

Supplementary Materials of “Sparse Point Cloud Patches Rendering via Splitting 2D Gaussians”

Changfeng Ma¹, Ran Bi¹, Jie Guo¹, Chongjun Wang¹, Yanwen Guo^{12*}

¹Nanjing University, Nanjing, China ²School of Software, North University of China

{changfengma, 211250233}@smail.nju.edu.cn

{guojie, chjwang, ywguo}@nju.edu.cn

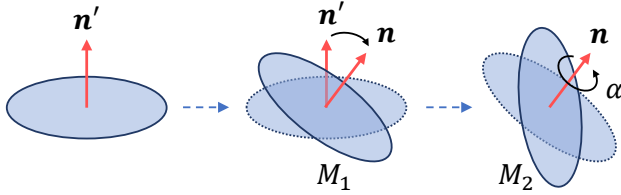


Figure 1. The illustration of our conversion process from the normal vector \mathbf{n} and angle α to the quaternion \mathbf{q} .

1. Implement Details

1.1. Quaternion Representation

In this paper, we utilize the normal vector \mathbf{n} and the rotation angle α about this normal to describe the orientation of Gaussians, rather than employing quaternions as used by 3DGS[8] and 2DGS[6]. This representation approach simplifies the initialization and prediction of Gaussian rotations in our method. To facilitate the splatting rendering proposed by 2DGS, we subsequently convert the normal vector \mathbf{n} and the rotation angle α into the quaternion \mathbf{q} . As depicted in Figure 1, the conversion process consists of two steps. Firstly, we determine the rotation matrix M_1 from the initial normal $\mathbf{n}' = [0, 0, 1]$ to the predicted normal \mathbf{n} . Subsequently, we compute the rotation matrix M_2 that represents the rotation around the normal vector \mathbf{n} by an angle α . These two processes can be accomplished using Rodrigues’ rotation formula. Finally, the quaternion \mathbf{q} is converted from the rotation matrix $M = M_1 M_2$.

1.2. Normalization and De-normalization

The normalization and de-normalization processes are as follows in the pseudocode below.

```
1 def normalize(points):
2     maxp = max(points, axis=0)
3     minp = min(points, axis=0)
4     c = (maxp + minp) / 2
5     s = max(maxp - c)
```

```
6     points = (points - c) / s
7     return points, c, s
8 def denormalize(X, S, c, s):
9     X = X * s + c
10    S = S * s
11    return X, S
```

1.3. 2D Gaussian Prediction Module

Encoder. We employ the basic PointMLP[9] as our encoder E to extract features F_l from points. The input channel number of E is $11 = 3 + 3 + 3 + 2$ (positions X^i (3), colors C^i (3), normals N^i (3) and initialized scale S^i (2)). The output channel number that is the channel of the features is 640, where each point contains one feature.

Splitting Decoder. We utilize a weight-shared Multi-Layer Perceptron (MLP) [2] to complement our splitting decoder. Each parameter of the 2D Gaussian is predicted by a splitting decoder. The input channel numbers of all the splitting decoders are $640 + 11$ (feature F_l (640), positions X^i (3), colors C^i (3), normals N^i (3) and initialized scale S^i (2)). The hidden layers of all the decoders are 512, 512, 512, 256 and 128. The output channels of the decoder are $K \times c$, where K is the number of splits and c varies depending on the specific decoder. The values of c for the decoders $D_x, D_s, D_c, D_n, D_\alpha$ and D_o are 3, 2, 27, 3, 1 and 1, respectively. The shape of the output from the decoder is $[N, K \times c]$. Following the reshape operation, the shape of the output becomes $[N \times K, c]$, thereby achieving the splitting of the predicted Gaussian. Here is the pseudocode for the splitting decoder, using D_x as an example.

```
1 def D_x(F_l, X_i, C_i, N_i, S_i):
2     shift_x = MLP(F_l, X_i, C_i, N_i, S_i)
3     # [N, K*3]
4     shift_x = shift_x.reshape([N*K, 3])
5     # [N*K, 3]
6     X_p = X_i.reshape([N, 1, 3]).repeat([1,
7     K, 1]).reshape([N*K, 3])
7     # [N*K, 3]
```

```

8   X_p = X_p + shift_x
9   return X_p

```

1.4. Entire-Patch Architecture

As we mentioned in the paper, we initially train \mathcal{N}_e using complete point cloud \mathcal{P}_e and corresponding complete images. Then, we freeze \mathcal{N}_e in order to train \mathcal{N}_p . The training process is illustrated in the pseudocode below.

```

1  N_e.requires_grad = False
2  optimizer = Optimizer(N_p, lr)
3  for step in range(step_N):
4      P_e, I_gt = Training_Data[step]
5      G_e = N_e(P_e)
6      P_p, mask = get_random_patch(P_e)
7      G_p = N_p(P_p)
8      G = concatenate(G_e[~mask], G_p[mask])
9      I_pred = splatting_render(G)
10     loss = L(I_pred, I_gt)
11     optimizer.zero_grad()
12     loss.backward()
13     optimizer.step()

```

The pseudocode for sampling the entire point cloud \mathcal{P}_e into multiple patches, as utilized in the inference of \mathcal{N}_p , is provided below.

```

1  def get_patches(P_e):
2      P_rest = P_e
3      P_p = []
4      while True:
5          center = random_select(P_rest)
6          mask = KNN(P_e, center, patch_N)
7          p = P_e[mask]
8          P_p.append(p)
9          P_rest = P_rest[~mask]
10         if P_rest is empty:
11             break

```

2. More Experiments and Results

2.1. Evaluation on Different Point Number

Figure 2 presents the rendering images of our method, TriVol [5] and PFGS [11] on different point numbers for qualitative comparison. Here, the methods are trained using Car category with 2K, 10K, 20K and 40K points. Our rendering results maintain clear details on sparse point clouds. The taillights predicted by our method exhibit two distinct lights even when the input point cloud contains only 2K points. Likewise, the wheels predicted by our method are clearer compared to those predicted by other methods. This comparison illustrates that our method adeptly handles sparse point clouds with the aid of splitting decoders.

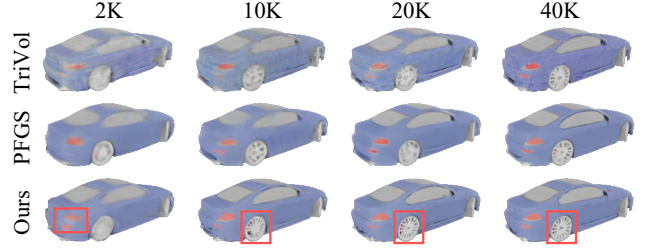


Figure 2. The rendering results of different methods trained on the Car category with 2K, 10K, 20K and 40K points.

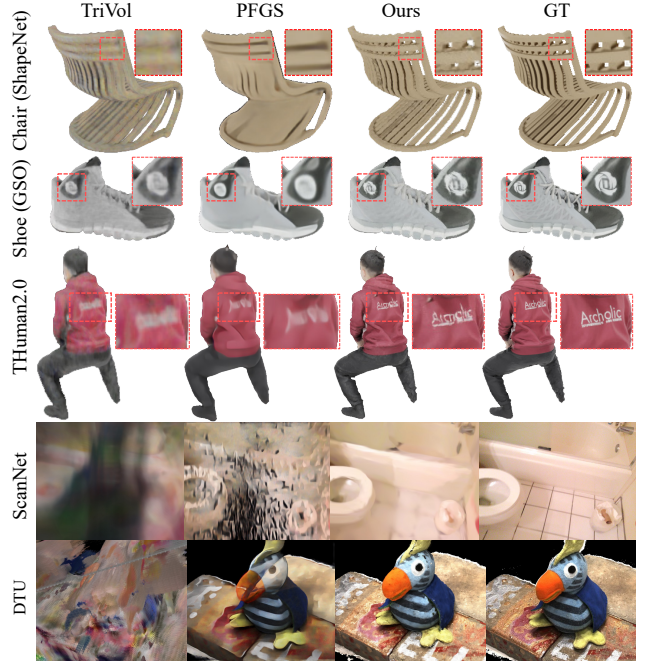


Figure 3. The evaluation results of our methods and previous methods on different datasets, where all methods are trained on the Car category with 20K points.

2.2. Evaluation of Generalization Capability

For a qualitative comparison, we train our method, TriVol, and PFGS on the car category using 20K points, and subsequently evaluate their performance across different categories. Figure 3 depicts the rendering outcomes of the methods on additional categories. These additional categories encompass object-level categories such as chair (ShapeNet [1]), shoe (ShapeNet), and human (THuman2.0 [12]), as well as scene-level datasets including ScanNet [3] and DTU [7]. The rendered results on objects of our method exhibit intricate details, such as the lattice of the chair, the pattern of the shoe, and the text on the clothing. Conversely, other methods only predict blurred results. Additionally, our method is also applicable to scene-level data, whereas other methods fail to produce accurate results. Ta-

Table 1. The evaluation of our method on different datasets including scenes, objects and human bodies, where our method is trained on the Car category with 20K point number.

Method	Point Number	ScanNet[3]			Car (ShapeNet[1])			Chair (ShapeNet[1])			Shoe (GSO[4])			THuman2.0[12]		
		PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Ours Car(ShapeNet) 20K points	20K	16.98	0.659	0.623	25.13	0.934	0.080	26.57	0.940	0.074	28.22	0.952	0.052	30.29	0.963	0.049
	40K	17.59	0.669	0.591	24.46	0.928	0.076	26.28	0.942	0.077	27.15	0.948	0.047	30.87	0.968	0.038
	100K	17.86	0.672	0.560	26.52	0.931	0.050	25.25	0.935	0.060	27.18	0.949	0.037	29.58	0.966	0.033

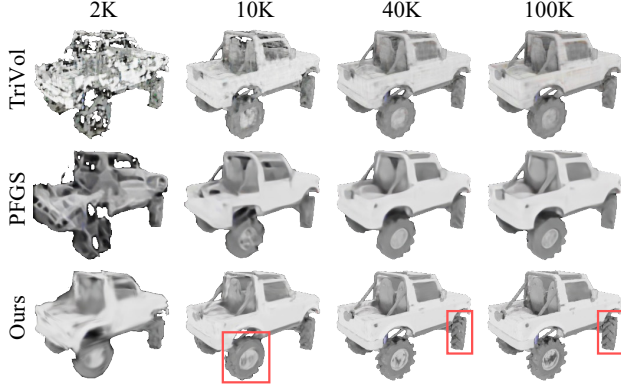


Figure 4. The evaluation results of our methods and previous methods on the Car category with 2K, 10K, 40K and 100K points, where all methods are trained on the Car category with 20K points.

ble 1 shows evaluation results of our method trained on Car category with 20K on other datasets with varying input point numbers. Although the performance of our method declined slightly, it was still comparable to the performance of previous works.

As depicted in Figure 4, we also compared the trained methods on the Car category with varying point numbers to verify the robustness of the methods across different point clouds. Here, the methods are evaluated on 2K, 10K, 40K and 100K points. The results of our method not only exhibit more details when the point number is high, but also maintain the basic details of objects when the point number is low. On the contrary, when the point number is high, the predictions of other methods are blurry and lack detail, while they struggle to generate complete images when the point number is low.

Both quantitative and qualitative evaluations demonstrate the exceptional generalization capability of our method across different categories and its robustness on different input point numbers.

2.3. Multi-View Consistency

We also render consecutive views around objects to verify the multi-view consistency of different methods. The results are presented in the video included in our supplemen-

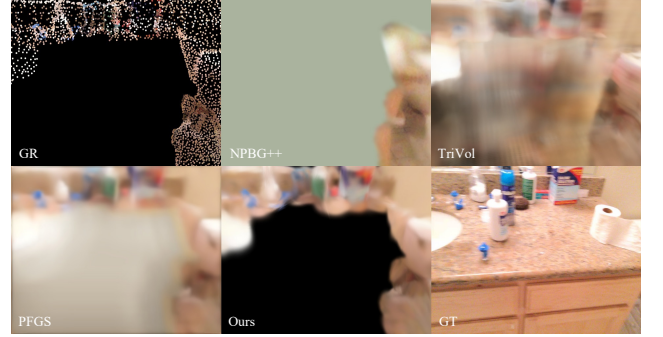


Figure 5. The illustration of our limitation.

tary materials, demonstrating that our method exhibits excellent multi-view consistency and rich detail. PFGS employs a 2-stage image refinement process to enhance the rendered images of predicted Gaussians, hence its results lack multi-view consistency and exhibit noticeable abrupt changes in the imagery.

2.4. More Comparison

Figures 6, 7, 8, and 9 present additional comparisons of our method with NPBG++[10], TriVol[5], and PFGS[11].

2.5. Limitation

A limitation of our method may arise when a portion of a point cloud is absent. As depicted in Figure 5, our method is unable to render a complete image without the direct support of the points, leaving areas unfilled and appearing as black. Such large missing areas are also beyond the predictive capabilities of other methods. Therefore, our future work aims to address this limitation by incorporating point cloud completion techniques.

References

- [1] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015. 2, 3
- [2] Kaichun Mo Leonidas J. Guibas Charles R. Qi, Hao Su. Pointnet: Deep learning on point sets for 3d classification

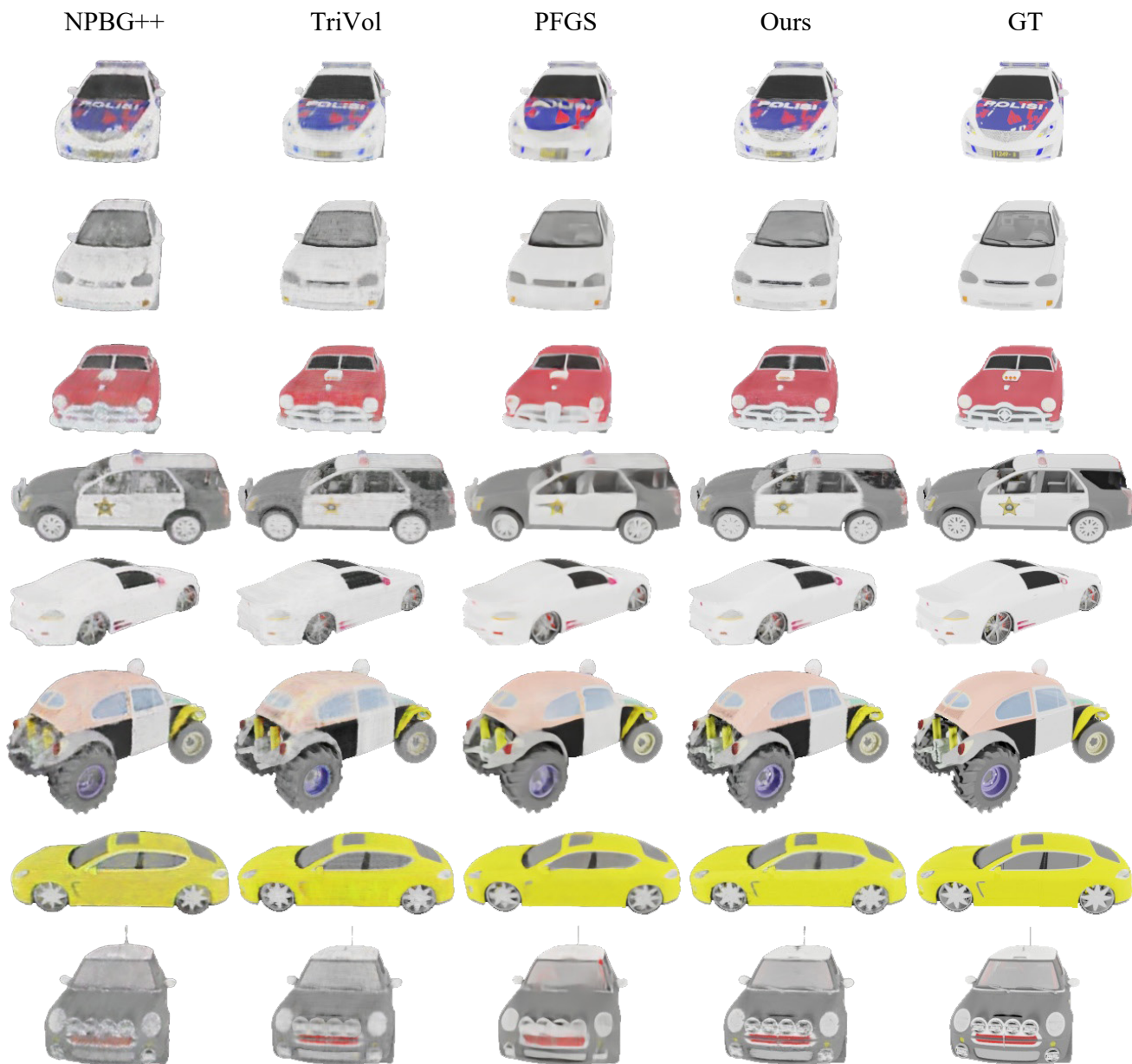


Figure 6. Additional comparison on Car category.

and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. 1

- [3] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 2, 3
- [4] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Michael Hickman, Krista Reymann, Thomas Barlow McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. *2022 International Conference on Robotics and*

Automation (ICRA), pages 2553–2560, 2022. 3

- [5] Tao Hu, Xiaogang Xu, Ruihang Chu, and Jiaya Jia. Trivol: Point cloud rendering via triple volumes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20732–20741, 2023. 2, 3
- [6] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. 1
- [7] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanaes. Large scale multi-view stereopsis evaluation. In *2014 IEEE Conference on Computer Vision and*

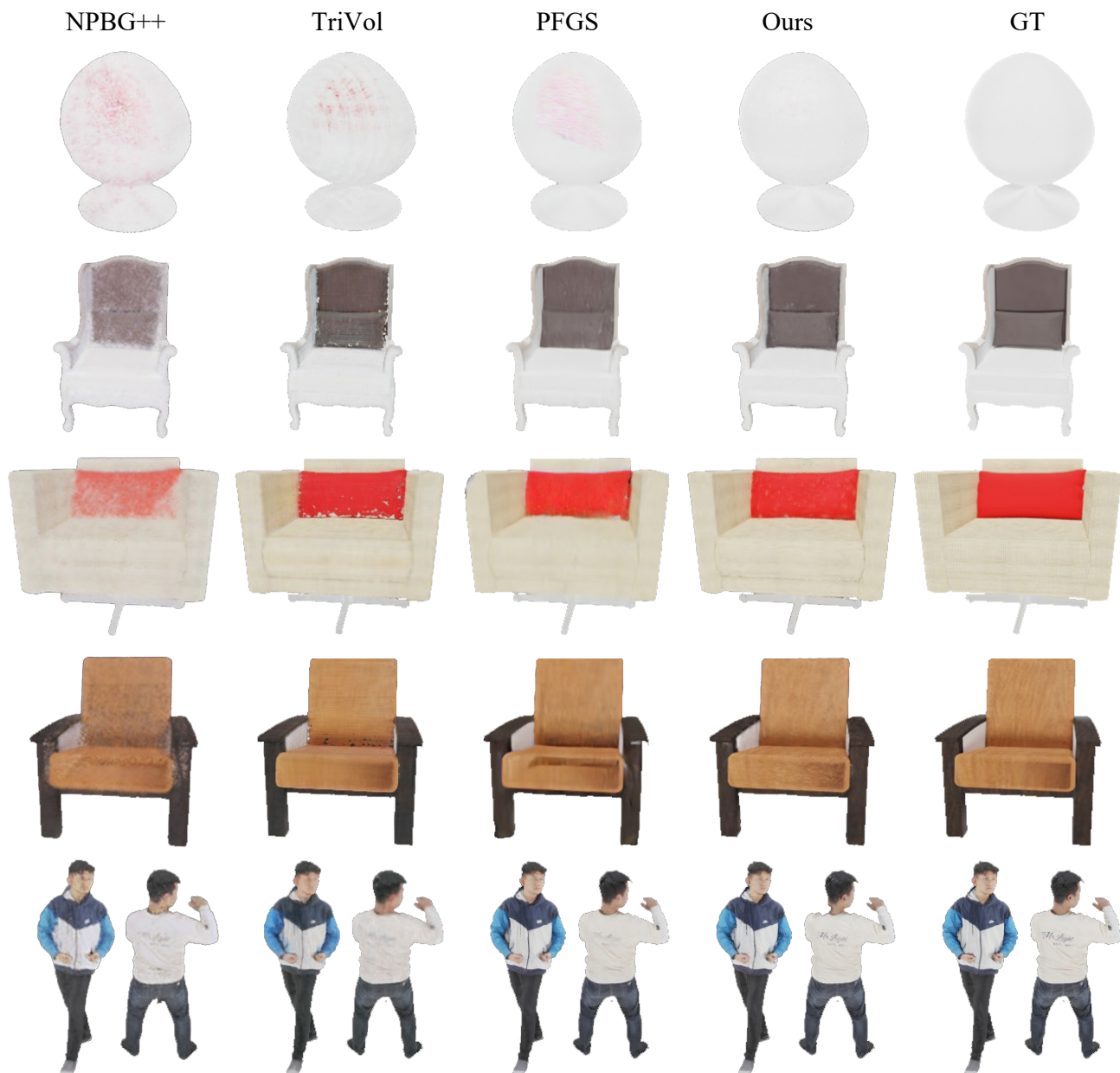


Figure 7. Additional comparison on Chair and Human categories.

Pattern Recognition, pages 406–413, 2014. [2](#)

- [8] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. [1](#)
- [9] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *ICLR*, 2022. [1](#)
- [10] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lepitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), pages 15969–15979, 2022. [3](#)

- [11] Jiaxu Wang, Ziyi Zhang, Junhao He, and Renjing Xu. Pfgs: High fidelity point cloud rendering via feature splatting. *ECCV 2024*, 2024. [2](#), [3](#)
- [12] Tao Yu, Zerong Zheng, Kaiwen Guo, Pengpeng Liu, Qionghai Dai, and Yebin Liu. Function4d: Real-time human volumetric capture from very sparse consumer rgbd sensors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR2021)*, 2021. [2](#), [3](#)



Figure 8. Additional comparison on Shoe category.

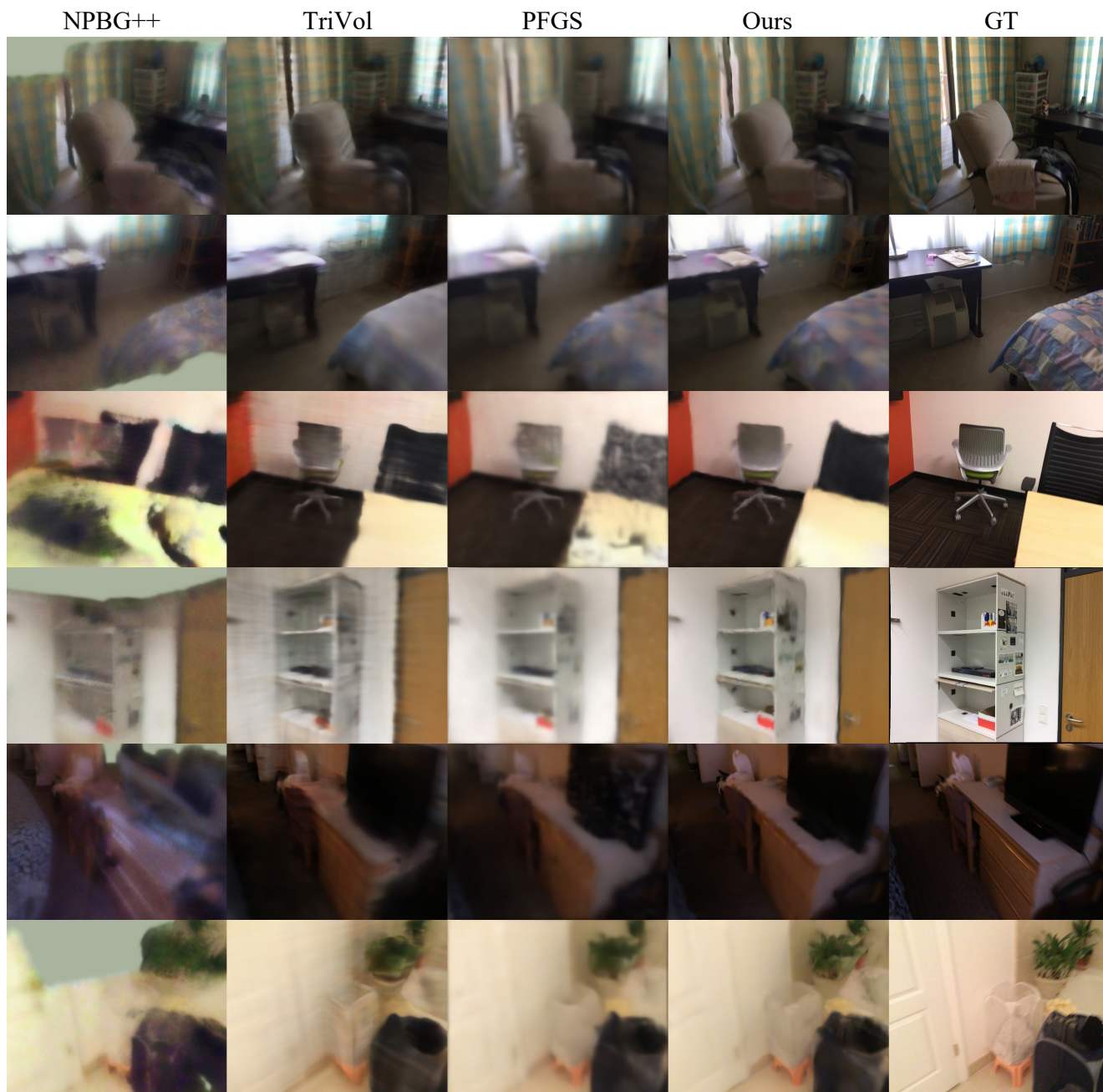


Figure 9. Additional comparison on ScanNet.