Supplementary Material for AIR-Nets: An Attention-Based Framework for Locally Conditioned Implicit Representations

Simon Giebenhain Bastian Goldlücke
University of Konstanz, Germany
{simon.giebenhain, bastian.goldluecke}@uni-konstanz.de

In this supplementary document, we provide additional details regarding network architectures, training and evaluation in section 1, experiments evaluating all models in the single-view 3D reconstruction setting in section 3 and a memory and runtime analysis comparing AIR-Nets to all baseline models in section 4. Furthermore, we conduct additional ablation experiments for some critical hyperparameters and the proposed attentive set abstraction module in section 5. Finally, in section 6, we present additional qualitative results.

Our code is available at https://github.com/SimonGiebenhain/AIR-Nets.

1. Architectural and Implementation Details

1.1. Network Architectures

In this section we briefly discuss the baselines' network architectures and highlight the novelty of our decoder.

Occupancy Networks (ONets) [7]. ONets are an early and effective model for encoder-based implicit shape learning. While the authors work with several different kinds of inputs, we are concerned with their PointNet [10] based encoder to infer a 512 dimensional, global latent representation from an input point cloud. We used their official code release[1] with the default parameter settings.

Convolutional Occupancy Networks (ConvONets) [9]. ConvONets are a recent local implicit shape representation framework, offering multiple operating modes (single-plane, multi-plane and 3D), from which we chose the multi-plane version, since it reached the highest accuracy on ShapeNet. This model projects encoded input points on the xy, xz and yz-plane, resorts to 2D convolutional neural networks (CNNs) in the encoder and linearly interpolates features on the encoded planes. The interpolated features of all three planes are added and fed into a feed-forward network (FFN) to make the final occupancy prediction. We believe that ConvONets generalize less well to the FAUST dataset due to the projection procedure and since this model is not permutation equivariant (for a different dataset the authors do propose a change to the encoder to achieve permutation equivariance). We used their official code release[2] with the default parameter settings.

Implicit Feature Networks (IF-Nets) [4]. IF-Nets were concurrently proposed to ConvONets and use information drawn from the feature tensor of a 3D CNN to represent an implicit function. We again used the official code release[3] with the default parameter settings.

Attentive Implicit Representation Networks (AIR-Nets). We already provided all architectural hyperparameters in the *Implementation Details* section of the main paper. Due to space constraints, we did not mention a special case for the dense setting. Here we reduce the internal dimensionality d from 256 to 128 and neighborhood size k_{enc} from 16 to 10 for the initial encoding layer, since this is the only layer operating on 3.000 neighborhoods. After this initial layer, we use the default hyperparameters, as the next layer only operates on $n_1 = 500$ neighborhoods.

Finally, we want to highlight that our decoder is fundamentally different from the baselines'. Both ConvONets and IF-Nets simply interpolate features from a dense grid, since the grid cells can hold very local and specific information. As we ablated in the main paper, interpolating features between local latent vectors leads to a major loss in performance. The reason is that each local latent vector stores more high-level information than grid cells. Therefore, our decoder is equipped with the more expressive vector cross attention (VCA) module to aggregate information appropriately. In section 5.2 we describe both decoder alternatives we compared against in more details.

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

[2] https://github.com/autonomousvision/convolutional_occupancy_networks

[3] https://github.com/jchibane/if-net

1.2. Training

All baseline models were trained using the official training procedure. Due to dependency issues, ConvONets and IF-Nets were trained on a single NVIDIA Quadro RTX 6000. ONets were trained on the same graphics card, since the memory was sufficient for the proposed batch size of 64, which was used for all models, except for IF-Nets. Here, we used a batch size of 20 and reduced the number of supervision points to $|J| = 2.000$ (as used for all other baselines). All AIR-Nets were trained on a single NVIDIA A100-40GB card. More details about memory and runtime requirements are found in section 4.

Since the decoder of AIR-Nets contains a memory intensive VCA module, we only sample $|J| = 1.000$ points for supervision, instead of 2.000. Furthermore, we find that using a slightly higher learning rate of $5e^{-4}$ (instead of $1e^{-4}$ that the baselines use) leads to better results for AIR-Nets when decaying it by a factor of 0.2 after every 200 epochs. All models were trained until the validation error ceased to improve for several hours and at most for 800 epochs.

1.3. Evaluation

To reconstruct meshes from the implicit function we use the MISE algorithm [7] with an initial resolution of 64 and 2 refinement steps. For the main results on ShapeNet [3] in the dense setting we used 3 refinement steps, in order to preserve more details in the reconstructions.

For experiments on $\mathcal{D}_A$ and on the FAUST [1] dataset we use official code from [4] to calculate the metrics. Similarly we use the code of [7] to evaluate models on their dataset $\mathcal{D}_B$.

2. Connection to Graph Neural Networks

While we formulate AIR-Nets as an attention-based framework in the main paper, we provide a formulation as graph neural network (GNN) here. According to Bronstein *et al*. [2], most GNN models fall in the broadest category of *message passing flavoured* GNN [6] and aggregate features from their neighbors according to

$$x_i' = \bigoplus_{j \in \mathcal{N}_i} \varphi(x_i, x_j), \tag{1}$$

where x_i are the features of node i, $\mathcal{N}_i$ holds the neighboring indices of i and φ is the *message function*, encoding both adjacent feature vectors of an edge into a *message*. All incoming messages at node i are then aggregated using a permutation invariant function $\bigoplus$, *e.g*. sum or maximum, to construct new features x_i'. Often residual connections, normalization layers, activation functions and element wise multilayer perceptrons (MLPs) appear between message passing layers.

Message passing GNNs are general enough to include vector self attention (VSA) on the k nearest neighbor (kNN) graph, where the message function

$$\varphi(x_i, x_j) = s(Q_i, K_j) \odot V_{ij} \tag{2}$$

and $\bigoplus$ applies channel-wise softmax normalization over the similarity scores before summing over the neighborhood. Therefore VSA is a message passing flavoured GNN, yet its computational structure is quite restricted (*e.g*. see the use of a similarity function and that positional information and other features are strictly separated, for details see the definition in the main paper). In fact, the dependence of V on i is the only reason that moves VSA from the *(vector valued) attentional flavour* to the *message passing flavour* category. Attentional flavoured GNNs are slightly less general by restricting the structure of φ to

$$\varphi(x_i, x_j) = a(x_i, x_j)\xi(x_j), \tag{3}$$

where a typically is a scalar valued function (in our case it is vector valued) and ξ transforms the features before the aggregation.

For the full attention layers the underlying graph changes to the fully connected graph. The VCA module can be similarly defined by only considering edges from the query to the key-value points.

Note that there is no particular reason why AIR-Nets use this specific GNN structure, except for the reported results in [16]. Trying different GNN architectures or even continuous convolution operators, *e.g*. [14, 15, 11], are of interest for future research.

3. Single-View Reconstructions

In this section, we compare AIR-Nets against the baseline models in a single-view setting on $\mathcal{D}_B$ with additive measurement noise of 0.005. We generate input point clouds using adapted code from [8]. We first randomly select one of 16 cameras equally distributed over the upper hemisphere of the unit sphere and then extract 3D coordinates from the depth buffer of a rendered 400×400 image. To simulate the observation we then randomly sample 300 and 3000 of these points for the sparse and dense setting respectively.

Except for the different observations, everything else remains the same compared to experiments in the main paper, *i.e*. all models, hyperparameters and training settings.

The quantitative results in table 1 again show that AIR-Nets outperform all baselines by a large margin in the sparse and dense setting. Note that in the dense setting in the main paper IF-Nets and AIR-Nets performed similarly on $\mathcal{D}_B$. We explain this change in relative performances by the fact that the dense single view setting is much more challenging

	Model	IoU↑	L_1-CD ↓	NC↑	F-Score↑
Sparse Input	ONet	0.647	0.01472	0.864	0.650
	ConvONet	0.693	0.01127	0.876	0.720
	IF-Net	0.687	0.01223	0.873	0.710
	ours	**0.730**	**0.00988**	**0.890**	**0.764**
Dense Input	ONet	0.616	0.01851	0.855	0.602
	ConvONet	0.731	0.00968	0.890	0.779
	IF-Net	0.755	0.00947	0.900	0.799
	ours	**0.785**	**0.00802**	**0.911**	**0.836**

Table 1: **Single-View 3D Reconstruction:** Quantitative results for 3D reconstructions from single view point clouds.

than the regular dense setting, which renders full attention layers more valuable again.

Qualitative results are shown in figure 4. Note that ONet visually performs fairly well on some of the examples. This shows the benefits of using a single global representation in the single view setting, i.e. the network learns that every chair has four legs. While this global consistency is less present in all of the locally operating models, they do better at reconstructing uncommon shapes and details.

Finally, note that AIR-Nets are not particularly well-suited for single view reconstructions, as the anchor positions are a subset of the input points. In the future, our framework could be extended to propose new anchor positions for large unoccupied volumes (*e.g.* perform point cloud completion as an intermediary task) and to enforce global consistency more thoroughly.

4. Memory and Runtime Comparisons

In this section, we provide a runtime and memory analysis of AIR-Nets and compare them to the baselines. Table 2 presents measured training speed, memory demand and the epoch number minimizing the validation error for $\mathcal{D}_A$ and $\mathcal{D}_B$. Furthermore, we provide the inference speed of encoding a point cloud and extracting a mesh using the MISE algorithm. Please note that MISE refines the resolution close to the object's surface, meaning that the computation time is influenced by the area of the objects surface. Therefore, slightly degenerate predictions (*e.g.* very bumpy surfaces or artifacts floating in the air) can result in higher inference times.

All numbers were obtained using a single NVIDIA Quadro RTX 6000, in order to evaluate all models on the same GPU. Please note that numbers related to the training are influenced by the number of supervision points $|J|$. For all baselines we used $|J| = 2.000$, as suggested by the authors of ONets and ConvONets. For AIR-Nets we use $J = 1.000$, since it resulted in similar performance and saves memory. When using $|J| = 2.000$ instead, the time per batch item of the vanilla AIR-Net increases to 0.0216 and 0.0448 for the sparse and dense setting respectively. Similarly, the memory demands are increased to 0.66 and 0.83 GB per batch item respectively.

Generally, we note that grid-based methods are less affected by the number of input points, since their computational complexity is mostly determined by their grid resolution. Furthermore, ConvONet require the least and IF-Nets the most amount of memory the during training. AIR-Nets take the most time to train until reaching stagnation, *e.g.* when $M = 50$ training takes roughly 29 hours on the sparse setting and using $M = 100$ in the dense setting takes roughly 3 days. As the full attention layers constitute a large part of the computation, reducing M from 100 to 50 significantly reduces the memory footprint and training time. This effect is less pronounced in the dense setting, since the first layer has to process 3000 points. In contrast the performance is only slightly negatively influenced, *e.g.* on $\mathcal{D}_A$ the IoU drops from 91.3 to 89.2 and from 95.8 to 94.0 in the sparse and dense setting respectively. Therefore AIR-Nets still significantly outperform all baselines on $\mathcal{D}_A$ with $M = 50$.

Furthermore note that in this work focused on high quality reconstructions instead of runtime and memory efficiency. There is still great potential to increase efficiency without negatively influencing effectiveness too much, *e.g.* tuning hyperparameters like M, k_{enc}, k_{dec} and d (see section 5.3), using a cheaper downsampling mechanism (see section 5.1) or resorting to the much more efficient multi-head scaled dot-product attention [13] or even a completely different GNN module, as mention in section 2.

5. Ablation Studies

In the following, we present additional ablation studies on the proposed set abstraction mechanism in section 5.1, provide an explanation of the decoder alternatives ablated in the main paper in section 5.2 and finally analyse the effect of three crucial hyperparameters in section 5.3.

5.1. Set Abstraction

Here we present a thorough comparison of the proposed attentive set abstraction mechanism with the common maxpooling alternative. Since maxpooling based set abstraction demands much less memory (reducing the complete model's total demand to 32.8 GB from 37.8 GB at a batch size of 64), we additionally compare against a compensated maxpooling version, which adds a point transformer block (PTB, see main paper) and elementwise FFN after each downsampling (demanding 35.8 GB in total). Furthermore, we justify the usage of two iterations of VCA, by comparing against a version which only facilitates a single step of VCA (demanding 35.5GB). Figure 1 shows the validation

	Model	Training Speed	Training Memory	Inference Speed	Opt. Epoch $\mathcal{D}_A$	Opt. Epoch $\mathcal{D}_B$
Sparse Input	ONet	0.0031 s/item	0.13 GB/item	2.0 s/mesh	301	532
	ConvONet	0.0023 s/item	0.09 GB/item	2.5 s/mesh	65	499
	IF-Net	0.0465 s/item	1.23 GB/item	2.8 s/mesh	10	43
	Ours$_{M=50}$	0.0113 s/item	0.33 GB/item	3.1 s/mesh	470	-
	Ours	0.0185 s/item	0.57 GB/item	3.1 s/mesh	763	606
Dense Input	ONet	0.0122 s/item	0.31 GB/item	1.9 s/mesh	185	442
	ConvONet	0.0027 s/item	0.10 GB/item	1.9 s/mesh	47	345
	IF-Net	0.0465 s/item	1.23 GB/item	3.1 s/mesh	27	140
	Ours$_{M=50}$	0.0394 s/item	0.53 GB/item	2.8 s/mesh	401	-
	Ours	0.0421 s/item	0.71 GB/item	2.3 s/mesh	468	296

Table 2: **Runtime and Memory Demands:** *Inference speed* denotes the time to encode a point cloud and reconstruct a 3D shape using MISE with 2 refinement steps. The last two columns denote the epoch achieving the minimum validation error. Hyphens are placed where we did not train a model.

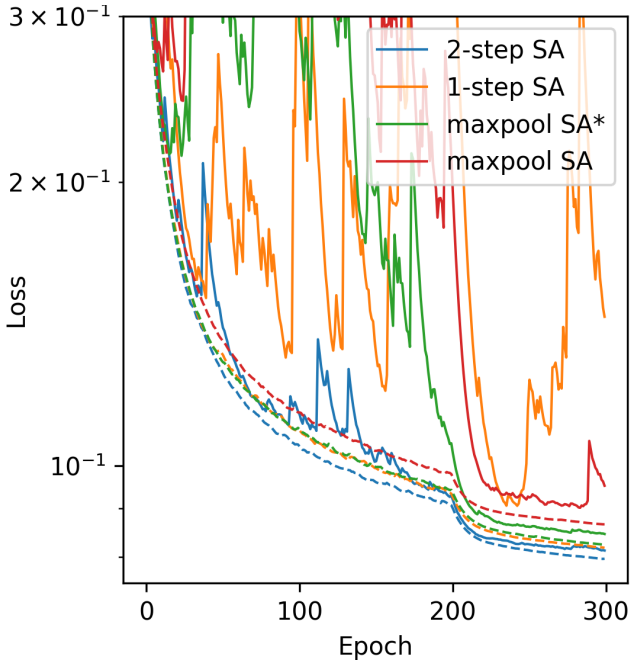


Figure 1: **Loss Curves of Set Abstraction Modules:** Validation and training (dashed lines) loss during 300 epochs of training. The models were trained on $\mathcal{D}_A$ in the sparse setting. *Maxpool SA** denotes the compensated maxpool version. Note the logarithmic scaling of the y-axis.

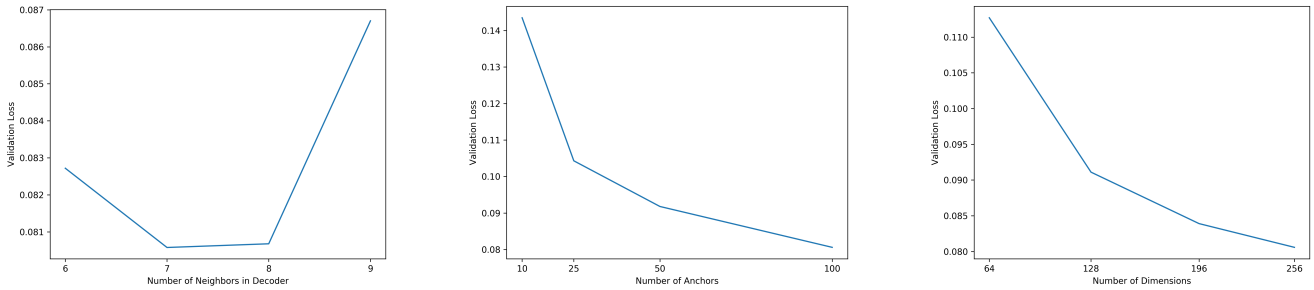
and training loss during the first 300 epochs, indicating that the proposed method performs best. While the maxpooling-based alternatives exhibit a less smooth behavior before the learning rate is reduced at epoch 200, the single-step version still evolves unreliable afterwards. We believe that the reason is the strong dependence on the selection of the *central point*, as explained in the main paper. Table 3 presents quantitative results, showing that the proposed method consistently performs best, at the cost of higher memory demands. In the dense setting the single-step version outperforms both maxpooling alternatives.

	Model	IoU↑	L_1-CD ↓	NC↑	F-Score↑
Sparse Input	maxpool	0.888	0.0037	0.940	0.952
	maxpool*	0.893	0.0036	0.943	0.955
	singleStep	0.891	0.0036	0.942	0.954
	ours	**0.896**	**0.0035**	**0.946**	**0.957**
Dense Input	maxpool	0.907	0.0040	0.946	0.965
	maxpool*	0.905	0.0041	0.944	0.961
	singleStep	0.914	0.0034	0.949	0.969
	ours	**0.919**	**0.0034**	**0.951**	**0.972**

Table 3: **Set Abstraction Ablation:** Quantitative results of different set abstraction modules after 300 epochs of training on $\mathcal{D}_A$.

5.2. Decoder

In the main paper, we ablated our proposed decoder to two alternatives, described in more detail below. As a reminder, we propose to use a VCA module to extract information from the local latent vectors, guided by the global latent description. The extracted information is fed into a FFN, which predicts an occupancy probability. Our experiments showed the great importance of using a computationally expressive decoder, capable of thoroughly understanding its surrounding local latent vectors in the context of the global latent description. Previously existing decoders are either grid-based as in [9, 4] (and therefore not applicable in our scenario) or are based on simplistic interpolation mechanisms. The latter alternatives, which are unable to decoded complex scenarios or fine grained details, are explained below.



(a) k_{dec}, Neighborhood Size of Decoder (b) M, Number of Local Latent Vectors (c) d, Internal Dimensionality

Figure 2: **Hyperparameter Ablations:** Models were trained for 300 epochs on $\mathcal{D}_A$ in the sparse setting. The plots show the minimum validation error achieved for a given parameter setting.

Feature Interpolation. The first decoder alternative was proposed as a baseline in [9], which interpolates between the local latent features. Similar to the proposed decoder, this method then feeds the interpolated feature vector into a FFN to predict an occupancy value. The interpolation is realized using Gaussian kernel regression on point coordinates, see the supplementary of [9] for more details.

Implicit Function Interpolation. Instead of interpolating between local feature vectors, [5] and [12] calculate the implicit function's values at each local latent description, before interpolating the function values. In the following we give a brief description of our implementation.

Let $\mathbf{z}_i$ be the i-th local latent vector, $\mathbf{a}_i$ its spatial anchor and p the queried 3D coordinate. We calculate the occupancy values

$$o_i = g_\psi([p - \mathbf{a}_i, \mathbf{z}_i])$$

for each local latent vector separately. The centered queried coordinate and feature vector are concatenated and fed into the same kind of FFN as before. Note that additionally learning an orientation for each local latent vector and rotating $p - \mathbf{a}_i$ accordingly resulted in slightly worse performance.

Finally, in order to weight the individual occupancies $o_1, \dots o_M$ we predict for each anchor three parameters σ_i of an axis aligned 3D Gaussian $\mathcal{N}(\mathbf{0}, \text{diag}(\sigma_i))$. Again learning a rotation of the covariance matrix resulted in slightly worse results. The final occupancy prediction

$$o = \sum_{i=1}^{M} f\left(p - \mathbf{a}_i \mid \mathbf{0}, \text{diag}(\sigma_i)\right) \cdot o_i$$

is the weighted sum of individual occupancies, where f is the probability density function of the Gaussian.

5.3. Hyperparameters

While there are certainly much more hyperparameters and other design choices to be optimized in the AIR-Net framework, we restrict our ablation to the k_{dec}, the neighborhood size of the decoder, M, the number of local latent vectors and d, the internal dimensionality of all linear layers and attention mechanisms. Figure 2 shows how the minimum validation error after 300 epochs on $\mathcal{D}_A$ varies with respect to these hyperparameters in the sparse setting. Please note that more rigorous ablations are left for future work, *e.g.* increasing d with layer depth (contrary to a constant choice as done here).

Interestingly, figure 2a shows a non-monotonous relation of k_{dec} with model performance. We believe that choosing k_{dec} too small for a certain choice of M supplies the decoder with too little context information. On the other hand choosing k_{dec} too big weakens the inductive bias, that shapes can be described locally (*i.e.* it becomes harder to extract the relevant information). The effect of M and d is more straight forward. In both cases increasing the model complexity results in better performance and choosing an extremely low capacity results in a disproportionate drop in performance. Please note the different scale of the y-axes of the three plots, *e.g.* choosing $k_{\text{dec}} = 6$ and $d = 196$ only results in a small performance decline, while setting $M = 50$ already has a bigger impact. While the vanilla version of AIR-Nets is quite memory intensive the framework has many opportunities to tune the model to the given desires. For example setting $d = 196$ results in a 20% decrease in memory demand or setting $M = 50$ and $k_{\text{dec}} = 6$ results in a 44% decline in memory consumption.

6. Additional Experimental Results

Figure 3 shows additional qualitative results of the zero-shot generalization to the FAUST dataset, figure 4 shows reconstructions of the sparse and dense single view setting, and figure 5 and 6 show additional qualitative results for the sparse and dense setting on $\mathcal{D}_A$ respectively. Similarly, figure 7 and 8 present additional reconstructions on $\mathcal{D}_B$. Furthermore, we provide rotating renderings of the main exper-

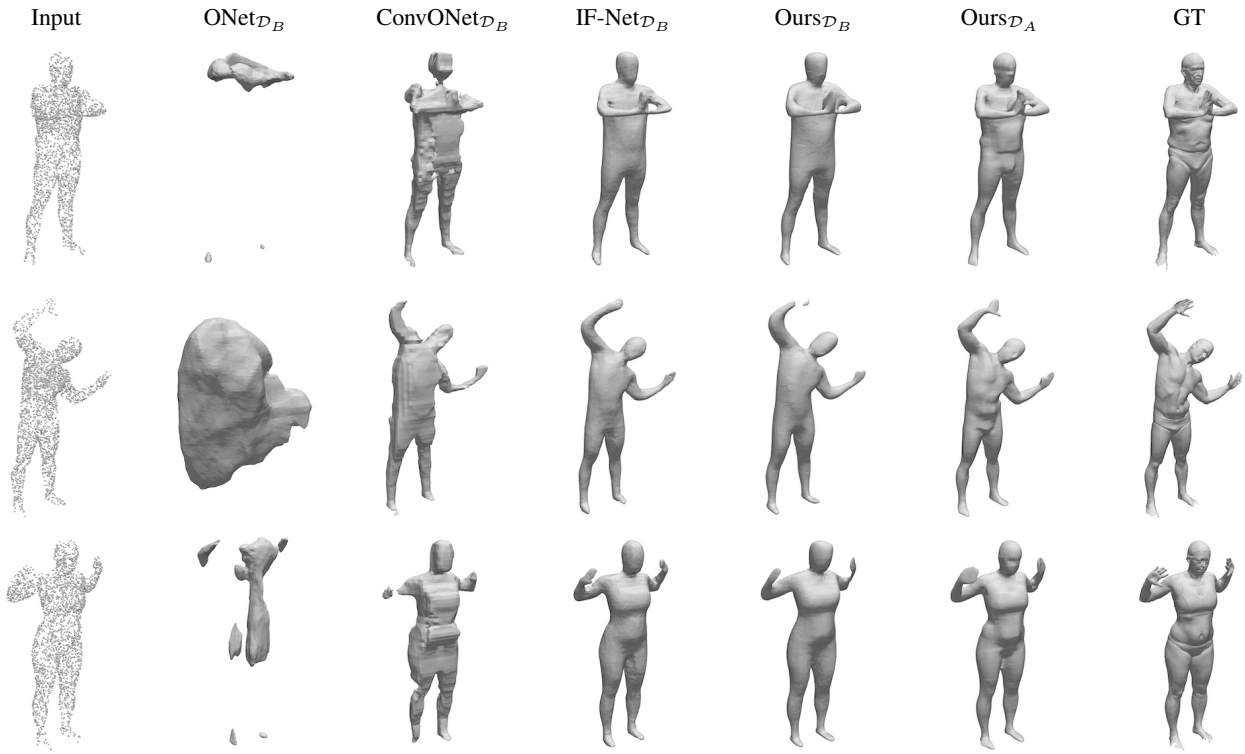


Figure 3: **Zero-Shot 3D Reconstructions on FAUST.** We test the zero-shot generalization ability of models, that were trained in the dense setting on the ShapeNet dataset. Subscripts indicate which dataset the models were trained on. We chose the best performing dataset for each model and included both datasets for our model. Note that $\mathcal{D}_A$ does not contain any noise, while the FAUST data contains real measurement noise and holes, *e.g.* see the feet of the ground trouth.

iments as videos and as GIFs in GitHub.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in the SFB Transregio 161 'Quantitative Methods for Visual Computing' (Project-ID 251654672) and the Cluster of Excellence 'Centre for the Advanced Study of Collective Behaviour'.

References

[1] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE. 2

[2] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. https://arxiv.org/abs/2104.13478, 2021. 2

[3] 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. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 2

[4] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020. 1, 2, 4

[5] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4857–4866, 2020. 5

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017. 2

[7] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2

[8] Jeong Joon Park, Peter Florence, Julian Straub, Richard

Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. 2

[9] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, Cham, Aug. 2020. Springer International Publishing. 1, 4, 5

[10] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. 1

[11] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds, 2019. 2

[12] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Carsten Stoll, and Christian Theobalt. Patchnets: Patch-based generalizable deep implicit 3d shape representations. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 293–309, Cham, 2020. Springer International Publishing. 5

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3

[14] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[15] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2

[16] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. https://arxiv.org/abs/2012.09164, 2020. 2

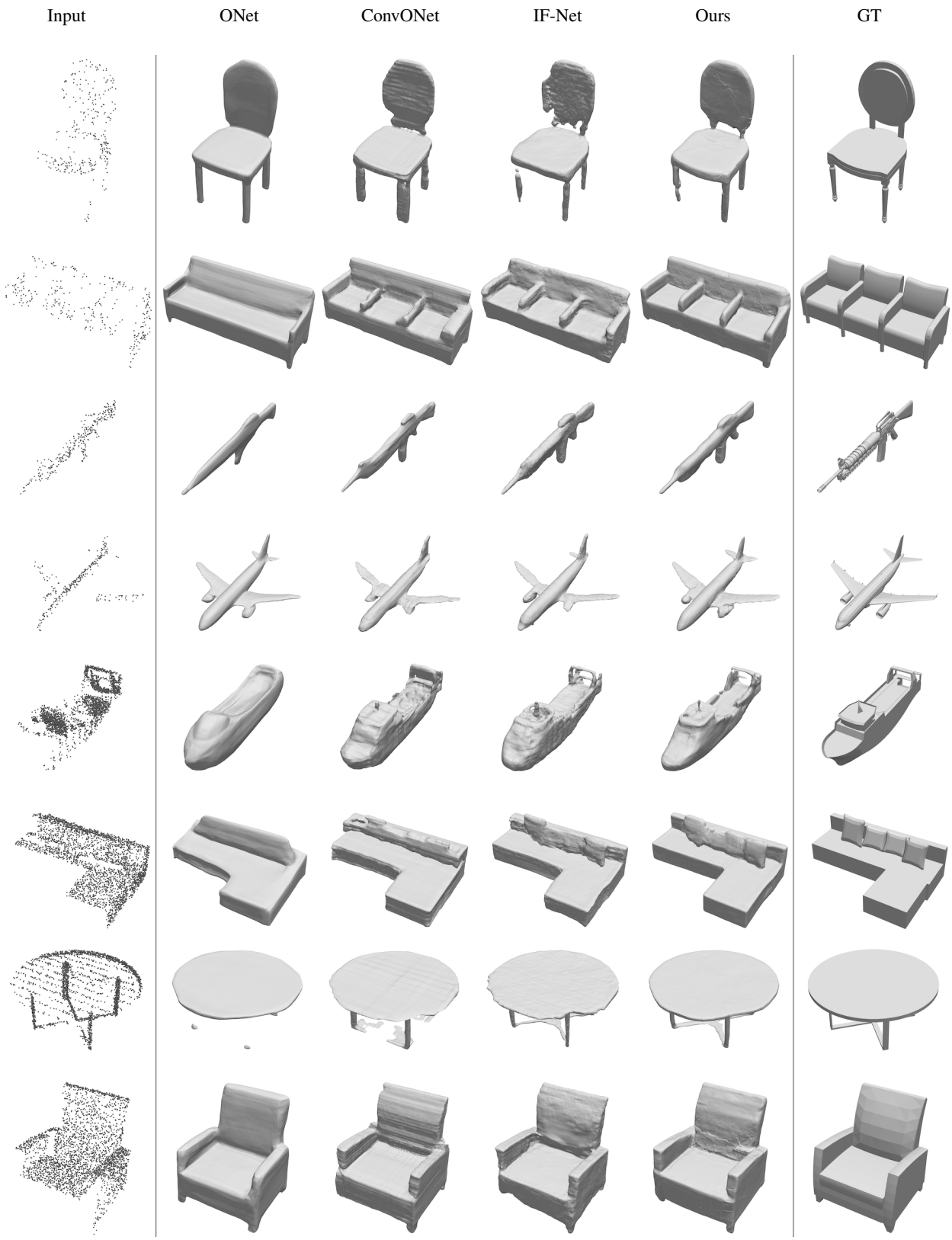


Figure 4: **Single View Reconstructions:** The top half shows the sparse setting, the bottom half the dense setting.

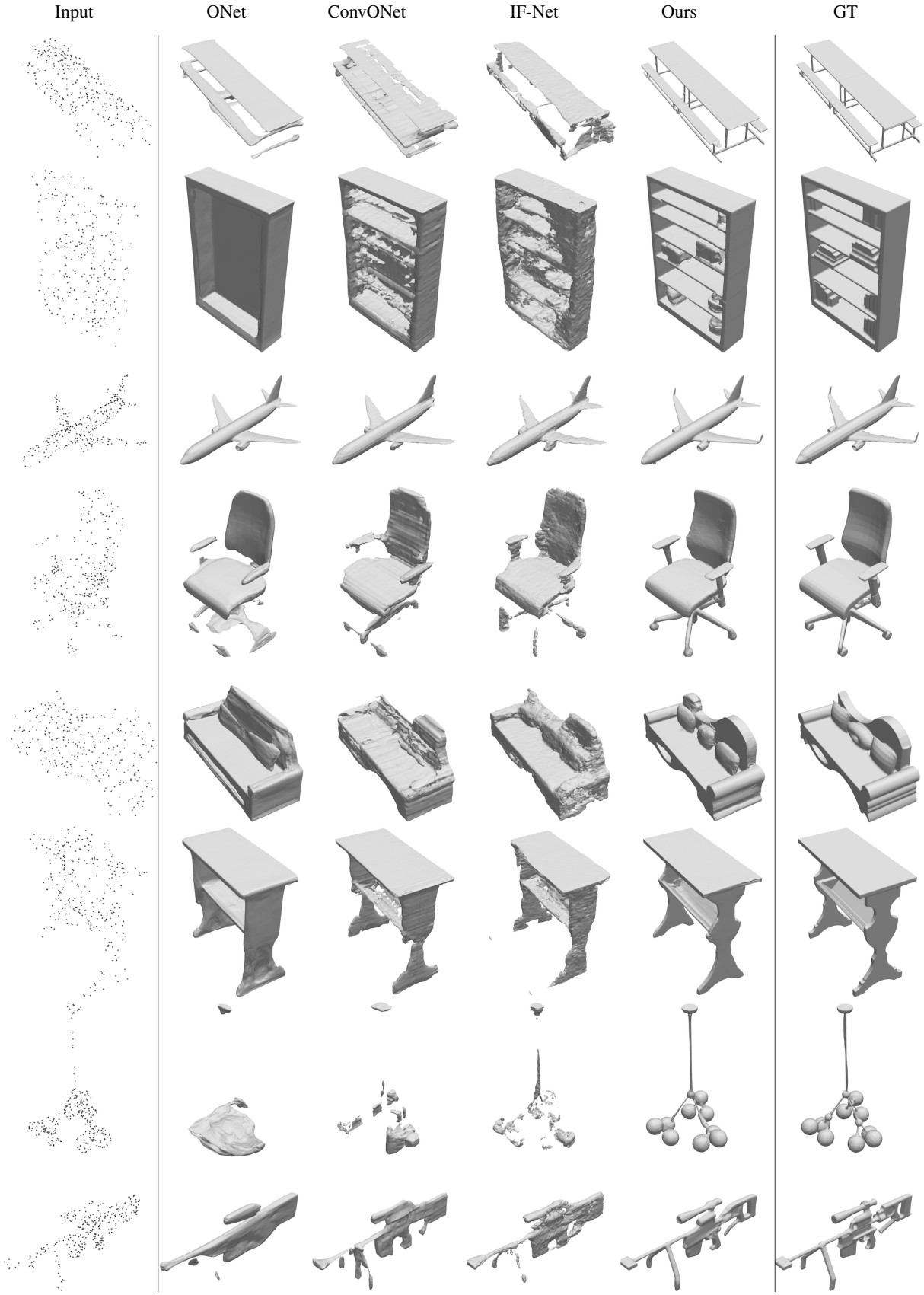


Figure 5: 3D Reconstructions on $\mathcal{D}_A$ in the sparse setting.

Figure 6: 3D Reconstructions on $\mathcal{D}_A$ in the dense setting.

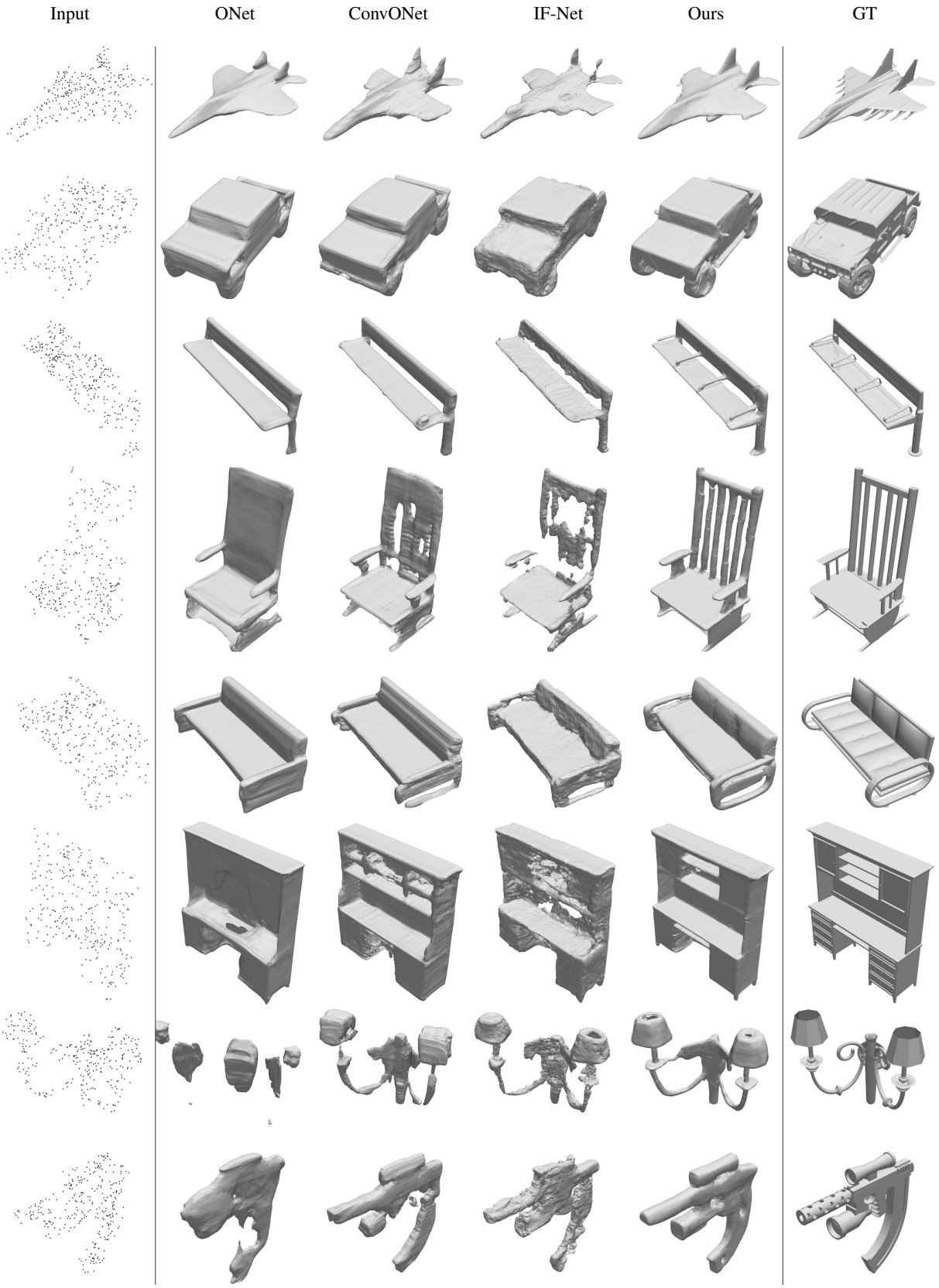


Figure 7: 3D Reconstructions on $\mathcal{D}_B$ in the sparse setting.

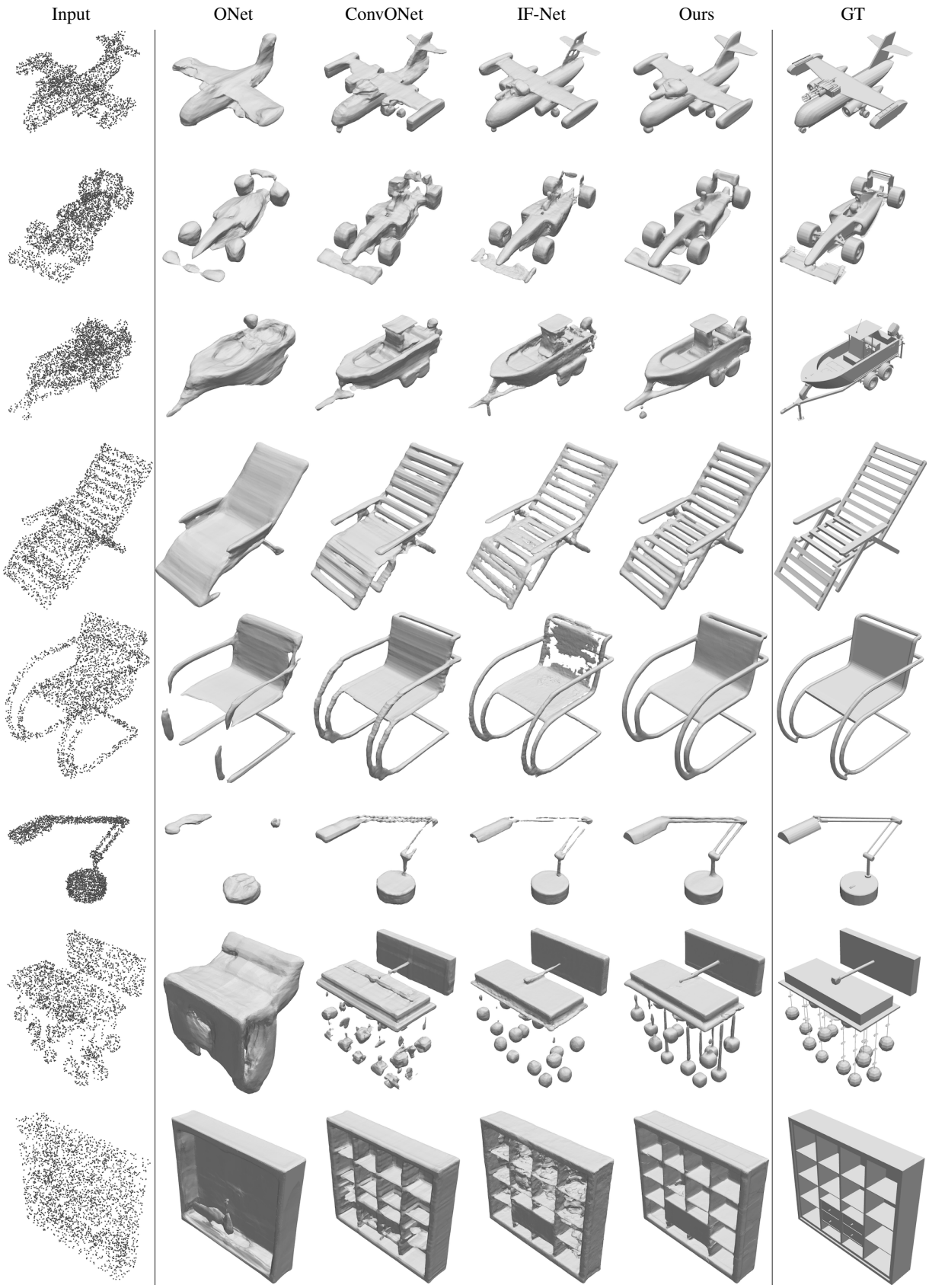


Figure 8: 3D Reconstructions on $\mathcal{D}_B$ in the dense setting.