PlaneTR: Structure-Guided Transformers for 3D Plane Recovery

We are grateful to the reviewers and area chairs for their efforts and constructive comments. We provide one-to-one responses to the major concerns as follows.

R1, R2, R3: *Justifying the usage of Transformer, as well as the vanilla model specification of Transformer as in DETR.*

Ans: We address this main concern in three-fold as follows. **(i) We follow the insightful suggestion to test an alternative to the Transformer in our network.** We replace the Transformer module by a convolution module. The tokenized line segments are substituted by rasterized line heatmaps computed using the off-the-shelf HAWP method in training and testing due to the difficulty of using the tokenized lines for non-Transformers. The resulting model obtains 1.024/0.904/0.787 for VI/RI/SC respectively in the ScanNetV1 dataset, which are comparable to the PlaneAE method, but consistently inferior to PlaneTR, 0.767/0.925/0.838 for VI/RI/SC respectively (see Table 1 in the submission). Together with the new results in Table I and II, the proposed PlaneTR shows strong empirical advantages. **(ii) Our motivation is to study how to leverage sparse line segments in learning plane recovery**. On the one hand, the proposed tokenized line segment representation naturally leads to consider the Transformer model for its contextual adaptation capability. Exploiting an off-the-shelf line segment detector (HAWP) is to decouple the two tasks and due to the lack of jointly annotated datasets. On the other hand, we completely agree that the end-to-end joint learning of line segment detection and plane recovery is much more appealing. To that end, the progress made by our PlaneTR may shed light on developing a large-scale jointly annotated dataset by the community in the first place. **(iii) The vanilla Transformer decoder is exploited for simplicity to verify our basic ideas of tokenized line segments helping plane recovery**. In the meanwhile, we are aware of the quadratic complexity of the vanilla Transformer decoder w.r.t. the number of tokens, which will hinder applying the proposed PlaneTR in really complex scenarios. State-of-the-art variants of Transformer such as the Swin-Transformer may be leveraged, as well as other more specialized variants for 3D plane recovery, which we leave for future work. We will elaborate on these in the revision.

R1: *The "w/o line segment" ablation study.*

Ans: (i) In Table 3, the results without line segments are obtained by a model trained without using line segments. (ii) The "w/ line" setting in which "inputs an empty line segment sequence" only refers to the results in Fig 7. It is to show how line segments influence the plane segmentation in the full model. We will make them clear in the revision.

R1: *Why not train PlaneTR on ScanNetV2?*

Ans: We train our PlaneTR on ScanNetV2 and follow the evaluation protocol on the testing set used in PlaneRCNN. As reported in Table I, our PlaneTR consistently obtains better results for the two versions of ScanNet dataset.

	ScanNet V1			ScanNet V2			NYUv2 (trained on ScanNet V2, 640 Res.)					
	VI ↓	RI ↑	SC ↑	VI ↓	RI ↑	SC ↑	Rel ↓	log_{10} ↓	RMSE ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
PlaneRCNN	1.337	0.845	0.690	1.336	0.895	0.718	0.183	0.076	**0.619**	71.8	**93.1**	98.3
PlaneTR (Ours)	**0.767**	**0.925**	**0.838**	**1.126**	**0.927**	**0.756**	**0.180**	**0.076**	0.634	**72.5**	92.8	**98.3**

Table I: Results on ScanNet V1, ScanNet V2 and NYUv2 datasets.

R1: *The resolution used for PlaneRCNN.*

Ans: Since PlaneRCNN is built on the MaskRCNN, it entails higher-resolution images to ensure the success of plane mask detection. We do not change the original configuration of PlaneRCNN to avoid other potential unfairness. Meanwhile, we retrain PlaneTR with the same resolution as PlaneRCNN on the ScanNetV2 dataset and achieve comparable results on the NYUv2 dataset (Table I) for depth prediction.

R1: *Why not use all the points along the line segments?*

Ans: Intuitively, the endpoints are more important representationally than other intermediate points. In the experiments, we did not observe any noticeable improvement when using more points on the line segments. Therefore, we adopt the minimally-simple design.

R2: *More evaluation on outdoor datasets.*

Ans: We quantitatively evaluate our PlaneTR on the outdoor HoliCity and SYNTHIA datasets without fine-tuning. As reported in Table II, our PlaneTR consistently outperforms other approaches by large margins.

	HoliCity				SYNTHIA			
	PlaneTR	PlaneAE	PlaneNet	PlaneRCNN	PlaneTR	PlaneAE	PlaneNet	PlaneRCNN
VI ↓	**1.938**	2.437	2.461	2.476	**1.325**	1.825	1.977	1.957
RI ↑	**0.665**	0.654	0.552	0.601	**0.842**	0.811	0.664	0.717
SC ↑	**0.544**	0.440	0.413	0.414	**0.664**	0.568	0.502	0.527

Table II: Evaluation results on two outdoor datasets (*i.e.*, HoliCity and SYNTHIA).

R3: *Computational cost for PlaneTR.*

Ans: Our PlaneTR obtains 12.2 FPS with 200M parameters for the model, which is slower than PlaneAE (18.93 FPS/ 160M) due to the Transformers branch existed. Compared with PlaneNet (1.54 FPS/ 248M) and PlaneRCNN (5.94 FPS/ 265M), our PlaneTR is more efficient. A small scarification of the inference time in our method brings much better plane recovery results.

R4: *The performance for different line detectors?*

Ans: We replace HAWP to LSD to train our PlaneTR under the same protocol. As reported in Table III, the input line segments of HAWP perform better than LSD as HAWP provides better line segments of images. A more comprehensive comparison will be added to the revision.

PlaneTR (with HAWP)			PlaneTR (with LSD)		
VI ↓	RI ↑	SC ↑	VI ↓	RI ↑	SC ↑
0.767	**0.925**	**0.838**	0.790	0.921	0.829

Table III: Performance for different use of line segment detection algorithm.

R4: *How was the non-plane ground-truth defined?*

Ans: According to PlaneNet, plane regions are generated from 3D mesh models and then projected onto images. The pixels that are not covered by any projected mesh face are defined as the non-plane regions.

R4: *The dimension of the plane instance embedding.*

Ans: We tried the dimensions of 2, 8 and 16, and found that 8D embedding performs best. We did not observe any significant improvement for the larger dimensions.