Supplementary Material for Convolutional Occupancy Networks

Songyou Peng[1,2] Michael Niemeyer[2,3] Lars Mescheder[2,4⋆]
Marc Pollefeys[1,5] Andreas Geiger[2,3]

[1]ETH Zurich [2]Max Planck Institute for Intelligent Systems, Tübingen
[3]University of Tübingen [4]Amazon, Tübingen [5]Microsoft

In this supplementary document, we first give a detailed overview of our architectures and baseline implementations in Section 1. We then discuss our training procedure for both object- and scene-level reconstruction in Section 2. In Section 3, we describe the generation of the synthetic indoor scene dataset. Detailed definitions for all three evaluation metrics are then provided in Section 4. Additional quantitative and qualitative results for all ShapeNet classes can be found in Section 5. Finally, we provide implementation details of our fully-convolutional model and more qualitative results on the Matterport3D dataset in Section 6.

1 Network Architectures

In this section, we provide a detailed description of our network architectures as well as the implementation of our PointConv baseline.

Point Cloud Encoder: We first use a fully-connected layer followed by a fully-connected ResNet [5] block to map the three-dimensional input point coordinates into the feature space. Next, unlike PointNet [12] which pools over all points to acquire a global feature, we perform the pooling operation locally. Depending on the defined plane/volume feature maps, we perform max-pooling only over the features falling into the same pixel/voxel cell. The locally pooled features are concatenated with the features before pooling, and then fed into the next ResNet block. We use 5 of these ResNet blocks with intermediate pooling to obtain the final point-wise features.

Voxel Encoder: Given an occupancy grid as input, we use a single 3D convolutional layer with convolution kernel size $3 \times 3 \times 3$ to extract voxel-wise features with dimension of 32.

U-Net: We use a U-Net [14,4] to process the plane or volume features. We follow [14] and adapt a modified implementation from [6] for our 2D variants. For our 3D variant, we adapt the 3D U-Net [4] implementation from [16]. We set the input and output feature dimensions to 32. Note that we choose the depth of the U-Net such that the receptive field is equal or larger than the size of the feature plane or volume. For example, when considering a 3D feature volume of 32^3 or a 2D feature plane of 128^2, the depth is set to 3 or 5, respectively.

⋆ This work was done prior to joining Amazon.

No. Blocks	Hidden Dim.	GPU Memory	IoU	Chamfer-L_1	Normal Consistency
3	32	2.2G	0.857	0.050	0.936
5	32	2.4G	0.861	0.048	0.937
5	256	3.8G	**0.864**	**0.047**	**0.941**

Table 1. Ablation Study on Network Architecture. We train our 3-plane method with resolution of 64^2 on the ShapeNet "chair" class with different numbers of ResNet blocks and hidden feature dimensions.

Occupancy Prediction Decoder: To predict the occupancy probability of query points, we use the network of [11] comprising a stack of fully-connected ResNet blocks. Table 1 provides an overview over the number of ResNet blocks and hidden dimensions. For all experiments in the main paper, we use a hidden feature dimension of 32 and 5 ResNet blocks for the occupancy prediction network.

Architecture Comparison with ONet [9]: For point cloud inputs, ONet uses a PointNet [12] as point cloud encoder and 5 fully-connected ResNet blocks as occupancy decoder. Both networks have a hidden dimension of 512, resulting in 10.4 million parameters in total. In contrast, our method uses shallow variants for both networks with a hidden dimension of 32: as discussed, we use a shallow local PointNet and consider the less memory-intense conditioning in the decoder from [11]. Combined, our shallow PointNet and our decoder have 43k parameters. Our 2D/3D U-Net has roughly 1 million parameters depending on the depth. Thus, our final model is more memory-efficient than ONet. Moreover, we perform batch-processing over instances as well as points. Hence, the decoder is queried more often than the encoder. As we are able to use a shallow decoder, this further reduces memory consumption in practice.

PointConv: We provide implementation details of our baseline method PointConv in the following. The point-CNN based PointNet++ [13] is used to extract point-wise features. We use the semantic and part segmentation architecture provided in [13] which contains three set abstraction layers and two point feature propagation layers. We adapt the PyTorch implementation from [17]. Given the point features, we interpolate them using Gaussian kernel regression for query points. More specifically, we first calculate the Euclidean distance between a query point and all points in the input point cloud. The weights are then computed using a Gaussian kernel with 0.1 as the defined variance. After performing weight normalization, we acquire interpolated point-wise features for query points and estimate its occupancy probability with an occupancy network as discussed before. We train PointConv end-to-end by backpropagating through the convolutional operations and the Gaussian kernel regression.

2 Training Details

All methods are trained for at least 300000 iterations. We use the Adam optimizer [8] with a learning rate of 10^{-4} for all methods. We perform evaluations on the validation set every 10000 iterations and pick the model for testing which performs best wrt. volumetric IoU on the validation set.

Object-Level Reconstruction: For the reconstruction from point cloud experiments, we use a batch size of 32 for all our methods including ONet [9], and 24 for the baseline PointConv. For the voxel super-resolution tasks, we train all methods with a batch size of 64.

Scene-Level Reconstruction: We use the official implementation[1] of ONet [9] but change the batch size to 12 in order to fit into GPU memory. For the baseline PointConv the batch size is set to 16. Our lightweight architectures allow us to set the batch size to 32 for our plane encoder for a resolution of 128^2 and 3×128^2, as well as the volumetric encoder for a resolution of 32^3. For our variant combining 2D and 3D features, the batch size is 24, while we use a batch size of 6 for our volumetric approach with a resolution of 64^3.

3 Data Generation

In this section, we provide details regarding data generation of our synthetic scenes dataset. We use the watertight meshes from the Choy et al. [3] subset of the ShapeNet dataset from [9]. Training, validation, and test splits are from [3].

We generate scenes with 4 to 8 objects and for each type we generate 1000 scenes, so there are 5000 scenes in total. For a single scene, we sample the x-y ratio of the ground plane uniformly between 0.3 and 1.0. For each object in the scene, we sample a rotation angle around the z-axis and a scaling factor uniformly from an interval which depends on how many objects are in the scene in total. We place the objects randomly in the scene via rejection sampling. We draw 4 samples from a Bernoulli distribution to decide whether to add a wall to the respective border of the scene. We sample the wall height uniformly from the interval between 0.2 and 0.4.

We divide the generated scenes into training, validation, and test sets. We further adhere to the object-level splits from [3] to not have similar objects in scenes of different splits.

4 Metrics

In this section, we provide the formal definitions of all three metrics that we use for evaluation. We follow the definition from [9].

Volumetric Intersection over Union (IoU): Let $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$ be the set of all points that are inside or on the surface of the predicted and ground truth

[1] https://github.com/autonomousvision/occupancy_networks

mesh, respectively. The volumetric IoU is the volume of two meshes' intersection divided by the volume of their union:

$$\text{IoU}(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) \equiv \frac{|\mathcal{M}_{\text{pred}} \cap \mathcal{M}_{\text{GT}}|}{|\mathcal{M}_{\text{pred}} \cup \mathcal{M}_{\text{GT}}|}. \tag{1}$$

We randomly sample 100k points from the bounding boxes and determine if the points lie inside or outside $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$, respectively.

Chamfer-L_1: Define an accuracy and completeness score of $\mathcal{M}_{\text{pred}}$ wrt. $\mathcal{M}_{\text{GT}}$:

$$\text{Accuracy}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}) \equiv \frac{1}{|\partial\mathcal{M}_{\text{pred}}|} \int_{\partial\mathcal{M}_{\text{pred}}} \min_{\mathbf{q}\in\partial\mathcal{M}_{\text{GT}}} \|\mathbf{p}-\mathbf{q}\| \mathrm{d}\mathbf{p} \tag{2}$$

$$\text{Completeness}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}) \equiv \frac{1}{|\partial\mathcal{M}_{\text{GT}}|} \int_{\partial\mathcal{M}_{\text{GT}}} \min_{\mathbf{p}\in\partial\mathcal{M}_{\text{pred}}} \|\mathbf{p}-\mathbf{q}\| \mathrm{d}\mathbf{q} \tag{3}$$

where $\partial\mathcal{M}_{\text{pred}}$ and $\partial\mathcal{M}_{\text{GT}}$ denote the surfaces of the two meshes. Then, the Chamfer-L_1 distance between two meshes is defined as below:

$$\begin{aligned}\text{Chamfer-}L_1(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) = \\ \frac{1}{2}(\text{Accuracy}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}) + \text{Completeness}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}))\end{aligned} \tag{4}$$

Normal Consistency: we define the normal consistency score as

$$\begin{aligned}\text{Normal-Con.}(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) \equiv \frac{1}{2\,|\partial\mathcal{M}_{\text{pred}}|} \int_{\partial\mathcal{M}_{\text{pred}}} |\langle n(\mathbf{p}), n(\text{proj}_2(\mathbf{p}))\rangle| \mathrm{d}\mathbf{p} \\ + \frac{1}{2\,|\partial\mathcal{M}_{\text{GT}}|} \int_{\partial\mathcal{M}_{\text{GT}}} |\langle n(\text{proj}_1(\mathbf{q})), n(\mathbf{q})\rangle| \mathrm{d}\mathbf{q}\end{aligned} \tag{5}$$

where $\langle\cdot,\cdot\rangle$ indicates the inner product, $n(\mathbf{p})$ and $n(\mathbf{q})$ the (unit) normal vectors on the mesh surface $\partial\mathcal{M}_{\text{pred}}$ and $\partial\mathcal{M}_{\text{GT}}$, respectively and $\text{proj}_2(\mathbf{p})$ and $\text{proj}_1(\mathbf{q})$ denote the projections of $\mathbf{p}$ and $\mathbf{q}$ onto $\partial\mathcal{M}_{\text{GT}}$ and $\partial\mathcal{M}_{\text{pred}}$ respectively.

F-Score: We first define recall and precision. As discussed in [15], recall counts how many points on the GT mesh lie within a certain distance to the reconstruction. Precision counts the percentage of points on the reconstructed mesh that lie within a certain distance to the GT. The F-Score is then defined as the harmonic mean between precision and recall:

$$\text{F-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

5 3D Reconstruction on ShapeNet

In this section, we provide additional experimental results on all 13 classes of the ShapeNet subset of Choy et al. [3].

category	IoU PointConv	IoU ONet	IoU Ours-2D (64^2)	IoU Ours-2D (3×64^2)	IoU Ours-3D (32^3)	Chamfer-L_1 PointConv	Chamfer-L_1 ONet	Chamfer-L_1 Ours-2D (64^2)	Chamfer-L_1 Ours-2D (3×64^2)	Chamfer-L_1 Ours-3D (32^3)
airplane	0.579	0.734	0.844	**0.849**	**0.849**	0.140	0.064	0.034	0.034	**0.033**
bench	0.537	0.682	0.756	**0.830**	0.791	0.120	0.067	0.047	**0.035**	0.041
cabinet	0.824	0.855	0.882	**0.940**	0.923	0.115	0.082	0.067	**0.046**	0.054
car	0.767	0.830	0.870	**0.886**	0.877	0.149	0.104	0.081	**0.075**	0.080
chair	0.667	0.720	0.791	**0.871**	0.853	0.129	0.095	0.067	**0.046**	0.049
display	0.743	0.799	0.863	**0.927**	0.904	0.106	0.082	0.056	**0.036**	0.042
lamp	0.495	0.546	0.668	0.785	**0.792**	0.215	0.159	0.098	**0.059**	0.066
loudspeaker	0.807	0.826	0.878	**0.918**	0.914	0.148	0.118	0.086	**0.064**	0.065
rifle	0.565	0.668	0.798	**0.846**	0.826	0.098	0.066	0.037	**0.028**	0.031
sofa	0.811	0.865	0.912	**0.936**	0.923	0.104	0.073	0.050	**0.042**	0.046
table	0.654	0.739	0.803	**0.888**	0.860	0.113	0.076	0.056	**0.038**	0.043
telephone	0.856	0.896	0.936	**0.955**	0.942	0.061	0.046	0.032	**0.027**	0.030
vessel	0.652	0.729	0.826	**0.865**	0.860	0.138	0.094	0.054	**0.043**	0.045
mean	0.689	0.761	0.833	**0.884**	0.870	0.126	0.087	0.059	**0.044**	0.048

category	Normal Consistency PointConv	Normal Consistency ONet	Normal Consistency Ours-2D (64^2)	Normal Consistency Ours-2D (3×64^2)	Normal Consistency Ours-3D (32^3)	F-Score PointConv	F-Score ONet	F-Score Ours-2D (64^2)	F-Score Ours-2D (3×64^2)	F-Score Ours-3D (32^3)
airplane	0.819	0.886	0.924	0.931	**0.932**	0.562	0.829	0.967	0.965	**0.968**
bench	0.811	0.871	0.888	**0.921**	0.911	0.617	0.827	0.917	**0.964**	0.944
cabinet	0.895	0.913	0.915	**0.956**	0.953	0.719	0.833	0.874	**0.956**	0.931
car	0.845	0.874	0.884	**0.893**	0.891	0.577	0.747	0.827	**0.849**	0.834
chair	0.851	0.886	0.902	**0.943**	0.942	0.618	0.730	0.847	**0.939**	0.930
display	0.910	0.926	0.935	**0.968**	0.965	0.679	0.795	0.891	**0.971**	0.956
lamp	0.779	0.809	0.855	0.900	**0.910**	0.453	0.581	0.761	0.892	**0.910**
loudspeaker	0.894	0.903	0.918	0.939	**0.942**	0.647	0.727	0.827	**0.892**	0.881
rifle	0.796	0.849	0.905	**0.929**	0.924	0.682	0.818	0.947	**0.980**	0.969
sofa	0.900	0.928	0.945	**0.958**	0.956	0.697	0.832	0.925	**0.953**	0.943
table	0.878	0.917	0.932	**0.959**	0.956	0.694	0.824	0.897	**0.967**	0.953
telephone	0.961	0.970	0.976	**0.983**	0.981	0.880	0.930	0.974	**0.989**	0.987
vessel	0.817	0.857	0.898	**0.919**	**0.919**	0.550	0.734	0.881	**0.931**	0.927
mean	0.858	0.891	0.914	**0.938**	0.937	0.644	0.785	0.887	**0.942**	0.933

Table 2. 3D Reconstruction from Point Clouds on ShapeNet. This table shows a per-category comparison of baselines and different variants of our approach. For this experiment, we train all methods simultaneously on all 13 classes.

Reconstruction from Noisy Point Clouds: Table 2 shows a quantitative comparison of the baselines and our methods. Our method outperforms all baselines on all classes and often by a significant margin. Fig. 2 and Fig. 3 show additional qualitative results.

Generalization: In this experiment, we train only on the "chair" category and test on "table". In contrast to the baselines, our method degrades gracefully (see Fig. 1). This emphasizes the importance of equivariant representations and geometric reasoning using both local and global features.

Reconstruction from Noisy Partial Point Clouds: We also investigate our method's capability to reconstruct shapes from partial point clouds. To this end, we first randomly select one axis of the x, y, z directions and calculate its coordinate range r. Then, we uniformly sample an offset between $[0.7r,\ r]$ and

Fig. 1. Generalization (Chair → Table). We analyze the generalization performance of our method and the baselines by training them on the ShapeNet "chair" category and evaluating them on the "table" category.

filter out all points with coordinates larger than the offset along that axis. The offset is always a positive value, so that e.g. for the z axis, we always crop from the top. Finally, 3000 points are uniformly sampled from the cropped point clouds. Fig. 4 shows our qualitative results.

6 3D Reconstruction on Matterport3D

In this section, we provide implementation details of our fully-convolutional model as well as more qualitative results on the Matterport3D dataset [1].

Implementation Details of Fully-Convolutional Model: We perform training and inference within the unit cube for both, the object-level as well as the synthetic scene reconstruction experiments. However, this does not fully exploit the translation equivariant property of convolution networks and does not scale to arbitrarily sized scenes.

In this section, our goal is to apply our model to scenes of arbitrary size represented in metric real-world units (i.e., in meters). Therefore, our model should be translational equivariant and not depend on a global coordinate system, but only on relative local coordinates.

For this experiment, we use our synthetic scene dataset for training. A scene consists of multiple objects from the ShapeNetCore [2] dataset, see Section 3 in supplementary for details. While no real-world units are provided in this dataset, we find that the synthetic scenes roughly correlate to a real-world unit of 4.4m $\times$ 4.4m $\times$ 4.4m. The voxel size s is a hyperparameter of our model and determines the granularity of the convolutional part. In all experiments, we set $s = 0.02$m. Therefore, each scene is contained in a regular grid of size $220 \times 220 \times 220$ voxels.

To train the network, we predict the occupancy of query points inside grid volumes cropped randomly within the scene. Specifically, at each iteration, we randomly sample one point within the scene as the center of the crop. The crop size is $H \times W \times D$, which is defined as $25 \times 25 \times 25$ voxels in our experiments. Since the receptive field of our network is $r = 64$, the corresponding input crop has a size of $(H + 63) \times (W + 63) + (D + 63) = 88 \times 88 \times 88$ voxels. We use the point cloud encoder described in Section 1 to encode the input point clouds inside each input crop. We use a batch size of 2 crops in practice.

Similarly, at inference time, we split the scene into overlapping crops so that we can perform occupancy prediction of every crop in a sliding-window manner. The crop size is determined to fit into GPU memory. Note that the input crops overlap, such that no padding is needed.

In contrast to Occupancy Networks [9], our method is equivariant to translations that are multiples of the voxel size s. This is achieved by first mapping the global 3D coordinates $\mathbf{p}$ to local voxel-centric 3D coordinates $\mathbf{p}' = h((\mathbf{p} \bmod s)/s)$, where "mod" denotes the element-wise modulo operation for real numbers. Note that $\mathbf{p} \in [0, 1]$, $h(\cdot)$ is a continuous transfer function which in the simplest case is linear $h(x) = x$, but which could also be a non-linear mapping such as the positional encoding described in [10]. In this work, we restrict our analysis to using the identity function, but plan to explore more complex transfer functions in the future.

Additional Qualitative Results: In order to better visualize the reconstruction, we first cut out the ceiling in the z direction and also cut one side of the walls in either x or y direction. This preprocessing was done on the provided region meshes. Next, we combine all the processed meshes into one big mesh for the entire scene. Surface points are then uniformly sampled from this final mesh and used as input to our method. For SPSR, we additionally provide the surface normals for every point in the input point cloud.

In Fig. 5 and Fig. 6, we show more qualitative results of our fully-convolutional model on Matterport3D dataset. Given a reasonable amount of surface points, our method is able to reconstruct scenes of different sizes, ranging from apartments to entire buildings.

Fig. 8 and Fig. 7 show a comparison over SPSR [7]. Note that SPSR requires additional surface normals as input, whereas our method only needs raw point clouds. Moreover, SPSR requires a carefully chosen trimming factor. In contrast, our method does not require any such hyperparameter tuning. Our results indicate that our method better preserves details and the reconstructions contain less artifacts.

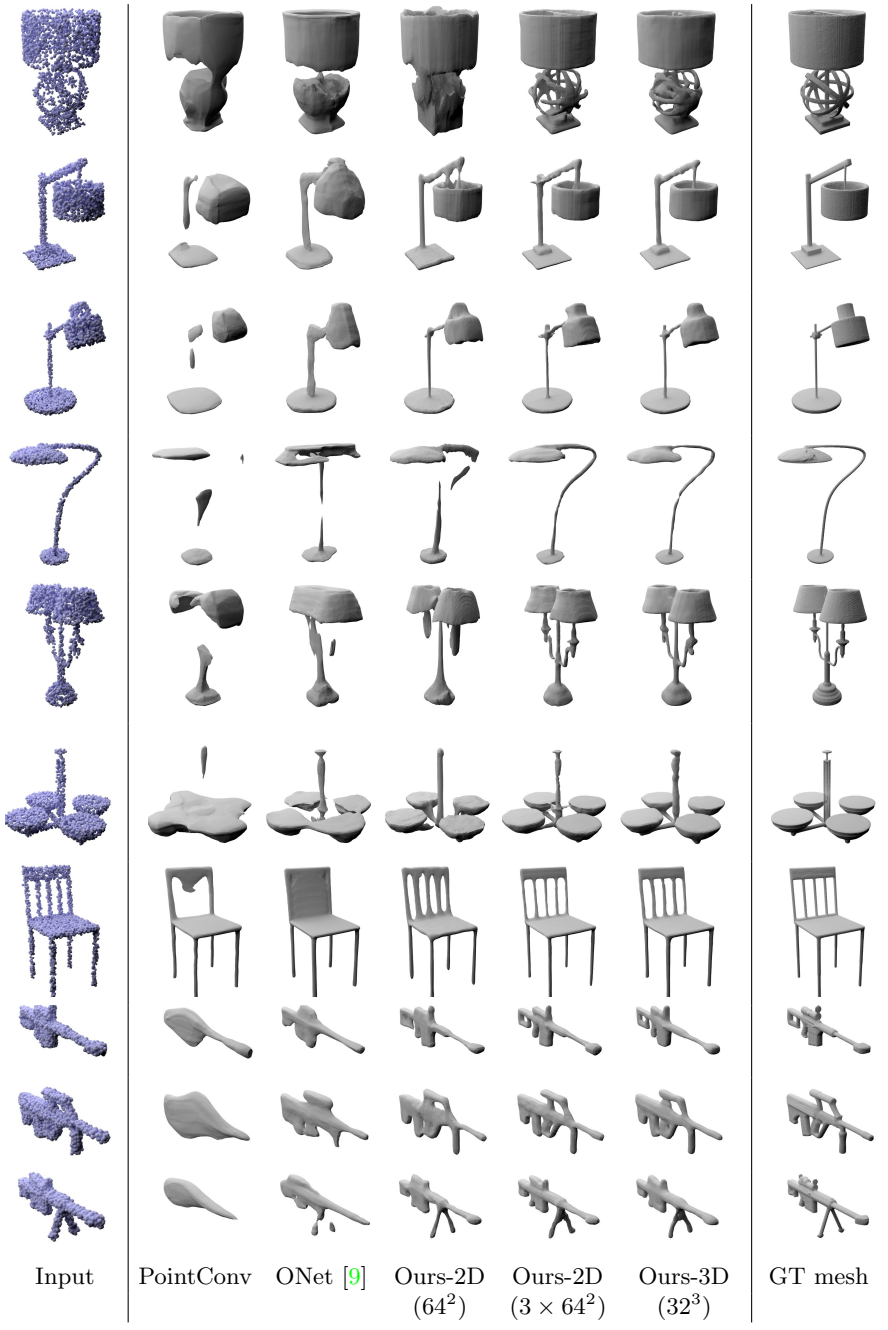


Fig. 2. Object-Level 3D Reconstruction from Point Clouds. We show qualitative results of ONet, PointConv, and our variants on the ShapeNet "lamp" and "rifle" categories. We choose these two categories since many objects have fine geometric details and thin structures. Note that the models are trained on all classes.

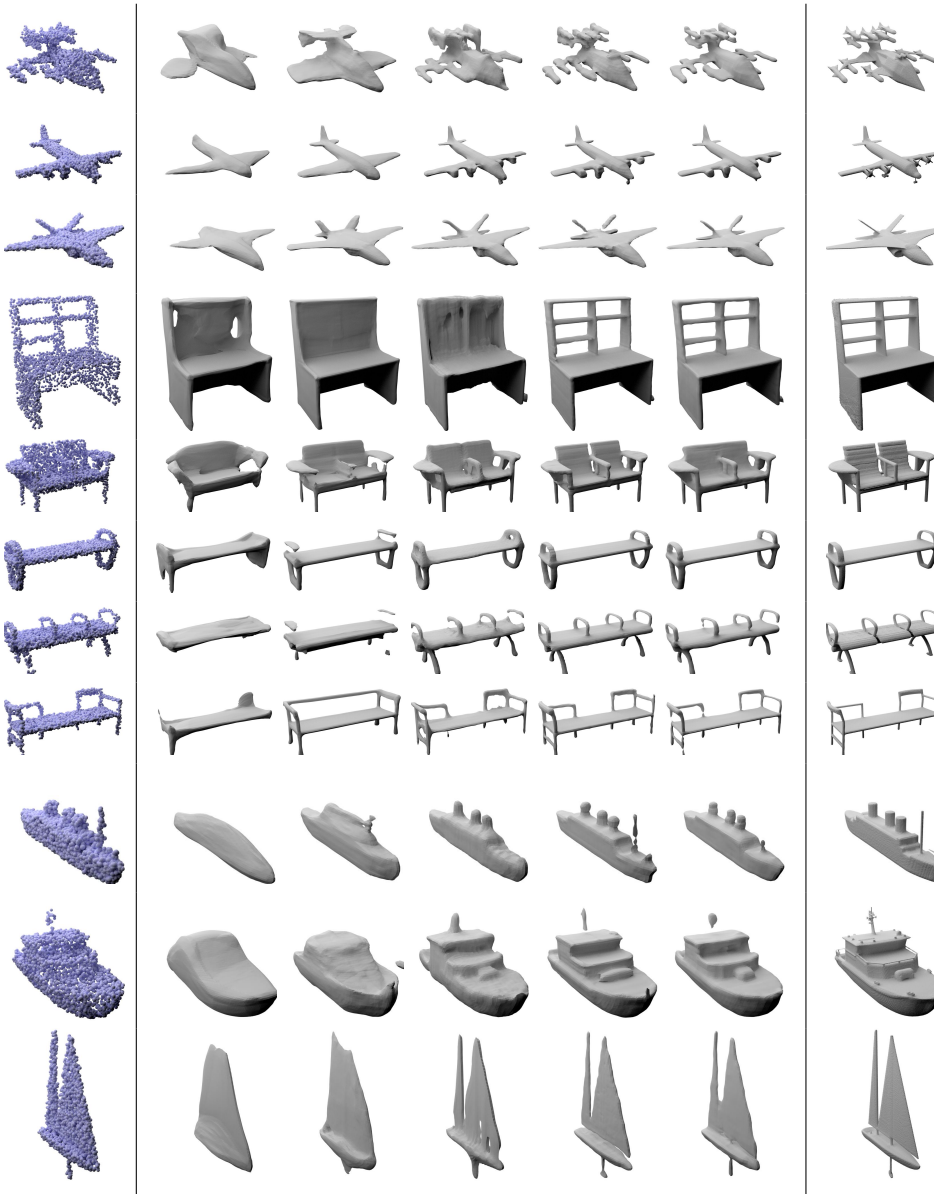


Fig. 3. Object-Level 3D Reconstruction from Point Clouds. We show qualitative results for ONet, PointConv and our variants on the ShapeNet "plane", "bench" and "vessel" categories. Here we select objects with complicated details to visualize the reconstruction quality among different methods. Note that the models are trained on all classes.

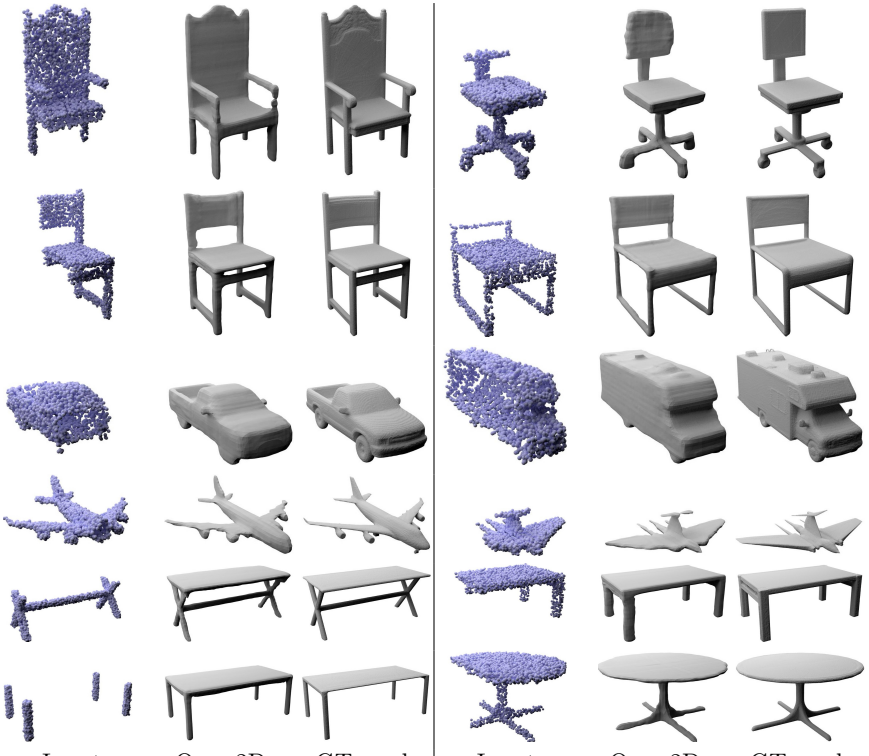
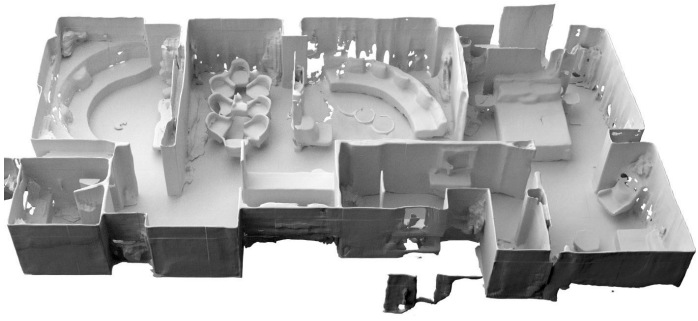
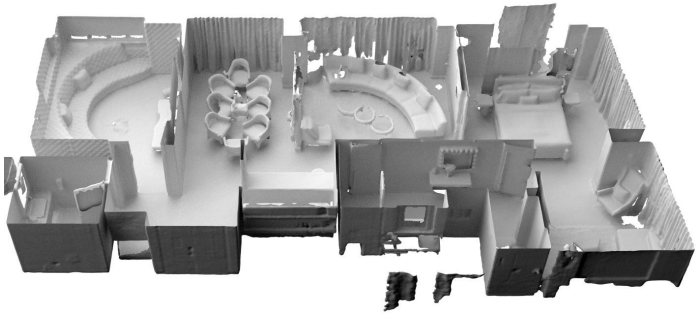


Fig. 4. Object-Level 3D Reconstruction from Partial Point Clouds. We show qualitative results on the ShapeNet "plane", "car", "chair" and "table" categories. Our method is able to correctly reconstruct 3D shapes from partial point clouds. Note that the models are trained on all classes.

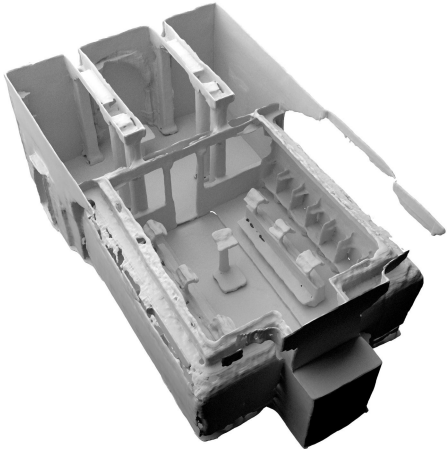


(a) Ours

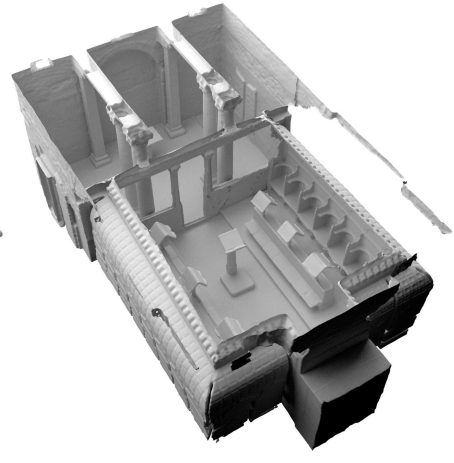


(b) GT Mesh

Fig. 5. Scene-Level Reconstruction on Matterport3D. Scene size: 18.5m × 9.6m × 2.2m. No. points in input point cloud: 60K.



(a) Ours



(b) GT Mesh

Fig. 6. Scene-Level Reconstruction on Matterport3D. Scene size: 11.3m × 6.6m × 4.0m. No. points in input point cloud: 100K.

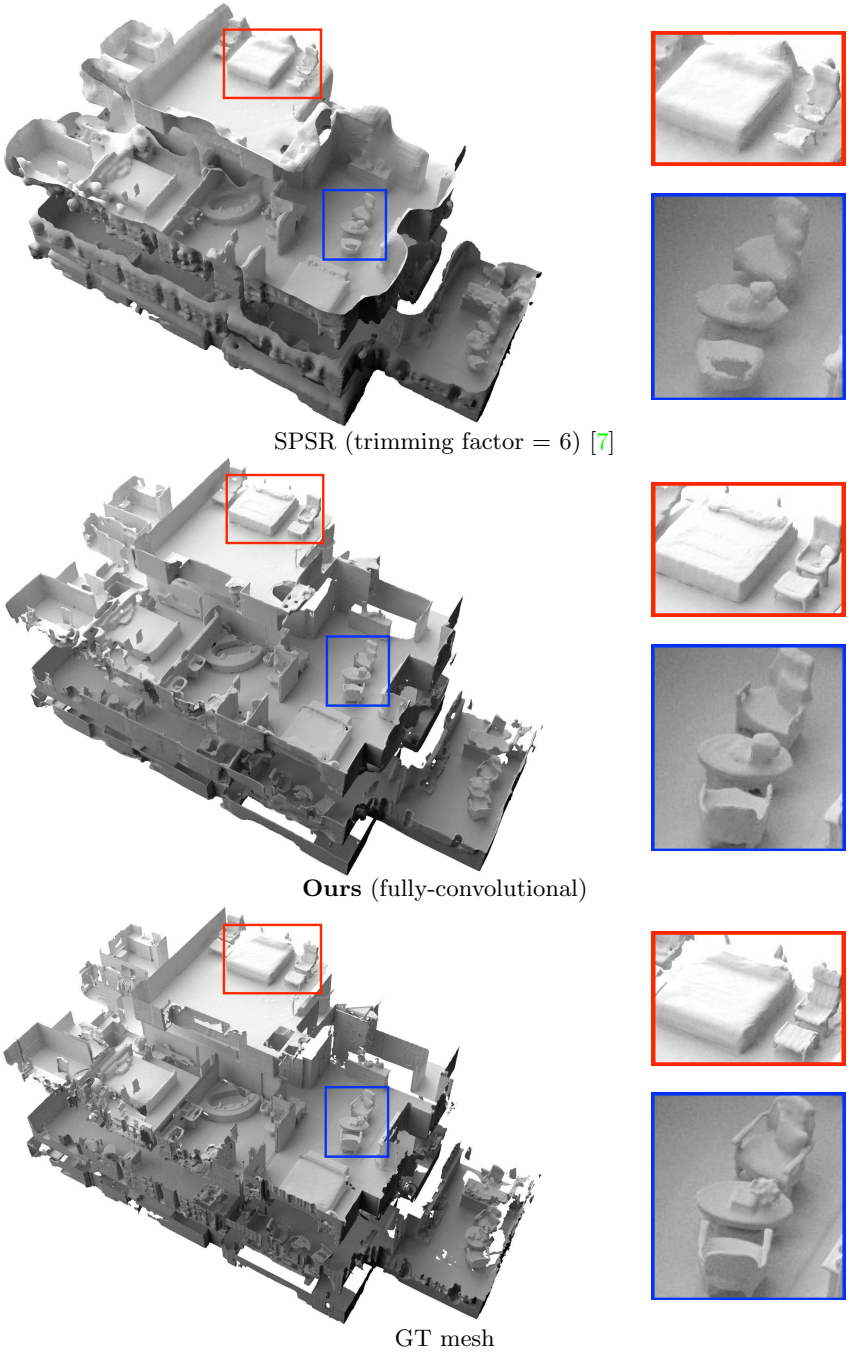


Fig. 7. Comparison of Building-Level Reconstruction on Matterport3D. Scene size: 19.7m × 10.9m × 9.4m. 200K points are sampled from the GT mesh and used as the input to SPSR and our method.

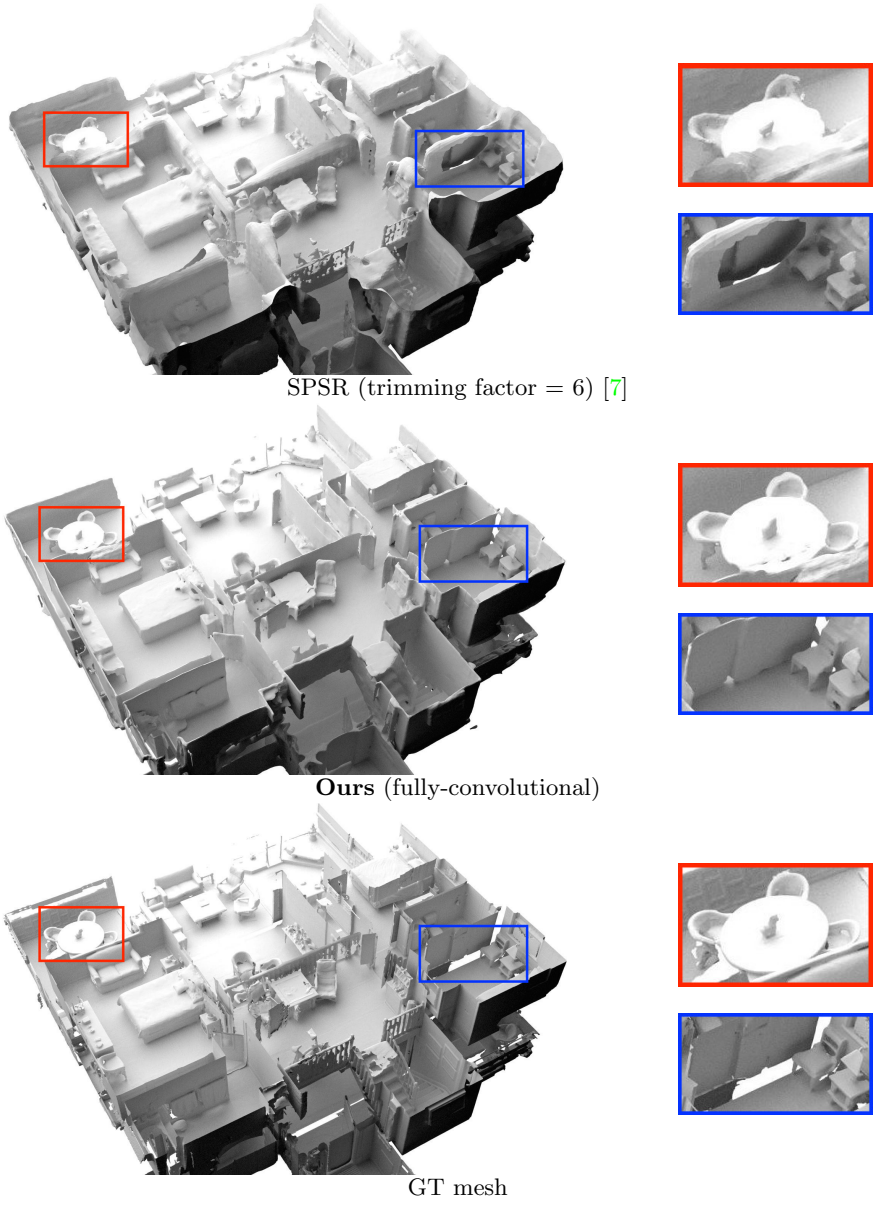


Fig. 8. Comparison of Building-Level Reconstruction on Matterport3D. Scene size: 15.7m × 12.3m × 4.5m. 200K points are sampled from the GT mesh and used as the input to SPSR and our method.

References

1. Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3D: Learning from RGB-D data in indoor environments. Proc. of the International Conf. on 3D Vision (3DV) (2017) 6
2. Chang, A.X., Funkhouser, T.A., Guibas, L.J., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: Shapenet: An information-rich 3d model repository. arXiv.org **1512.03012** (2015) 6
3. Choy, C.B., Xu, D., Gwak, J., Chen, K., Savarese, S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In: Proc. of the European Conf. on Computer Vision (ECCV) (2016) 3, 4
4. Çiçek, Ö., Abdulkadir, A., Lienkamp, S.S., Brox, T., Ronneberger, O.: 3d u-net: Learning dense volumetric segmentation from sparse annotation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI) (2016) 1
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2016) 1
6. Huang, J.: U-net implementation in pytorch. https://github.com/jaxony/unet-pytorch (2017) 1
7. Kazhdan, M.M., Hoppe, H.: Screened poisson surface reconstruction. ACM Trans. on Graphics **32**(3), 29 (2013) 7, 12, 13
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proc. of the International Conf. on Machine learning (ICML) (2015) 3
9. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019) 2, 3, 6, 7, 8, 9
10. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: Proc. of the European Conf. on Computer Vision (ECCV) (2020) 7
11. Niemeyer, M., Mescheder, L.M., Oechsle, M., Geiger, A.: Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2020) 2
12. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2017) 1, 2
13. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems (NeurIPS) (2017) 2
14. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI) (2015) 1
15. Tatarchenko, M., Richter, S.R., Ranftl, R., Li, Z., Koltun, V., Brox, T.: What do single-view 3d reconstruction networks learn? In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019) 4
16. Wolny, A.: 3d u-net model for volumetric semantic segmentation written in pytorch. https://github.com/wolny/pytorch-3dunet (2020) 1
17. Yan, X.: Pytorch implementation of pointnet and pointnet++. https://github.com/yanx27/Pointnet_Pointnet2_pytorch (2020) 2