Supplementary Material for Clutter Detection and Removal in 3D Scenes with View-Consistent Inpainting

Fangyin Wei Thomas Funkhouser Szymon Rusinkiewicz
Princeton University

The supplementary material includes this document as well as a video for method overview and qualitative results. In this document, we first explain in detail the datasets used to evaluate 3D segmentation and 3D inpainting in Sec. 1. Then we describe more implementation details in Sec. 2. Finally, we present additional results of our method in Sec. 3.

1. Datasets

1.1. Clutter Segmentation

We select eight representative scenes from ScanNet test set to cover as many as the scene types in the dataset as possible. We use the annotation tool from ScanNet [3] to annotate vertices on clutter objects. In Fig. 1, we show the bird eye views of all eight scenes that we manually labeled to evaluate the clutter segmentation models. In particular, the manual test set covers a wide range of cleanness levels from rooms with very few common clutter objects to highly cluttered rooms. The covered room types include study room, meeting room, bedroom, bathroom, living room, studio, and craft room. As we can see from the third column, in original ScanNet labels, some clutter objects are unlabeled (*e.g.*, second row, objects on the round table) and some objects are mislabeled (*e.g.*, last row, clutter objects in the dark blue closet). In comparison, our manually annotated dataset correctly labels all clutter in the scenes. The time required for each scene varies from 30 minutes (only for extremely clean scenes) to hours. This also demonstrates that it is impractical to annotate a dataset of thousands of scenes with precise clutter/non-clutter labels.

For all experiments, we sample once every 5 frames from the raw captured sequences to reduce redundancy and improve efficiency. To render virtual views for clutter segmentation, we use cameras with fixed intrinsics. We randomly put cameras at a distance of 2-5 meters from the mesh surface looking at the scene to render views for training.

1.2. 3D Inpainting

The metrics used to quantitatively evaluate 3D inpainting (main Tab. 2) requires ground truth, which is not available in any existing datasets. Therefore, we construct a synthetic evaluation set. We use the official test split from Matterport3D [1] which originally has 391 rooms. For each room, we identify regions that belong to non-clutter categories and have a normalized normal $n = (n_x, n_y, n_z)$ where $|n_z| = max(|n_x|, |n_y|, |n_z|)$ (z is pointing up). In the meantime, we cut instances that belong to the class pure clutter from the scenes and drop those instances on the identified non-clutter regions. We find the convex hull of the dropped clutter and any surface of the room that is contained within the convex hull is removed to create holes. In other words, the original room is the ground truth room after 3D inpainting, and the room with introduced holes is the input to different 3D inpainting methods.

Both baseline methods use TSDF representation as input and output. To create the input, we mask out the TSDF and color values for the voxels that are within the convex hull. This is very similar to how FF [7] creates free-form holes in their methods.

For each hole of the room, we find a view from the dataset where the hole is placed as close to image center as possible. We also find other views that have at least 1000 pixel overlap with the centered view. We project the clutter onto all the above views to create holes on RGB-D data. We then use them to run our RGBD inpainting and mesh reconstruction.

Below we explain how we compute 3D and 2D metrics once the inpainted 3D is obtained (from either our method or baselines). To evaluate the proposed method's 3D performance, we compare Chamfer Distance between our inpainted mesh and groundtruth mesh reconstructed (using PSR) using original (unmasked) views. To evaluate baselines' 3D performance, we compare Chamfer Distance between marching cubes mesh from predicted TSDF and marching cubes mesh from groundtruth TSDF. This is fairer than using mesh reconstructed from PSR as groundtruth for baseline methods (using PSR results as groundtruth would result in worse results than using marching cubes results from TSDF, so we don't use PSR as groundtruth for baselines). To evaluate 2D performance, we render from the centered view and compute the metrics between the ground truth renderings (without dropped clutter) and renderings

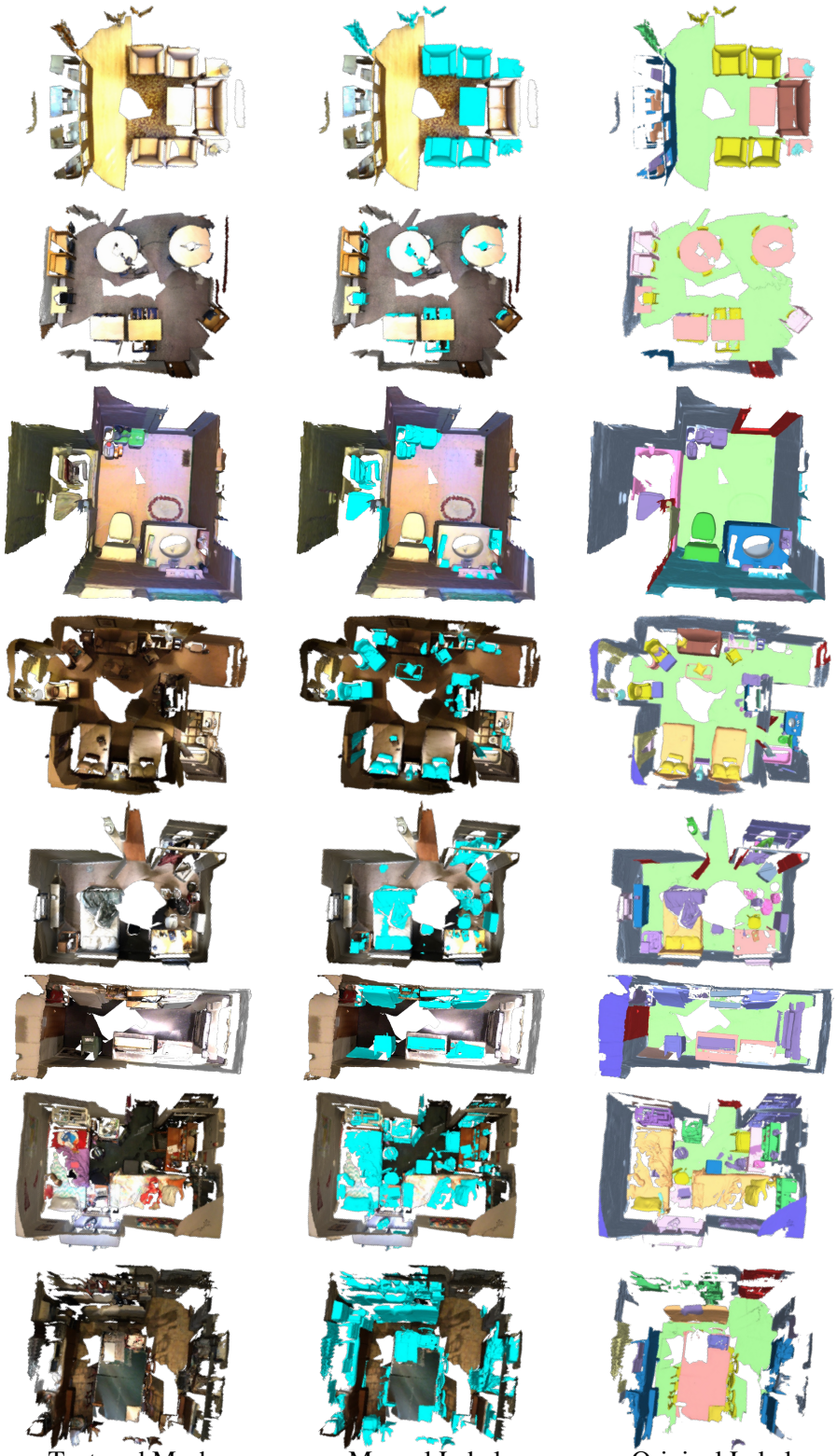

Figure 1. **Clutter Segmentation Manually-Labeled Evaluation Dataset.** We show the bird eye views of all eight scenes (scaled, rotated, and ceiling surfaces removed for better view) that we manually labeled to evaluate the clutter segmentation model of our method and the BPNet [6] baseline.

Table 1. **Runtime analysis.** We compute average Timing based on scenes that have approximately 200 frames and a $150,000$-vertex input mesh from original scan data. All GPUs are NVIDIA TITAN X with 12 GB memory. We can see that it takes slightly over 30 min to process a 200-frame RGB-D sequence, and the majority of the time is spent on consistency checks and surface reconstruction.

Step	Time	Device	Output size
Clutter Segmentation	2s	GPU	$150,000$ points
Mask Projection	30s	GPU	200 images
Image Inpainting	50s	GPU	200 images 320×240
Depth Completion	80s	GPU	200 images 304×228
Consistency Checks	18min	GPU	200 images 304×228
Surface Reconstruction	11min	CPU	1,500,000 vertices

from our method or baselines.

2. Implementation Details

We show in Tab. 1 the average timing of each step for a 200-frame sequence. Specifically, we compute average Timing based on scenes that have approximately 200 frames and a $150,000$-vertex input mesh from original scan data. All GPUs are NVIDIA TITAN X with 12 GB memory. We can see that it takes slightly over 30 min to process a 200-frame RGB-D sequence, and the majority of the time is spent on consistency checks and surface reconstruction.

We find empirically Cross Entropy loss for two classes works better than Binary Cross Entropy loss.

2.1. Clutter Definition

Our focus is to propose a new type of scene segmentation based on shared properties (*e.g.*, across clutter) rather than common benchmark semantic categories. Therefore, a concrete clutter definition is crucial to test our method but the exact form can be very flexible.

The definition we chose for our result is as follows: an uninstalled object is considered clutter if it's moved at any time within 3 months and with probability $> 95\%$ so that the IoU between the bounding boxes before and after the movement is < 0.5. We want to note that although we need a concrete definition to run the experiments, our proposed techniques for clutter segmentation and 3D inpainting are general and not specifically tailored to work for only one definition. For example, one can easily extend the current clutter definition to be with variable time length (*e.g.*, 1 or 9 months) without affecting the effectiveness of our proposed methods.

2.2. Clutter Segmentation

The model is implemented in PyTorch, trained with batch size 16 on two NVIDIA TITAN X with 12 GB memory. Following BPNet [6], we use SGD optimizer with base learning rate of 0.01 and employ a poly learning rate scheduler with the power set to 0.9. Momentum and weight decay are set to 0.9 and 0.0001, respectively. The model is trained in total for 100 epochs.

During testing, we run the inference on the same mesh multiple times (each time with different camera views) until all camera views have been used. We accumulate the model's predicted class probability for the final 3D segmentation results.

After projecting the 3D segmentation masks onto RGBD images, we further dilate the 2D masks with 6 iterations (pixels) to account for the misalignment between 3D reconstruction and 2D captures. The final masks are applied to color and depth images to remove regions with clutter.

2.3. Depth Completion

As introduced in main paper, we adopt NLSPN [10] that performs non-local depth propagation for image-guided depth completion. The model was originally designed for the task of sparse depth completion that predicts the complete depth map from hundreds of sampled pixels. One of the method's key components is the deformable convolution [11] whose effective sampling field can be the entire image. Therefore, despite its different original goals, this method is still proper for our usage of completing the dense map with (potentially big) holes. We retrain the model on our depth maps with synthetic holes for depth completion. We explain below in detail how we prepare the training data.

We use captured depth data with modifications. Since there is no ground truth depth map before and after clutter removal, we need synthesize training data by ourselves. To do so, we start from a captured depth map d_1. We assume m_1 is the mask for the coarse clutter/non-clutter groupings (main paper Sec. 3.2.1) of d_1. We use the clutter mask m_2 from a second view to mask out regions on d_1. In the meantime, we make sure that the regions originally masked as clutter in m_1 are not masked out. By copying and pasting m_2 from another view, we make sure that the masks during training have realistic boundaries of clutter objects. By not masking clutter from m_1, we guarantee that the depth completion model does not need to hallucinate clutter objects out of the hole areas. And this is exactly the expected behaviors of a depth completion model. We note that since the clutter/non-clutter grouping is very noisy, the training data created as described above still contains noise. However, we empirically find the model trained this way already works much better than using pre-trained model weights trained on the sparse-to-dense depth completion task.

The model is implemented in PyTorch, trained with batch size 12 on one NVIDIA TITAN X with 12 GB memory. Following NLSPN [10], we use Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$ and initial learning rate of 0.001. The model is trained in total for 20 epochs. We set the propagation time of each forward pass as 18.

Table 2. **Ablation on Segmentation Loss.** We compare our area-sensitive loss with original Cross Entropy (CE), balanced Cross Entropy (BCE) [2], median frequency loss (MF) [5], and Focal Loss (FL) [9]. Note: hyper-parameters were set for other methods (β in BCE and γ in FL) using best practices suggested in the original papers.

Method	Manual Test Set (clean)		
	IoU(NC)	IoU(C)	mIoU
CE	0.83	0.48	0.66
BCE	0.74	0.38	0.56
FL	0.83	0.45	0.64
MF	**0.84**	0.54	0.69
Ours	**0.84**	**0.58**	**0.71**

Table 3. **Ablation on Consistency Checks.** sp, cp, cv stand for single-frame pruning, cross-frame pruning, and cross-frame voting, respectively.

ID	sp	cp	cv	L1(↓)	L2(↓)	PSNR(↑)	SSIM(↑)	LPIPS(↓)	CD(cm)(↓)
a				0.034	0.075	21.820	0.860	0.201	9.013
b	✓			0.029	0.051	22.403	0.873	0.189	5.245
c	✓	✓		0.023	0.043	22.965	0.880	0.171	2.612
d	✓	✓	✓	0.019	0.035	23.820	0.891	0.150	0.931

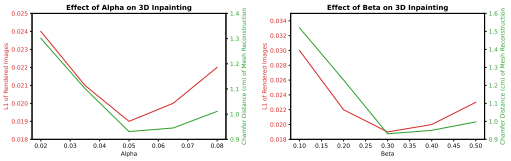

Figure 2. **Ablation on α, β on 3D geometry and rendered image**.

3. More Results

3.1. Ablation on Segmentation Loss

Tab. 2 shows an ablation study comparing different losses designed to combat imbalance. Our per-instance area-sensitive loss performs the best. BCE [2] and FL [9] perform worse than the CE baseline for clutter segmentation. MF [5] improves over CE by re-weighting classes based on cumulative class surface area, but it is worse than ours. Since clutter usually comprises many small object instances, per-instance weighting is advantageous for this task.

3.2. Ablation on Consistency Checks

Tab. 3 studies the contribution of each consistency check on reconstruction for 100 Matterport3D test scenes (Sec. 1.2). Every consistency check improves the 3D inpainting, especially the mesh reconstruction quality.

3.3. Ablation on α, β

Fig. 2 studies the effects of varying α and β on 3D inpainting. β is relatively high because there are limited overlapping views for the same region; a lower β increases the sensitivity to noise.

3.4. Qualitative Results

In Fig. 3 , we show additional qualitative results for our entire pipeline of object removal and 3D inpainting, where we compare with baselines of 3D inpainting. The layout is the same as in the main paper. Alongside input mesh texture and geometry, we first show our predicted 3D clutter segmentation. Then we show the texture and geometry of our final inpainted mesh. In the last two columns, we compare with the geometry from Poisson surface reconstruction (PSR) [8] and a hole-filling algorithm. For fair comparison, the input meshes (first two columns) are also reconstructed using the same Poisson surface reconstruction settings as described in main paper Sec. 3.3.3. The mesh used to visualize segmentation (third row) is from original ScanNet dataset. This is a much smoothed and simplified version than the mesh reconstructed using Sec. 3.3.3.

In Fig. 4 and Fig. 5, we show additional results of 3D inpainting compared with two baselines, SPSG [4] and FF [7]. We can see that our method outputs results of higher resolution than prior work that uses volumetric representations. The proposed method also fills in the hole with more coherent texture and geometry compared to baselines.

Figure 3. **Additional results for automatic clutter removal and 3D inpainting.** Each row shows a rendered view of an input 3D scene, together with our clutter segmentation and a rendered view of the scene with inpainted color and geometry. We compare the latter to removing clutter and filling the resulting holes with either Poisson Surface Reconstruction (PSR) or a triangulation-based hole-filling algorithm.

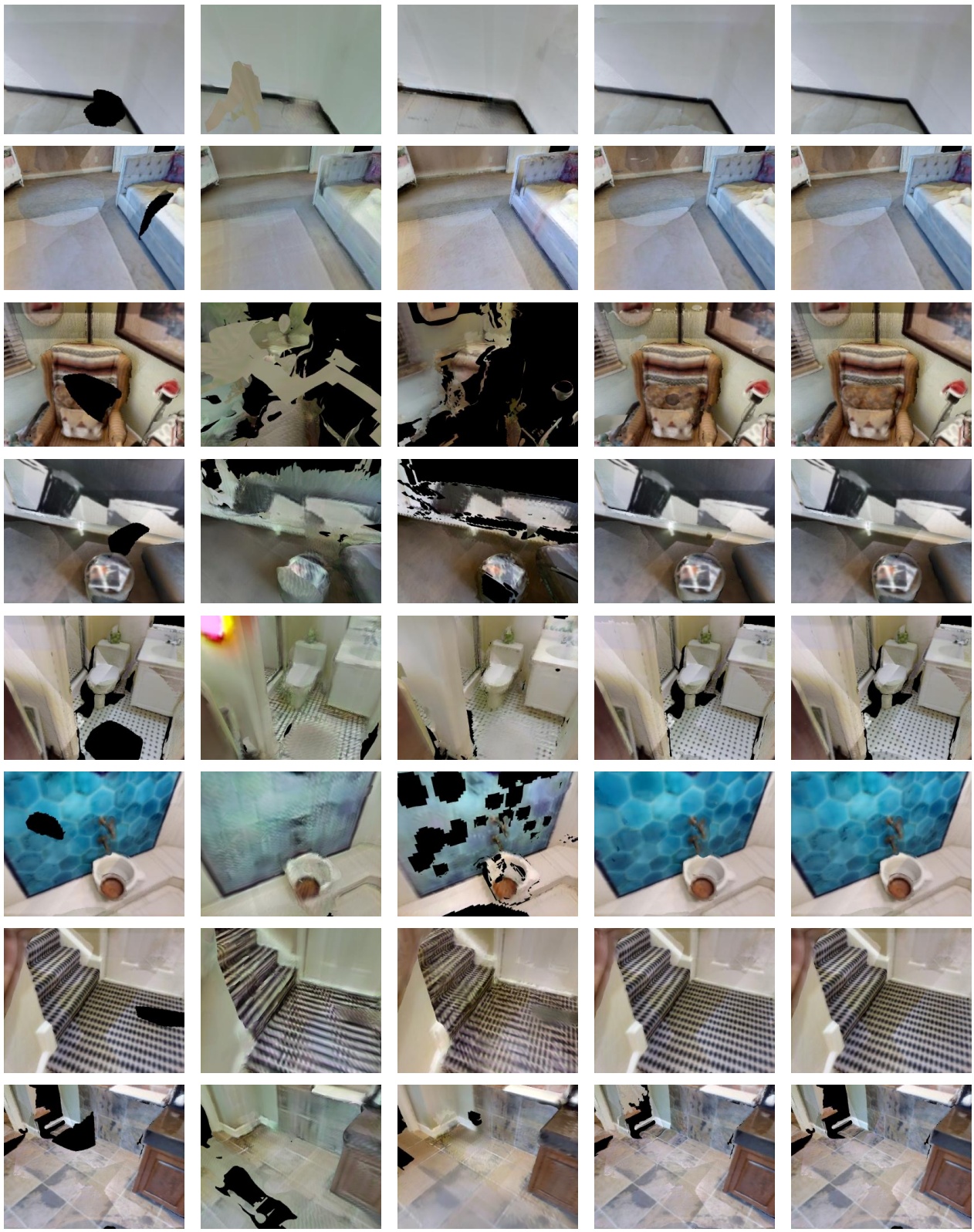

Figure 4. **Additional results for 3D inpainting.** The first column shows a rendered view of an input 3D scene, The second and third rows are results from baseline SPSG and FF. The fourth column is results of proposed method. The last column is ground truth mesh.

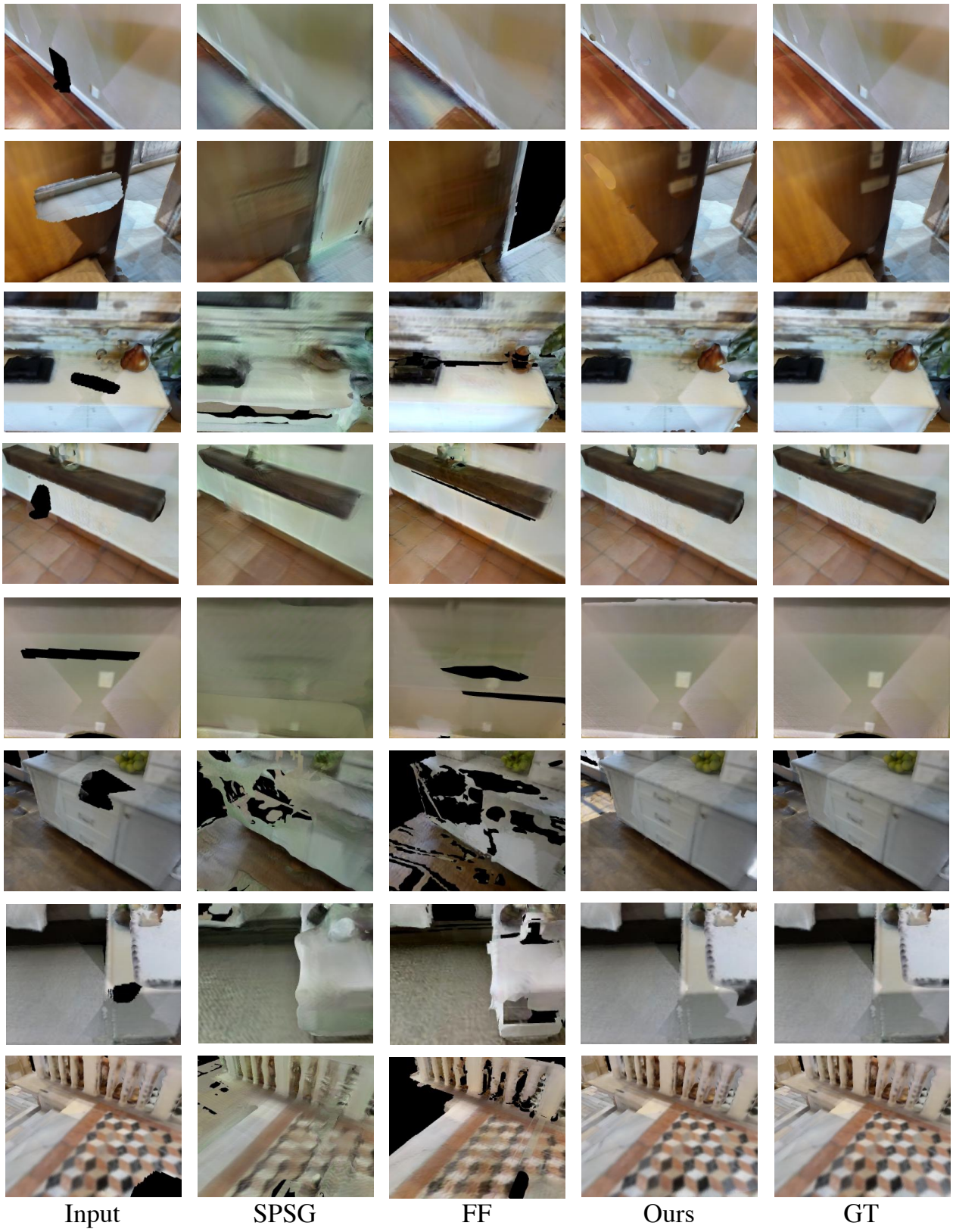

Figure 5. **Additional results for 3D inpainting.** The first column shows a rendered view of an input 3D scene, The second and third rows are results from baseline SPSG and FF. The fourth column is results of proposed method. The last column is ground truth mesh.

References

[1] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017. 1

[2] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *CVPR*, pages 9268–9277, 2019. 4

[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, pages 5828–5839, 2017. 1

[4] Angela Dai, Yawar Siddiqui, Justus Thies, Julien Valentin, and Matthias Nießner. Spsg: Self-supervised photometric scene generation from rgb-d scans. In *CVPR*, pages 1747–1756, 2021. 4

[5] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, pages 2650–2658, 2015. 4

[6] Wenbo Hu, Hengshuang Zhao, Li Jiang, Jiaya Jia, and Tien-Tsin Wong. Bidirectional projection network for cross dimensional scene understanding. In *CVPR*, 2021. 2, 3

[7] Ru-Fen Jheng, Tsung-Han Wu, Jia-Fong Yeh, and Winston H Hsu. Free-form 3d scene inpainting with dual-stream gan. *BMVC*, 2022. 1, 4

[8] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 4

[9] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017. 4

[10] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. Non-local spatial propagation network for depth completion. In *ECCV*, 2020. 3

[11] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *CVPR*, pages 9308–9316, 2019. 3