Supplementary Material for Occupancy Networks: Learning 3D Reconstruction in Function Space

Lars Mescheder[1] Michael Oechsle[1,2] Michael Niemeyer[1] Sebastian Nowozin[3†] Andreas Geiger[1]
[1]Autonomous Vision Group, MPI for Intelligent Systems and University of Tübingen
[2]ETAS GmbH, Stuttgart
[3]Google AI Berlin
{firstname.lastname}@tue.mpg.de nowozin@gmail.com

Abstract

*In this **supplementary document**, we first give a detailed overview of our architectures and training procedure in Section 1. We then discuss our implementation of the baselines in Section 2 and compare them to the implementation in the original publications. Finally, we provide additional experimental results, both qualitatively and quantitatively in Section 3. The **supplementary video** shows 3D animations of the output of our method for the conditional tasks as well as latent space interpolations for our generative model.*

1. Implementation Details

In this section, we first give a detailed description of our architectures. We then describe all our data preprocessing steps and provide details on the metrics we use both for training and testing. Finally, we give details on our training protocol and inference procedure.

1.1. Architectures

We employ the same occupancy network architecture (Fig. 1) in all experiments. Our model takes the output $c \in \mathbb{R}^C$ of a task-specific encoder network and a batch of T 3D-coordinates as input. The points are passed through a fully-connected layer to produce a 256-dimensional feature vector for each point. This feature vector is then passed through 5 pre-activation ResNet-blocks: Each ResNet-block first applies Conditional Batch-Normalization (CBN) [4, 6] to the current feature vector followed by a ReLU activation function. The output is then fed into a fully-connected layer, a second CBN layer, a ReLU activation and another fully-connected layer. The output of this operation is then added to the input of the ResNet-block.

The output is finally passed through a last CBN layer and ReLU activation followed by a fully-connected layer that projects the features down to one dimension. To obtain the occupancy probability, this vector can simply be passed through a sigmoid activation.

The CBN layers are implemented in the following way: First, we pass the conditional encoding c which we obtained from the encoder network through two fully-connected layers to obtain 256-dimensional vectors $\beta(c)$ and $\gamma(c)$. We then normalize the 256-dimensional input feature vector f_{in} using first and second order moments, multiply the output with $\gamma(c)$ and add the bias term $\beta(c)$:

$$f_{out} = \gamma(c)\frac{f_{in} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta(c), \tag{1}$$

where μ and σ are the empirical mean and standard-deviation (over the batch- and T-dimensions) of the input features f_{in} and $\epsilon = 10^{-5}$ (the default value in PyTorch[1]). Moreover, we compute a running mean over μ and σ^2 with momentum 0.1 during training. At test time, we replace μ and σ^2 with the corresponding running mean.

[†]Part of this work was done while at MSR Cambridge.
[1]https://pytorch.org

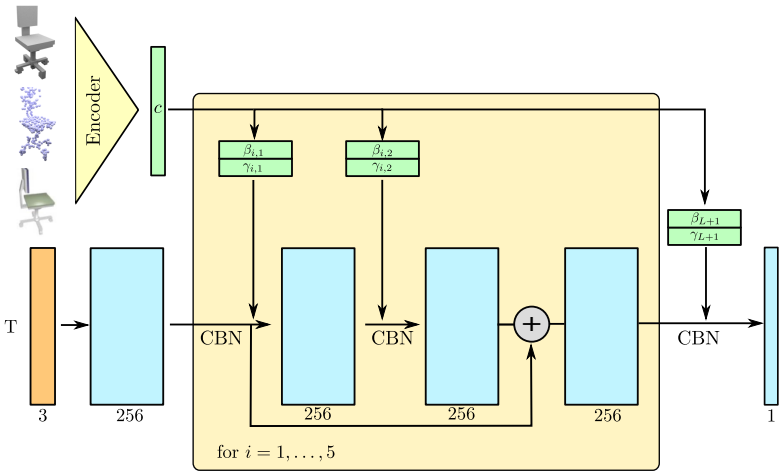


Figure 1: **Occupancy Network Architecture.** We first compute an embedding c of the input. We then feed the input points through multiple fully-connected ResNet-blocks. In these ResNet-blocks, we use Conditional Batch-Normalization (CBN) to condition the network on c. Finally, we project the output of our network to one dimension using a fully-connected layer and apply the sigmoid function to obtain occupancy probabilities.

The respective encoder network depends on the task (Fig. 2): For single view 3D reconstruction we use a ResNet-18 architecture [10] (Fig. 2a) which was pretrained on the ImageNet dataset [5]. However, we adjust the last fully-connected layer to project the features to a 256-dimensional embedding c. For point cloud completion we use a PointNet encoder [7] (Fig. 2b) with 5 ResNet-blocks. To enable communication between points at lower layers, we also add pooling and expansion layers between the ResNet-blocks. After the ResNet-blocks, the final output is pooled using max-pooling and then projected to a 512 embedding vector using a fully-connected layer. For voxel super-resolution we use a 3D convolutional neural network (Fig. 2c) that encodes the 32^3 input into a 256-dimensional embedding vector c. The encoder network g_ψ for unconditional mesh generation is similar to the model shown in Fig. 2b, but we simply use four fully-connected blocks instead of the five ResNet blocks. Moreover, we replace the last fully-connected layer with two fully-connected layers to produce both the mean μ_ψ and log-standard-deviation $\log \sigma_\psi$ of the 128 dimensional latent code z.

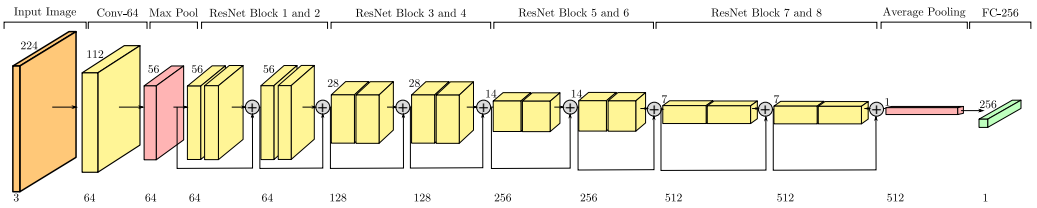
1.2. Data Preprocessing

For our experiments we use the ShapeNet [2] subset of Choy et al. [3]. We also use the same 32^3 voxelization, image renderings and train/test split (80% and 20% of the whole dataset) as Choy et al. Moreover, we subdivide the training data into a training (70% of the whole dataset) and a validation set (10% of the whole dataset) on which we track the loss of our method and the baselines to determine when to stop training.

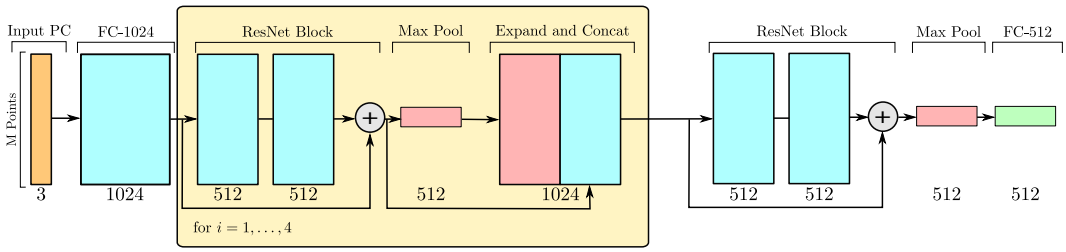
In order to determine if a point lies in the interior of a mesh (e.g., for measuring IoU), we need the meshes to be watertight. We therefore use the code provided by Stutz et al. [18][2], which performs TSDF-fusion on random depth renderings of the object, to create watertight versions of the meshes. We center and rescale all meshes so that they are aligned with the voxelizations from [3]. In practice, this means that we transform the meshes so that the 3D bounding box of the mesh is centered at 0 and its longest edge has a length of 1. We then sample 100k points offline in the unit cube centered at 0 with an additional small padding of 0.05 on both sides and determine if the points lie inside or outside the watertight mesh. To this end, we count the number of triangles that a ray which starts at the given point and which is parallel to the z-axis intersects. If this number is even the point lies outside the mesh, otherwise it lies inside. We save both the positions of the 100k points and their occupancies to a file. During training, we subsample 2048 points from this set (with replacement) as training data. Similarly, we also sample 100k points from the surface of the object and save them to a file. When we train the Point Set Generation Network (PSGN) [7], Pixel2Mesh [21] or Deep Marching Cubes (DMC) [14] we subsample a certain number of points (the exact number depends on the method) from this set of points. We accelerated data preprocessing using the GNU parallel tool [20].

Because Choy et al. [3] only provide 32^3 voxelizations in their dataset, we compute the voxelizations for the experiment in

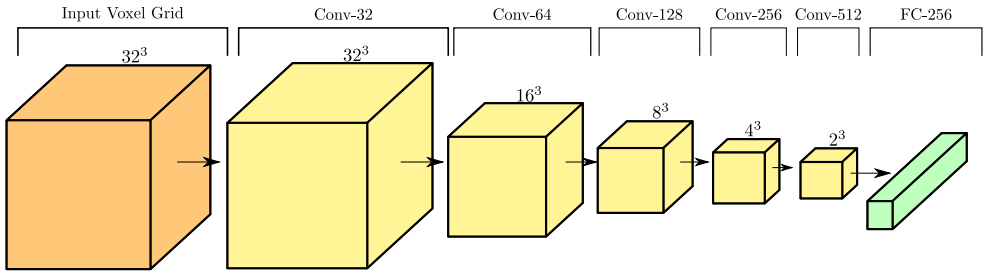
[2] https://github.com/davidstutz/mesh-fusion



(a) Single Image 3D Reconstruction.



(b) Point Cloud Completion.



(c) Voxel Super-Resolution.

Figure 2: **Encoder Architectures.** Depending on the task, we leverage different encoder architectures: (a) For image input, we use a ResNet-18 architecture [10], which was pretrained on ImageNet. (b) For point cloud input, we use a PointNet encoder [16] with additional pooling and expansion layers. (c) For voxel input, we use a 3D convolutional neural network.

Section 4.1 of the main paper ourselves. To this end, we first detect all voxels that intersect the surface of the mesh and mark them as occupied. We then test for a random point in the interior of each voxel if it lies inside or outside the mesh. We mark the corresponding voxel as occupied if the first case is true. This procedure marks all voxels as occupied which intersect the mesh (or its interior).

1.3. Metrics

As described in the main text, we evaluate our method and baselines using the volumetric Intersection over Union (IoU), the Chamfer-L_1 distance and a normal consistency score.

In the following, let $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$ be the set of all points that are inside or on the predicted and ground truth mesh, respectively. The volumetric IoU is defined as the quotient of the volume of the two meshes' intersection and the volume of their union:

$$\text{IoU}(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) \equiv \frac{|\mathcal{M}_{\text{pred}} \cap \mathcal{M}_{\text{GT}}|}{|\mathcal{M}_{\text{pred}} \cup \mathcal{M}_{\text{GT}}|}. \tag{2}$$

We obtain unbiased estimates of these volumes by randomly sampling 100k points from the bounding volume and determining if the points lie inside or outside $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$, respectively.

We define the Chamfer-L_1 distance between the two meshes as

$$\text{Chamfer-}L_1(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) \equiv \frac{1}{2\,|\partial\mathcal{M}_{\text{pred}}|} \int_{\partial\mathcal{M}_{\text{pred}}} \min_{q \in \partial\mathcal{M}_{\text{GT}}} \|p - q\| \mathrm{d}p + \frac{1}{2\,|\partial\mathcal{M}_{\text{GT}}|} \int_{\partial\mathcal{M}_{\text{GT}}} \min_{p \in \partial\mathcal{M}_{\text{pred}}} \|p - q\| \mathrm{d}q, \tag{3}$$

where $\partial\mathcal{M}_{\text{pred}}$ and $\partial\mathcal{M}_{\text{GT}}$ denote the surfaces of the two meshes. Moreover, we define an accuracy and completeness score of $\mathcal{M}_{\text{pred}}$ wrt. $\mathcal{M}_{\text{GT}}$:

$$\text{Accuracy}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}) \equiv \frac{1}{|\partial\mathcal{M}_{\text{pred}}|}\int_{\partial\mathcal{M}_{\text{pred}}} \min_{q\in\partial\mathcal{M}_{\text{GT}}} \|p-q\|\mathrm{d}p \tag{4}$$

$$\text{Completeness}(\mathcal{M}_{\text{pred}}|\mathcal{M}_{\text{GT}}) \equiv \frac{1}{|\partial\mathcal{M}_{\text{GT}}|}\int_{\partial\mathcal{M}_{\text{GT}}} \min_{p\in\partial\mathcal{M}_{\text{pred}}} \|p-q\|\mathrm{d}q \tag{5}$$

Note that this definition implies that *lower* accuracy and completeness scores are *better*. Moreover, note that the Chamfer-L_1 distance is just the mean of the accuracy and the completeness score.

Similarly, we define the normal consistency score as

$$\text{Normal-Consistency}(\mathcal{M}_{\text{pred}},\mathcal{M}_{\text{GT}}) \equiv \frac{1}{2\,|\partial\mathcal{M}_{\text{pred}}|}\int_{\partial\mathcal{M}_{\text{pred}}} |\langle n(p), n(\text{proj}_2(p))\rangle|\mathrm{d}p + \frac{1}{2\,|\partial\mathcal{M}_{\text{GT}}|}\int_{\partial\mathcal{M}_{\text{GT}}} |\langle n(\text{proj}_1(q)), n(q)\rangle|\mathrm{d}q \tag{6}$$

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

We estimate all four quantities efficiently by sampling 100k points from the surface of both meshes and employing a KD-tree to determine the corresponding nearest neighbors from the other mesh.

1.4. Training

In all experiments, we use the Adam optimizer [13] with a learning rate of $\eta = 10^{-4}$ and no weight decay. For other hyperparameters of Adam we use PyTorch defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. For image input, we augment the input by scaling it with a random factor between 0.75 and 1 and taking a random crop.

During training, we track all losses and other quantities of interest on the validation set to determine when to stop training. For our method and 3D-R2N2, we use the volumetric IoU to the ground truth mesh as a stopping criterion. For PSGN and Pixel2Mesh we use the Chamfer-L_2 distance to the ground truth mesh. While these quantities can be computed in a fast way for the corresponding methods, for DMC it is not straightforward to compute the volumetric IoU and the Chamfer-distance on the validation set. Therefore, we use the validation loss to determine when to stop training.

1.5. Inference

We introduce *Multiresolution IsoSurface Extraction (MISE)* as a method to extract high resolution meshes from the output of a trained occupancy network for new observations. In this supplementary material, we restrict the discussion to only relevant additional information. We refer to the main publication for an in-depth description of the approach.

We chose a voxel grid of size 32^3 as initial resolution where we evaluate the output of the occupancy network. Next, we subdivide voxels which are at current mesh intersection positions into 8 subvoxels and perform another evaluation of the newly introduced grid points. In practice, we found that repeating this process two times is enough for our purposes. Finally, we extract the approximate isosurface by using the Marching Cubes algorithm [15].

As indicated in the main publication, we further refine the extracted mesh by first simplifying it to 5,000 faces using [8] and next, minimizing

$$\sum_{k=1}^{K}(f_\theta(p_k,x)-\tau)^2 + \lambda\left\|\frac{\nabla_p f_\theta(p_k,x)}{\|\nabla_p f_\theta(p_k,x)\|} - n(p_k)\right\|^2 \tag{7}$$

for points p_k sampled on each face of the mesh. In practice, we set $\lambda = 0.01$. We perform 30 optimization steps using RMSprop with a learning rate of $\eta = 10^{-4}$. Note that gradient based refinement is only possible because we learn a continuous representation of the occupancy function which we can differentiate.

2. Baselines

In order to conduct controlled experiments and to disentangle the individual components of the different 3D reconstruction approaches, we created a PyTorch package comprising our method and several baselines [3, 7, 14, 21]. To make sure that the performance of our reimplementations of the baselines matches the performance of the original implementations, we conducted thorough comparisons to the original models.

2.1. 3D-R2N2

The 3D Recurrent Reconstruction Neural Network (3D-R2N2) [3] is a method to map one or multiple images of an object to its 3D shape. The images are encoded as a low-dimensional vector and subsequently used as input to a 3D-LSTM network [11] to combine features from multiple views. Finally, the decoder uses the LSTM network's hidden states to produce a voxel representation of the object.

In our experiments, we used a ResNet-18 as encoder and a decoder consisting of one fully connected and four transposed convolutional layers with LeakyReLU activation. As we did not use multiple input views for 3D reconstruction, we did not utilize the proposed 3D-LSTM module. A quantitative comparison of the authors' implementation and our version can be found in Table 1.

	3D-R2N2 (authors)			3D-R2N2 (resnet18)
	1 view	3 views	5 views	1 view
category				
airplane	0.513	0.549	0.561	**0.606**
bench	0.421	0.502	0.527	**0.562**
cabinet	0.716	0.763	**0.772**	0.764
car	0.798	0.829	0.836	**0.846**
chair	0.466	0.533	**0.550**	0.532
display	0.468	0.545	**0.565**	0.528
lamp	0.381	0.415	**0.421**	0.389
loudspeaker	0.662	0.708	**0.717**	0.685
rifle	0.544	0.593	**0.600**	0.595
sofa	0.628	0.690	**0.706**	**0.706**
table	0.513	0.564	**0.580**	0.571
telephone	0.661	0.732	**0.754**	0.740
vessel	0.513	0.596	**0.610**	0.606
mean	0.560	0.617	**0.631**	0.625

Table 1: **3D-R2N2 Reimplementation.** This table shows the IoU reported in the original 3D-R2N2 paper and our reimplementation wrt. the *voxelized* ground truth mesh. In contrast to other IoU values reported in this paper, in this table we compare with the 32^3 voxelization of the ground truth meshes and not the high resolution meshes themselves.

2.2. Point Set Generation Networks

Point Set Generation Network [7] is a point-based 3D object reconstruction model that takes a single image as input and outputs 1024 3D point coordinates.

The comparison of results from our architecture for PSGN and the deterministic two-branch version proposed in the original publication can be found in Table 2. The former uses a ResNet-18 as an encoder and four fully connected layers with a hidden dimension of 512 and ReLU activation as a decoder. The latter consists of 7 blocks of multiple convolutional layers as an encoder, and four blocks of transposed and ordinary convolutional layers as a decoder where connections to the respective mirrored encoder blocks are introduced similar to U-Net [17]. We refer to the original publication [7] for more details. We trained the models 100 hours on a Nvidia GTX 1080 GPU with a batch size of 64.

As indicated in the table, the ResNet-18 based version achieves slightly better accuracy, completeness and Chamfer-L_1 scores. We suspect that the higher scores are caused by using a more powerful ResNet-18 as the encoder.

2.3. Pixel2Mesh

Pixel2Mesh [21] is a mesh-based approach to reconstruct 3D shapes from single images. The initial start mesh of an ellipsoid is progressively deformed to match the object's shape by extracting perceptual features from the input image. The model consists of three feature pooling and mesh deformation blocks, each containing 12 (first two) or 13 (latter) graph convolution layers with ResNet-inspired [10] skip connections and ReLU activations. In the latter two blocks, the perceptual and hidden features from the last graph convolution layer are concatenated before an unpooling operation is applied. This way, the number of vertices is increased proportionally to the number of edges in the mesh. The final outputs of the network are locations for 156, 628, and 2466 vertices, each output corresponding to one block.

In our experiments, we adhered to the implementation provided in the official GitHub repository[3] which differs slightly from details given in the paper [21]. We compare the results from the authors' pretrained model on data they provided as well as our implementation on our ground truth data in different settings. We found that while our ground truth point clouds are a fixed number of uniformly sampled points from the CAD models, the Pixel2Mesh authors performed Poisson-disk sampling which results in different numbers of points for each CAD model. In addition, they multiplied the vertex coordinates by a factor of 0.57 and inverted the y and z axes. To provide a fair comparison, we conducted our experiments using a template

[3] Source: https://github.com/nywang16/Pixel2Mesh

category	Accuracy			Completeness			Chamfer-L_1		
	PSGN (2 branch)	PSGN (resnet18)	ONet	PSGN (2 branch)	PSGN (resnet18)	ONet	PSGN (2 branch)	PSGN (resnet18)	ONet
airplane	0.113	**0.101**	0.133	0.191	0.173	**0.161**	0.152	**0.137**	0.147
bench	0.161	**0.133**	0.154	0.262	0.230	**0.156**	0.212	0.181	**0.155**
cabinet	0.154	**0.138**	0.150	0.307	0.291	**0.184**	0.231	0.215	**0.167**
car	0.126	0.117	**0.116**	0.235	0.221	**0.203**	0.181	0.169	**0.159**
chair	0.233	**0.187**	0.223	0.333	0.307	**0.233**	0.283	0.247	**0.228**
display	0.258	**0.235**	0.281	0.351	0.333	**0.275**	0.304	0.284	**0.278**
lamp	0.301	**0.275**	0.402	0.372	**0.354**	0.557	0.337	**0.314**	0.479
loudspeaker	0.253	**0.245**	0.285	0.403	0.386	**0.315**	0.328	0.316	**0.300**
rifle	0.113	**0.110**	0.148	0.159	0.158	**0.134**	0.136	**0.134**	0.141
sofa	0.206	**0.171**	0.194	0.307	0.278	**0.195**	0.257	0.224	**0.194**
table	0.165	**0.152**	0.189	0.307	0.293	**0.189**	0.236	0.222	**0.189**
telephone	0.140	**0.122**	0.149	0.219	0.201	**0.130**	0.179	0.161	**0.140**
vessel	0.186	**0.153**	0.197	0.249	**0.223**	0.238	0.217	**0.188**	0.218
mean	0.185	**0.164**	0.202	0.284	0.265	**0.228**	0.235	**0.215**	**0.215**

Table 2: **PSGN Reimplementation.** Comparison of our architecture for PSGN and the two branch architecture proposed in the original publication.

category	IoU			Chamfer-L_1			Normal Consistency		
	Pix2Mesh (authors)	Pix2Mesh (reimpl.)	ONet	Pix2Mesh (authors)	Pix2Mesh (reimpl.)	ONet	Pix2Mesh (authors)	Pix2Mesh (reimpl.)	ONet
airplane	0.400	0.420	**0.571**	0.191	0.187	**0.147**	0.775	0.759	**0.840**
bench	0.272	0.323	**0.485**	0.198	0.201	**0.155**	0.736	0.732	**0.813**
cabinet	0.609	0.664	**0.733**	0.228	0.196	**0.167**	0.828	0.834	**0.879**
car	0.544	0.552	**0.737**	0.190	0.180	**0.159**	0.762	0.756	**0.852**
chair	0.371	0.396	**0.501**	0.270	0.265	**0.228**	0.745	0.746	**0.823**
display	0.440	**0.490**	0.471	0.245	**0.239**	0.278	0.840	0.830	**0.854**
lamp	0.304	0.323	**0.371**	0.315	**0.308**	0.479	0.688	0.666	**0.731**
loudspeaker	0.563	0.599	**0.647**	0.306	**0.285**	0.300	0.796	0.782	**0.832**
rifle	0.341	0.402	**0.474**	0.167	0.164	**0.141**	0.715	0.718	**0.766**
sofa	0.544	0.613	**0.680**	0.233	0.212	**0.194**	0.806	0.820	**0.863**
table	0.323	0.395	**0.506**	0.229	0.218	**0.189**	0.785	0.784	**0.858**
telephone	0.625	0.661	**0.720**	0.156	0.149	**0.140**	0.912	0.907	**0.935**
vessel	0.357	0.397	**0.530**	0.221	**0.212**	0.218	0.716	0.699	**0.794**
mean	0.438	0.480	**0.571**	0.227	0.216	**0.215**	0.777	0.772	**0.834**

Table 3: **Pixel2Mesh Reimplementation.** Comparison of our implementation of Pixel2Mesh, the pretrained Pixel2Mesh network provided by the authors and our method. While both methods achieve a similar Chamfer-L_1 distance, our method still performs significantly better regarding IoU and normal consistency. Note that our method does not directly optimize the Chamfer-distance during training whereas Pixel2Mesh does.

ellipsoid adjusted to our data by inverting the transformation and used a high number of target points (8,000). We trained our model for several days on a Nvidia Titan X GPU with batch size 12.

The quantitative results can be seen in Table 3. We observe that we can almost completely reproduce the results from the pretrained model with our reimplementation. In addition, we found that qualitative results and failure cases are very similar as well. For the car class, we noted that while the car meshes of our implementation sometimes have cavities on the roof, the ones from the pretrained model show the same artifacts on the bottom. In our experience, the results depend on a carefully selected ellipsoid alignment as well as number of ground truth points. In contrast to Pixel2Mesh, the occupancy network is not directly trained on Chamfer loss and, surprisingly, yet achieves almost the same Chamfer-L_1 distance. In addition, our proposed method outperforms the Pixel2Mesh method significantly in both, IOU and normal consistency.

2.4. AtlasNet

We evaluate AtlasNet [9] using the code and pretrained model with 25 parameterizations provided by the authors[4]. As AtlasNet uses a slightly different test/train split as our code[5], we test AtlasNet on the intersection of the test splits of [9] and [3]. Because the data was normalized differently in [9] than in our framework, we use the ground truth point clouds provided by [9] to renormalize the output of [9] to the unit cube.

[4] https://github.com/ThibaultGROUEIX/AtlasNet

[5] We found that almost all samples in the test split from [9] are also in the test split from [3], but not the other way around.

	IoU				Chamfer-L_1				Normal Consistency			
category	3D-R2N2	PSGN	DMC	ONet	3D-R2N2	PSGN	DMC	ONet	3D-R2N2	PSGN	DMC	ONet
airplane	0.464	-	0.570	**0.760**	0.145	0.135	0.106	**0.056**	0.668	-	0.814	**0.897**
bench	0.441	-	0.536	**0.716**	0.145	0.181	0.102	**0.059**	0.695	-	0.808	**0.878**
cabinet	0.718	-	0.802	**0.867**	0.167	0.238	0.114	**0.073**	0.790	-	0.887	**0.916**
car	0.681	-	0.731	**0.835**	0.197	0.191	0.166	**0.098**	0.719	-	0.825	**0.875**
chair	0.523	-	0.642	**0.736**	0.181	0.236	0.121	**0.089**	0.676	-	0.842	**0.890**
display	0.603	-	0.753	**0.817**	0.167	0.228	0.096	**0.076**	0.755	-	0.902	**0.926**
lamp	0.356	-	0.484	**0.566**	0.231	0.247	0.170	**0.135**	0.601	-	0.777	**0.813**
loudspeaker	0.705	-	0.789	**0.828**	0.200	0.278	0.151	**0.116**	0.741	-	0.882	**0.898**
rifle	0.419	-	0.580	**0.694**	0.145	0.115	0.083	**0.060**	0.707	-	0.780	**0.864**
sofa	0.704	-	0.800	**0.872**	0.158	0.211	0.103	**0.069**	0.760	-	0.886	**0.928**
table	0.483	-	0.624	**0.759**	0.172	0.236	0.103	**0.071**	0.742	-	0.874	**0.917**
telephone	0.699	-	0.848	**0.915**	0.128	0.165	0.062	**0.041**	0.851	-	0.949	**0.970**
vessel	0.554	-	0.603	**0.748**	0.163	0.165	0.150	**0.085**	0.647	-	0.798	**0.859**
mean	0.565	-	0.674	**0.778**	0.169	0.202	0.117	**0.079**	0.719	-	0.848	**0.895**

Table 4: **Point Cloud Completion.** This table shows a per-category numerical comparison of our approach and the baselines for Point Cloud Completion on the ShapeNet dataset. We measure the IoU, Chamfer-L_1 distance and Normal Consistency for various methods wrt. the ground truth mesh. Note that in contrast to prior work, we compute the IoU wrt. the high-resolution mesh and not a coarse voxel representation. Due to missing connectivity information, IoU and the normal consistency score can not be calculated for PSGN.

2.5. Deep Marching Cubes

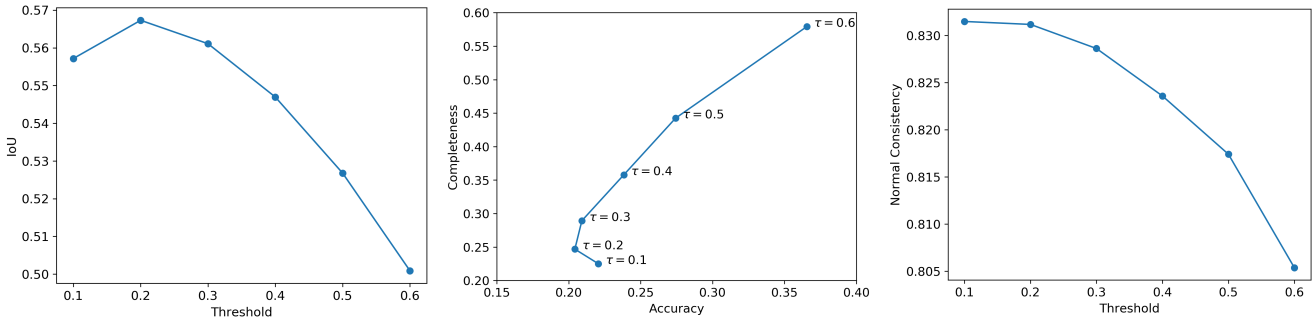
We apply Deep Marching Cubes (DMC) [14] as a baseline for mesh-based point cloud completion. Given a point cloud, the method predicts mesh objects in an explicit representation consisting of occupancy probabilities and vertex displacement variables on a 3D grid. The first part of the pipeline distills point cloud information into a 3D grid by extracting point features and performing 3D grid pooling. Then, a U-Net-based [17] architecture with 3D convolutional layers parameterizes the mapping between the extracted grid features and the predicted mesh representation. The full architecture is trained in an end-to-end fashion using four different losses. The first loss penalizes the mean Euclidean distance between each point of the ground truth point cloud and its closest mesh face. A second loss (the weight prior) encourages the occupancy probabilities at the boundary of the scene to be small. The absolute difference between a pair of adjacent occupancy probabilities is penalized to enforce smoothness. Furthermore, an additional loss preserves smoothness between adjacent faces.

In the original implementation these loss functions are implemented as a CFFI Cuda extension for PyTorch since these loss functions are computationally very expensive. We adapted the original implementation in PyTorch $0.3.0$ to our framework in PyTorch 1.0 without algorithmic changes. Unfortunately, the CFFI Cuda extensions in [14] are not compatible with PyTorch versions $> 0.3.0$, so that we transferred the original extensions to C++ / Cuda extensions for PyTorch 1.0.

To ensure a fair comparison, we use point clouds consisting of 300 instead of 3,000 points as input, in contrast to the original implementation. However, during training, we still apply the point-to-mesh loss to the full point cloud with 3,000 points following [14]. We observed that the default weighting of the weight prior 10 as chosen in the original implementation causes artifacts in form of rectangular shapes for flat 3D objects, thus we reduce the weight to 5.

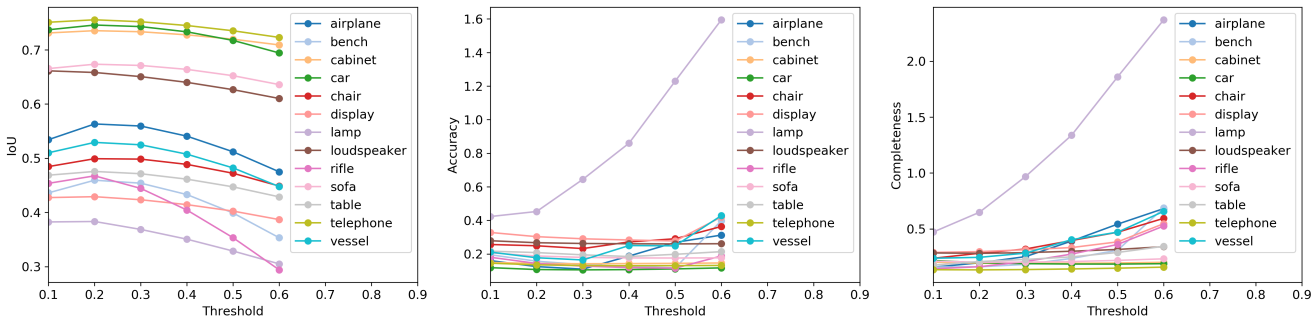
3. Additional Experimental Results

In this section, we present additional quantitative and qualitative experimental results. We first analyze the effect of the threshold parameter τ. We then discuss some additional qualitative results for the single image 3D reconstruction task. Here, we also discuss some failure cases of our method. Next, we provide some additional quantitative and qualitative results for point cloud completion and voxel super-resolution. Finally, we show some qualitative results for unconditional mesh generation and discuss the effects of the different stages of the Multiresolution IsoSurface Extraction (MISE) algorithm.



(a) IOU (b) ROC-curve (c) Normal consistency

Figure 3: **Threshold Parameter.** Effect of threshold parameter on IoU, accuracy, completeness and normal consistency for the single-view 3D reconstruction experiment.



(a) IoU (b) Accuracy (c) Completeness

Figure 4: **Threshold Parameter per Category.** Effect of threshold parameter on IoU, accuracy, completeness and normal consistency for the single-view 3D reconstruction experiment for individual classes.

3.1. Effect of the Threshold Parameter

To understand the effect of the threshold parameter τ, we examine its effect on various metrics on a validation set. To this end, we vary τ from $\tau = 0.1$ to $\tau = 0.6$ and use MISE to extract the corresponding meshes. For each value of τ, we measure the volumetric IoU, the completeness, the accuracy and the normal consistency score.

The results are shown in Figure 3. We observe that IoU improves as we decrease τ until we reach the critical level of $\tau \approx 0.2$. For smaller τ, the IoU becomes worse again. As expected, we observe that the completeness score becomes better with smaller values of τ. Surprisingly, we find that the accuracy also improves with smaller τ until it reaches its optimum at $\tau \approx 0.2$. This is counterintuitive, because a small value of τ could a priori lead to spurious geometries which should deteriorate the accuracy score. We explain this counterintuitive observation by the fact that both the accuracy score and the completeness score are better when the gap between the two meshes is smaller. Although we might add some spurious geometry for some objects for small τ, the meshes are closer to each other which leads to better accuracy and completeness scores. This effect dominates the accuracy score until we reach $\tau \approx 0.2$. For smaller τ the accuracy score becomes worse again because spurious geometry appears.

For class specific results see Figure 4. We observe that some categories are more difficult to learn. In particular the "lamp" category is very challenging for our method due to very fine geometry.

All in all, we see that the selection of the threshold parameter is a critical component of our method and a threshold parameter of $\tau \approx 0.2$ yields the best performance.

3.2. Single Image 3D Reconstruction

Additional qualitative results for the single image 3D reconstruction task are shown in Fig. 5 and Fig. 6. While all methods are able to reconstruct the 3D geometry, we observe that our method can better capture high frequencies details than other methods.

category	IoU			Chamfer-L_1			Normal Consistency		
	PSGN	ONet	ONet + PSGN	PSGN	ONet	ONet + PSGN	PSGN	ONet	ONet + PSGN
airplane	-	0.571	**0.598**	0.137	0.147	**0.119**	-	0.840	**0.854**
bench	-	**0.485**	0.438	0.181	0.155	**0.152**	-	**0.813**	0.791
cabinet	-	**0.733**	0.711	0.215	0.167	**0.166**	-	**0.879**	0.869
car	-	**0.737**	0.723	0.169	0.159	**0.151**	-	**0.852**	0.838
chair	-	**0.501**	0.498	0.247	0.228	**0.207**	-	**0.823**	0.813
display	-	0.471	**0.507**	0.284	0.278	**0.248**	-	**0.854**	**0.854**
lamp	-	**0.371**	0.364	0.314	0.479	**0.305**	-	0.731	**0.741**
loudspeaker	-	**0.647**	0.637	0.316	0.300	**0.284**	-	**0.832**	0.824
rifle	-	0.474	**0.481**	**0.134**	0.141	0.135	-	0.766	**0.778**
sofa	-	**0.680**	0.673	0.224	0.194	**0.188**	-	**0.863**	0.856
table	-	**0.506**	0.452	0.222	**0.189**	0.192	-	**0.858**	0.835
telephone	-	0.720	**0.733**	0.161	0.140	**0.128**	-	0.935	**0.936**
vessel	-	0.530	**0.544**	**0.188**	0.218	**0.188**	-	**0.794**	0.786
mean	-	**0.571**	0.566	0.215	0.215	**0.190**	-	**0.834**	0.829

Table 5: **ONet + PSGN.** While occupancy networks can be trained ent-to-end, we can also combine them with PSGN.

Some failure cases are shown in Fig. 11. While in general our method performs well, we observe that our method sometimes has problems with very thin object parts. Objects with such parts are especially frequent in the "lamp" category of the ShapeNet dataset. This can be seen in Figure 4: IOU as well as accuracy and completeness are much worse for the "lamp" category than for other categories. Moreover, the "lamp" category is also the category which is most sensitive to the threshold parameter. Note that thin object parts are especially problematic for volumetric approaches to 3D-reconstruction like our approach and 3D-R2N2 [3]. Approaches that take the metric of the 3D-space into account such as PSGN, Pixel2Mesh and AtlasNet are less prone to these kinds of problems. However, these methods either only produce points clouds that have to be meshed in a postprocessing step, can only represent meshes with very simple topology or do not lead to watertight meshes.

In order to take the metric of the 3D space into account for our approach, we can also combine occupancy networks with PSGN. To this end, we use PSGN to reconstruct a pointcloud of 1024 points and use occupancy networks to generate the final mesh. Quantitative results are shown in Table 5. We find that this improves Chamfer-L_1 distance with little degradation of IOU and normal consistency score.

Our experiments suggest that PSGN is good at reconstructing 3D geometry. Unfortunately, however, PSGN lacks connectivity information and hence requires additional postprocessing steps to obtain the final mesh. In our experiments we found that reconstructing the final mesh from the output of PSGN is indeed very challenging. To reconstruct the mesh, we tried both Screened Poisson Surface Reconstruction (PSR) [12] and Ball Pivoting (BP) [1]. Unfortunately, however, we were not able to find hyperparameters for PSR that were able to produce reasonable meshes for all input classes. We believe that this is partly due to the fact that PSGN does not output any normals. As PSR requires normals, we therefore have to estimate the normals in a preprocessing step using a moving window. BP worked better in our experiments and some qualitative results are shown in Fig. 10. While BP succeeds in generating reasonable meshes from the output of PSGN, the meshes are very rough and have a lot of missing parts. In particular, the meshes generated by BP are not watertight.

3.3. Pix3D dataset

We also evaluated the networks which were trained on ShapeNet on the Pix3D dataset [19] with ground truth masks. For all methods, we mask out the background and crop the image so that the object is roughly in the middle of the image. As all methods are only trained on 13 ShapeNet classes, we only use the 4 object classes (chair, desk, sofa, table) from Pix3D that are similar to one of those 13 ShapeNet classes.

Qualitative and quantitative results are shown in Fig. 12 and Table 6, respectively. We find that all methods generalize reasonably well to real data, but achieve worse results than on synthetic data. We also find that ONet trained on more random views (ONet+) generalizes better than ONet trained on the renderings by Choy et al. [3]. This shows that the main limiting factor on this dataset is the realism of the training dataset. We therefore expect that it is possible to further narrow the gap to synthetic data by using even more varied and higher quality renderings as well as better data augmentation strategies.

	IoU					Chamfer-L_1					Normal Consistency				
category	3D-R2N2	PSGN	AtlasNet	ONet	ONet+	3D-R2N2	PSGN	AtlasNet	ONet	ONet+	3D-R2N2	PSGN	AtlasNet	ONet	ONet+
chair	0.146	-	-	0.224	**0.319**	2.572	0.750	**0.392**	0.628	0.531	0.401	-	0.731	0.715	**0.747**
desk	0.170	-	-	0.268	**0.414**	1.776	0.760	**0.509**	1.047	0.568	0.478	-	0.717	0.682	**0.780**
sofa	0.483	-	-	0.580	**0.699**	0.537	0.592	**0.367**	0.427	0.413	0.645	-	0.792	0.817	**0.861**
table	0.137	-	-	0.204	**0.345**	2.665	0.979	0.556	1.071	**0.502**	0.431	-	0.768	0.720	**0.809**
mean	0.234	-	-	0.319	**0.444**	1.888	0.770	**0.456**	0.793	0.504	0.489	-	0.752	0.733	**0.799**

Table 6: **Pix3D dataset.** We evaluate the networks which were trained on ShapeNet on the Pix3D dataset [19] with ground truth masks. ONet+ was trained on our own renderings with more varied random views.

3.4. Inference time

The inference time of our algorithm with simplification and refinement steps is about 3s / mesh. When we remove the simplification and refinement steps, the inference time of our method reduces to 603ms / mesh with little degradation of the results. While our method is slower compared to the baselines (3D-R2N2: 11ms, PSGN: 10ms, Pixel2Mesh: 31ms), we can trade off accuracy and efficiency in our inference procedure by adaptively choosing the resolution: if we extract meshes at 32^3 resolution instead, our inference time reduces to 41ms (32ms w/o marching cubes).

3.5. Other Results

Qualitative results for point cloud completion are shown in Fig. 7 and Fig. 8. We observe that our method is able to reconstruct high frequency details from the sparse input point clouds. The full quantitative results for point cloud completion are shown in Table 4. Qualitative results for the voxel super-resolution task are shown in Fig. 9.

Figure 13, 14, 15 and 16 show interpolations in latent space for our unconditional model. We find that our model learns a meaningful representation of the input data.

Figure 17 shows results from the different stages of the MISE algorithm. We find that even without further refinement our algorithm generates high-quality meshes. Moreover, when we additionally apply simplification and refinement, the remaining discretization artifacts (e.g. on the wings of the airplane in Fig. 17) disappear.

References

[1] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics (VCG)*, 5(4):349–359, 1999. 9, 17

[2] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *arXiv.org*, 1512.03012, 2015. 2

[3] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2016. 2, 5, 6, 9, 12, 13, 14, 15, 19

[4] H. de Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 1

[5] J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009. 2

[6] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2017. 1

[7] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3D object reconstruction from a single image. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 5, 12, 13, 14, 15, 17, 19

[8] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Visualization'98. Proceedings*, pages 263–269. IEEE, 1998. 4

[9] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. AtlasNet: A papier-mâché approach to learning 3d surface generation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 6, 12, 13, 19

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 3, 5

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 5

[12] M. M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Trans. on Graphics (SIGGRAPH)*, 32(3):29, 2013. 9

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2015. 4

[14] Y. Liao, S. Donne, and A. Geiger. Deep marching cubes: Learning explicit surface representations. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 5, 7, 14, 15

[15] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Trans. on Graphics (SIGGRAPH)*, 1987. 4

[16] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[17] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015. 5, 7

[18] D. Stutz and A. Geiger. Learning 3D shape completion from laser scan data with weak supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[19] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 9, 10, 19

[20] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb. 2011. 2

[21] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2018. 2, 5, 12, 13

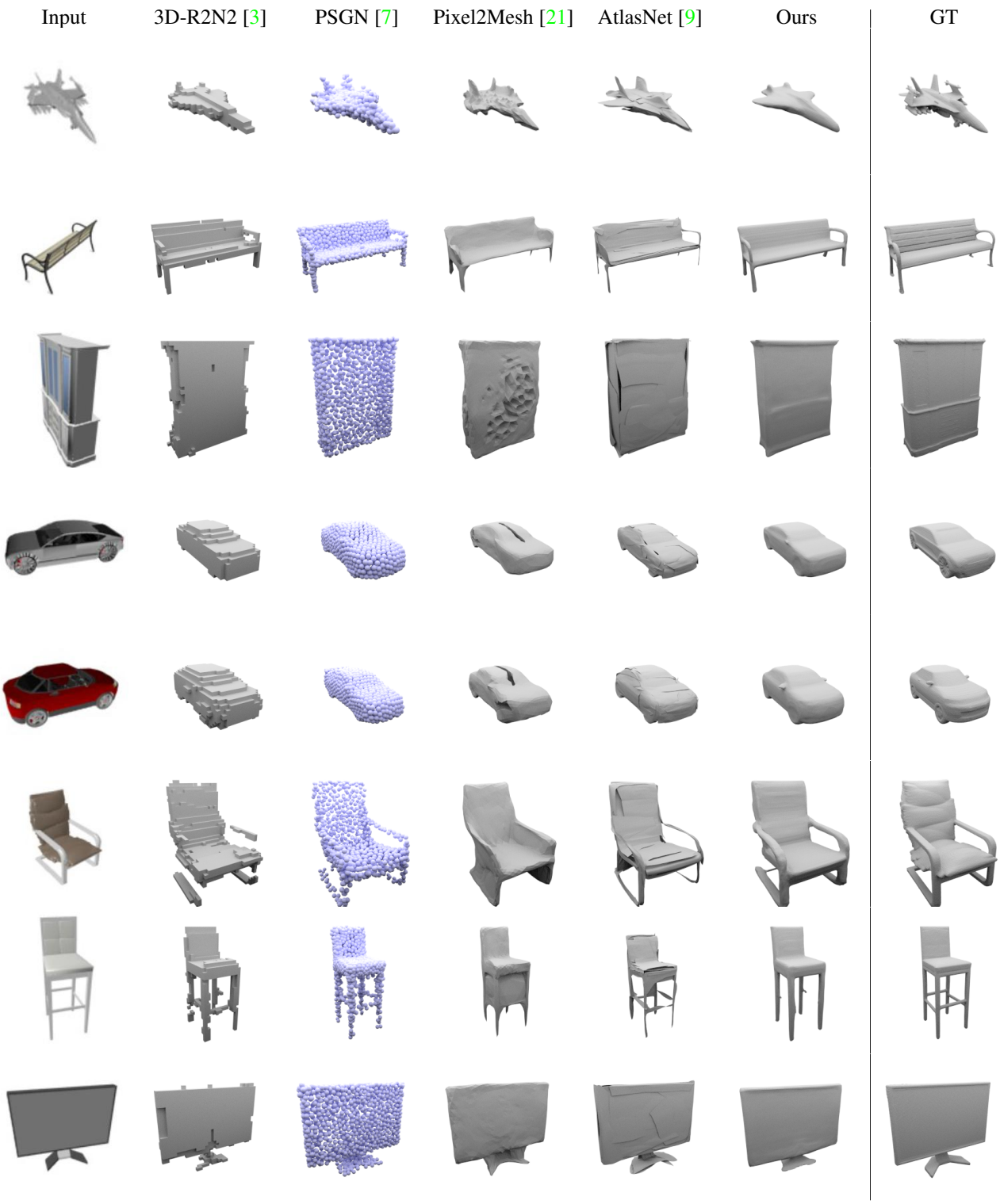


Figure 5: **Single Image 3D Reconstruction.** The input image is shown in the first column, the other columns show the results for our method compared to various baselines.

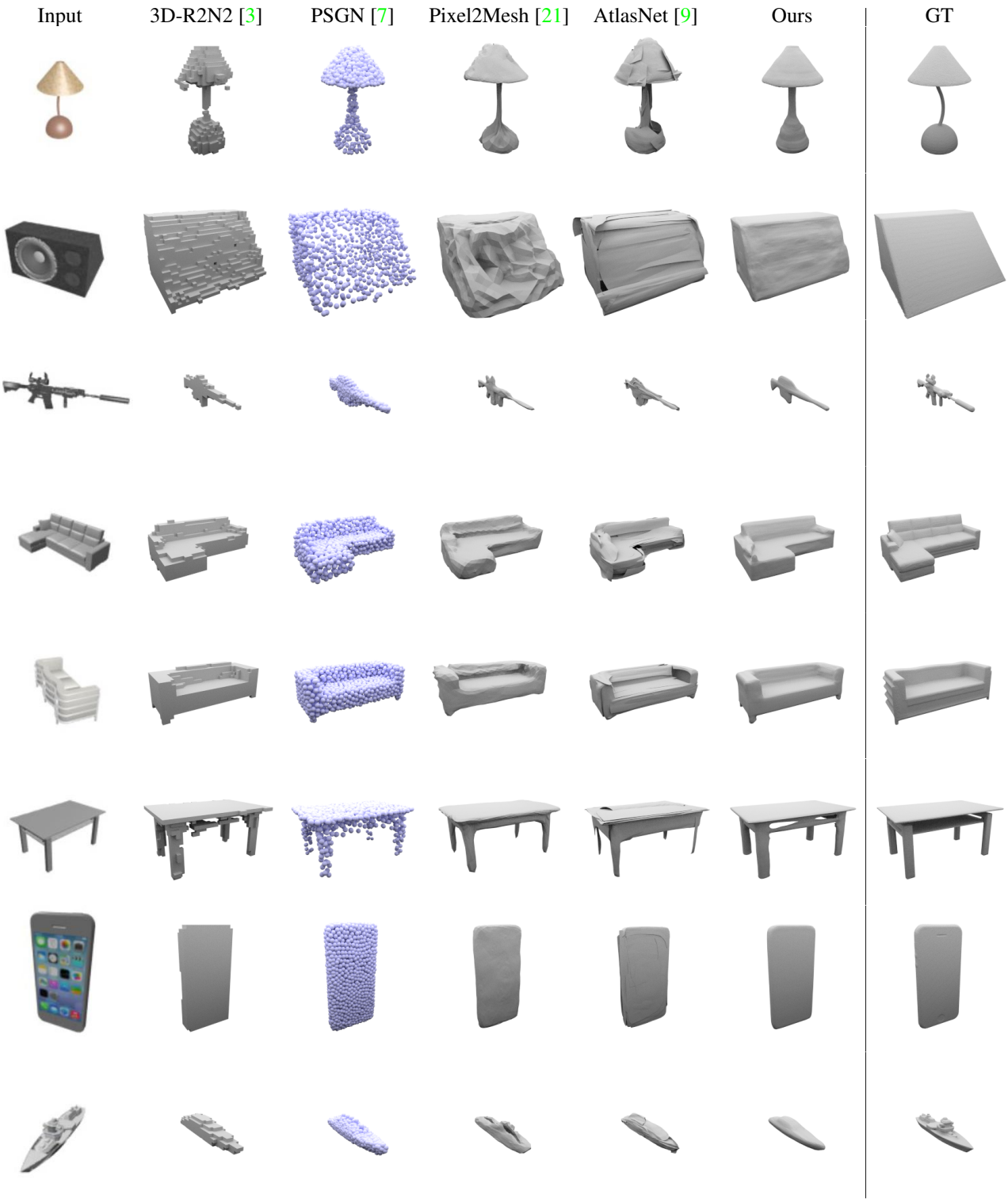


Figure 6: **Single Image 3D Reconstruction.** The input image is shown in the first column, the other columns show the results for our method compared to various baselines.

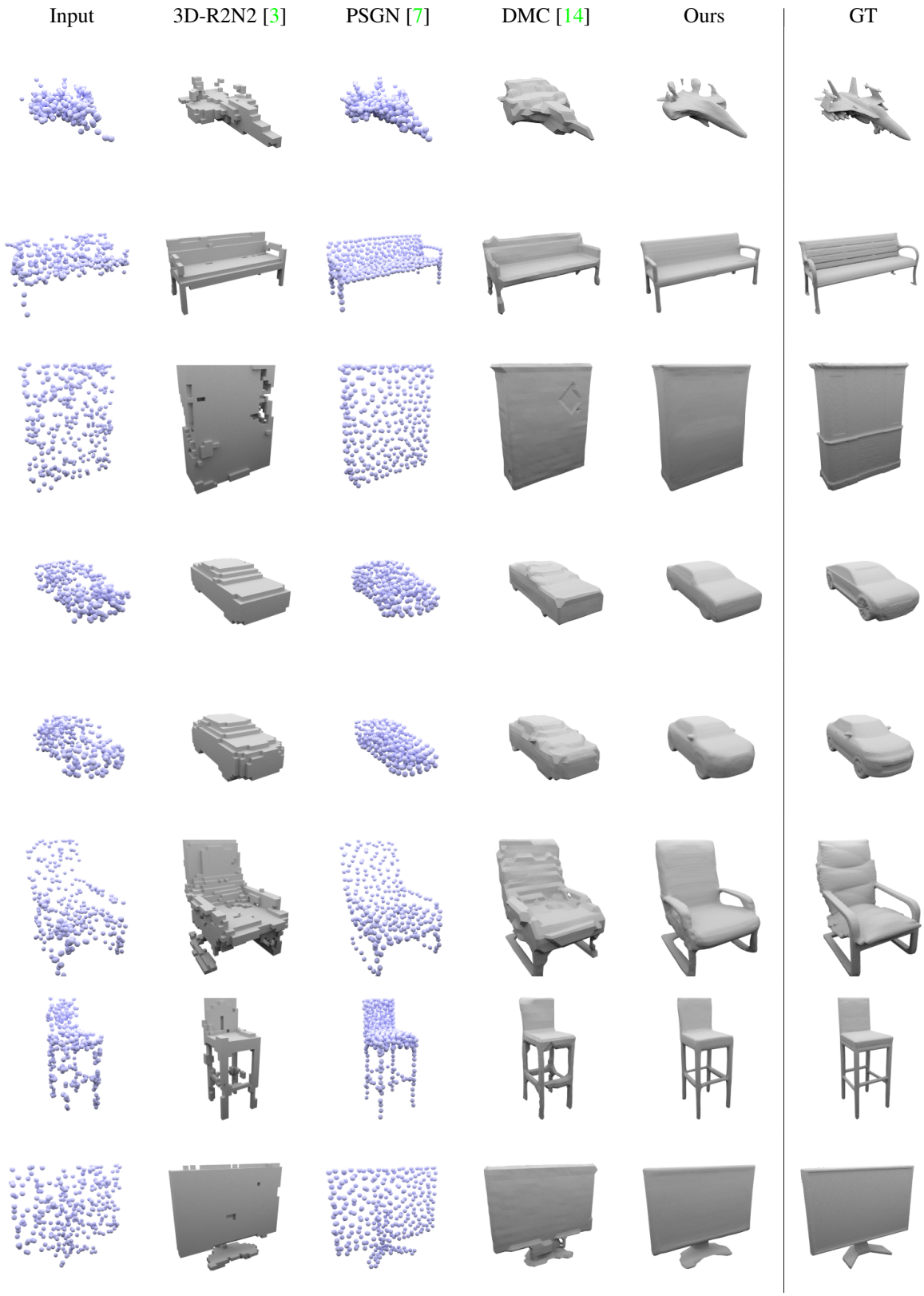


Figure 7: **Point Cloud Completion.** The input point cloud is shown in the first column, the other columns show the results for our method compared to various baselines.

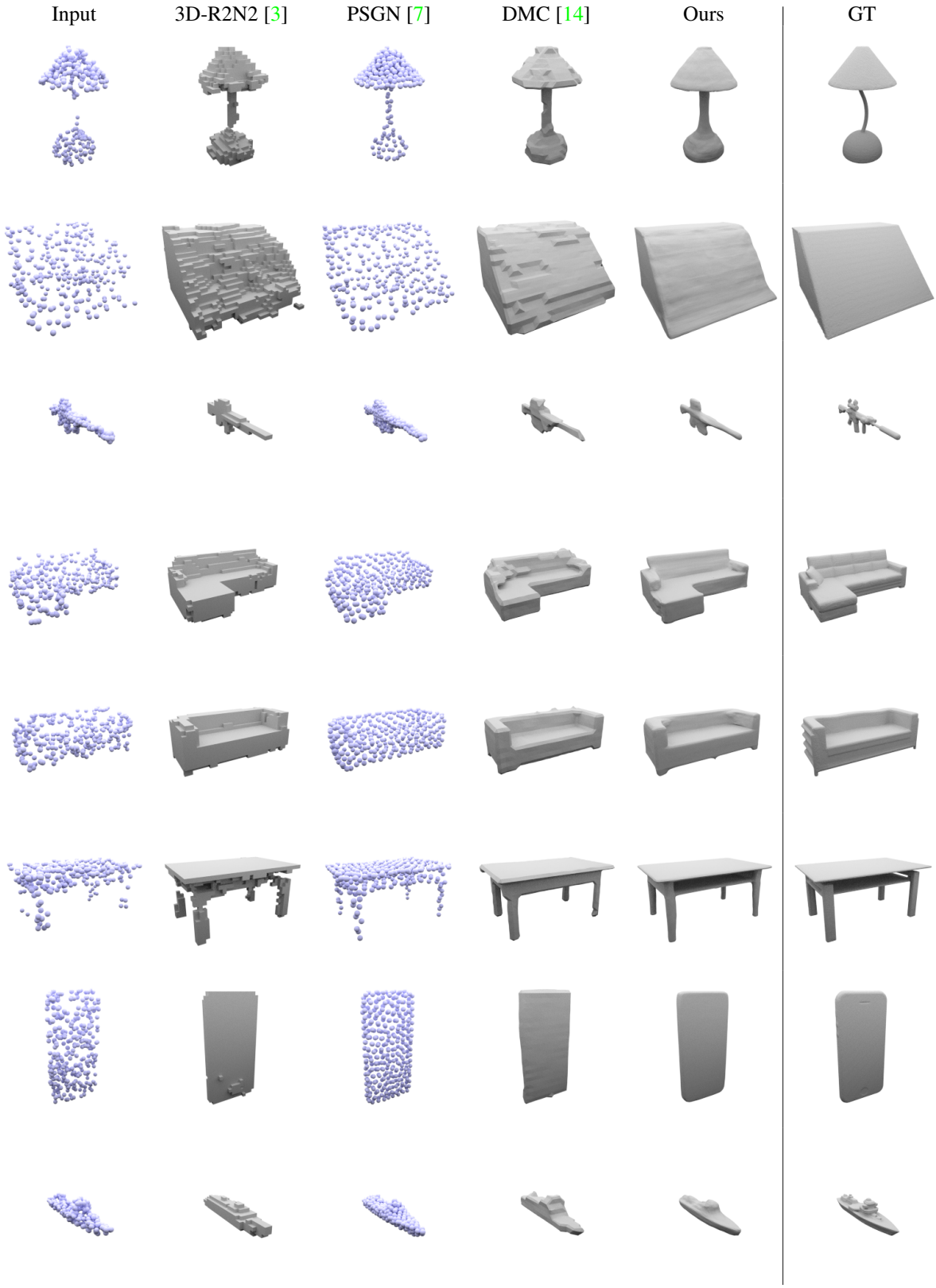


Figure 8: **Point Cloud Completion.** The input point cloud is shown in the first column, the other columns show the results for our method compared to various baselines.

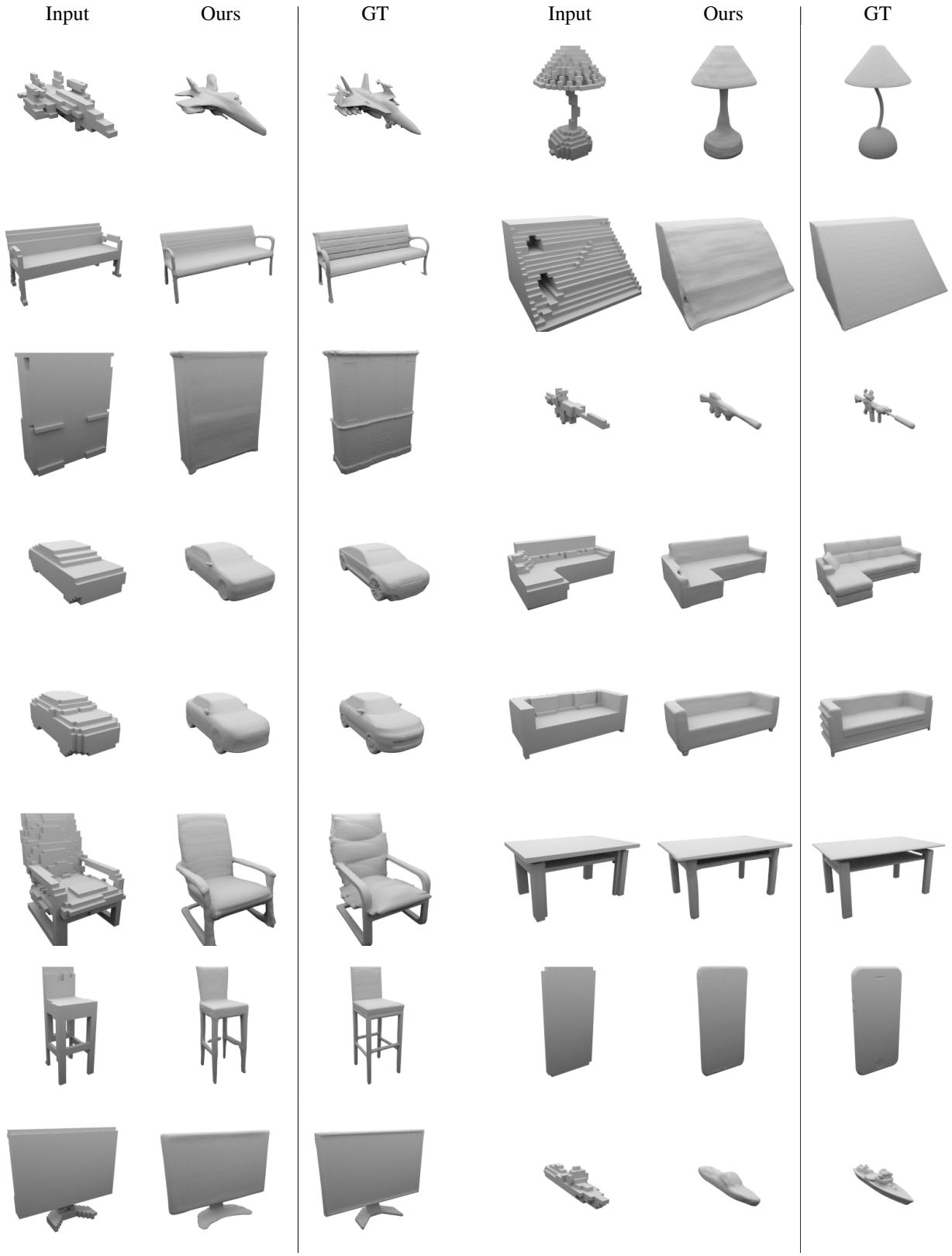


Figure 9: **Voxel Super-Resolution.** The input is shown in the first column, the other columns show the results for our method compared to the ground truth.

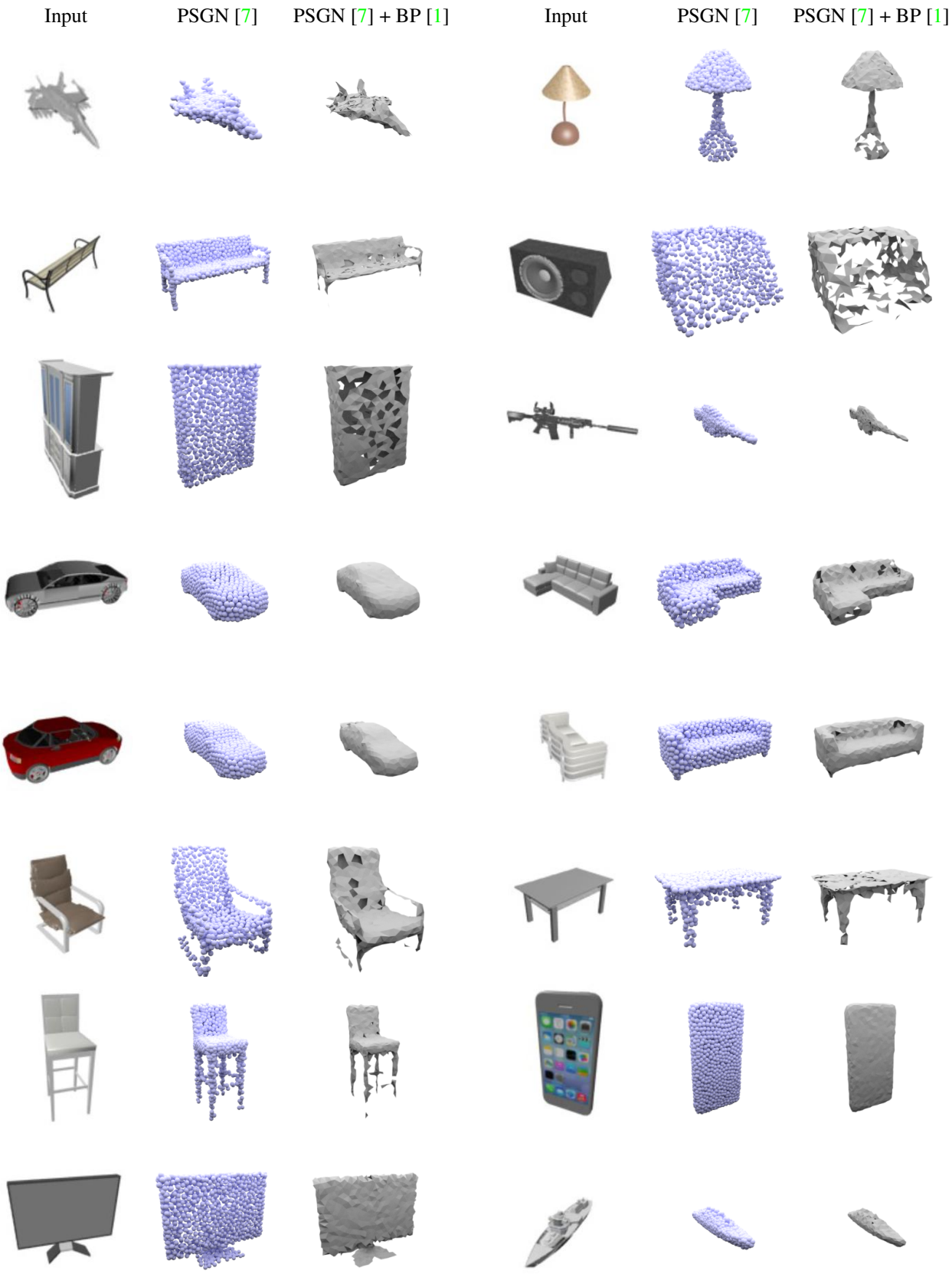


Figure 10: **PSGN Meshing Results.** To obtain a mesh for the output of PSGN [7], we applied the Ball-Pivoting (BP) algorithm [1]. We find that meshing the output of PSGN directly is non-trivial: While the combination of PSGN and BP allows to obtain coarse meshes from single images, the output has many missing parts and the method does not lead to closed meshes.

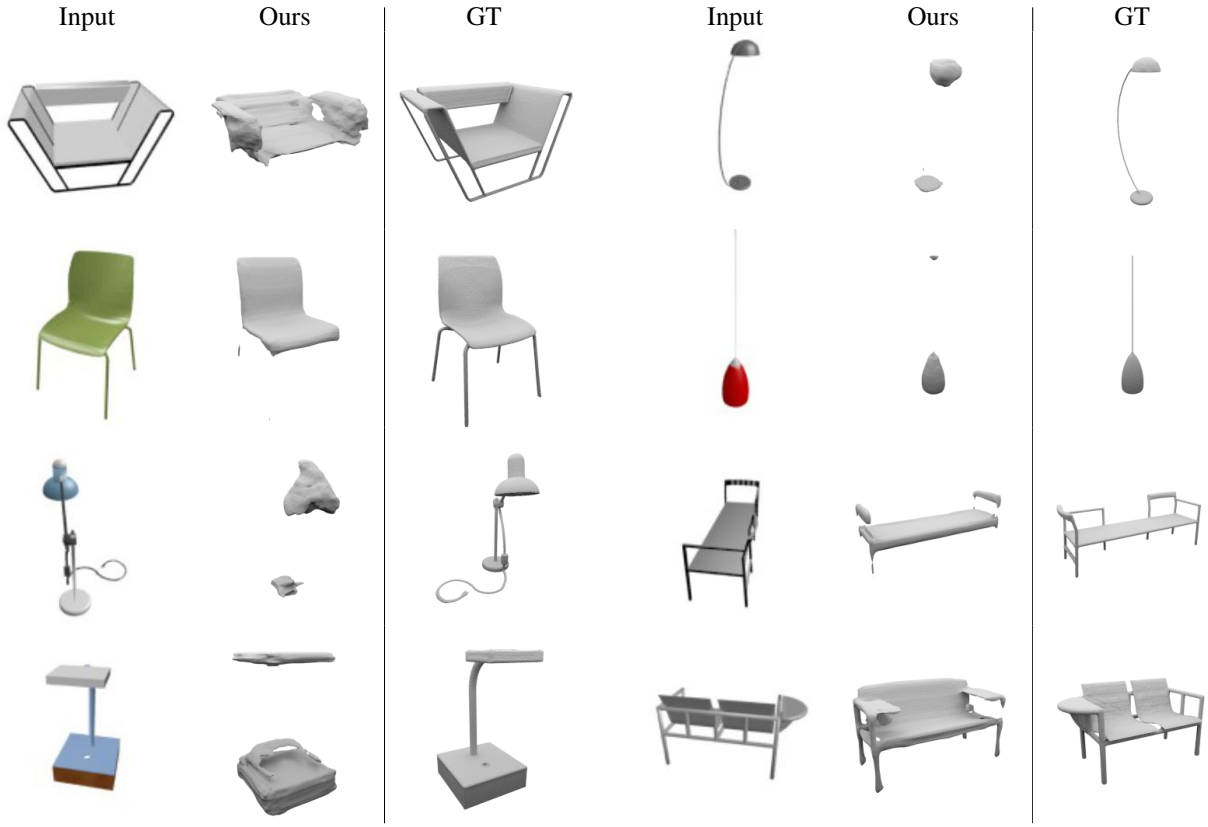


Figure 11: **Failure Cases.** While our method generally performs well, it struggles with extremely thin object parts and objects that are very different from the objects seen during training. These kinds of objects are especially frequent for the "lamp' category of the ShapeNet dataset. The input is shown in the first column, the other columns show the results for our method compared to the ground truth.

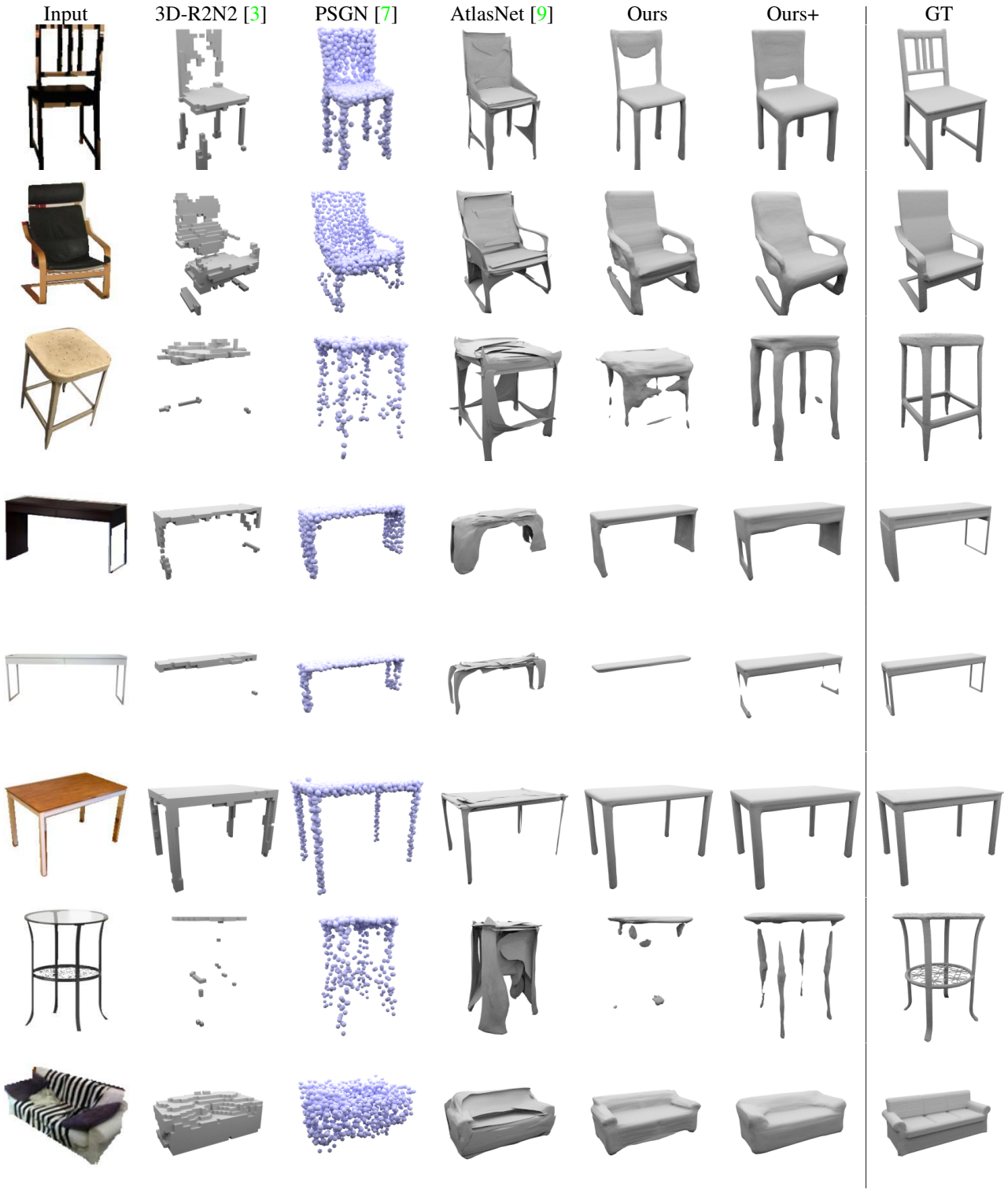


Figure 12: **Pix3D Dataset.** While all method generalize reasonably well to Pix3D [19], they perform worse and show more artifacts than on synthetic data.

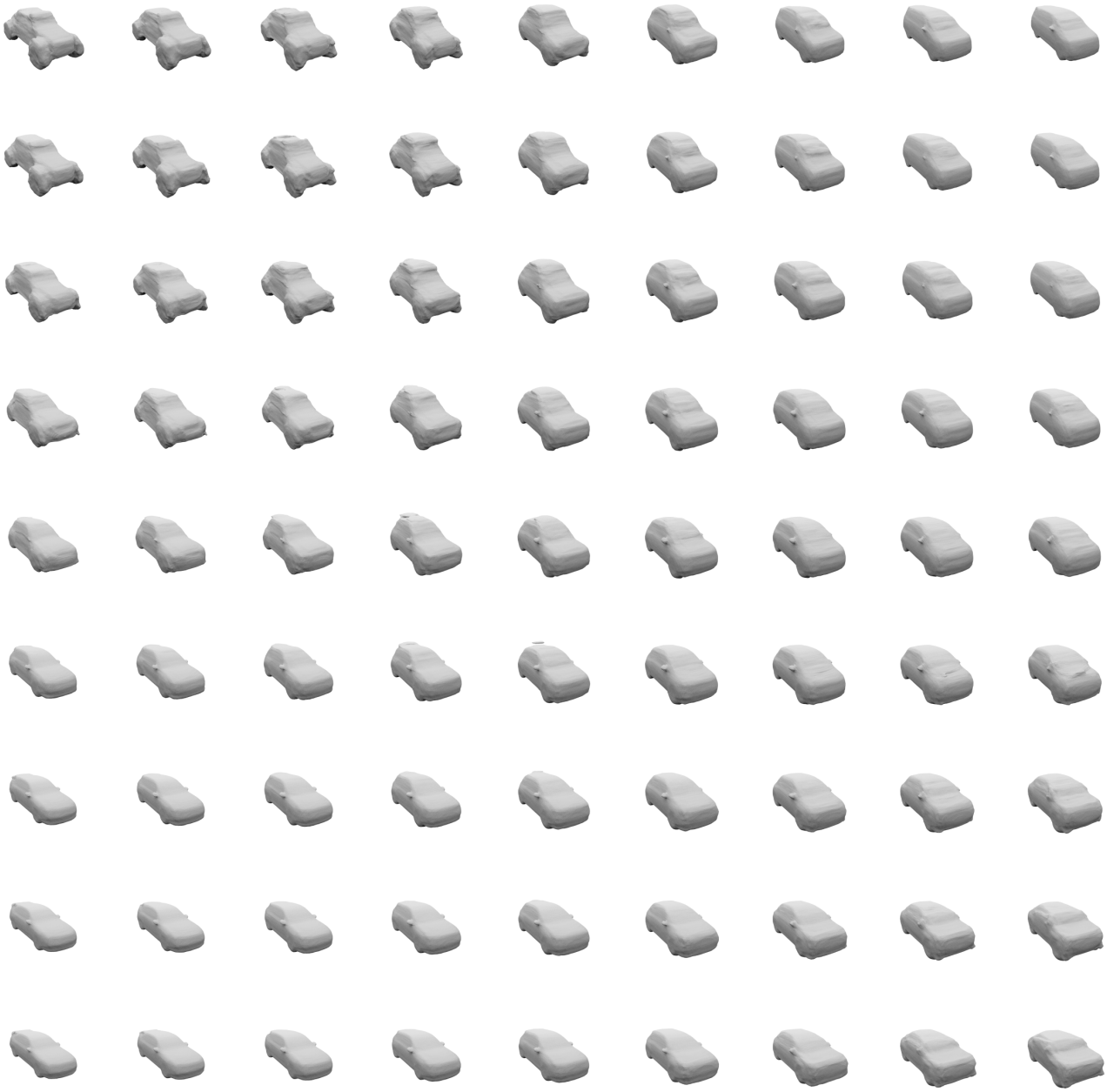


Figure 13: **Unconditional Model.** Interpolations in latent space for “car” category of the ShapeNet dataset.

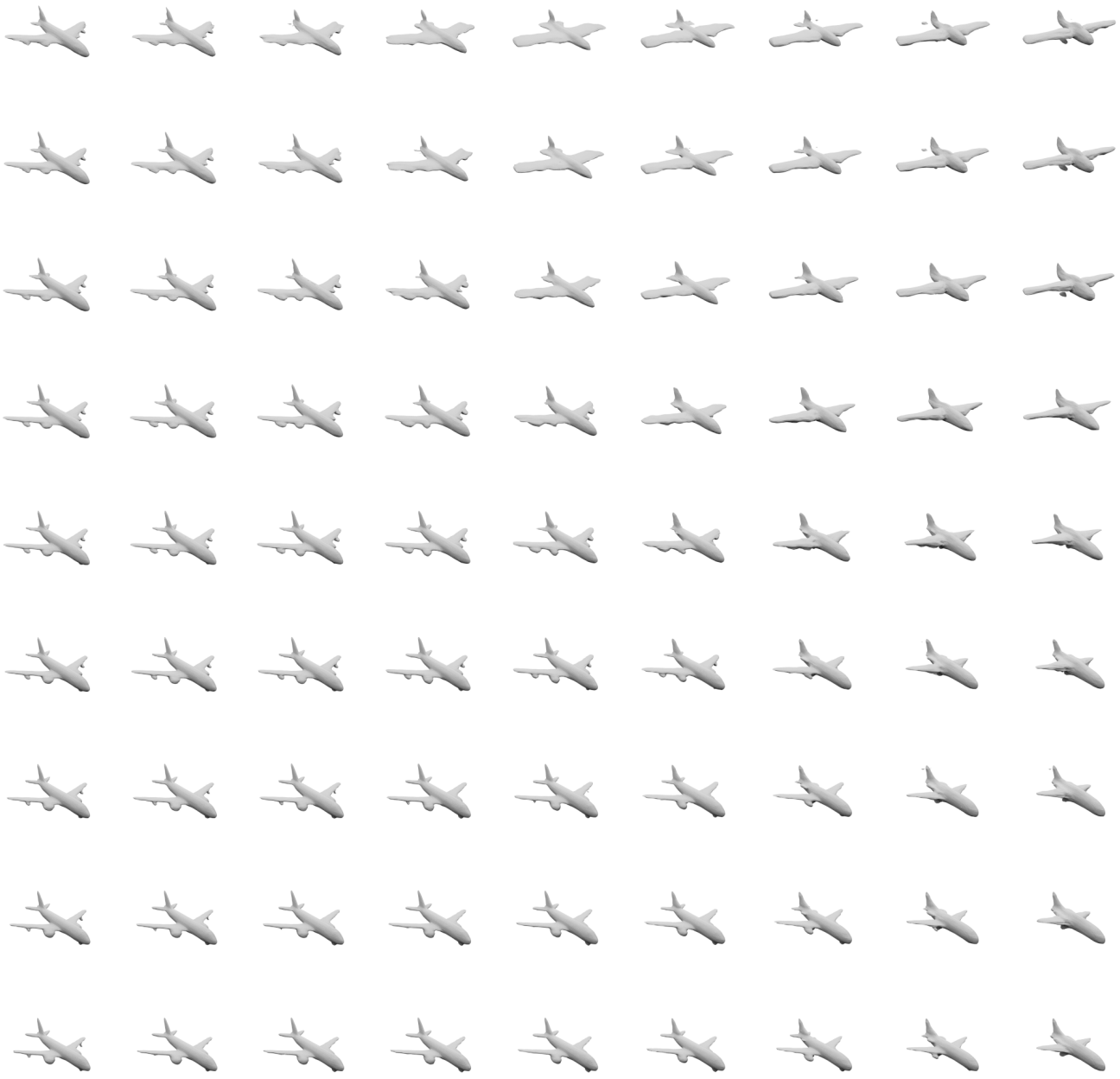


Figure 14: **Unconditional Model.** Interpolations in latent space for “airplane” category of the ShapeNet dataset.

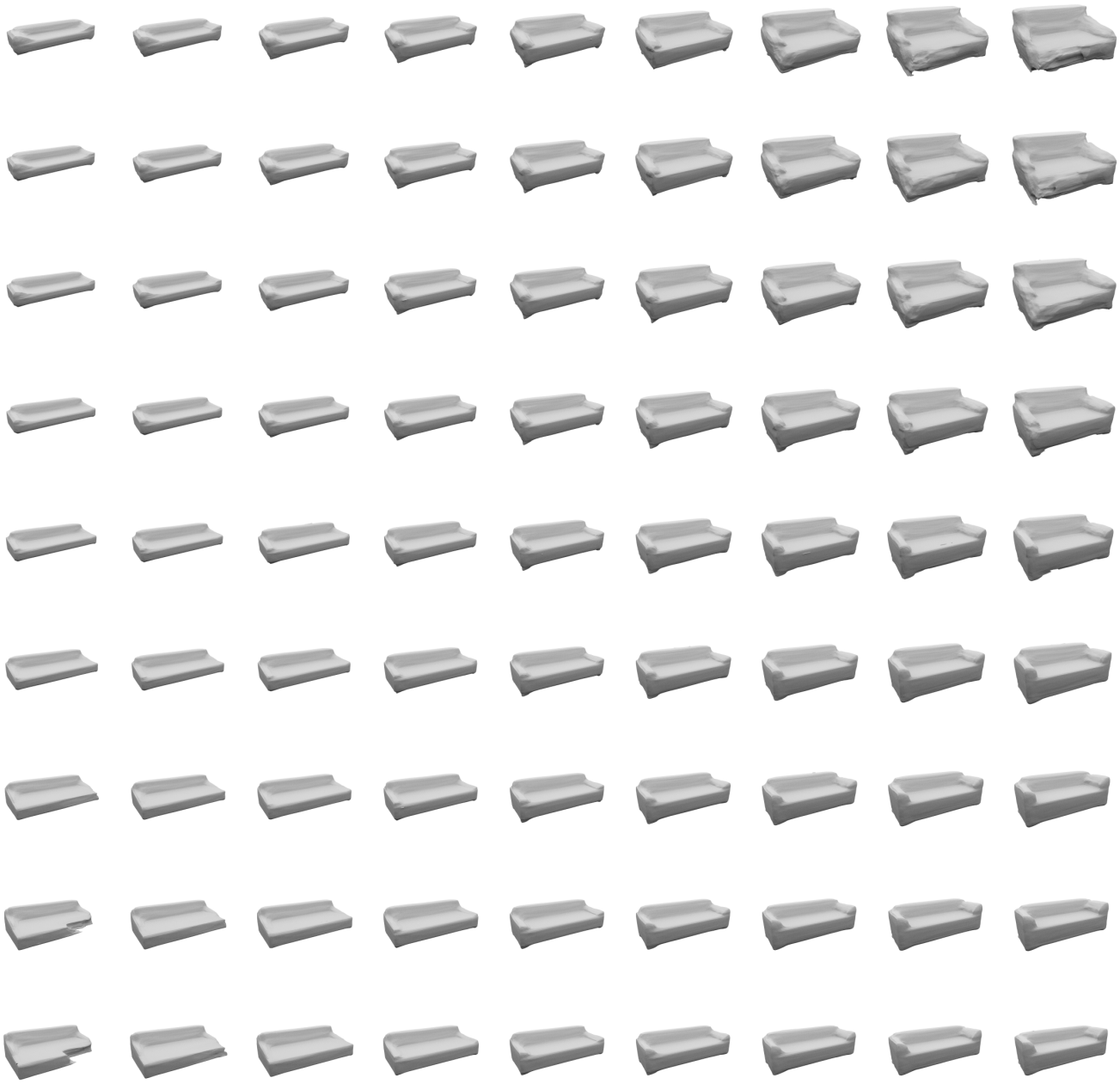


Figure 15: **Unconditional Model.** Interpolations in latent space for “sofa” category of the ShapeNet dataset.

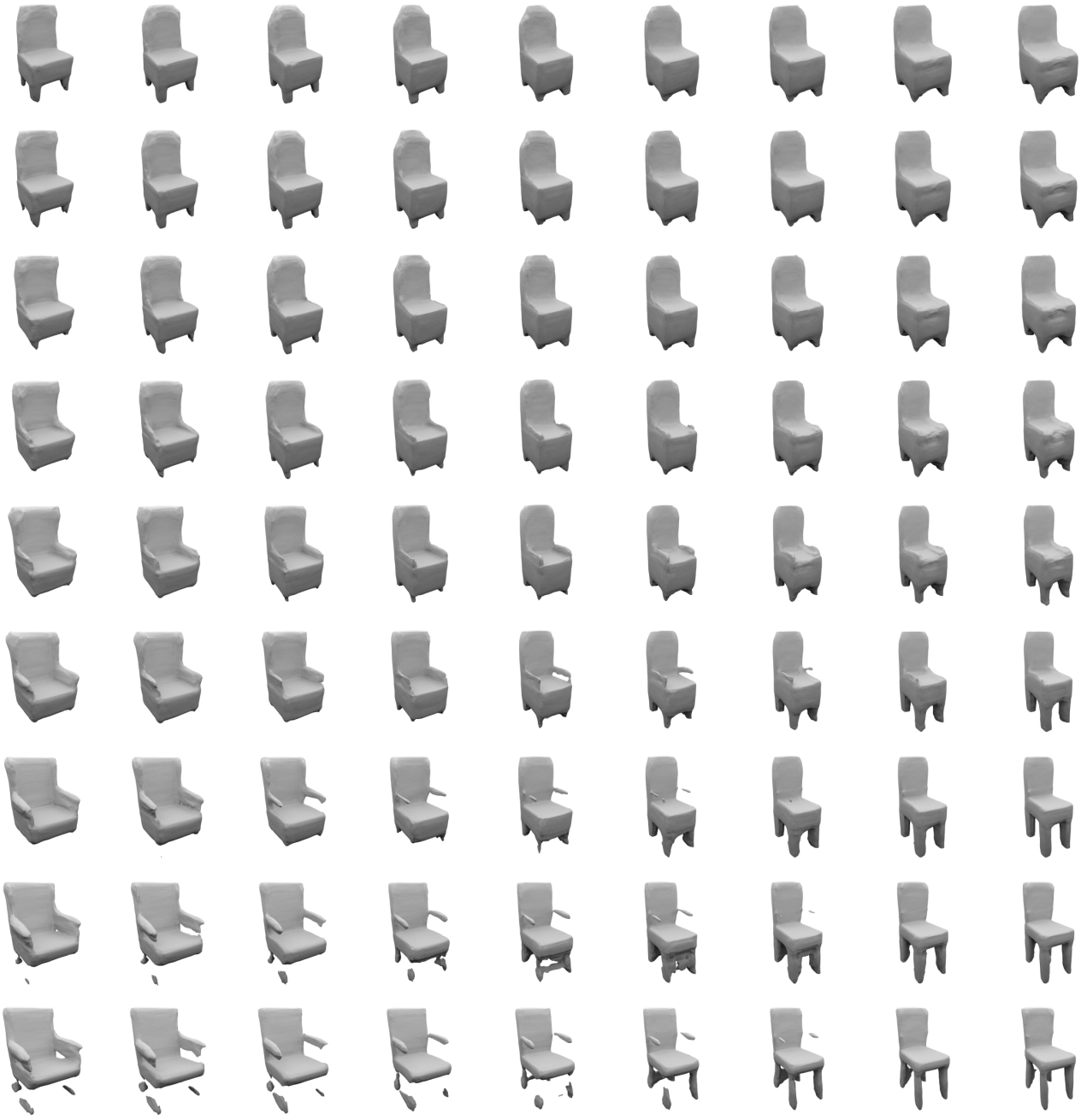
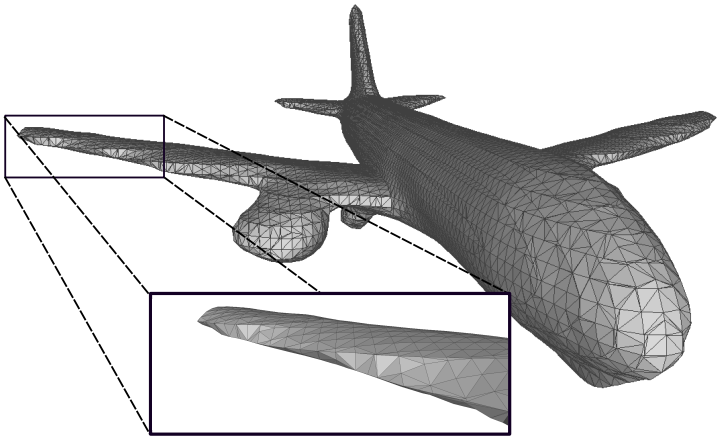
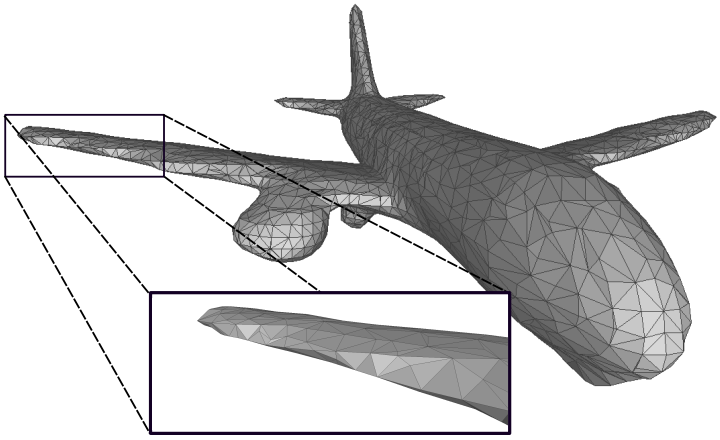


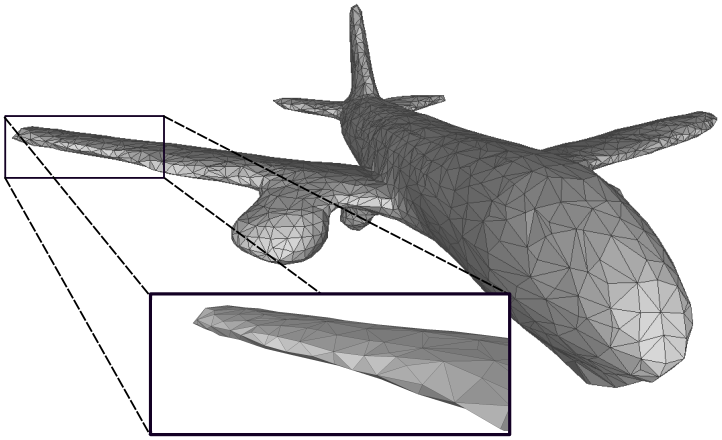
Figure 16: **Unconditional Model.** Interpolations in latent space for “chair” category of the ShapeNet dataset.



(a) After Marching Cubes



(b) After Simplification



(c) After Refinement

Figure 17: **Multiresolution IsoSurface Extraction.** This figure visualizes the different stages of Multiresolution IsoSurface Extraction (MISE).