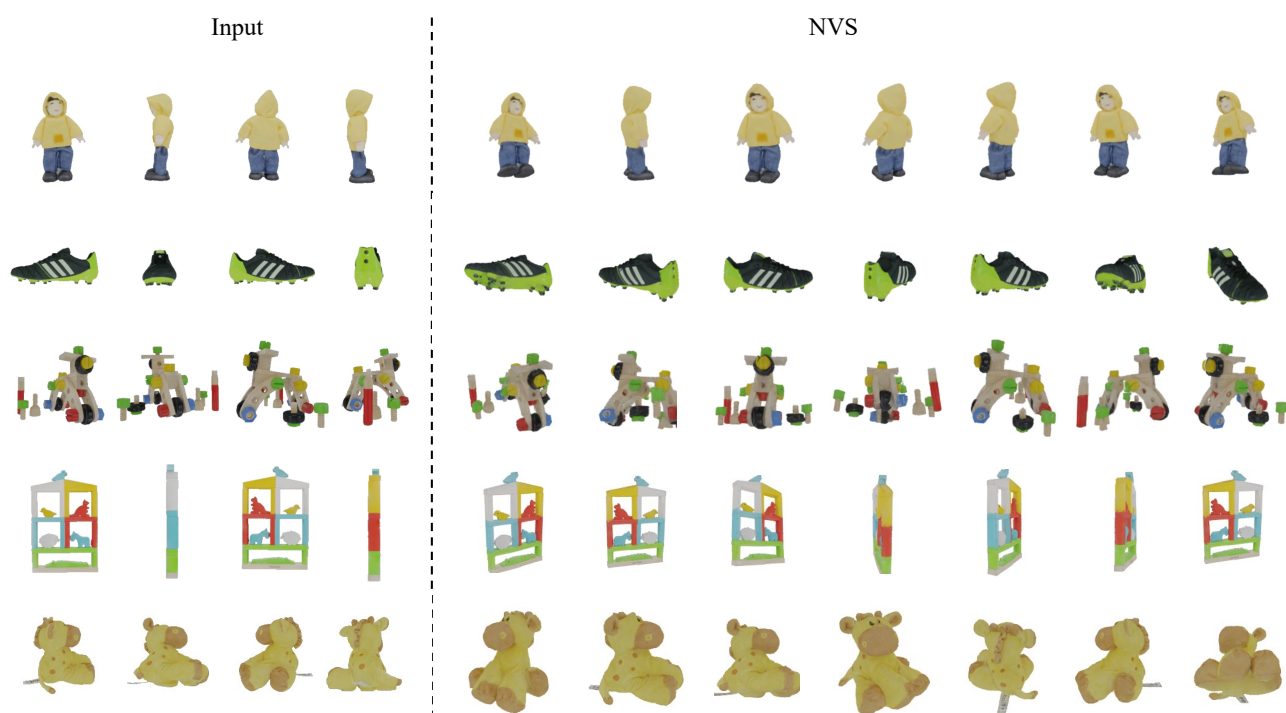


Figure 20. Visualization for our method with resolution 512