

RayletDF: Raylet Distance Fields for Generalizable 3D Surface Reconstruction from Point Clouds or Gaussians

Supplementary Material

The appendix includes:

- Details for all the datasets.
- The definitions of all the evaluation metrics.
- Implementation details for all the baselines.
- Implementation details for our model, including Raylet Feature Extractor, Raylet Distance Field Network, and the calculation of ray-gaussian intersection.
- Details for the surface normal derivation.
- Computation cost analysis
- Results on different point density
- More quantitative and qualitative results.

5.1. Datasets

5.1.1. ScanNet++

We use the official training and test splits from ScanNet++ dataset for novel view synthesis. There are 855 scenes for training and 50 scenes for test. For each scene in train set, we uniformly sample every 5th frame from the video sequence for training, and reserve every 10th frame in the test set for testing. We use the iPhone sequences with COLMAP registered poses and initial sparse points for pre-estimating 3D Gaussians. We do not use additional control over the number of Gaussian points during the training process.

5.1.2. ScanNet

We use the official training and test splits of ScanNetV2 dataset, with 1201 training scenes and 100 test scenes. Similar to ScanNet++, we also load every 5th frame from the trajectories for training, and then reserve every 10th frame in test set for evaluation.

5.1.3. ARKitScenes

We use the official split of “3dod” parts from ARKitScenes including 5047 rooms in total. After removing several rooms which cannot be pre-estimated by 3DGS, the training set contains 4496 rooms while the test dataset has 549 rooms. For training views, we sample every 5th image from the train set for training, and then reserve every 10th image in the test set for testing.

5.1.4. MultiScan

We use the whole MultiScan dataset with 207 scenes for testing, after removing a few scenes which cannot be pre-estimated by 3DGS. We sample every 10th image for test.

5.2. Ray-surface Distance Metrics

We report a series of ray distance metrics including Absolute Distance Error (ADE), RMSE, Absolute Relative Distance (Abs-Rel), Squared Relative Distance (Sq Rel) and Threshold accuracy ($\delta < t$) to measure the error between the estimated i^{th} ray-surface distance t_i^{pred} and the ground truth distance t_i^{gt} . These metrics are calculated over N samples and defined as following:

Absolute Distance Error (ADE):

$$\frac{1}{N} \sum_{i=1}^N |t_i^{pred} - t_i^{gt}|$$

RMSE:

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (t_i^{pred} - t_i^{gt})^2}$$

Absolute Relative Distance (Abs Rel):

$$\frac{1}{N} \sum_{i=1}^N \frac{|t_i^{pred} - t_i^{gt}|}{t_i^{gt}}$$

Squared Relative Distance (Sq Rel):

$$\frac{1}{N} \sum_{i=1}^N \frac{(t_i^{pred} - t_i^{gt})^2}{t_i^{gt}}$$

Threshold accuracy, δ :

$$\frac{1}{N} \sum_{i=1}^N [\max(\frac{t_i^{pred}}{t_i^{gt}}, \frac{t_i^{gt}}{t_i^{pred}}) < \delta]$$

5.3. Evaluation Metrics on Meshes

The evaluation metrics for comparing predicted meshes \hat{P} and ground truth meshes P are defined as follows:

Accuracy:

$$\frac{1}{|\hat{P}|} \sum_{\hat{\mathbf{p}} \in \hat{P}} (\min_{\mathbf{p} \in P} \|\mathbf{p} - \hat{\mathbf{p}}\|)$$

Completion

$$\frac{1}{|P|} \sum_{\mathbf{p} \in P} (\min_{\hat{\mathbf{p}} \in \hat{P}} \|\mathbf{p} - \hat{\mathbf{p}}\|)$$

Chamber-L1

$$\frac{\text{Accuracy} + \text{Completion}}{2}$$

Normal Accuracy

$$\frac{1}{|\hat{P}|} \sum_{\hat{\mathbf{p}} \in \hat{P}} (\mathbf{n}_{\mathbf{p}}^T \mathbf{n}_{\hat{\mathbf{p}}}) \text{ s.t. } \mathbf{p} = \underset{\mathbf{p} \in P}{\operatorname{argmin}} \|\mathbf{p} - \hat{\mathbf{p}}\|$$

Normal Completion

$$\frac{1}{|P|} \sum_{\mathbf{p} \in P} (\mathbf{n}_{\mathbf{p}}^T \mathbf{n}_{\hat{\mathbf{p}}}) \text{ s.t. } \hat{\mathbf{p}} = \underset{\hat{\mathbf{p}} \in \hat{P}}{\operatorname{argmin}} \|\mathbf{p} - \hat{\mathbf{p}}\|$$

Normal Consistency

$$\frac{\text{Normal Accuracy} + \text{Normal Completion}}{2}$$

Precision

$$\frac{1}{|P|} \sum_{\hat{\mathbf{p}} \in \hat{P}} (\min_{\mathbf{p} \in P} \|\mathbf{p} - \hat{\mathbf{p}}\|) < 5cm$$

Recall

$$\frac{1}{|P|} \sum_{\mathbf{p} \in P} (\min_{\hat{\mathbf{p}} \in \hat{P}} \|\mathbf{p} - \hat{\mathbf{p}}\|) < 5cm$$

F-score

$$\frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5.4. Baselines

3DGS: We use the official implementation of Gaussian splatting [30] and increase densification gradient threshold to 0.0005 on datasets used in this paper to remove floaters which reduce the render performance significantly. Each scene Gaussian model of ARKitScenes dataset is trained for 7K iterations to obtain better render performance while scenes of other datasets are trained for 30K iterations. Additionally, we modify the original CUDA kernel to support depth rendering and convert depth predictions to ray-surface distances according to camera intrinsics.

GOF: We use the official GOF [74] source code. The densification gradient threshold and training iterations are the same with 3DGS models mentioned above.

PGSR We use the official PGSR [9] source code for comparison. Similarly to the GOF and 3DGS training strategy, we also increase the densification gradient threshold to 0.0005. Gaussian models of ARKitScenes dataset are optimized for 20K iterations until its loss converges.

DepthanythingV2: We use the official depth estimation checkpoint with ViT-L backbone in the original paper to estimate depth d^{pred} for RGB images rendered by 3DGS. Then we align the scale of estimation results with ground truth depth d^{gt} by forcing their median and variance to be the same:

$$m^{pred} = \operatorname{median}(d^{pred})$$
$$s^{pred} = \frac{1}{N} \sum_{i=1}^N |d_i^{pred} - m^{pred}|$$

$$d_{norm}^{pred} = \frac{d^{pred} - m^{pred}}{s^{pred}}$$

$$m^{gt} = \operatorname{median}(d^{gt})$$

$$s^{gt} = \frac{1}{N} \sum_{i=1}^N |d_i^{gt} - m^{gt}|$$

$$d_{align}^{pred} = d_{norm}^{pred} * s^{gt} + m^{gt}$$

Depth-Pro: We also use the official checkpoint to estimate depth values for rendered RGB images. Since the outputs of Depth-Pro are depth values in meter, we directly align the median and variance of estimation to ground truth depths.

MVSGaussians: We use the official MVSGaussians source code. For a fair comparison, we also add ground truth depth signals to supervise rendering results in training. During inference stage, in favor of the baseline, we use MVSGaussian model to get better initialization and then optimize the Gaussian per scene for 7000 iterations.

Pointersect: We use the official source code to re-train the Pointersect model. For a fair comparison, the same RGB-D images are used as supervision signals as ours.

RayDF: We implement a ray-surface distance field conditioned on the same local geometry features as ours to compare generalization ability fairly in this paper.

PFGS: We use the official implementation code. We re-train the network using the same depth supervision signals as ours.

Notably, we use ground truth depth values to align the scale of estimated depths from **DepthanythingV2** and **Depth-Pro**, which is strongly in favor of these baselines. The ground truth depth is not available in real practice. Thus, we compare the performances of these two models aligned to the gaussian-rendered depth values, as shown in Table 8.

5.5. Details of Raylet Feature Extractor

We simply employ the SparseConv architecture as our Raylet Feature extractor. Particularly, we use the existing U-Net implementation of SPconv PyTorch package <https://github.com/traveller59/spconv>. As illustrated in Figure 7, the encoder and decoder have 8 layers respectively, which produce 32-dimensional features. Block means a sequential application of SparseConvolution in encoder or SparseInverseConvolution in decoder, Group-Norm and ReLU activation.

5.6. Details of Raylet Distance Field Network

As shown in 8, we first use an MLP layer with ReLU to map the input feature embeddings to 256 dimensions, and then we use 8 dense layers of 256 neurons with ReLU activation network. Lastly, we use a linear layer with no activation function to output the raylet distance and confidence score.

Table 8. Quantitative results of depth estimation models aligned with different method.

	<i>test on</i> \rightarrow ARKitScenes					<i>test on</i> \rightarrow ScanNet/ScanNet++					<i>test on</i> \rightarrow MultiScan				
	ADE \downarrow	RMSE \downarrow	Abs-Rel \downarrow	Sq-Rel \downarrow	$\delta \uparrow$	ADE \downarrow	RMSE \downarrow	Abs-Rel \downarrow	Sq-Rel \downarrow	$\delta \uparrow$	ADE \downarrow	RMS \downarrow E	Abs-Rel \downarrow	Sq-Rel \downarrow	$\delta \uparrow$
DepthAnythingV2 [†] [69]	0.206	0.294	0.144	0.093	0.860	0.168	0.248	0.118	0.066	0.887	0.228	0.303	0.205	0.117	0.775
Depth-Pro [†] [5]	0.294	0.403	0.202	0.242	0.750	0.220	0.319	0.120	0.106	0.861	0.280	0.363	0.234	0.157	0.659
DepthAnythingV2 [69]	0.325	0.419	0.215	0.135	0.637	1.022	1.267	0.580	1.053	0.303	0.460	0.538	0.347	0.237	0.352
Depth-Pro [5]	0.379	0.495	0.246	0.225	0.594	0.374	0.478	0.187	0.163	0.712	0.483	0.571	0.361	0.258	0.333

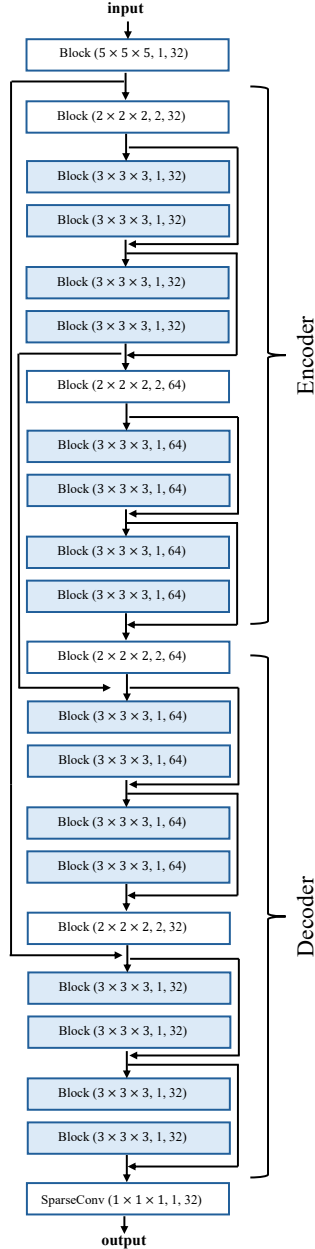


Figure 7. Details of U-Net feature extractor. Block means a sequential application of SparseConvolution in encoder or SparseInverseConvolution in decoder, GroupNorm and ReLU activation.

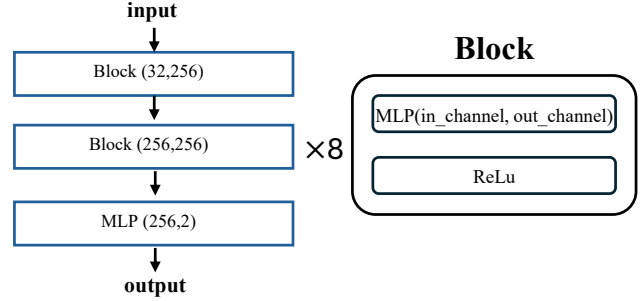


Figure 8. Details Raylet Distance Field Network.

5.7. Calculation of Intersection

We provide the details of calculating intersections using Gaussians as an example. Given a ray $x = r_o + t * r_d$ and a Gaussian $G(x) = \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu))$, we can compute the Gaussian value on the ray as:

$$G^{1D}(t) = \exp(-\frac{1}{2}(o+t*r_d-\mu)^T \Sigma^{-1}(o+t*r_d-\mu)) \quad (6)$$

Follow [31], the intersection is defined as the point that maximizes the 1D Gaussian $G^{1D}(t)$. The distance t_i can be calculated in closed-form by making the derivative of function $G^{1D}(t)$ equal to zero as:

$$t_i = \frac{r_d \Sigma^{-1}(\mu - r_o)}{r_d^T \Sigma^{-1} r_d} \quad (7)$$

Lastly, bringing t_i back to ray function, we can obtain the intersection coordinates $x = r_o + t_i * r_d$.

To utilize the tile-based splitting strategy in 3DGS, we have customized the CUDA kernels to select the top K intersection points with the highest alpha blending weight.

5.8. Derivation of Surface Normal

In this section, we derive the formula of surface normal from our raylet distance field using a single raylet sample as an example. Given a ray from camera origin o , we can first parameter its direction in spherical coordinate system as $(\theta, \phi, 1)$, we convert the ray direction to world coordinate as $d = (x, y, z) = (\sin \theta \cos \phi, \cos \theta, \sin \theta \sin \phi)$. Then we use the ray direction and origin to compute intersection between the Gaussians or points analytically and obtain the

ray distance t_i from camera origin to intersection point. We form a raylet \mathbf{l} at the sample point position and use our network to predict the raylet distance. The complete ray distance from camera to surface is $D = F(\mathbf{l}) + t_i$. Due to the differentiability of network and intersection calculations, we can calculate the derivative of surface point with respect to θ, ϕ and the unit normal can be computed similarly as in [36]. Specifically, the surface point coordinate can be formed as:

$$\Phi(\theta, \phi) = (D \sin \theta \cos \phi, D \cos \theta, D \sin \theta \sin \phi) \quad (8)$$

From Eq. (8), we have:

$$\frac{\partial \Phi}{\partial \phi} = \begin{bmatrix} \left(\frac{\partial D}{\partial \phi} \cos \phi - D \sin \phi \right) \sin \theta \\ \frac{\partial D}{\partial \phi} \cos \theta \\ \left(\frac{\partial D}{\partial \phi} \sin \phi + D \cos \phi \right) \sin \theta \end{bmatrix} \quad (9)$$

$$\frac{\partial \Phi}{\partial \theta} = \begin{bmatrix} \left(\frac{\partial D}{\partial \theta} \sin \theta + D \cos \theta \right) \cos \phi \\ \frac{\partial D}{\partial \theta} \cos \theta - D \sin \theta \\ \left(\frac{\partial D}{\partial \theta} \sin \theta + D \cos \theta \right) \sin \phi \end{bmatrix} \quad (10)$$

Finally, the formula for a unit normal vector is :

$$\mathbf{n} = \frac{\frac{\partial \Phi}{\partial \phi} \times \frac{\partial \Phi}{\partial \theta}}{\left\| \frac{\partial \Phi}{\partial \phi} \times \frac{\partial \Phi}{\partial \theta} \right\|}. \quad (11)$$

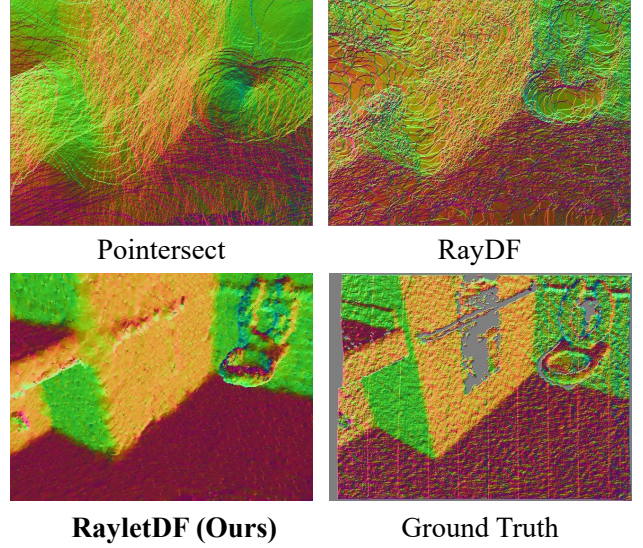
5.9. Quantitative and Qualitative Results on Normal Derivation

We report derived normals on sparse point cloud of ScanNet/++ test split in Table 9 and Figure 9. Our method demonstrates superior performance compared to Pointersect and RayDF under this setting. Notably, our framework does not incorporate normal supervision during training, for a fair comparison normals for Pointersect are computed post-hoc from predicted depth maps. In contrast to RayletDF, RayDF lacks explicit local geometry-aware features in its architecture, preventing analytical normal estimation from the network. Consequently, we also derive RayDF’s normals via depth map post-processing for fair comparison. The results suggest that RayletDF’s ray-based feature encoding and geometric reasoning enable more accurate normal estimation without direct supervision, outperforming depth-derived normals from competing methods.

Table 9. Comparison of derived normals. The Mean Angle Error ($^\circ$), RMSE, mean of L1 error and median of L1 error are reported.

	MAE↓	RMSE↓	Mean L1↓	Median L1↓
Pointersect [8]	62.37	0.60	0.46	0.37
RayDF [36]	62.81	0.61	0.47	0.38
RayletDF (Ours)	48.86	0.50	0.37	0.28

Figure 9. Qualitative results of derived normals.



5.10. More Results of Raylet Sampling in Testing

We train our model on ScanNet/ScanNet++ given $T = \{1, 10, 20\}$ during training, and then test the three models on ScanNet/ScanNet++ with $T = \{1, 5, 10, 20\}$ samples in testing. Results are provided in Tables 10&11&12.

Table 10. Qualitative results for different number of raylet samples per ray when evaluating on the test split of ScanNet/ScanNet++

test on \rightarrow ScanNet/ScanNet++					
(train samples: $T=1$)	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	$\delta \uparrow$
test samples: $T = 1$	0.174	0.326	0.085	0.144	0.898
test samples: $T = 5$	0.168	0.292	0.082	0.086	0.908
test samples: $T = 10$	0.174	0.306	0.087	0.085	0.904
test samples: $T = 20$	0.172	0.301	0.0852	0.0851	0.905

Table 11. Qualitative results for different number of raylet samples per ray when evaluating on the test split of ScanNet/ScanNet++

test on \rightarrow ScanNet/ScanNet++					
(train samples: $T=10$)	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	$\delta \uparrow$
test samples: $T = 1$	0.190	0.357	0.093	0.184	0.884
test samples: $T = 5$	0.146	0.278	0.072	0.076	0.920
test samples: $T = 10$	0.141	0.274	0.070	0.066	0.925
test samples: $T = 20$	0.142	0.275	0.071	0.069	0.924

5.11. Computation Cost Analysis

We report the total train time, memory consumption and rendering speed in test for coordinate-based methods, 3DGS and ours on ScanNet/++ test split in Table 13. Voxel grid KNN https://github.com/janericlenssen/torch_knnquery/ is employed

Table 12. Qualitative results for different number of raylet samples per ray when evaluating on the test split of ScanNet/ScanNet++

	<i>test on</i> → ScanNet/ScanNet++				
<i>(train samples: T=20)</i>	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	δ ↑
test samples: $T = 1$	0.194	0.368	0.096	0.329	0.881
test samples: $T = 5$	0.146	0.277	0.072	0.076	0.920
test samples: $T = 10$	0.139	0.267	0.069	0.068	0.926
test samples: $T = 20$	0.141	0.271	0.070	0.070	0.924

to accelerate our algorithm. Since our method only needs 5 raylets sampled on each ray and each raylet only has 5 neighboring points sampled, for a fair comparison, we also only sample 25 points for coordinate-based methods in addition to their original dense samples. For clarification, our training time includes 30 hours to train RayletDF on ScanNet/++ train split, and 37.5 hours of pre-estimating 3D Gaussians on ScanNet/++ test split. We can see that: 1) our method has a clear advantage over coordinate-based methods in rendering speed; 2) though being slower than 3DGS, our method achieves much higher accuracy in surface reconstruction.

5.12. Results of Pre-trained Models

Results of pre-trained Pointersect on sparse point clouds are given in Table 14. The model is pre-trained on dense point clouds, thus being inferior to our trained version. RayDF is a per-scene method and we cannot use its pre-trained models.

5.13. Quantitative Results on Different Point Density

Given that Gaussian point clouds typically contain hundreds of thousands to millions of points per scene, we adopt sparse point cloud inputs in our primary experiments to rigorously evaluate the performance of RayletDF under low-density conditions. Furthermore, advancements in RGB image-based reconstruction techniques—such as Structure-from-Motion (SfM)—have made it increasingly practical to generate sparse point cloud data, aligning with our experimental design.

In this section, we test models on one million of points on ScanNet/++ test split in Table 15 to investigate the sensitivity to point density of our algorithm. Shown in the Table, our method is robust to high density points though only trained on sparse point settings. The pre-trained Pointersect model is originally trained on millions of points, we give its results for reference. As the input points already contain accurate surfaces, the performance gap between RayletDF and other methods decreases to some degree.

5.14. Ablation of Transformer Layer for Neighbor Points Aggregation

We present results of a Transformer layer to aggregate neighbor features on Gaussians ScanNet/++ test split in Table 16. Employing transformer layer does not yield performance improvements in this setting, we hypothesize that much larger dataset is needed to fully unleash the ability of Transformer.

5.15. Standard Deviation Results

We report standard deviation results on ray-based methods in Table 18 and Table 19. Shown in the Table, our method achieves the most robust performance.

5.16. Scene Scale and Point Density Information

We report the scale of scenes and distance to the nearest point of sparse point clouds used in our three group experiments in Table 17.

5.17. Additional Quantitative and Qualitative Results

More results are provided in the following table and figures. **Models trained on Gaussians of ScanNet/ScanNet++:** in Figure 13&14, we show more qualitative results of Group 1.

Models trained on Gaussians of ARKitScens: in Figure 10&11&12, we show more qualitative results of Group 2.

Table 13. The time, memory consumption, and the speed of rendering a 640×480 image given different number of points sampled.

	Total Train Time (hours)	Memory (GB)	No. of Samples per ray	Render Speed (FPS)	ADE ↓
NeRF [39]	~ 645	4.39 / 0.64	192 (dense) / 25	0.11 / 1.16	-
InstantNGP [40]	~ 27.5	14.39 / 12.94	1024 (dense) / 25	1.09 / 1.17	-
NeuS [60]	~ 469	6.20 / 1.22	160 (dense) / 25	0.06 / 0.17	-
3DGS [30]	~ 37.5	4.45	-	293	0.321
RayletDF (Ours)	~ (37.5+30)	11.63	5×5	5.35	0.145

Table 14. Results of pre-trained pointersect model.

	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	$\delta \uparrow$
ARKitScenes	0.57	1.14	0.39	1.33	0.65
ScanNet/ScanNet++	0.48	0.97	0.23	0.69	0.77
Multiscan	0.42	0.89	0.31	0.96	0.71

Table 15. Results on one million points as input point clouds.

	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	$\delta \uparrow$
Pre-trained Pointersect [8]	0.098	0.264	0.040	0.051	0.958
Pointersect [8]	0.197	0.354	0.126	0.105	0.844
RayDF [36]	0.160	0.271	0.100	0.056	0.918
RayletDF (Ours)	0.088	0.232	0.043	0.043	0.954

Table 16. Comparison between Transformer layer and MLPs.

	ADE↓	RMSE↓	Abs-Rel↓	Sq-Rel↓	$\delta \uparrow$
Transformer	0.155	0.286	0.077	0.082	0.916
MLPs	0.145	0.276	0.072	0.079	0.922

Table 17. Scene size and average distance (meter) between nearest neighbors in sparse point clouds.

Method	Mean X	Mean Y	Mean Z	Max X	Max Y	Max Z	Dist
ARKitScenes	7.74	7.65	3.24	22.23	17.98	9.15	0.08
ScanNet/++	5.87	4.94	2.61	19.72	19.04	7.68	0.07
Multiscan	6.32	2.89	6.36	16.49	5.36	19.30	0.07

Table 18. Standard deviation results on Gaussians dataset.

		<i>test on</i> → ARKitScenes		<i>test on</i> → ScanNet/ScanNet++		<i>test on</i> → MultiScan	
		ADE ↓	RMSE ↓	ADE ↓	RMSE ↓	ADE ↓	RMSE ↓
Pointersect [8]	<i>train on</i> ARKitScenes	0.286±0.179	0.397±0.272	0.366±0.184	0.259±0.381	0.266±0.135	0.311±0.176
RayDF [36]		0.183±0.113	0.303±0.211	0.175±0.124	0.320±0.214	0.326±0.183	0.425±0.233
RayletDF (Ours)		0.115±0.084	0.218±0.181	0.175±0.123	0.320±0.217	0.216±0.151	0.311±0.209
Pointersect [8]	<i>train on</i> ScanNet++	0.328±0.253	0.462±0.351	0.433±0.209	0.604±0.274	0.404±0.215	0.504±0.249
RayDF [36]		0.587±0.276	0.704±0.334	0.202±0.120	0.337±0.198	0.604±0.298	0.690±0.327
RayletDF (Ours)		0.132±0.094	0.241±0.193	0.145±0.100	0.276±0.186	0.259±0.167	0.353±0.213

Table 19. Standard deviation results on point cloud dataset.

		<i>test on</i> → ARKitScenes		<i>test on</i> → ScanNet/ScanNet++		<i>test on</i> → MultiScan	
		ADE ↓	RMSE ↓	ADE ↓	RMSE ↓	ADE ↓	RMSE ↓
Pointersect [8]	<i>train on</i> ARKitScenes	0.335±0.138	0.456±0.187	0.389±0.193	0.553±0.278	0.254±0.077	0.330±0.099
RayDF [36]		0.166±0.119	0.303±0.216	0.186±0.089	0.321±0.172	0.154±0.098	0.234±0.152
RayletDF (Ours)		0.088±0.071	0.205±0.167	0.107±0.068	0.269±0.164	0.067±0.048	0.149±0.109
Pointersect [8]	<i>train on</i> ScanNet++	0.299±0.115	0.398±0.156	0.344±0.178	0.477±0.240	0.233±0.086	0.289±0.096
RayDF [36]		0.289±0.178	0.407±0.268	0.161±0.085	0.291±0.158	0.349±0.189	0.429±0.222
RayletDF (Ours)		0.096±0.077	0.217±0.179	0.093±0.056	0.234±0.133	0.130±0.177	0.229±0.255

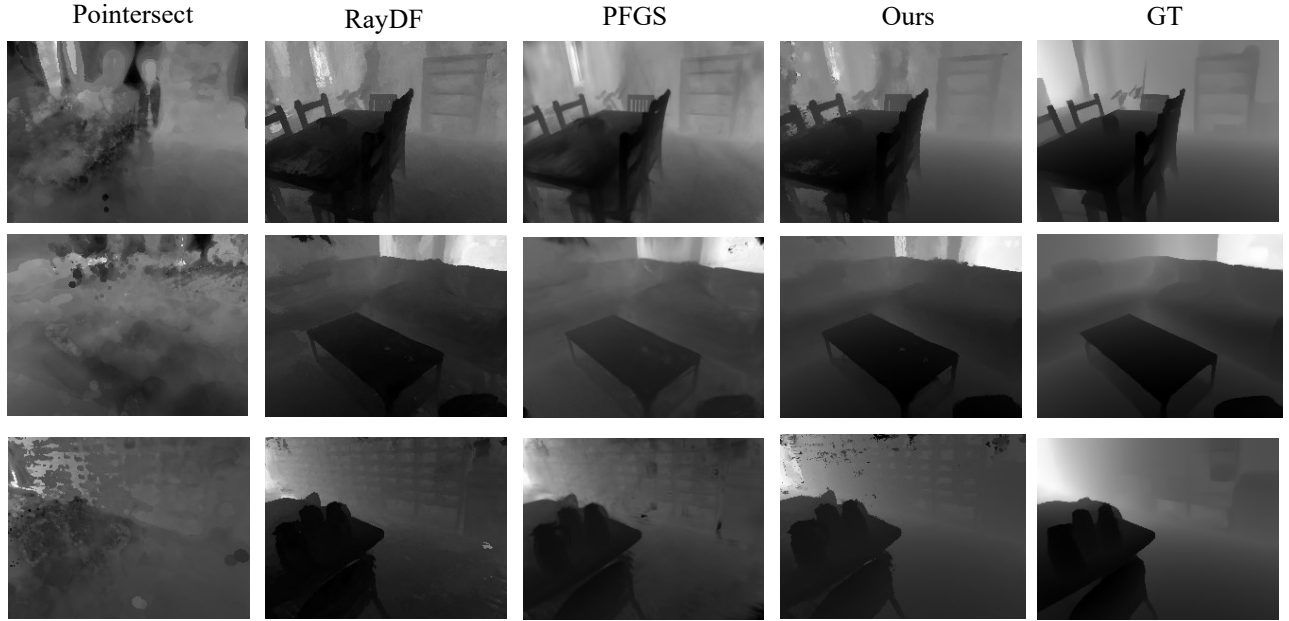


Figure 10. Qualitative results of all methods trained on Gaussians of ARKitScenes and tested on ScanNet/ScanNet++.

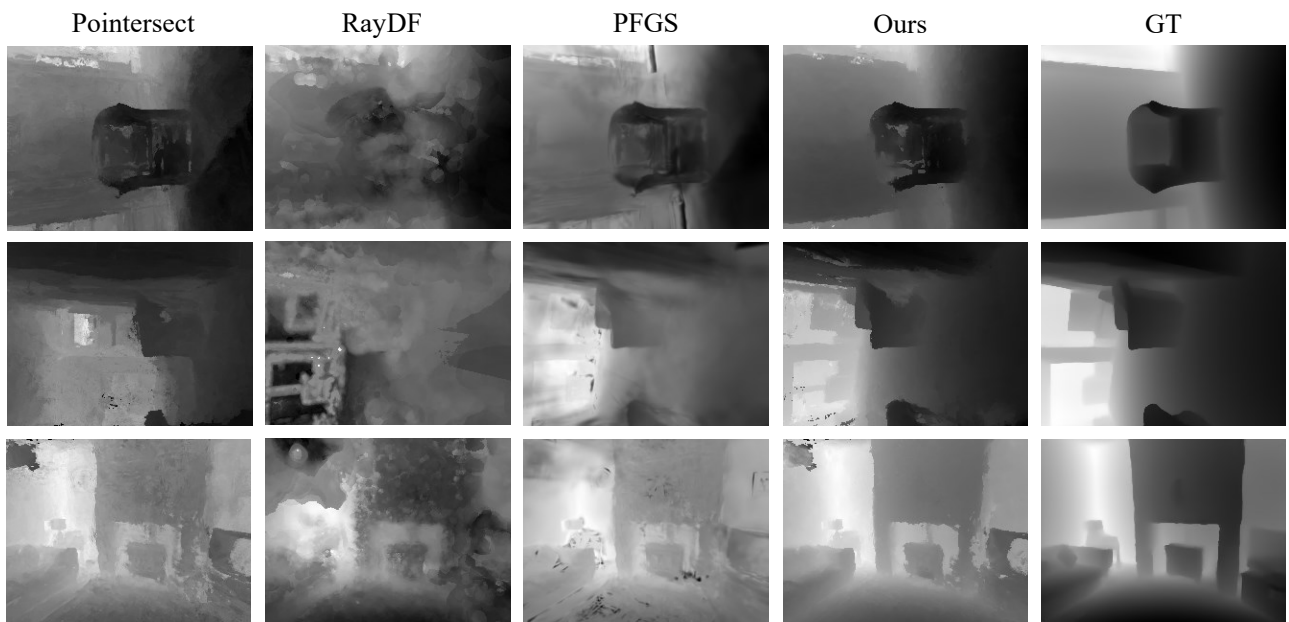


Figure 11. Qualitative results of all methods trained on Gaussians of ARKitScenes and tested on ARKitScenes.

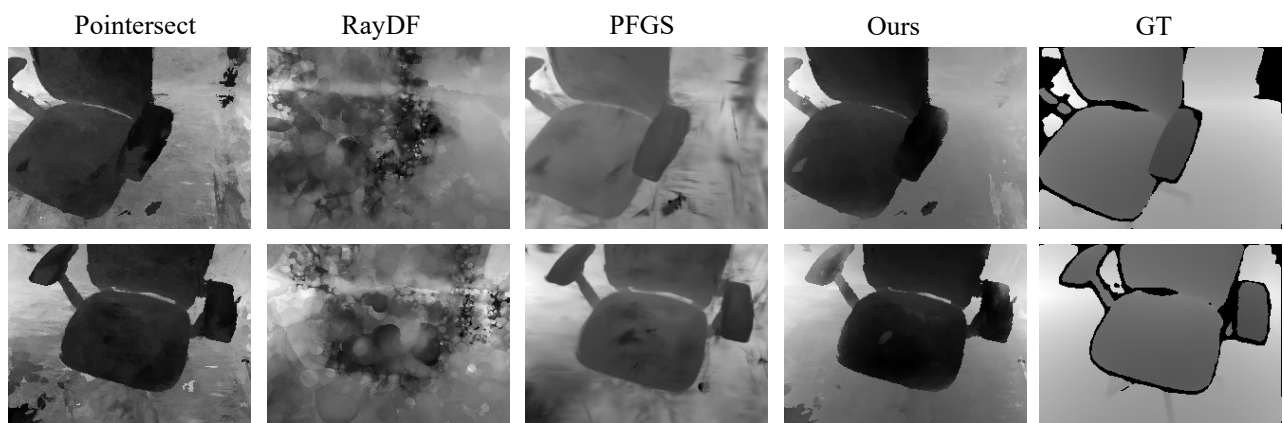


Figure 12. Qualitative results of all methods trained on Gaussians of ARKitScenes and tested on MultiScan.

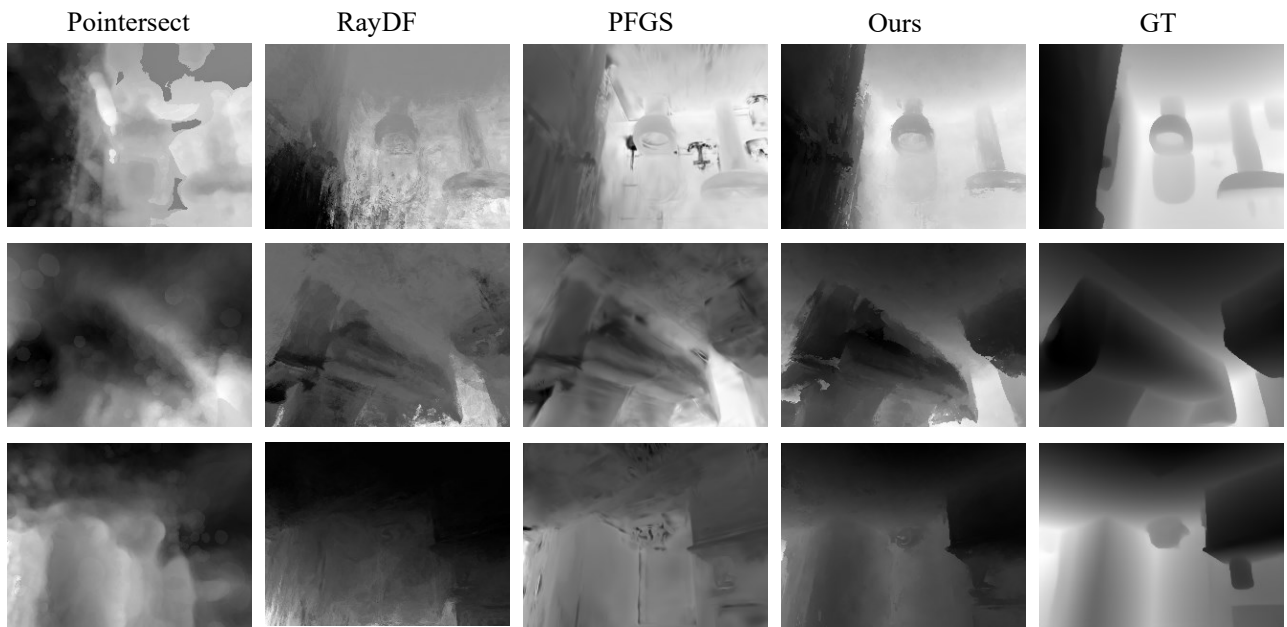


Figure 13. Qualitative results of all methods trained on Gaussians of ScanNet/ScanNet++ and tested on ARKitScenes.

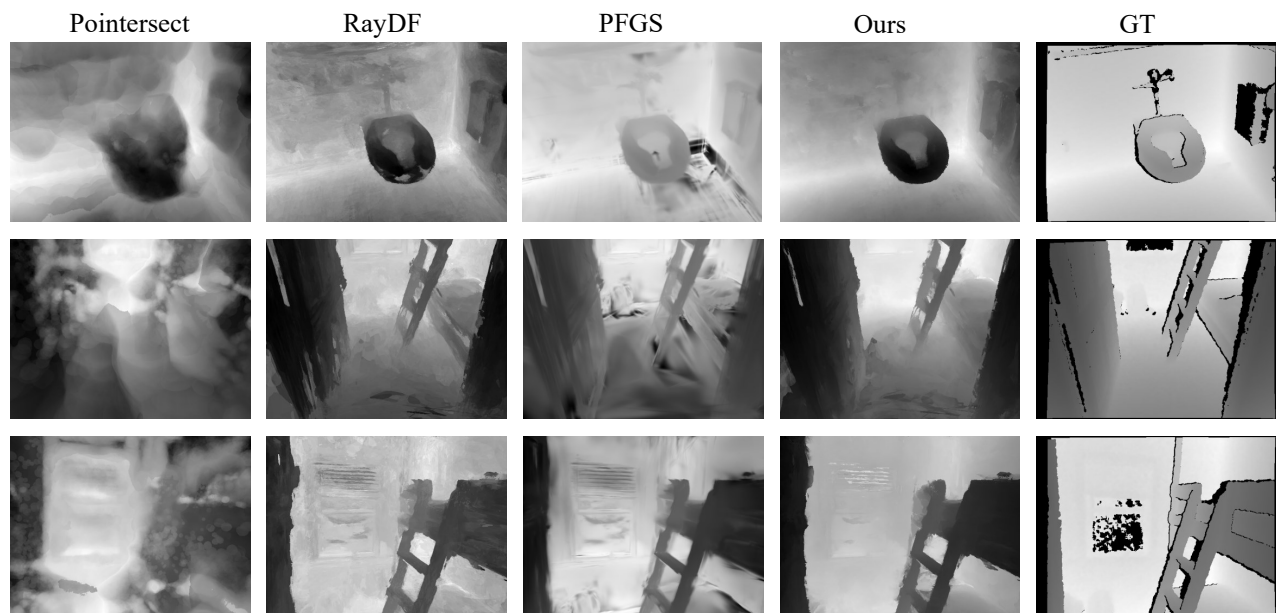


Figure 14. Qualitative results of all methods trained on Gaussians of ScanNet/ScanNet++ and tested on ScanNet/ScanNet++.