Supplementary Material
DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation

Jeong Joon Park Peter Florence Julian Straub Richard Newcombe Steven Lovegrove

A. Overview

This supplementary material provides quantitative and qualitative experimental results along with extended technical details that are supplementary to the main paper. We first describe the shape completion experiment with noisy depth maps using DeepSDF (Sec. B). We then discuss architecture details (Sec. C) along with experiments exploring characteristics and tradeoffs of the DeepSDF design decisions (Sec. D). In Sec. G we compare auto-decoders with variational and standard auto-encoders. Further, additional details on data preparation (Sec. E), training (Sec. F), the auto-decoder learning scheme (Sec. H), and quantitative evaluations (Sec. I) are presented, and finally in Sec. J we provide additional quantitative and qualitative results.

B. Shape Completion from Noisy Depth Maps

We test the robustness of our shape completion method by using noisy depth maps as input. Specifically, we demonstrate the ability to complete shapes given partial noisy point clouds obtained from consumer depth cameras. Following [7], we simulate the noise distribution of typical structure depth sensors, including Kinect V1 by adding zero-mean Guassian noise to the inverse depth representation of a ground truth input depth image:

$$D_{\text{noise}} = \frac{1}{(1/D) + \mathcal{N}(0, \alpha^2)}, \tag{1}$$

where α is standard deviation of the normal distribution.

For the experiment, we synthetically generate noisy depth maps from the ShapeNet [2] plane models using the same benchmark test set of Dai et al. [3] used in the main paper. We perturb the depth values using standard deviation α of 0.01, 0.02, 0.03, and 0.05. Given that the target shapes are normalized to a unit sphere, one can observe that the inserted noise level is significant (Fig. 1).

The shape completion results with respect to added Guassian noise on the input synthetic depth maps are shown in Fig. 1. The Chamfer distance of the inferred shape versus the ground truth shape deteriorates approximately linearly with increasing standard deviation of the noise. Compared to the Chamfer distance between raw perturbed point cloud and ground truth depth map, which increases superlinearly with increasing noise level (Fig. 1), the shape completion quality using DeepSDF degrades much slower, implying that the shape priors encoded in the network play an important role regularizing the shape reconstruction.

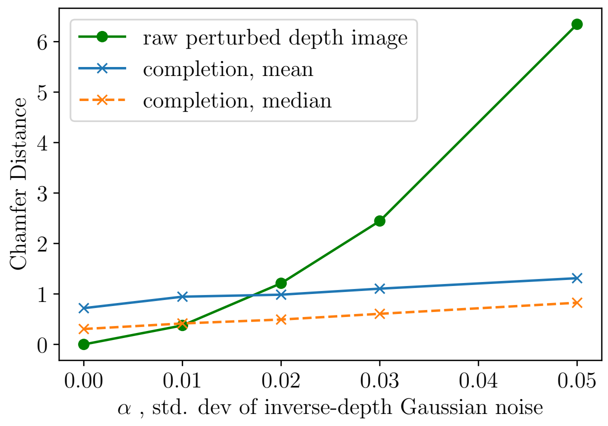


Figure 1: Chamfer distance (multiplied by 10^3) as a function of α, the standard deviation of inverse-depth Gaussian noise as shown in Eq. 1, for shape completions on planes from ShapeNet. Green line describes Chamfer distance between the perturbed depth points and original depth points, which shows the superlinear increase with increased noise. Blue and orange show respectively the mean and median of the shape completion's Chamfer distance (over a dataset of 85 plane completions) relative to the ground truth mesh which deteriorates approximately linearly with increasing standard deviation of noise. The same DeepSDF model was used for inference, the only difference is in the noise of the single depth image provided from which to perform shape completion. Example qualitative resuls are shown in Fig. 2.

C. Network Architecture

Fig. 4 depicts the overall architecture of DeepSDF. For all experiments in the main paper we used a network com-

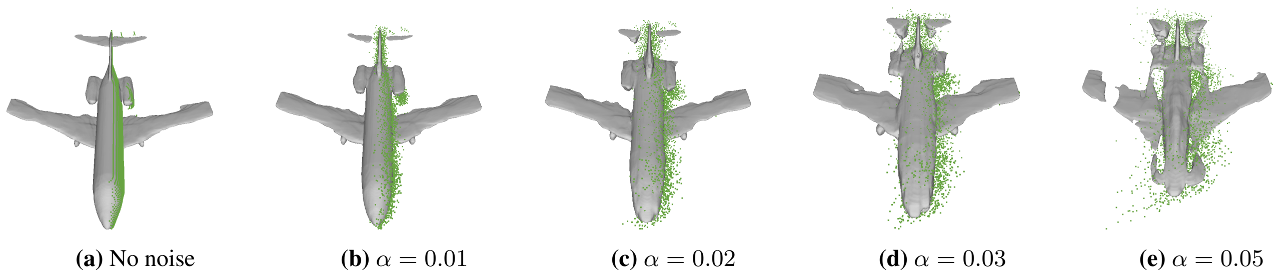


Figure 2: Shape completion results obtained from the partial and noisy input depth maps shown below. Input point clouds are overlaid on each completion to illustrate the scale of noise in the input.

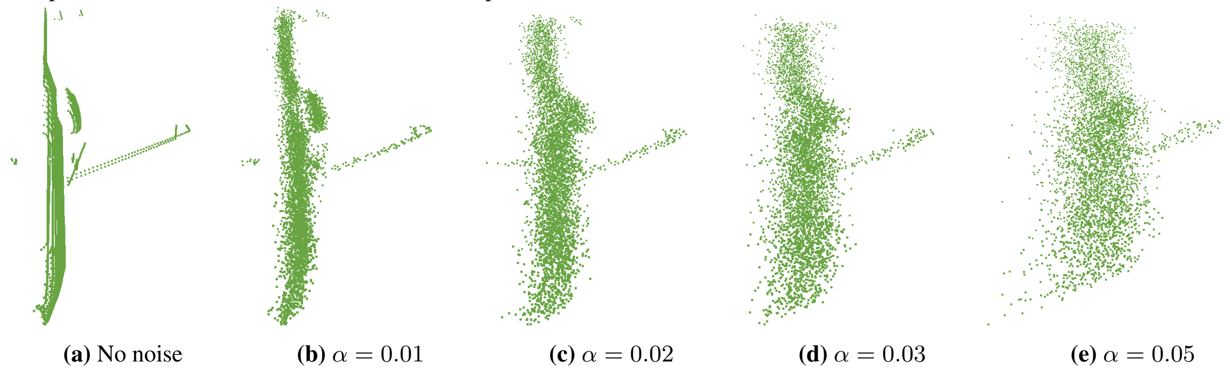


Figure 3: Visualization of partial and noisy point-clouds used to test shape completion with *DeepSDF*. Here, α is the standard deviation of Gaussian noise in Eq. 1. Corresponding completion results are shown above.

posed of 8 fully connected layers each of which are applied with weight-normalization, and each intermediate vectors are processed with RELU activation and 0.2 dropout except for the final layer. A skip connection is included at the fourth layer.

D. DeepSDF Network Design Decisions

In this section, we study system parameter decisions that affect the accuracy of SDF regression, thereby providing insight on the tradeoffs and scalability of the proposed algorithm.

D.1. Effect of Network Depth on Regression Accuracy

In this experiment we test how the expressive capability of DeepSDF varies as a function of the number of layers. Theoretically, an infinitely deep feed-forward network should be able to memorize the training data with arbitrary precision, but in practice this is not true due to finite compute power and the vanishing gradient problem, which limits the depth of the network.

We conduct an experiment where we let DeepSDF memorize SDFs of 500 chairs and inspect the training loss with varying number of layers. As described in Fig. 4, we find that applying the input vector (latent vector + xyz query) both to the first and a middle layer improves training. Inspired by this, we split the experiment into two cases: 1) train a regular network without skip connections, 2) train a network by concatenating the input vector to every 4 layers (e.g. for 12 layer network the input vector will be concatenated to the 4th, and 8th intermediate feature vectors).

Experiment results in Fig. 5 shows that the DeepSDF architecture without skip connections gets quickly saturated at 4 layers while the error keeps decreasing when trained with latent vector skip connections. Compared to the architecture we used for the main experiments (8 FC layers), a network with 16 layers produces significantly smaller training error, suggesting a possibility of using a deeper network for higher precision in some application scenarios. Further, we observe that the test error quickly decrease from four-layer architecture (9.7) to eight layer one (5.7) and subsequently plateaued for deeper architectures. However, this does not suggest conclusive results on generalization, as we used the same number of small training data for all architectures even though a network with more number of parameters tends to require higher volume of data to avoid overfitting.

D.2. Effect of Truncation Distance on Regression Accuracy

We would like to highlight that we can directly use the Ray Marching algorithm [8] for visualization: since the

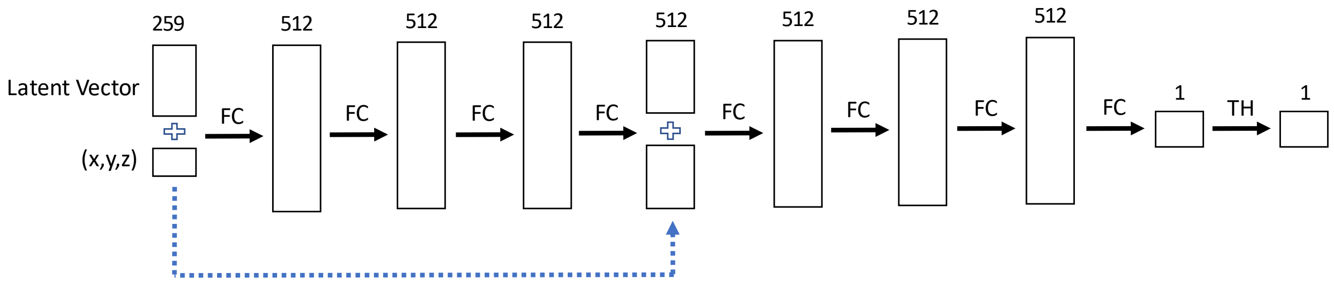


Figure 4: DeepSDF architecture used for experiments. Boxes represent vectors while arrows represent operations. The feed-forward network is composed of 8 fully connected layers, denoted as "FC" on the diagram. Each of the "FC" layers except for the last one is processed with weight-normalization, ReLU activation and Dropout. We used 256 and 128 dimensional latent vectors for reconstruction and shape completion experiments, respectively. The latent vector is concatenated, denoted "+", with the xyz query, making 259 length vector, and is given as input to the first layer. We find that inserting the latent vector again to the middle layers significantly improves the learning, denoted as dotted arrow in the diagram: the 259 vector is concatenated with the output of fourth fully connected layer to make a 512 vector. Final SDF value is obtained with hypberbolic tangent non-linear activation denoted as "TH".

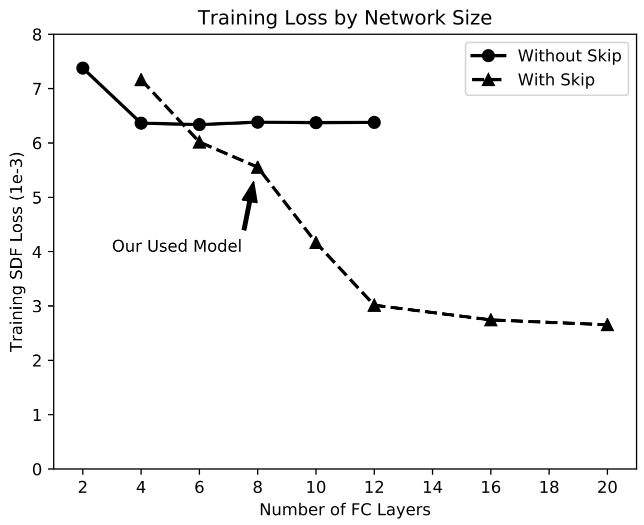


Figure 5: Regression accuracy (measured by the SDF loss used in training) as a function of network depth. Without skip connections, we observe a plateau in training loss past 4 layers. With skip connections, training loss continues to decrease although with diminishing returns past 12 layers. The model size chosen for all other experiments, 8 layers, provides a good tradeoff between speed and accuracy.

trained DeepSDF predicts SDF value at each spatial point, we may march along each ray by the SDF value at current position until intersecting a zero-crossing please refer to [8] for details. To speed up the ray-tracing we may train DeepSDF with a larger truncation region in exchange for the regression accuracy.

The truncation distance δ (from Eq. 4 of the manuscript) controls the extent from the surface over which we expect the network to learn a metric SDF. Fig. 6 plots the surface accuracy in terms of Chamfer distance as a function of truncation distance. We observe a moderate decrease in the accuracy of the surface representation as the truncation distance is increased. A hypothesis for an explanation is that it becomes more difficult to approximate a larger truncation region (a strictly larger domain of the function) to the same absolute accuracy as a smaller truncation region. The benefit, however, of larger truncation regions is that there is a larger region over which the metric SDF is maintained – in our application this reduces raycasting time, and there are other applications as well, such as physics simulation and robot motion planning for which a larger SDF of shapes may be valuable. We chose a δ value of 0.01 for all experiments presented in the manuscript, which provides a good tradeoff between raycasting speed and surface accuracy.

E. Data Preparation Details

For data preparation, we are given a mesh of a shape to sample spatial points and their SDF values. We begin by normalizing each shape so that the shape model fits into a unit sphere with some margin (in practice fit to sphere radius of 1/1.03). Then, we virtually render the mesh from 100 virtual cameras regularly sampled on the surface of the unit sphere. Then, we gather the surface points by back-projecting the depth pixels from the virtual renderings, and the points' normals are assigned from the triangle to which it belongs. Triangle surface orientations are set such that they are towards the camera. When a triangle is visible from both orientations, however, the given mesh is not watertight, making true SDF values hard to calculate, so we discard a mesh with more than 2% of its triangles being double-sided. For a valid mesh, we construct a KD-tree for the oriented surface points.

As stated in the main paper, it is important that we sample more aggressively near the surface of the mesh as we want to accurately model the zero-crossings. Specifically, we sample around 250,000 points randomly on the surface

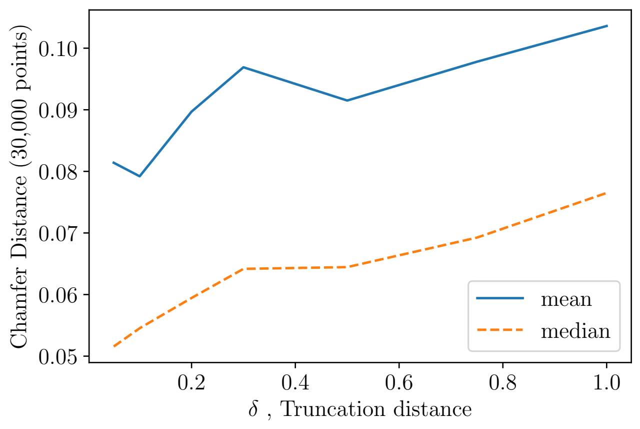


Figure 6: Chamfer distance (multiplied by 10^3) as a function of δ, the truncation distance, for representing a known small dataset of 100 cars from ShapeNet dataset [2]. All models were trained on the same set of SDF samples from these 100 cars. There is a moderate reduction in the accuracy of the surface, as measured by the increasing Chamfer distance, as the truncation distance is increased between 0.05 and 1.0. The bend in the curve at $\delta = 0.3$ is just expected to be due to the stochasticity inherent in training. Note that (a) due to the $\tanh()$ activation in the final layer, 1.0 is the maximum value the model can predict, and (b) the plot is dependent on the distribution of the samples used during training.

of the mesh, weighted by triangle areas. Then, we perturb each surface point along all xyz axes with mean-zero Gaussian noise with variance 0.0025 and 0.00025 to generate two spatial samples per surface point. For around 25,000 points we uniformly sample within the unit sphere. For each collected spatial samples, we find the nearest surface point from the KD-tree, measure the distance, and decide the sign from the dot product between the normal and their vector difference.

F. Training and Testing Details

For training, we find it important to initialize the latent vectors quite small, so that similar shapes do not diverge in the latent vector space – we used $\mathcal{N}(0, 0.01^2)$. Another crucial point is balancing the positive and negative samples both for training and testing: for each batch used for gradient descent, we set half of the SDF point samples positive and the other half negative.

Learning rate for the decoder parameters was set to be 1e-5 * B, where B is number of shapes in one batch. For each shape in a batch we randomly subsampled 16384 SDF samples (out of 500K available points). Learning rate for the latent vectors was set to be 1e-3. Also, we set the regularization parameter $\sigma = 10^{-2}$. We trained our models on 8 Nvidia GPUs approximately for 8 hours for 1000 epochs. For reconstruction experiments the latent vector size was set to be 256, and for the shape completion task we used models with 128 dimensional latent vectors.

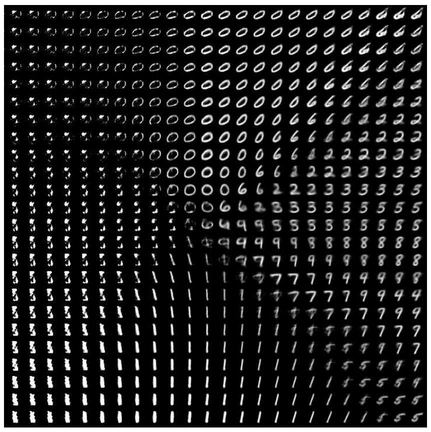
G. Comparison with Variational and Standard Auto-encoders on MNIST

To compare different approaches of learning a latent code-space for a given datum, we use the MNIST dataset and compare the variational auto encoder (VAE), the standard bottleneck auto encoder (AE), and the proposed auto decoder (AD). As the reconstruction error we use the standard binary cross-entropy and match the model architectures such that the decoders of the different approaches have exactly the same structure and hence theoretical capacity. We show all evaluations for different latent code-space dimensions of 2D, 5D and 15D.

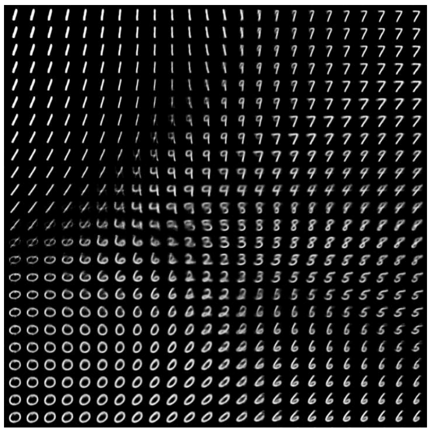
For 2D codes the latent spaces learned by the different methods are visualized in Fig. 7. All code spaces can reasonably represent the different digits. The AD latent space seems more condensed than the ones from VAE and AE. For the optimization-based encoding approach we initialize codes randomly. We show visualizations of such random samples in Fig. 8. Note that samples from the AD- and VAE-learned latent code spaces mostly look like real digits, showing their ability to generate realistic digit images.

We also compare the train and test reconstruction errors for the different methods in Fig. 9. For VAE and AE we show both the reconstruction error obtained using the learned encoder and obtained via code optimization using the learned decoder only (denoted “(V)AE decode”). The test error for VAE and AE are consistently minimized for all latent code dimensions. “AE decode ” diverges in all cases hinting at a learned latent space that is poorly suited for optimization-based decoding. Optimizing latent codes using the VAE encoder seems to work better for higher dimensional codes. The proposed AD approach works well in all tested code space dimensions. Although “VAE decode” has slightly lower test error than AD in 15 dimensions, qualitatively the AD’s reconstructions are better as we discuss next.

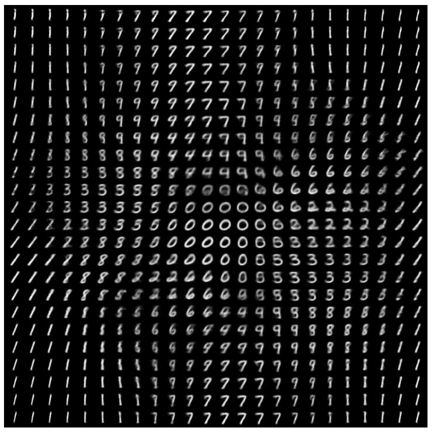
In Fig. 10 we show example reconstructions from the test dataset. When using the learned encoders VAE and AE produce qualitatively good reconstructions. When using optimization-based encoding “AE decode” performs poorly indicating that the latent space has many bad local minima. While the reconstructions from “VAE decode“ are, for the most part, qualitatively close to the original, AD’s reconstructions more closely resemble the actual digit of the test data. Qualitatively, AD is on par with reconstructions from end-to-end-trained VAE and AE.



(a) Auto Encoder (AE)

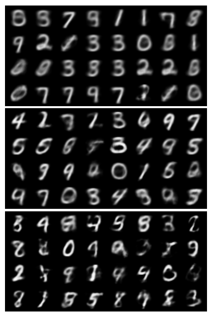


(b) Variational Auto Encoder (VAE)

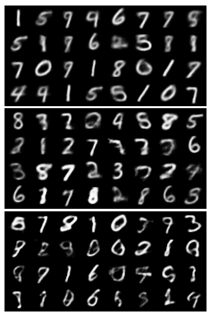


(c) Auto Decoder (AD)

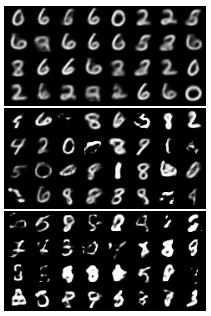
Figure 7: Comparison of the 2D latent code-space learned by the different methods. Note that large portion of the regular auto-encoder's (AE) latent embedding space contains images that do not look like digits. In contrast, both VAE and AD generate smooth and complete latent space without outstanding artifacts. Best viewed digitally.



(a) AD



(b) VAE



(c) AE

Figure 8: Visualization of random samples from the latent 2D (top), 5D (middle), and 15D (bottom) code space on MNIST. Note that the sampling from regular auto-encoder (AE) suffers from artifacts. Best viewed digitally.

H. Full Derivation of Auto-decoder-based DeepSDF Formulation

To derive the auto-decoder-based shape-coded DeepSDF formulation we adopt a probabilistic perspective. Given a dataset of N shapes represented with signed distance function ${SDF^i}_{i=1}^N$, we prepare a set of K point samples and their signed distance values:

$$X_i = \{(\boldsymbol{x}_j, s_j) : s_j = SDF^i(\boldsymbol{x}_j)\}\,. \tag{2}$$

The SDF values can be computed from mesh inputs as detailed in the main paper.

For an auto-decoder, as there is no encoder, each latent code $\boldsymbol{z}_i$ is paired with training shape data X_i and randomly initialized from a zero-mean Gaussian. We use $\mathcal{N}(0, 0.001^2)$. The latent vectors $\{\boldsymbol{z}_i\}_{i=1}^N$ are then jointly optimized during training along with the decoder parameters θ.

We assume that each shape in the given dataset $\boldsymbol{X} = \{X_i\}_{i=1}^N$ follows the joint distribution of shapes:

$$p_\theta(X_i, \boldsymbol{z}_i) = p_\theta(X_i|\boldsymbol{z}_i)p(\boldsymbol{z}_i)\,, \tag{3}$$

where θ parameterizes the data likelihood. For a given θ a shape code $\boldsymbol{z}_i$ can be estimated via Maximum-a-Posterior (MAP) estimation:

$$\hat{\boldsymbol{z}}_i = \arg\max_{\boldsymbol{z}_i} p_\theta(\boldsymbol{z}_i|X_i) = \arg\max_{\boldsymbol{z}_i} \log p_\theta(\boldsymbol{z}_i|X_i)\,. \tag{4}$$

We estimate θ as the parameters that maximizes the posterior across all shapes:

$$\begin{aligned}\hat{\theta} &= \arg\max_\theta \sum_{X_i \in \boldsymbol{X}} \max_{\boldsymbol{z}_i} \log p_\theta(\boldsymbol{z}_i|X_i) \\ &= \arg\max_\theta \sum_{X_i \in \boldsymbol{X}} \max_{\boldsymbol{z}_i} (\log p_\theta(X_i|\boldsymbol{z}_i) + \log p(\boldsymbol{z}_i))\,,\end{aligned} \tag{5}$$

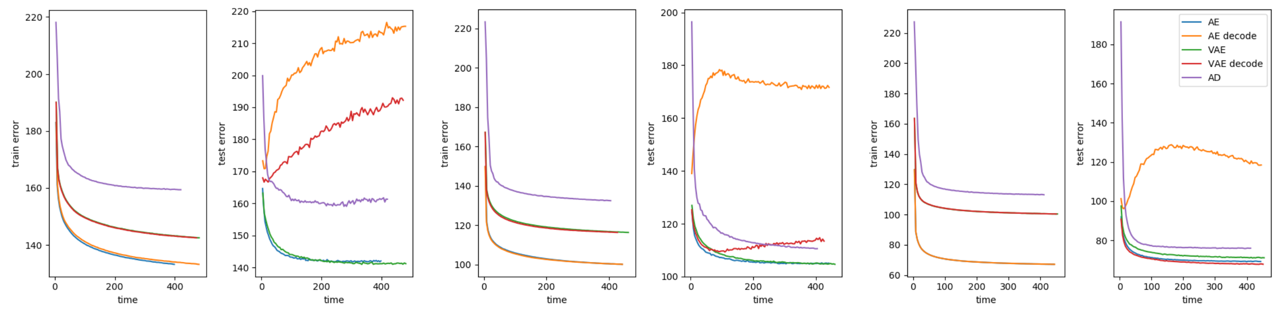
where the second equality follows from Bayes Law.

For each shape X_i defined via point and SDF samples $(\boldsymbol{x}_j, \boldsymbol{s}_j)$ as defined in Eq. 2 we make a conditional independence assumption given the code z_i to arrive at the decomposition of the posterior $p_\theta(X_i|z_i)$:

$$p_\theta(X_i|z_i) = \prod_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X_i} p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j)\,. \tag{6}$$

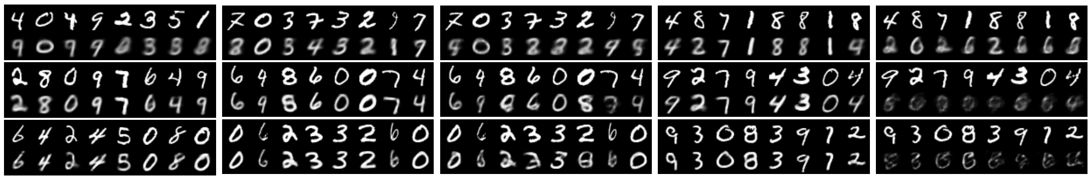
Note that the individual SDF likelihoods $p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j)$ are parameterized by the sampling location $\boldsymbol{x}_j$.

To derive the proposed auto-decoder-based DeepSDF approach we express the SDF likelihood via a deep feed-forward network $f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j)$ and, without loss of generality,



(a) 2D **(b)** 5D **(c)** 15D

Figure 9: Train and test error for different dimensions of the latent code for the different approaches.



(a) AD **(b)** VAE **(c)** VAE decode **(d)** AE **(e)** AE decode

Figure 10: Reconstructions for 2D (top), 5D (middle), and 15D (bottom) code space on MNIST. For each of the different dimensions we plot the given test MNIST image and the reconstruction given the inferred latent code.

assume that the likelihood takes the form:

$$p_\theta(\boldsymbol{s}_j|z_i;\boldsymbol{x}_j) = \exp(-\mathcal{L}(f_\theta(\boldsymbol{z}_i,\boldsymbol{x}_j), s_j)) \,. \tag{7}$$

The SDF prediction $\tilde{s}_j = f_\theta(\boldsymbol{z}_i,\boldsymbol{x}_j)$ is represented using a fully-connected network and $\mathcal{L}(\tilde{s}_j, s_j)$ is a loss function penalizing the deviation of the network prediction from the actual SDF value s_j. One example for the cost function is the standard L_2 loss function which amounts to assuming Gaussian noise on the SDF values. In practice we use the clamped L_1 cost introduced in the main manuscript.

In the latent shape-code space, we assume the prior distribution over codes $p(\boldsymbol{z_i})$ to be a zero-mean multivariate-Gaussian with a spherical covariance $\sigma^2 I$. Note that other more complex priors could be assumed. This leads to the final cost function via Eq. 5 which we jointly minimize with respect to the network parameters θ and the shape codes $\{z_i\}_{i=1}^N$:

$$\underset{\theta,\{\boldsymbol{z}_i\}_{i=1}^N}{\arg\min} \sum_{i=1}^{N}\left(\sum_{j=1}^{K}\mathcal{L}(f_\theta(\boldsymbol{z}_i,\boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}_i||_2^2\right) . \tag{8}$$

At inference time, we are given SDF point samples X of one underlying shape to estimate the latent code $\boldsymbol{z}$ describing the shape. Using the MAP formulation from Eq. 4 with fixed network parameters θ we arrive at:

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z}}{\arg\min} \sum_{(\boldsymbol{x}_j,\boldsymbol{s}_j)\in X} \mathcal{L}(f_\theta(\boldsymbol{z},\boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}||_2^2, \tag{9}$$

where $\frac{1}{\sigma^2}$ can be used to balance the reconstruction and regularization term. For additional comments and insights as well as the practical implementation of the network and its training refer to the main manuscript.

I. Details on Quantitative Evaluations

I.1. Preparation for Benchmarked Methods

I.1.1 DeepSDF

For quantitative evaluations we converted the DeepSDF model for a given shape into a mesh by using Marching Cubes [9] with 512^3 resolution. Note that while this was done for quantitative evaluation as a mesh, many of the qualitative renderings are instead produced by raycasting directly against the continuous SDF model, which can avoid some of the artifacts produced by Marching Cubes at finite resolution. For all experiments in representing known or unknown shapes, DeepSDF was trained on ShapeNet v2, while all shape completion experiments were trained on ShapeNet v1, to match 3D-EPN. Additional DeepSDF training details are provided in Sec. F.

I.1.2 OGN

For OGN we trained the provided decoder model ("shape_from_id") for 300,000 steps on the same train set of cars used for DeepSDF. To compute the point-based metrics, we took the pair of both the groundtruth 256-voxel training data provided by the authors, and the generated 256-voxel output, and converted both of these into point clouds of only the surface voxels, with one point for each of the voxels' centers. Specifically, surface voxels were defined as voxels which have at least one of 6 direct (non-diagonal) voxel neighbors unoccupied. A typical number of vertices in the resulting point clouds is approximately 80,000, and the points used for evaluation are randomly sampled from these sets. Additionally, OGN was trained based on ShapeNet v1, while AtlasNet was trained on ShapeNet v2. To adjust for the scale difference, we converted OGN point clouds into ShapeNet v2 scale for each model.

I.1.3 AtlasNet

Since the provided pretrained AtlasNet models were trained multi-class, we instead trained separate AtlasNet models for each evaluation. Each model was trained with the available code by the authors with all default parameters, except for the specification of class for each model and matching train/test splits with those used for DeepSDF. The quality of the models produced from these trainings appear comparable to those in the original paper.

Of note, we realized that AtlasNet's own computation of its training and evaluation metric, Chamfer distance, had the limitation that only the vertices of the generated mesh were used for the evaluation. This leaves the triangles of the mesh unconstrained in that they can connect across what are supposed to be holes in the shape, and this would not be reflected in the metric. Our evaluation of meshes produced by AtlasNet instead samples evenly from the mesh surface, i.e. each triangle in the mesh is weighted by its surface area, and points are sampled from the triangle faces.

I.1.4 3D-EPN

We used the provided shape completion inference results for 3D-EPN, which is in voxelized distance function format. We subsequently extracted the isosurface using MATLAB as described in the paper to obtain the final mesh.

I.2. Metrics

The first two metrics, Chamfer and Earth Mover's, are easily applicable to points, meshes (by sampling points from the surface) and voxels (by sampling surface voxels and treating their centers as points). When meshes are available, we also can compute metrics suited particularly for meshes: mesh accuracy, mesh completion, and mesh cosine similarity.

Chamfer distance is a popular metric for evaluating shapes, perhaps due to its simplicity [5]. Given two point sets S_1 and S_2, the metric is simply the sum of the nearest-neighbor distances for each point to the nearest point in the other point set.

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_2^2$$

Note that while sometimes the metric is only defined one-way (i.e., just $\sum_{x \in S_1} \underset{y \in S_2}{min} ||x - y||_2^2$) and this is not symmetric, the sum of both directions, as defined above, is symmetric: $d_{CD}(S_1, S_2) = d_{CD}(S_2, S_1)$. Note also that the metric is not technically a valid distance function since it does not satisfy the triangle inequality, but is commonly used as a psuedo distance function [5]. In all of our experiments we report the Chamfer distance for 30,000 points for both $|S_1|$ and $|S_2|$, which can be efficiently computed by use of a KD-tree, and akin to prior work [6] we normalize by the number of points: we report $\frac{d_{CD}(S_1, S_2)}{30{,}000}$.

Earth Mover's distance [10], also known as the Wasserstein distance, is another popular metric for measuring the difference between two discrete distributions. Unlike the Chamfer distance, which does not require any constraints on the correspondences between evaluated points, for the Earth Mover's distance a bijection $\phi : S_1 \to S_2$, i.e. a one-to-one correspondence, is formed. Formally, for two point sets S_1 and S_2 of equal size $|S_1| = |S_2|$, the metric is defined via the optimal bijection [5]:

$$d_{EMD}(S_1, S_2) = \min_{\phi : S_1 \to S_2} \sum_{x \in S_1} ||x - \phi(x)||_2$$

Although the metric is commonly approximated in the deep learning literature [5] by distributed approximation schemes [1] for speed during training, we compute the metric accurately for evaluation using a more modest number of point samples (500) using [4].

In practice the intuitive, important difference between the Chamfer and Earth Mover's metrics is that the Earth Mover's metric more favors distributions of points that are similarly evenly distributed as the ground truth distribution. A low Chamfer distance may be achieved by assigning just one point in S_2 to a cluster of points in S_1, but to achieve a low Earth Mover's distance, each cluster of points in S_1 requires a comparably sized cluster of points in S_2.

Mesh accuracy, as defined in [11], is the minimum distance d such that 90% of generated points are within d of the ground truth mesh. We used 1,000 points sampled evenly from the generated mesh surface, and computed the minimum distances to the *full* ground truth mesh. To clarify, the distance is computed to the closest point on any face of the mesh, not just the vertices. Note that unlike Chamfer and Earth Mover's metrics which require sampling of points from both meshes, with this metric the entire mesh for the ground truth is used – accordingly this metric has lower variance than for example Chamfer distance computed with only 1,000 points from each mesh. Note also that mesh accuracy does not measure how *complete* the generated mesh is – a low (good) mesh accuracy can be achieved by only generating one small portion of the ground truth mesh, ignoring the rest. Accordingly, it is ideal to pair mesh accuracy with the following metric, mesh completion.

Mesh completion, also as defined in [11], is the fraction of points sampled from the ground truth mesh that are within some distance Δ (a parameter of the metric) to the generated mesh. We used $\Delta = 0.01$, which well measured the differences in mesh completion between the different methods. With this metric the *full* generated mesh is used, and points (we used 1,000) are sampled from the ground truth mesh (mesh accuracy is vice versa). Ideal mesh completion is 1.0, minimum is 0.0.

Mesh cosine similarity is a metric we introduce to measure the accuracy of mesh normals. We define the metric as the mean cosine similarity between the normals of points sampled from the ground truth mesh, and the normals of the nearest faces of the generated mesh. More precisely, given the generated mesh M_{gen} and a set of points with normals S_{gt} sampled from the ground truth mesh, for each point x_i in S_{gt} we look up the closest face F_i in M_{gen}, and then compute the average cosine similarity between the normals associated with x_i and F_i,

$$\text{Cos. sim}(M_{gen}, S_{gt}) = \frac{1}{|S_{gt}|} \sum_{x_i \in S_{gt}} \hat{n}_{F_i} \cdot \hat{n}_{x_i} \ ,$$

where each $\hat{n} \in \mathbb{R}^3$ is a unit-norm normal vector. We use $|S_{gt}| = 2,500$ and in order to allow for [6] which does not provide oriented normals, we compute the $\min(\cdot)$ over both the generated mesh normal and its flipped normal: $\min(\hat{n}_{F_i} \cdot \hat{n}_{x_i}, -\hat{n}_{F_i} \cdot \hat{n}_{x_i})$. Ideal cosine similarity is 1.0, minimum (given the allowed flip of the normal) is 0.0.

J. Additional Results

J.1. Representing Unseen Objects

We provide additional results on representing test objects with trained DeepSDF (Fig. 11, 12). We provide additional data with the additional metrics, mesh completion and mesh cosine similarity, for the comparison of methods contained in the manuscript (Tab. 1). The success of this task for DeepSDF implies that 1) high quality shapes similar to the test shapes exist in the embedding space, and 2) the codes for the shapes can be found through simple gradient descent.

Mesh comp., mean	chair	plane	table	lamp	sofa
AtlasNet-Sph.	0.668	0.862	0.755	0.281	0.641
AtlasNet-25	0.723	0.887	0.785	0.528	0.681
DeepSDF	**0.947**	**0.943**	**0.959**	**0.877**	**0.931**
Mesh comp., median					
AtlasNet-Sph.	0.686	0.930	0.795	0.257	0.666
AtlasNet-25	0.736	0.944	0.825	0.533	0.702
DeepSDF	**0.970**	**0.970**	**0.982**	**0.930**	**0.941**
Cosine sim., mean					
AtlasNet-Sph.	0.790	0.840	0.826	0.719	0.847
AtlasNet-25	0.797	0.858	0.835	0.725	0.826
DeepSDF	**0.896**	**0.907**	**0.916**	**0.862**	**0.917**

Table 1: Comparison of metrics for representing unknown shapes (U) for various classes of ShapeNet. Mesh completion as defined in [11] i.e. the fraction of groundtruth sampled points that are within a Δ (we used $\Delta = 0.01$) of the generated mesh, and mean cosine similarity is of normals for nearest groundtruth-generated point pairs. Cosine similarity is defined in I.2. Higher is better for all metrics in this table.

J.2. Shape Completions

Finally we present additional shape completion results on unperturbed depth images of synthetic ShapeNet dataset (Fig. 13), demonstrating the quality of the auto-decoder learning scheme and the new shape representation.

References

[1] D. P. Bertsekas. A distributed asynchronous relaxation algorithm for the assignment problem. In *Decision and Control, 1985 24th IEEE Conference on*, pages 1703–1704. IEEE, 1985.

[2] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[3] A. Dai, C. Ruizhongtai Qi, and M. Niessner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *CVPR*, pages 5868–5877, 2017.

[4] G. Doran. PyEMD: Earth mover's distance for Python, 2014–. [Online; accessed ¡today¿].

[5] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image.

[6] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier-m\^ ach\'e approach to learning 3d surface generation. *arXiv preprint arXiv:1802.05384*, 2018.

[7] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *Robotics and automation (ICRA), 2014 IEEE international conference on*, pages 1524–1531. IEEE, 2014.

[8] J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

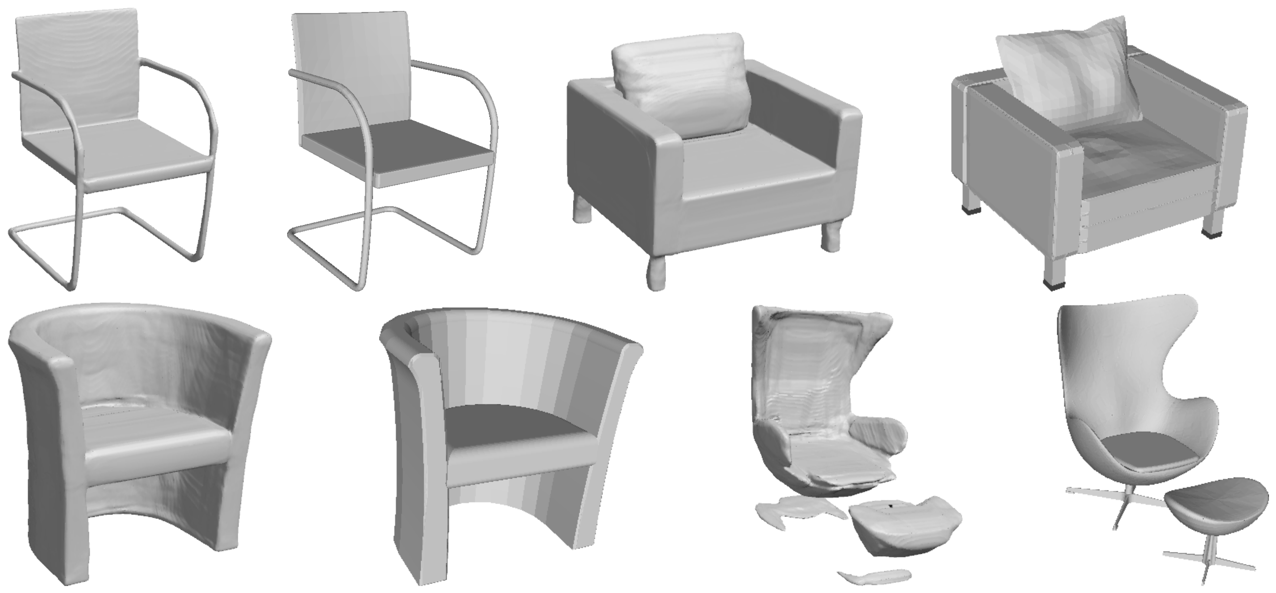


Figure 11: Additional test shape reconstruction results. Left to right alternatingly: DeepSDF reconstruction and ground truth.

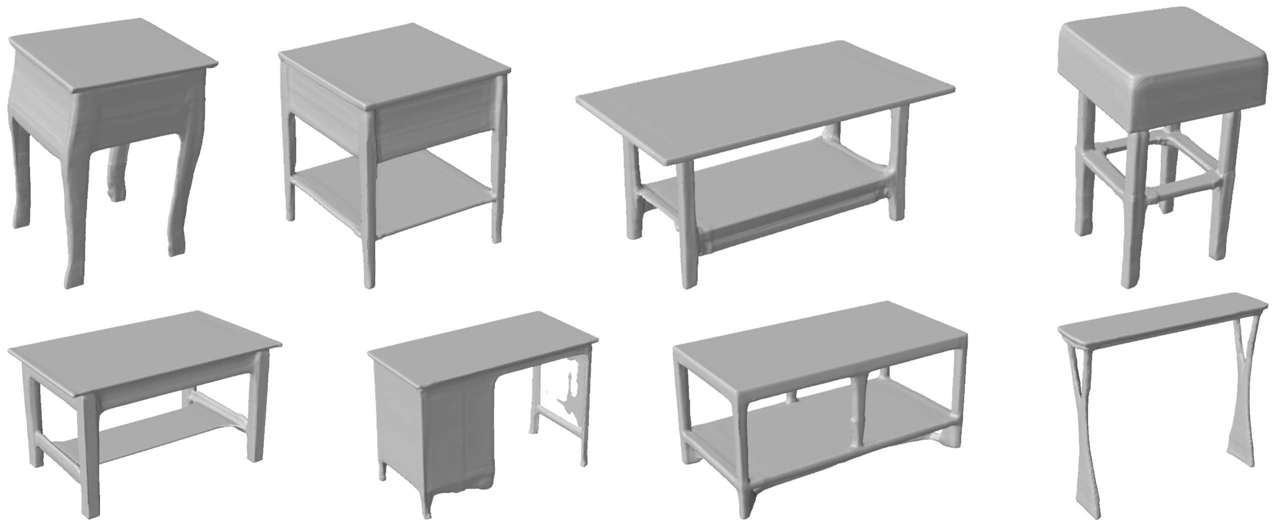


Figure 12: Additional test shape reconstruction results for Table ShapeNet class. All of the above images are test shapes represented with our DeepSDF network during inference time, showing the accuracy and expressiveness of the shape embedding.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, volume 21, pages 163–169. ACM, 1987.

[10] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66. IEEE, 1998.

[11] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. pages 519–528. IEEE, 2006.

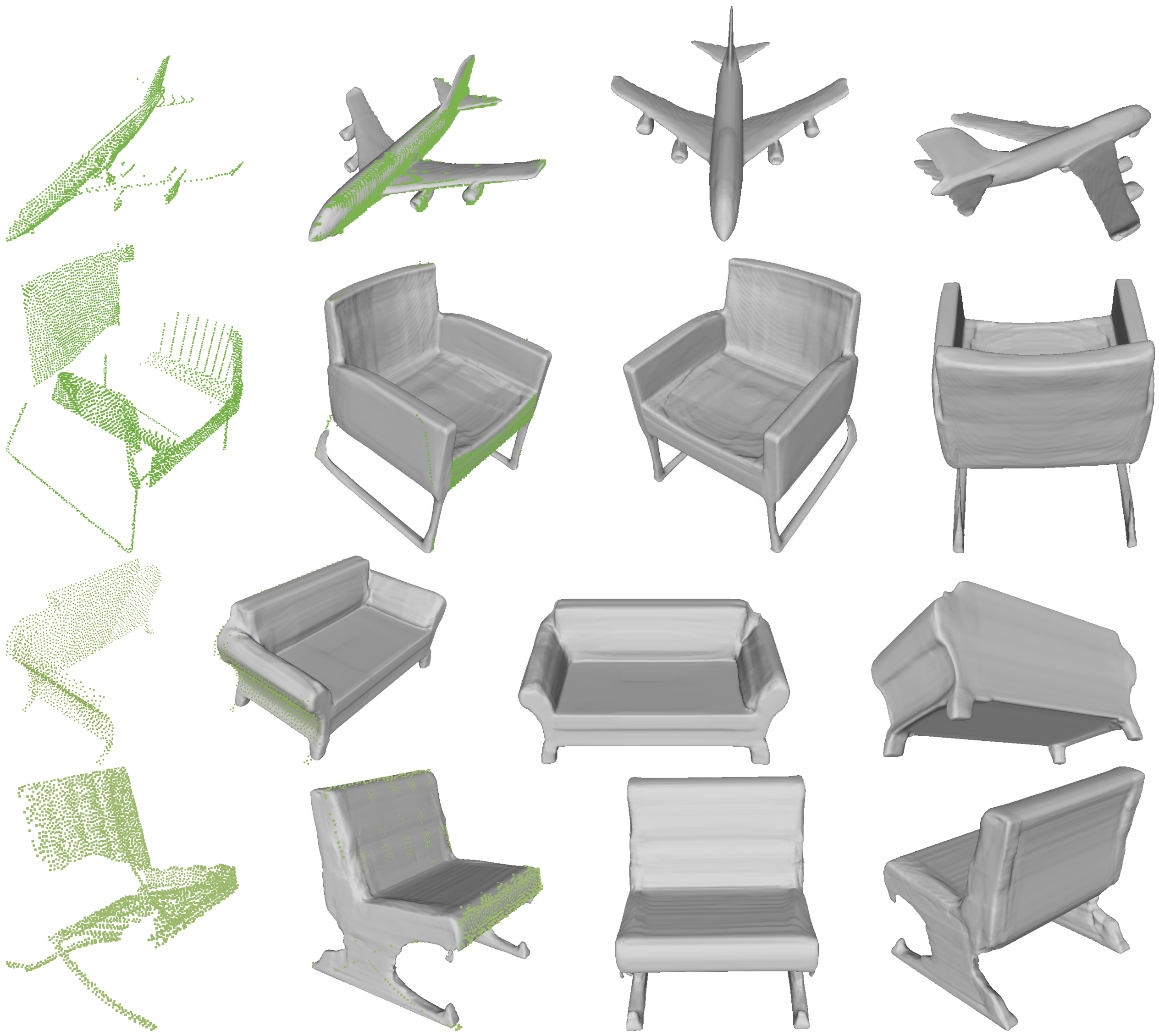


Figure 13: Additional shape completion results. Left to Right: input depth point cloud, shape completion using DeepSDF, second view, and third view.