# DeepSDF: Learning Continuous Signed Distance Functions
# for Shape Representation

Jeong Joon Park[1,3†]     Peter Florence [2,3†]     Julian Straub[3]     Richard Newcombe[3]     Steven Lovegrove[3]

[1]University of Washington     [2]Massachusetts Institute of Technology     [3]Facebook Reality Labs
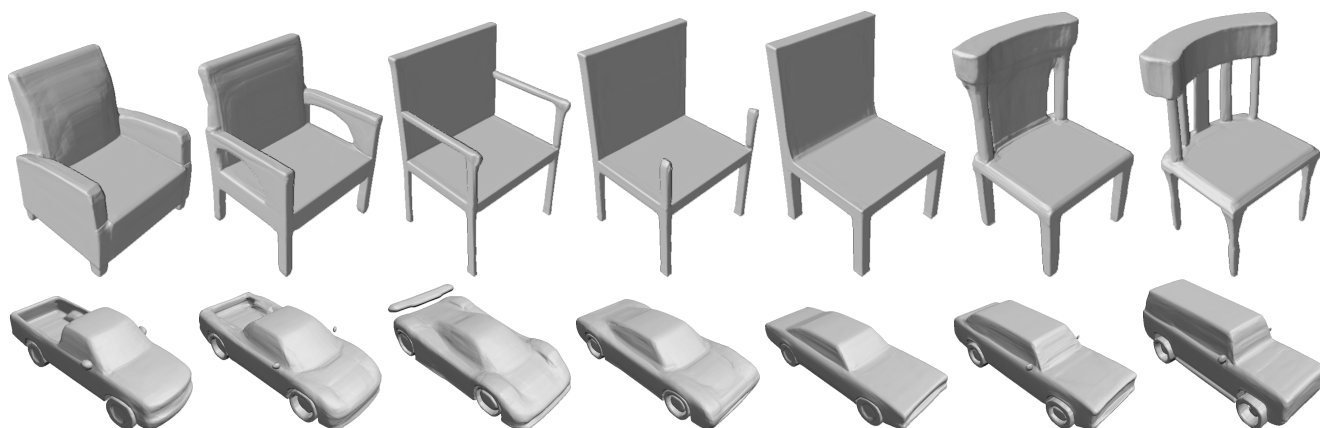
**Figure 1:** DeepSDF represents signed distance functions (SDFs) of shapes via latent code-conditioned feed-forward decoder networks. Above images are raycast renderings of DeepSDF interpolating between two shapes in the learned shape latent space. Best viewed digitally.

## Abstract

*Computer graphics, 3D computer vision and robotics communities have produced multiple approaches to representing 3D geometry for rendering and reconstruction. These provide trade-offs across fidelity, efficiency and compression capabilities. In this work, we introduce DeepSDF, a learned continuous Signed Distance Function (SDF) representation of a class of shapes that enables high quality shape representation, interpolation and completion from partial and noisy 3D input data. DeepSDF, like its classical counterpart, represents a shape's surface by a continuous volumetric field: the magnitude of a point in the field represents the distance to the surface boundary and the sign indicates whether the region is inside (-) or outside (+) of the shape, hence our representation implicitly encodes a shape's boundary as the zero-level-set of the learned function while explicitly representing the classification of space as being part of the shapes interior or not. While classical SDF's both in analytical or discretized voxel form typically represent the surface of a single shape, DeepSDF can represent an entire class of shapes. Furthermore, we show state-of-the-art performance for learned 3D shape representation and completion while reducing the model size by an order of magnitude compared with previous work.*

---

† Work performed during internship at Facebook Reality Labs.

## 1. Introduction

Deep convolutional networks which are a mainstay of image-based approaches grow quickly in space and time complexity when directly generalized to the 3rd spatial dimension, and more classical and compact surface representations such as triangle or quad meshes pose problems in training since we may need to deal with an unknown number of vertices and arbitrary topology. These challenges have limited the quality, flexibility and fidelity of deep learning approaches when attempting to either input 3D data for processing or produce 3D inferences for object segmentation and reconstruction.

In this work, we present a novel representation and approach for generative 3D modeling that is efficient, expressive, and fully continuous. Our approach uses the concept of a SDF, but unlike common surface reconstruction techniques which discretize this SDF into a regular grid for evaluation and measurement denoising, we instead learn a generative model to produce such a continuous field.

The proposed continuous representation may be intuitively understood as a learned shape-conditioned classifier for which the decision boundary is the surface of the shape itself, as shown in Fig. 2. Our approach shares the generative aspect of other works seeking to map a latent space to a distribution of complex shapes in 3D [54], but critically differs in the central representation. While the notion of an
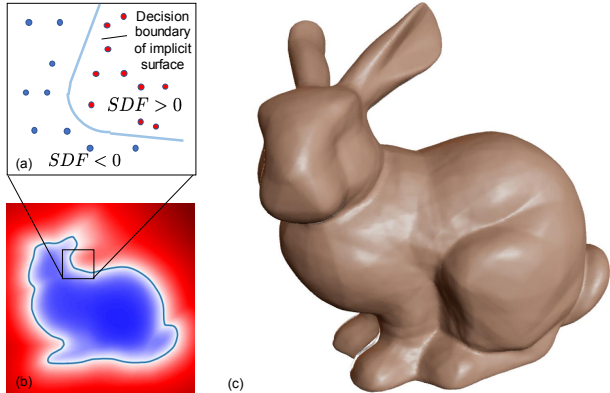
**Figure 2:** Our DeepSDF representation applied to the Stanford Bunny: (a) depiction of the underlying implicit surface $SDF = 0$ trained on sampled points inside $SDF < 0$ and outside $SDF > 0$ the surface, (b) 2D cross-section of the signed distance field, (c) rendered 3D surface recovered from $SDF = 0$. Note that (b) and (c) are recovered via DeepSDF.

implicit surface defined as a SDF is widely known in the computer vision and graphics communities, to our knowledge no prior works have attempted to directly learn continuous, generalizable 3D generative models of SDFs.

Our contributions include: (i) the formulation of generative shape-conditioned 3D modeling with a continuous implicit surface, (ii) a learning method for 3D shapes based on a probabilistic auto-decoder, and (iii) the demonstration and application of this formulation to shape modeling and completion. Our models produce high quality continuous surfaces with complex topologies, and obtain state-of-the-art results in quantitative comparisons for shape reconstruction and completion. As an example of the effectiveness of our method, our models use only 7.4 MB (megabytes) of memory to represent entire classes of shapes (for example, thousands of 3D chair models) – this is, for example, less than half the memory footprint (16.8 MB) of a single uncompressed $512^3$ 3D bitmap.

## 2. Related Work

We review three main areas of related work: 3D representations for shape learning (Sec. 2.1), techniques for learning generative models (Sec. 2.2), and shape completion (Sec. 2.3).

### 2.1. Representations for 3D Shape Learning

Representations for data-driven 3D learning approaches can be largely classified into three categories: point-based, mesh-based, and voxel-based methods. While some applications such as 3D-point-cloud-based object classification are well suited to these representations, we address their limitations in expressing continuous surfaces with complex topologies.

**Point-based.** A point cloud is a lightweight 3D representation that closely matches the raw data that many sensors (i.e. LiDARs, depth cameras) provide, and hence is a natural fit for applying 3D learning. PointNet [38, 39], for example, uses max-pool operations to extract global shape features, and the technique is widely used as an encoder for point generation networks [57, 1]. There is a sizable list of related works to the PointNet style approach of learning on point clouds. A primary limitation, however, of learning with point clouds is that they do not describe topology and are not suitable for producing watertight surfaces.

**Mesh-based.** Various approaches represent classes of similarly shaped objects, such as morphable human body parts, with predefined template meshes and some of these models demonstrate high fidelity shape generation results [2, 34]. Other recent works [3] use poly-cube mapping [51] for shape optimization. While the use of template meshes is convenient and naturally provides 3D correspondences, it can only model shapes with fixed mesh topology.

Other mesh-based methods use existing [48, 36] or learned [22, 23] parameterization techniques to describe 3D surfaces by morphing 2D planes. The quality of such representations depends on parameterization algorithms that are often sensitive to input mesh quality and cutting strategies. To address this, recent data-driven approaches [57, 22] learn the parameterization task with deep networks. They report, however, that (a) multiple planes are required to describe complex topologies but (b) the generated surface patches are not stitched, i.e. the produced shape is not closed. To generate a closed mesh, sphere parameterization may be used [22, 23], but the resulting shape is limited to the topological sphere. Other works related to learning on meshes propose to use new convolution and pooling operations for meshes [17, 53] or general graphs [9].

**Voxel-based.** Voxels, which non-parametrically describe volumes with 3D grids of values, are perhaps the most natural extension into the 3D domain of the well-known learning paradigms (i.e., convolution) that have excelled in the 2D image domain. The most straightforward variant of voxel-based learning is to use a dense occupancy grid (occupied / not occupied). Due to the cubically growing compute and memory requirements, however, current methods are only able to handle low resolutions ($128^3$ or below). As such, voxel-based approaches do not preserve fine shape details [56, 14], and additionally voxels visually appear significantly different than high-fidelity shapes, since when rendered their normals are not smooth. Octree-based methods [52, 43, 26] alleviate the compute and memory limitations of dense voxel methods, extending for example the ability to learn at up to $512^3$ resolution [52], but even this resolution is far from producing shapes that are visually compelling.

Aside from occupancy grids, and more closely related to our approach, it is also possible to use a 3D grid of voxels to represent a signed distance function. This inherits from the success of fusion approaches that utilize a truncated SDF (TSDF), pioneered in [15, 37], to combine noisy depth maps into a single 3D model. Voxel-based SDF representations have been extensively used for 3D shape learning [59, 16, 49], but their use of discrete voxels is expensive in memory. As a result, the learned discrete SDF approaches generally present low resolution shapes. [30] reports various wavelet transform-based approaches for distance field compression, while [10] applies dimensionality reduction techniques on discrete TSDF volumes. These methods encode the SDF volume of each individual scene rather than a dataset of shapes.

## 2.2. Representation Learning Techniques

Modern representation learning techniques aim at automatically discovering a set of features that compactly but expressively describe data. For a more extensive review of the field, we refer to Bengio et al. [4].

**Generative Adversial Networks.** GANs [21] and their variants [13, 41] learn deep embeddings of target data by training discriminators adversarially against generators. Applications of this class of networks [29, 31] generate realstic images of humans, objects, or scenes. On the downside, adversarial training for GANs is known to be unstable. In the 3D domain, Wu et al. [54] trains a GAN to generate objects in a voxel representation, while the recent work of Hamu et al. [23] uses multiple parameterization planes to generate shapes of topological spheres.

**Auto-encoders.** Auto-encoder outputs are expected to replicate the original input given the constraint of an information bottleneck between the encoder and decoder. The ability of auto-encoders as a feature learning tool has been evidenced by the vast variety of 3D shape learning works in the literature [16, 49, 2, 22, 55] who adopt auto-encoders for representation learning. Recent 3D vision works [6, 2, 34] often adopt a variational auto-encoder (VAE) learning scheme, in which bottleneck features are perturbed with Gaussian noise to encourage smooth and complete latent spaces. The regularization on the latent vectors enables exploring the embedding space with gradient descent or random sampling.

**Optimizing Latent Vectors.** Instead of using the full auto-encoder for representation learning, an alternative is to learn compact data representations by training *decoder-only* networks. This idea goes back to at least the work of Tan et al. [50] which simultaneously optimizes the latent vectors assigned to each data point and the decoder weights through back-propagation. For inference, an optimal latent vector is searched to match the new observation with fixed decoder parameters. Similar approaches have been exten-

sively studied in [42, 8, 40], for applications including noise reduction, missing measurement completions, and fault detections. Recent approaches [7, 20] extend the technique by applying deep architectures. Throughout the paper we refer to this class of networks as *auto-decoders*, for they are trained with *self*-reconstruction loss on decoder-only architectures.

## 2.3. Shape Completion

3D shape completion related works aim to infer unseen parts of the original shape given sparse or partial input observations. This task is anaologous to image-inpainting in 2D computer vision.

Classical surface reconstruction methods complete a point cloud into a dense surface by fitting radial basis function (RBF) [11] to approximate implicit surface functions, or by casting the reconstruction from oriented point clouds as a Poisson problem [32]. These methods only model a single shape rather than a dataset.

Various recent methods use data-driven approaches for the 3D completion task. Most of these methods adopt encoder-decoder architectures to reduce partial inputs of occupancy voxels [56], discrete SDF voxels [16], depth maps [44], RGB images [14, 55] or point clouds [49] into a latent vector and subsequently generate a prediction of full volumetric shape based on learned priors.

## 3. Modeling SDFs with Neural Networks

In this section we present DeepSDF, our continuous shape learning approach. We describe modeling shapes as the zero iso-surface decision boundaries of feed-forward networks trained to represent SDFs. A signed distance function is a continuous function that, for a given spatial point, outputs the point's distance to the closest surface, whose sign encodes whether the point is inside (negative) or outside (positive) of the watertight surface:

$$SDF(\boldsymbol{x}) = s : \boldsymbol{x} \in \mathbb{R}^3, \, s \in \mathbb{R} \,. \tag{1}$$

The underlying surface is implicitly represented by the iso-surface of $SDF(\cdot) = 0$. A view of this implicit surface can be rendered through raycasting or rasterization of a mesh obtained with, for example, Marching Cubes [35].

Our key idea is to directly regress the continuous SDF from point samples using deep neural networks. The resulting trained network is able to predict the SDF value of a given query position, from which we can extract the zero level-set surface by evaluating spatial samples. Such surface representation can be intuitively understood as a learned binary classifier for which the decision boundary is the surface of the shape itself as depicted in Fig. 2. As a universal function approximator [27], deep feed-forward networks in theory can learn the fully continuous shape

functions with arbitrary precision. Yet, the precision of the approximation in practice is limited by the finite number of point samples that guide the decision boundaries and the finite capacity of the network due to restricted compute power.

The most direct application of this approach is to train a single deep network for a given target shape as depicted in Fig. 3a. Given a target shape, we prepare a set of pairs $X$ composed of 3D point samples and their SDF values:

$$X := \{(\boldsymbol{x}, s) : SDF(\boldsymbol{x}) = s\}. \tag{2}$$

We train the parameters $\theta$ of a multi-layer fully-connected neural network $f_\theta$ on the training set $S$ to make $f_\theta$ a good approximator of the given SDF in the target domain $\Omega$:

$$f_\theta(\boldsymbol{x}) \approx SDF(\boldsymbol{x}), \forall \boldsymbol{x} \in \Omega. \tag{3}$$

The training is done by minimizing the sum over losses between the predicted and real SDF values of points in $X$ under the following $L_1$ loss function:

$$\mathcal{L}(f_\theta(\boldsymbol{x}), s) = |\operatorname{clamp}(f_\theta(\boldsymbol{x}), \delta) - \operatorname{clamp}(s, \delta)|, \tag{4}$$

where $\operatorname{clamp}(x, \delta) := \min(\delta, \max(-\delta, x))$ introduces the parameter $\delta$ to control the distance from the surface over which we expect to maintain a metric SDF. Larger values of $\delta$ allow for fast ray-tracing since each sample gives information of safe step sizes. Smaller values of $\delta$ can be used to concentrate network capacity on details near the surface.

To generate the 3D model shown in Fig. 3a, we use $\delta = 0.1$ and a feed-forward network composed of eight fully connected layers, each of them applied with dropouts. All internal layers are 512-dimensional and have ReLU non-linearities. The output non-linearity regressing the SDF value is tanh. We found training with batch-normalization [28] to be unstable and applied the weight-normalization technique instead [46]. For training, we use the Adam optimizer [33]. Once trained, the surface is implicitly represented as the zero iso-surface of $f_\theta(\boldsymbol{x})$, which can be visualized through raycasting or marching cubes. Another nice property of this approach is that accurate normals can be analytically computed by calculating the spatial derivative $\partial f_\theta(\boldsymbol{x})/\partial \boldsymbol{x}$ via back-propogation through the network.

## 4. Learning the Latent Space of Shapes

Training a specific neural network for each shape is neither feasible nor very useful. Instead, we want a model that can represent a wide variety of shapes, discover their common properties, and embed them in a low dimensional latent space. To this end, we introduce a latent vector $\boldsymbol{z}$, which can be thought of as encoding the desired shape, as a second input to the neural network as depicted in Fig. 3b. Conceptually, we map this latent vector to a 3D shape represented
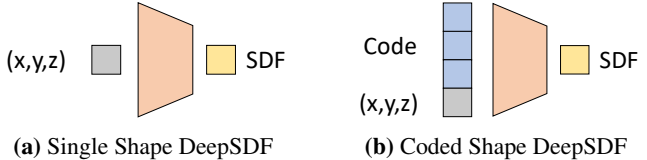


**(a)** Single Shape DeepSDF  **(b)** Coded Shape DeepSDF

**Figure 3:** In the single-shape DeepSDF instantiation, the shape information is contained in the network itself whereas the coded-shape DeepSDF, the shape information is contained in a code vector that is concatenated with the 3D sample location. In both cases, DeepSDF produces the SDF value at the 3D query location,
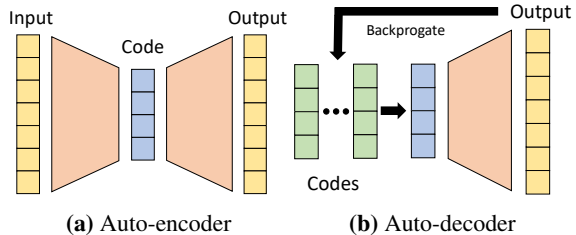


**(a)** Auto-encoder  **(b)** Auto-decoder

**Figure 4:** Different from an auto-encoder whose latent code is produced by the encoder, an auto-decoder directly accepts a latent vector as an input. A randomly initialized latent vector is assigned to each data point in the beginning of training, and the latent vectors are optimized along with the decoder weights through standard backpropagation. During inference, decoder weights are fixed, and an optimal latent vector is estimated.

by a continuous SDF. Formally, for some shape indexed by $i$, $f_\theta$ is now a function of a latent code $\boldsymbol{z}_i$ and a query 3D location $\boldsymbol{x}$, and outputs the shape's approximate SDF:

$$f_\theta(\boldsymbol{z}_i, \boldsymbol{x}) \approx SDF^i(\boldsymbol{x}). \tag{5}$$

By conditioning the network output on a latent vector, this formulation allows modeling multiple SDFs with a single neural network. Given the decoding model $f_\theta$, the continuous surface associated with a latent vector $\boldsymbol{z}$ is similarly represented with the decision boundary of $f_\theta(\boldsymbol{z}, \boldsymbol{x})$, and the shape can again be discretized for visualization by, for example, raycasting or Marching Cubes.

Next, we motivate the use of encoder-less training before introducing the 'auto-decoder' formulation of the shape-coded DeepSDF.

### 4.1. Motivating Encoder-less Learning

Auto-encoders and encoder-decoder networks are widely used for representation learning as their bottleneck features tend to form natural latent variable representations.

Recently, in applications such as modeling depth maps [6], faces [2], and body shapes [34] a full auto-encoder is trained but only the decoder is retained for inference, where they search for an optimal latent vector given some input observation. However, since the trained encoder is unused

at test time, it is unclear whether using the encoder is the most effective use of computational resources during training. This motivates us to use an *auto-**decoder*** for learning a shape embedding without an encoder as depicted in Fig. 4.

We show that applying an auto-decoder to learn continuous SDFs leads to high quality 3D generative models. Further, we develop a probabilistic formulation for training and testing the auto-decoder that naturally introduces latent space regularization for improved generalization. To the best of our knowledge, this work is the first to introduce the auto-decoder learning method to the 3D learning community.

## 4.2. 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:

$$X_i = \{(\boldsymbol{x}_j, s_j) : s_j = SDF^i(\boldsymbol{x}_j)\}. \quad (6)$$

For an auto-decoder, as there is no encoder, each latent code $\boldsymbol{z}_i$ is paired with training shape $X_i$. The posterior over shape code $\boldsymbol{z}_i$ given the shape SDF samples $X_i$ can be decomposed as:

$$p_\theta(\boldsymbol{z}_i|X_i) = p(\boldsymbol{z}_i) \prod_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X_i} p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j), \quad (7)$$

where $\theta$ parameterizes the SDF likelihood. 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 $\sigma^2 I$. This prior encapsulates the notion that the shape codes should be concentrated and we empirically found it was needed to infer a compact shape manifold and to help converge to good solutions.

In the auto-decoder-based DeepSDF formulation we express the SDF likelihood via a deep feed-forward network $f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j)$ and, without loss of generality, assume that the likelihood takes the form:

$$p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j) = \exp(-\mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j)). \quad (8)$$

The SDF prediction $\tilde{s}_j = f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j)$ is represented using a fully-connected network. $\mathcal{L}(\tilde{s}_j, s_j)$ 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 from Eq. 4 for reasons outlined previously.

At training time we maximize the joint log posterior over all training shapes with respect to the individual shape codes $\{z_i\}_{i=1}^N$ and the network parameters $\theta$:

$$\arg\min_{\theta,\{\boldsymbol{z}_i\}_{i=1}^N} \sum_{i=1}^{N} \left( \sum_{j=1}^{K} \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}_i||_2^2 \right). \quad (9)$$
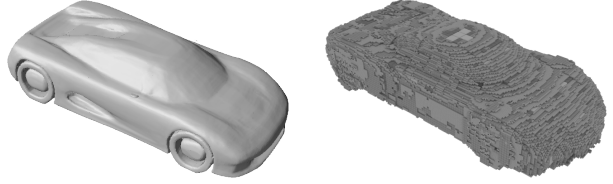


**Figure 5:** Compared to car shapes memorized using OGN [52] (right), our models (left) preserve details and render visually pleasing results as DeepSDF provides oriented surace normals.

At inference time, after training and fixing $\theta$, a shape code $\boldsymbol{z}_i$ for shape $X_i$ can be estimated via Maximum-a-Posterior (MAP) estimation as:

$$\hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \sum_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X} \mathcal{L}(f_\theta(\boldsymbol{z}, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}||_2^2. \quad (10)$$

Crucially, this formulation is valid for SDF samples $X$ of arbitrary size and distribution because the gradient of the loss with respect to $\boldsymbol{z}$ can be computed separately for each SDF sample. This implies that DeepSDF can handle any form of partial observations such as depth maps. This is a major advantage over the auto-encoder framework whose encoder expects a test input similar to the training data, e.g. shape completion networks of [16, 58] require preparing training data of partial shapes.

To incorporate the latent shape code, we stack the code vector and the sample location as depicted in Fig. 3b and feed it into the same fully-connected NN described previously at the input layer and additionally at the 4th layer. We again use the Adam optimizer [33]. The latent vector $\boldsymbol{z}$ is initialized randomly from $\mathcal{N}(0, 0.01^2)$.

Note that while both VAE and the proposed auto-decoder formulation share the zero-mean Gaussian prior on the latent codes, we found that the the stochastic nature of the VAE optimization did not lead to good training results.

## 5. Data Preparation

To train our continuous SDF model, we prepare the SDF samples $X$ (Eq. 2) for each mesh, which consists of 3D points and their SDF values. While SDF can be computed through a distance transform for any watertight shapes from real or synthetic data, we train with synthetic objects, (e.g. ShapeNet [12]), for which we are provided complete 3D shape meshes. To prepare data, we start by normalizing each mesh to a unit sphere and sampling 500,000 spatial points $\boldsymbol{x}$'s: we sample more aggressively near the surface of the object as we want to capture a more detailed SDF near the surface. For an ideal oriented watertight mesh, computing the signed distance value of $\boldsymbol{x}$ would only involve finding the closest triangle, but we find that human designed meshes are commonly not watertight and contain undesired internal structures. To obtain the *shell* of a

| Method | Type | Discretization | Complex topologies | Closed surfaces | Surface normals | Model size (GB) | Inf. time (s) | Eval. tasks |
|---|---|---|---|---|---|---|---|---|
| 3D-EPN [16] | Voxel SDF | $32^3$ voxels | ✓ | ✓ | ✓ | 0.42 | - | C |
| OGN [52] | Octree | $256^3$ voxels | ✓ | ✓ | | 0.54 | 0.32 | K |
| AtlasNet-Sphere [22] | Parametric mesh | 1 patch | | ✓ | | 0.015 | **0.01** | K, U |
| AtlasNet-25 [22] | Parametric mesh | 25 patches | ✓ | | | 0.172 | 0.32 | K, U |
| DeepSDF (ours) | Continuous SDF | **none** | ✓ | ✓ | ✓ | **0.0074** | 9.72 | K, U, C |

**Table 1:** Overview of the benchmarked methods. AtlasNet-Sphere can only describe topological-spheres, voxel/octree occupancy methods (i.e. OGN) only provide 8 directions for normals, and AtlasNet does not provide oriented normals. Our tasks evaluated are: (K) representing known shapes, (U) representing unknown shapes, and (C) shape completion.

mesh with proper orientation, we set up equally spaced virtual cameras around the object, and densely sample surface points, denoted $P_s$, with surface normals oriented towards the camera. Double sided triangles visible from both orientations (indicating that the shape is not closed) cause problems in this case, so we discard mesh objects containing too many of such faces. Then, for each $x$, we find the closest point in $P_s$, from which the $SDF(x)$ can be computed. We refer readers to supplementary material for further details.

## 6. Results

We conduct a number of experiments to show the representational power of DeepSDF, both in terms of its ability to describe geometric details and its generalization capability to learn a desirable shape embedding space. Largely, we propose four main experiments designed to test its ability to 1) represent training data, 2) use learned feature representation to reconstruct unseen shapes, 3) apply shape priors to complete partial shapes, and 4) learn smooth and complete shape embedding space from which we can sample new shapes. For all experiments we use the popular ShapeNet [12] dataset.

We select a representative set of 3D learning approaches to comparatively evaluate aforementioned criteria: a recent octree-based method (OGN) [52], a mesh-based method (AtlasNet) [22], and a volumetric SDF-based shape completion method (3D-EPN) [16] (Table 1). These works show state-of-the-art performance in their respective representations and tasks, so we omit comparisons with the works that have already been compared: e.g. OGN's octree model outperforms regular voxel approaches, while AtlasNet compares itself with various points, mesh, or voxel based methods and 3D-EPN with various completion methods.

### 6.1. Representing Known 3D Shapes

First, we evaluate the capacity of the model to represent *known* shapes, i.e. shapes that were in the training set, from only a restricted-size latent code — testing the limit of expressive capability of the representations.

| Method \metric | CD, mean | CD, median | EMD, mean | EMD, median |
|---|---|---|---|---|
| OGN | 0.167 | 0.127 | **0.043** | **0.042** |
| AtlasNet-Sph. | 0.210 | 0.185 | 0.046 | 0.045 |
| AtlasNet-25 | 0.157 | 0.140 | 0.060 | 0.060 |
| DeepSDF | **0.084** | **0.058** | **0.043** | **0.042** |

**Table 2:** Comparison for representing known shapes (K) for cars trained on ShapeNet. CD = Chamfer Distance ($30,000$ points) multiplied by $10^3$, EMD = Earth Mover's Distance (500 points).

Quantitative comparison in Table 2 shows that the proposed DeepSDF significantly beats OGN and AtlasNet in Chamfer distance against the true shape computed with a large number of points (30,000). The difference in earth mover distance (EMD) is smaller because 500 points do not well capture the additional precision. Figure 5 shows a qualitative comparison of DeepSDF to OGN.

### 6.2. Representing Test 3D Shapes (auto-encoding)

For encoding *unknown* shapes, i.e. shapes in the held-out test set, DeepSDF again significantly outperforms AtlasNet on a wide variety of shape classes and metrics as shown in Table 3. Note that AtlasNet performs reasonably well at classes of shapes that have mostly consistent topology without holes (like planes) but struggles more on classes that commonly have holes, like chairs. This is shown in Fig. 6 where AtlasNet fails to represent the fine detail of the back of the chair. Figure 7 shows more examples of detailed reconstructions on test data from DeepSDF as well as two example failure cases.

### 6.3. Shape Completion

A major advantage of the proposed DeepSDF approach for representation learning is that inference can be performed from an arbitrary number of SDF samples. In the DeepSDF framework, shape completion amounts to solving for the shape code that best explains a partial shape observation via Eq. 10. Given the shape-code a complete shape can be rendered using the priors encoded in the decoder.
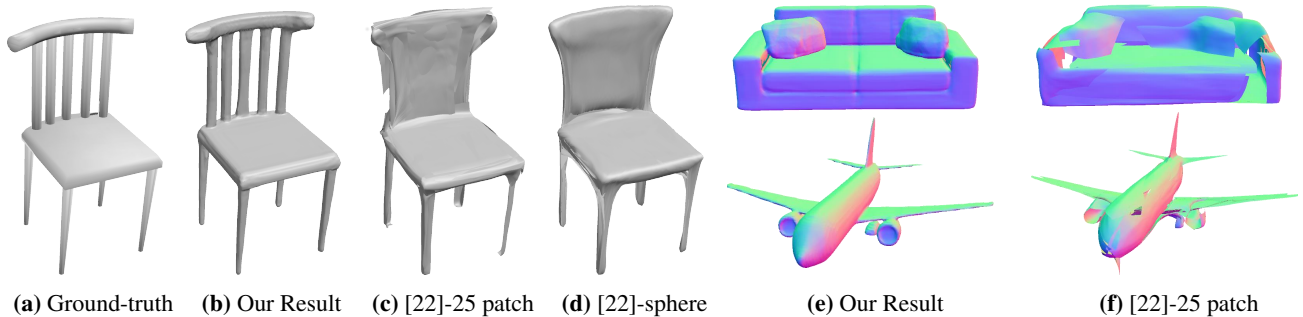
**(a)** Ground-truth    **(b)** Our Result    **(c)** [22]-25 patch    **(d)** [22]-sphere    **(e)** Our Result    **(f)** [22]-25 patch

**Figure 6:** Reconstruction comparison between DeepSDF and AtlasNet [22] (with 25-plane and sphere parameterization) for test shapes. Note that AtlasNet fails to capture the fine details of the chair, and that (f) shows holes on the surface of sofa and the plane.



**Figure 7:** Reconstruction of test shapes. From left to right alternating: ground truth shape and our reconstruction. The two right most columns show failure modes of DeepSDF. These failures are likely due to lack of training data and failure of minimization convergence.

| CD, mean | chair | plane | table | lamp | sofa |
|---|---|---|---|---|---|
| AtlasNet-Sph. | 0.752 | 0.188 | 0.725 | 2.381 | 0.445 |
| AtlasNet-25 | 0.368 | 0.216 | **0.328** | 1.182 | 0.411 |
| DeepSDF | **0.204** | **0.143** | 0.553 | **0.832** | **0.132** |
| CD, median | | | | | |
| AtlasNet-Sph. | 0.511 | 0.079 | 0.389 | 2.180 | 0.330 |
| AtlasNet-25 | 0.276 | 0.065 | 0.195 | 0.993 | 0.311 |
| DeepSDF | **0.072** | **0.036** | **0.068** | **0.219** | **0.088** |
| EMD, mean | | | | | |
| AtlasNet-Sph. | 0.071 | 0.038 | 0.060 | 0.085 | 0.050 |
| AtlasNet-25 | 0.064 | 0.041 | 0.073 | 0.062 | 0.063 |
| DeepSDF | **0.049** | **0.033** | **0.050** | **0.059** | **0.047** |
| Mesh acc., mean | | | | | |
| AtlasNet-Sph. | 0.033 | 0.013 | 0.032 | 0.054 | 0.017 |
| AtlasNet-25 | 0.018 | 0.013 | 0.014 | 0.042 | 0.017 |
| DeepSDF | **0.009** | **0.004** | **0.012** | **0.013** | **0.004** |

**Table 3:** Comparison for representing unknown shapes (U) for various classes of ShapeNet. Mesh accuracy as defined in [47] is the minimum distance $d$ such that 90% of generated points are within $d$ of the ground truth mesh. Lower is better for all metrics.

| Method \Metric | *lower is better* | | | | *higher is better* | |
|---|---|---|---|---|---|---|
| | CD, med. | CD, mean | EMD | Mesh acc. | Mesh comp. | Cos sim. |
| chair | | | | | | |
| 3D-EPN | 2.25 | 2.83 | 0.084 | 0.059 | 0.209 | 0.752 |
| DeepSDF | **1.28** | **2.11** | **0.071** | **0.049** | **0.500** | **0.766** |
| plane | | | | | | |
| 3D-EPN | 1.63 | 2.19 | 0.063 | 0.040 | 0.165 | 0.710 |
| DeepSDF | **0.37** | **1.16** | **0.049** | **0.032** | **0.722** | **0.823** |
| sofa | | | | | | |
| 3D-EPN | 2.03 | 2.18 | 0.071 | 0.049 | 0.254 | 0.742 |
| DeepSDF | **0.82** | **1.59** | **0.059** | **0.041** | **0.541** | **0.810** |

**Table 4:** Comparison for shape completion (C) from partial range scans of unknown shapes from ShapeNet.

We test our completion scheme using single view depth observations which is a common use-case and maps well to our architecture without modification. Note that we currently require the depth observations in the canonical shape frame of reference.

To generate SDF point samples from the depth image observation, we sample two points for each depth observation, each of them located $\eta$ distance away from the measured surface point (along surface normal estimate). With small $\eta$ we approximate the signed distance value of those points to be $\eta$ and $-\eta$, respectively. We solve for Eq. 10 with loss function of Eq. 4 using clamp value of $\eta$. Additionally, we incorporate free-space observations, (i.e. empty-space between surface and camera), by sampling points along the freespace-direction and enforce larger-than-zero constraints. The freespace loss is $|f_\theta(z, x_j)|$ if $f_\theta(z, x_j) < 0$ and 0 otherwise.

Given the SDF point samples and empty space points, we similarly optimize the latent vector using MAP estimation. Tab. 4 and Figs. (22, 9) respectively shows quantitative and qualitative shape completion results. Compared to one of the most recent completion approaches [16] using volu-
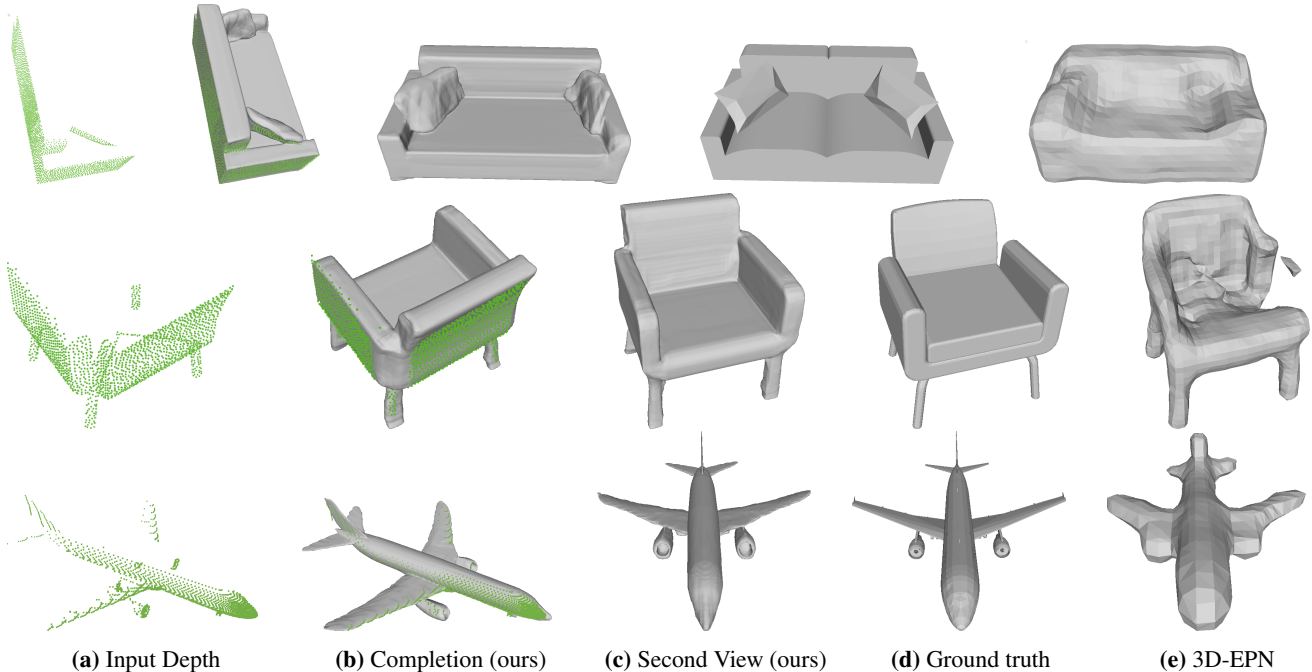
| **(a)** Input Depth | **(b)** Completion (ours) | **(c)** Second View (ours) | **(d)** Ground truth | **(e)** 3D-EPN |
| --- | --- | --- | --- | --- |

**Figure 8:** For a given depth image visualized as a green point cloud, we show a comparison of shape completions from our DeepSDF approach against the true shape and 3D-EPN.
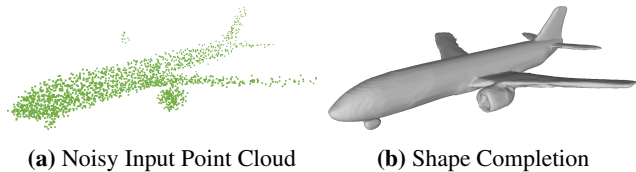


| **(a)** Noisy Input Point Cloud | **(b)** Shape Completion |
| --- | --- |

**Figure 9:** Demonstration of DeepSDF shape completion from a partial noisy point cloud. Input here is generated by perturbing the 3D point cloud positions generated by the ground truth depth map by 3% of the plane length. We provide a comprehensive analysis of robustness to noise in the supplementary material.

metric shape representation, our continuous SDF approach produces more visually pleasing and accurate shape reconstructions. While a few recent shape completion methods were presented [24, 55], we could not find the code to run the comparisons, and their underlying 3D representation is voxel grid which we extensively compare against.

### 6.4. Latent Space Shape Interpolation

To show that our learned shape embedding is complete and continuous, we render the results of the decoder when a pair of shapes are interpolated in the latent vector space (Fig. 1). The results suggests that the embedded continuous SDF's are of meaningful shapes and that our representation extracts common interpretable shape features, such as the arms of a chair, that interpolate linearly in the latent space.

## 7. Conclusion & Future Work

DeepSDF significantly outperforms the applicable benchmarked methods across shape representation and completion tasks and simultaneously addresses the goals of representing complex topologies, closed surfaces, while providing high quality surface normals of the shape. However, while point-wise forward sampling of a shape's SDF is efficient, shape completion (auto-decoding) takes considerably more time during inference due to the need for explicit optimization over the latent vector. We look to increase performance by replacing ADAM optimization with more efficient Gauss-Newton or similar methods that make use of the analytic derivatives of the model.

DeepSDF models enable representation of more complex shapes without discretization errors with significantly less memory than previous state-of-the-art results as shown in Table 1, demonstrating an exciting route ahead for 3D shape learning. The clear ability to produce quality latent shape space interpolation opens the door to reconstruction algorithms operating over scenes built up of such efficient encodings. However, DeepSDF currently assumes models are in a canonical pose and as such completion in-the-wild requires explicit optimization over a $SE(3)$ transformation space increasing inference time. Finally, to represent the true *space-of-possible-scenes* including dynamics and textures in a single embedding remains a major challenge, one which we continue to explore.

# Supplementary

## 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. E we compare auto-decoders with variational and standard auto-encoders. Further, additional details on data preparation (Sec. F), training (Sec. G), 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 [25], 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:

$$D_{\text{noise}} = \frac{1}{(1/D) + \mathcal{N}(0, \alpha^2)}, \tag{11}$$

where $\alpha$ is standard deviation of the normal distribution.

For the experiment, we synthetically generate noisy depth maps from the ShapeNet [12] plane models using the same benchmark test set of Dai et al. [16] used in the main paper. We perturb the depth values using standard deviation $\alpha$ 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. 10).

The shape completion results with respect to added Guassian noise on the input synthetic depth maps are shown in Fig. 11. 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. 11), 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.

## C. Network Architecture

Fig. 13 depicts the overall architecture of DeepSDF. For all experiments in the main paper we used a network com-
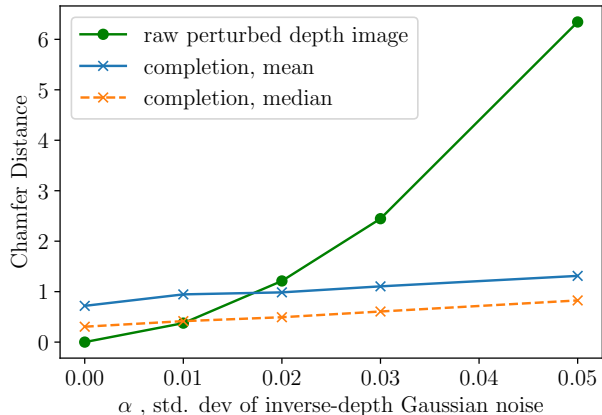


**Figure 10:** Chamfer distance (multiplied by $10^3$) as a function of $\alpha$, the standard deviation of inverse-depth Gaussian noise as shown in Eq. 11, for shape completions on planes from ShapeNet. Green line describes Chamfer distance between the perturbed depth points and original depth points, 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. 11.

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. 13, we find
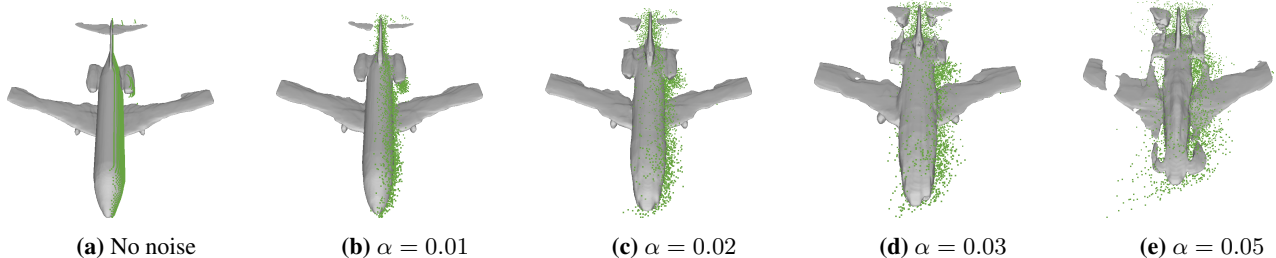
**(a)** No noise      **(b)** $\alpha = 0.01$      **(c)** $\alpha = 0.02$      **(d)** $\alpha = 0.03$      **(e)** $\alpha = 0.05$

**Figure 11:** 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.
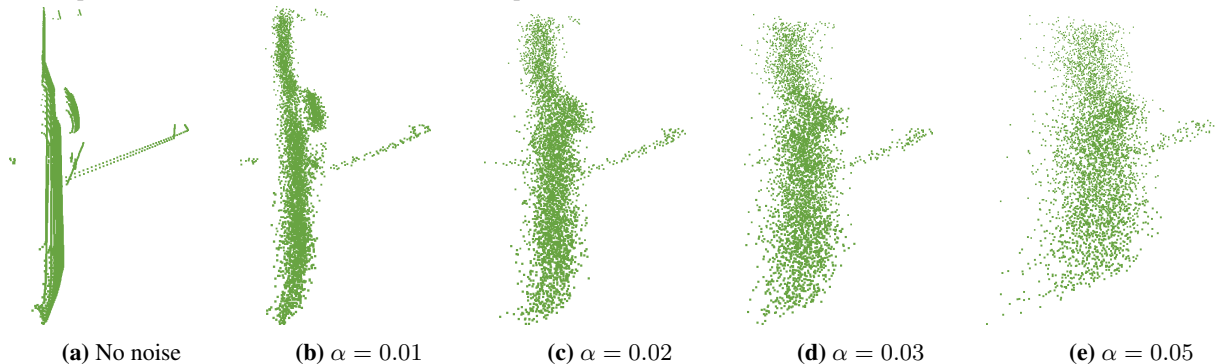


**(a)** No noise      **(b)** $\alpha = 0.01$      **(c)** $\alpha = 0.02$      **(d)** $\alpha = 0.03$      **(e)** $\alpha = 0.05$

**Figure 12:** Visualization of partial and noisy point-clouds used to test shape completion with *DeepSDF*. Here, $\alpha$ is the standard deviation of Gaussian noise in Eq. 11. Corresponding completion results are shown above.
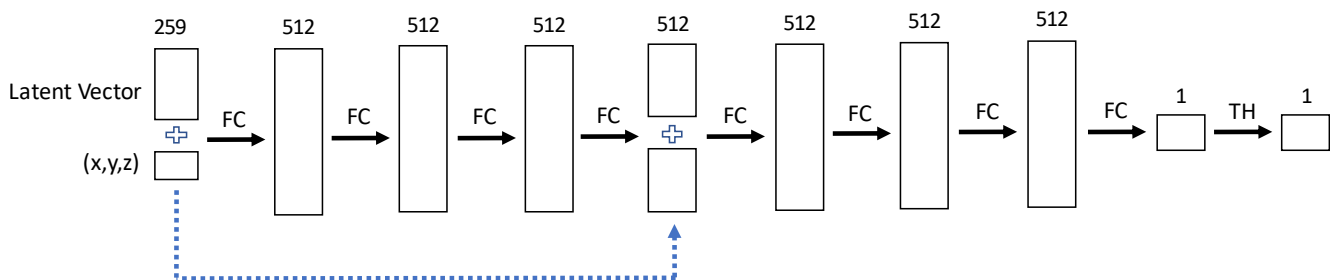


**Figure 13:** DeepSDF architecture used for experiments. Boxes represent vectors while arrows represent operations. The feed-forward network is composed of 8 fully connected layers, denoted as "FC" on the diagram. We used 256 and 128 dimensional latent vectors for reconstruction and shape completion 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".

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. 14 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 archi-
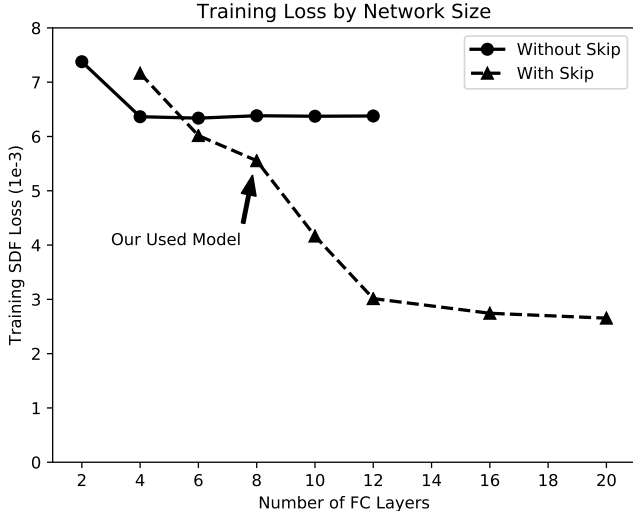
**Figure 14:** Regression accuracy (measured by the SDF loss used in training) as a function of network depth. Without skip connections, we observe a plateau in training loss past 4 layers. With skip connections, training loss continues to decrease although with diminishing returns past 12 layers. The model size chosen for all other experiments, 8 layers, provides a good tradeoff between speed and accuracy.



**Figure 15:** Chamfer distance (multiplied by $10^3$) as a function of $\delta$, the truncation distance, for representing a known small dataset of 100 cars from ShapeNet dataset [12]. 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.

tectures 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 study the effect of the truncation distance ($\delta$ from Eq. 4 of the manuscript) on the regression accuracy of the model. The truncation distance controls the extent from the surface over which we expect the network to learn a metric SDF. Fig. 15 plots the 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 $\delta$ value of 0.01 for all experiments presented in the manuscript, which provides a good tradeoff between raycasting speed and surface accuracy.

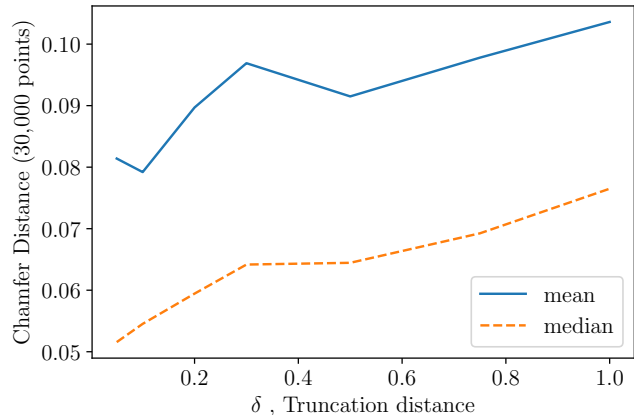### E. Comparison with Variational and Standard Auto-encoders on MNIST

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. 16. 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. 17. 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. 18. For VAE and AE we show both the reconstruction error obtained using the learned encoder and obtained via code optimization using
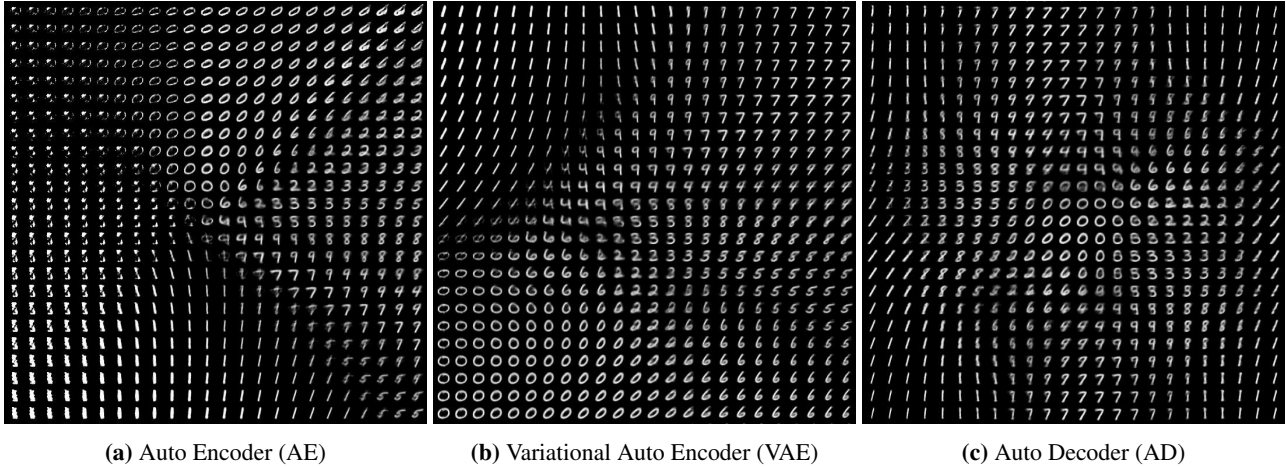
**(a)** Auto Encoder (AE)  **(b)** Variational Auto Encoder (VAE)  **(c)** Auto Decoder (AD)

**Figure 16:** 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.
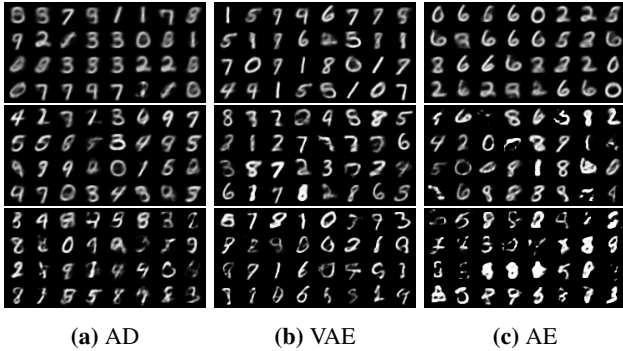


**(a)** AD  **(b)** VAE  **(c)** AE

**Figure 17:** 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.

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. 19 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.

## F. 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 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
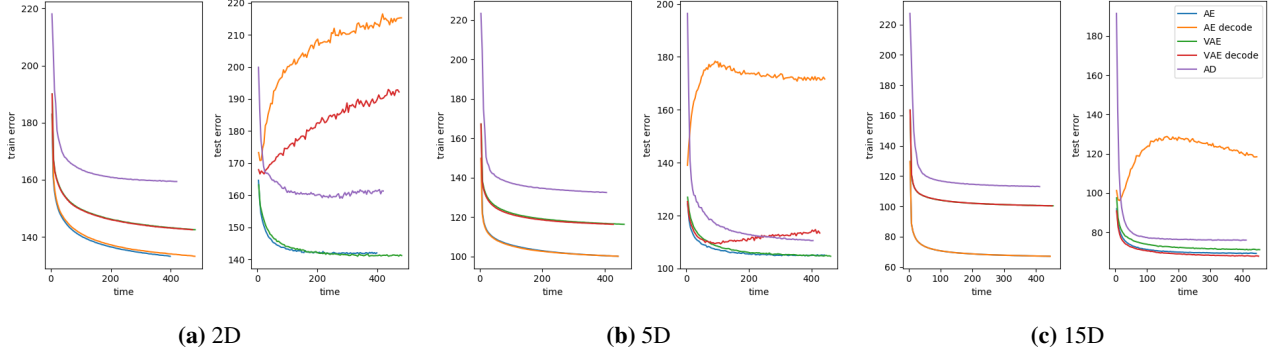
**(a)** 2D      **(b)** 5D      **(c)** 15D

**Figure 18:** 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
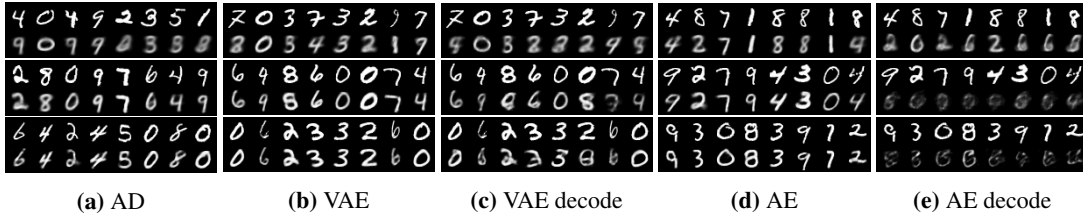
**Figure 19:** Reconstructions for 2D (top), 5D (middle), and 15D (bottom) code space on MNIST. For each of the different dimensions we plot the given test MNIST image and the reconstruction given the inferred latent code.

from the KD-tree, measure the distance, and decide the sign from the dot product between the normal and their vector difference.

## G. Training and Testing Details

For training, we find it is 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 subsampled 16384 SDF samples. 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.

## 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 func-

tion $SDF^i{}_{i=1}^N$, we prepare a set of $K$ point samples and their signed distance values:

$$X_i = \{(\boldsymbol{x}_j, s_j) : s_j = SDF^i(\boldsymbol{x}_j)\}. \quad (12)$$

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 $\theta$.

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

$$p_\theta(X_i, \boldsymbol{z}_i) = p_\theta(X_i|\boldsymbol{z}_i)p(\boldsymbol{z}_i), \quad (13)$$

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

$$\hat{\boldsymbol{z}}_i = \arg\max_{\boldsymbol{z}_i} p_\theta(\boldsymbol{z}_i|X_i) = \arg\max_{\boldsymbol{z}_i} \log p_\theta(\boldsymbol{z}_i|X_i). \quad (14)$$

We estimate $\theta$ as the parameters that maximizes the posterior across all shapes:

$$\hat{\theta} = \arg\max_{\theta} \sum_{X_i \in \boldsymbol{X}} \max_{\boldsymbol{z}_i} \log p_\theta(\boldsymbol{z}_i|X_i) \quad (15)$$

$$= \arg\max_{\theta} \sum_{X_i \in \boldsymbol{X}} \max_{\boldsymbol{z}_i} (\log p_\theta(X_i|\boldsymbol{z}_i) + \log p(\boldsymbol{z}_i)),$$

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. 12 we make a conditional independence assumption given the code $z_i$ to arrive at the decomposition of the posterior $p_\theta(X_i|z_i)$:

$$p_\theta(X_i|z_i) = \prod_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X_i} p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j). \qquad (16)$$

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, assume that the likelihood takes the form:

$$p_\theta(\boldsymbol{s}_j|z_i; \boldsymbol{x}_j) = \exp(-\mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j)). \qquad (17)$$

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. 15 which we jointly minimize with respect to the network parameters $\theta$ and the shape codes $\{z_i\}_{i=1}^N$:

$$\underset{\theta, \{\boldsymbol{z}_i\}_{i=1}^N}{\arg\min} \sum_{i=1}^N \left( \sum_{j=1}^K \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}_i||_2^2 \right). \quad (18)$$

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. 14 with fixed network parameters $\theta$ we arrive at:

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z}}{\arg\min} \sum_{(\boldsymbol{x}_j, \boldsymbol{s}_j) \in X} \mathcal{L}(f_\theta(\boldsymbol{z}, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}||_2^2, \quad (19)$$

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 [35] 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. G.

#### 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 [19]. 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 $\sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2$) 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 [19]. 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 [22] we normalize by the number of points: we report $\frac{d_{CD}(S_1, S_2)}{30,000}$.

**Earth Mover's distance** [45], 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 $\phi : S_1 \rightarrow S_2$, 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 [19]:

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

Although the metric is commonly approximated in the deep learning literature [19] by distributed approximation schemes [5] for speed during training, we compute the metric accurately for evaluation using a more modest number of point samples (500) using [18].

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 [47], 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 [47], is the fraction of points sampled from the ground truth mesh that are within some distance $\Delta$ (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 $F_i$,

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

| 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 5:** Comparison of metrics for representing unknown shapes (U) for various classes of ShapeNet. Mesh completion as defined in [47] i.e. the fraction of groundtruth sampled points that are within a $\Delta$ (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.

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 [22] 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. 20, 21). We provide additional data with the additional metrics, mesh completion and mesh cosine similarity, for the comparison of methods contained in the manuscript (Tab. 5). 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.

## J.2. Shape Completions

Finally we present additional shape completion results on unperturbed depth images of synthetic ShapeNet dataset (Fig. 22), demonstrating the quality of the auto-decoder learning scheme and the new shape representation.

# References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. 2018.

[2] T. Bagautdinov, C. Wu, J. Saragih, P. Fua, and Y. Sheikh. Modeling facial geometry using compositional vaes. 1:1.

[3] P. Baqué, E. Remelli, F. Fleuret, and P. Fua. Geodesic convolutional shape optimization. *arXiv preprint arXiv:1802.04016*, 2018.

[4] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828, 2013.

[5] 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.

[6] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison. Codeslam-learning a compact, optimisable representation for dense visual slam. *arXiv preprint arXiv:1804.00874*, 2018.

[7] P. Bojanowski, A. Joulin, D. Lopez-Pas, and A. Szlam. Optimizing the latent space of generative networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 600–609. PMLR, 10–15 Jul 2018.

[8] M. Bouakkaz and M.-F. Harkat. Combined input training and radial basis function neural networks based nonlinear principal components analysis model applied for process monitoring. In *IJCCI*, pages 483–492, 2012.

[9] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[10] D. R. Canelhas, E. Schaffernicht, T. Stoyanov, A. J. Lilienthal, and A. J. Davison. An eigenshapes approach to compressed signed distance fields and their utility in robot mapping. *arXiv preprint arXiv:1609.02462*, 2016.

[11] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*, pages 67–76. ACM, 2001.

[12] 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.

[13] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, pages 2172–2180, 2016.

[14] 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 *ECCV*, pages 628–644. Springer, 2016.

[15] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312. ACM, 1996.

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

[17] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016.

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

**Figure 20:** Additional test shape reconstruction results. Left to right alternatingly: DeepSDF reconstruction and ground truth.
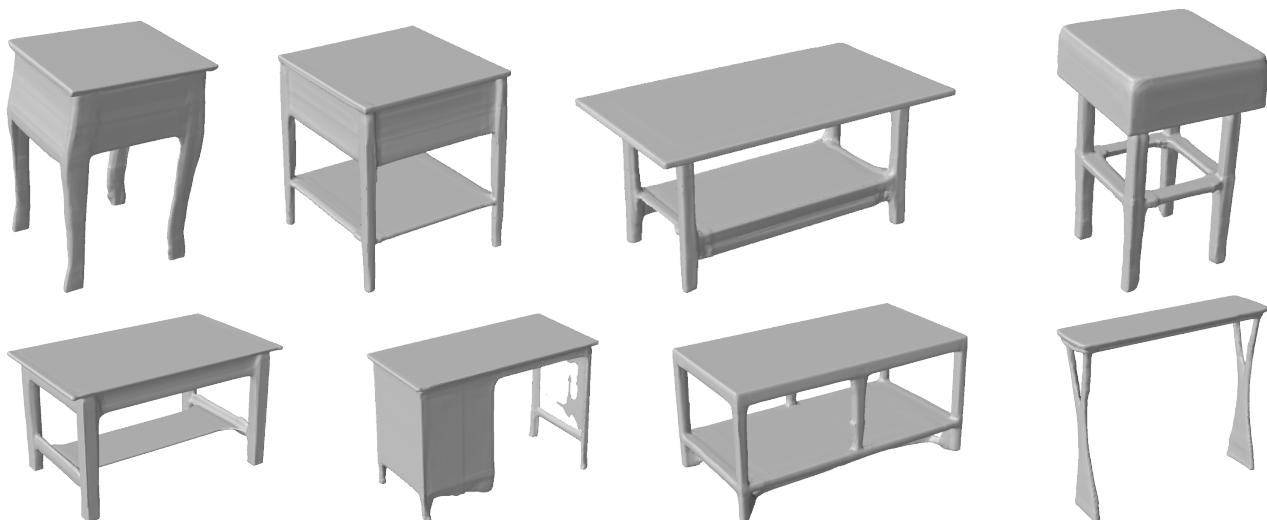


**Figure 21:** Additional test shape reconstruction results for Table ShapeNet class. All of the above images are test shapes represented with our DeepSDF network during inference time, showing the accuracy and expressiveness of the shape embedding.

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

[20] J. Fan and J. Cheng. Matrix completion by deep matrix factorization. *Neural Networks*, 98:34–41, 2018.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

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

[23] H. B. Hamu, H. Maron, I. Kezurer, G. Avineri, and Y. Lipman. Multi-chart generative surface modeling. *arXiv preprint arXiv:1806.02143*, 2018.

[24] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu. High-resolution shape completion using deep neural networks for global structure and local geometry inference.

[25] 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.

[26] C. Häne, S. Tulsiani, and J. Malik. Hierarchical surface prediction for 3d object reconstruction. In *3D Vision (3DV), 2017 International Conference on*, pages 412–420. IEEE, 2017.

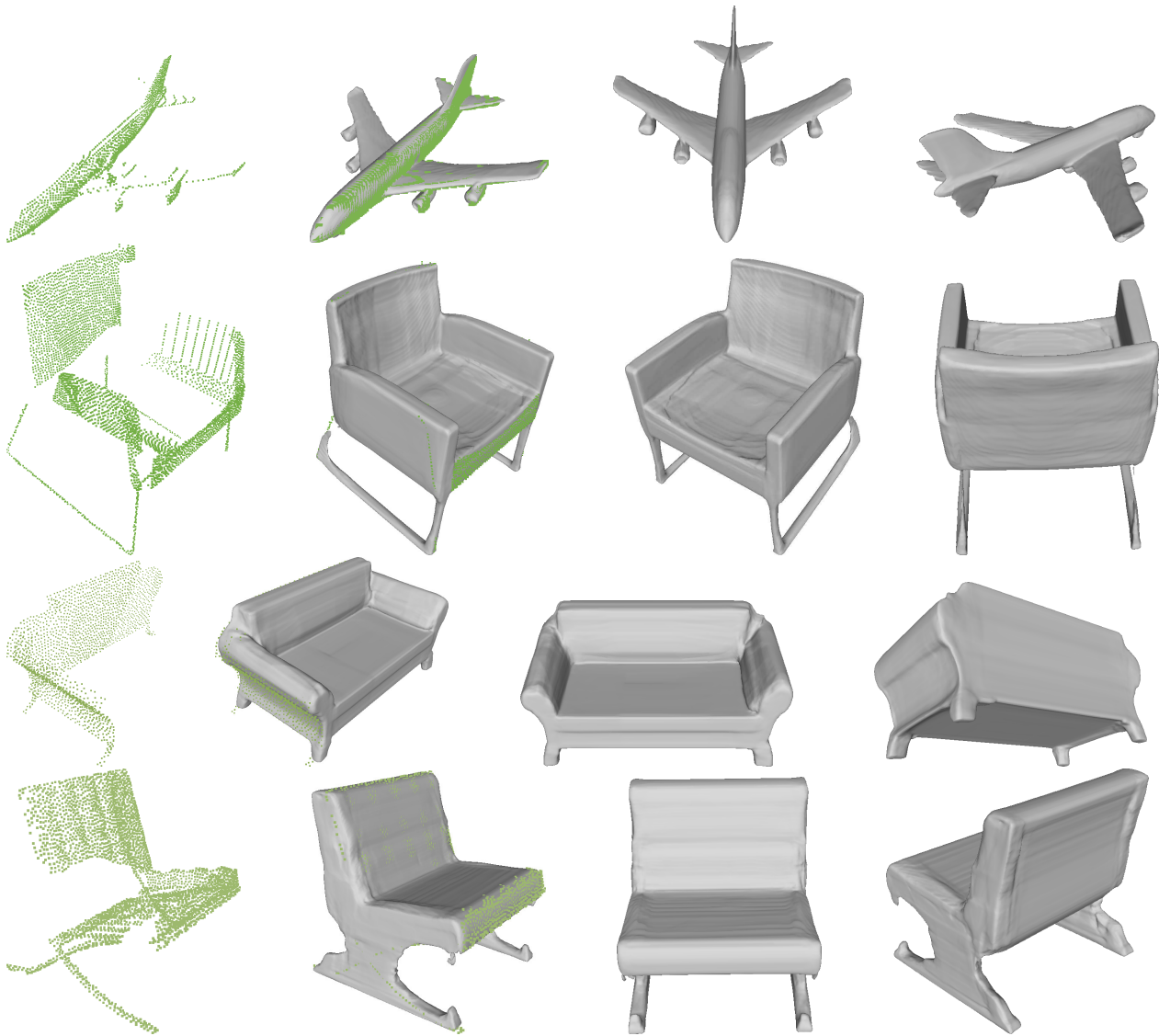[27] K. Hornik, M. Stinchcombe, and H. White. Multilayer feed-

**Figure 22:** Additional shape completion results. Left to Right: input depth point cloud, shape completion using DeepSDF, second view, and third view.

forward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[28] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[30] M. W. Jones. Distance field compression. *Journal of WSCG*, 12(2):199–204, 2004.

[31] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[32] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM TOG*, 32(3):29, 2013.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] O. Litany, A. Bronstein, M. Bronstein, and A. Makadia. Deformable shape completion with graph convolutional autoencoders. *CVPR*, 2017.

[35] 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.

[36] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. 2017.

[37] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136. IEEE, 2011.

[38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017.

[39] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, pages 5099–5108, 2017.

[40] Z. Qunxiong and L. Chengfei. Dimensionality reduction with input training neural network and its application in chemical process modelling. *Chinese Journal of Chemical Engineering*, 14(5):597–603, 2006.

[41] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[42] V. Reddy and M. Mavrovouniotis. An input-training neural network approach for gross error detection and sensor replacement. *Chemical Engineering Research and Design*, 76(4):478–489, 1998.

[43] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, pages 6620–6629. IEEE, 2017.

[44] J. Rock, T. Gupta, J. Thorsen, J. Gwak, D. Shin, and D. Hoiem. Completing 3d object shape from one depth image. In *CVPR*, pages 2484–2493, 2015.

[45] 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.

[46] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, pages 901–909, 2016.

[47] 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.

[48] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *ECCV*, pages 223–240. Springer, 2016.

[49] D. Stutz and A. Geiger. Learning 3d shape completion from laser scan data with weak supervision. In *CVPR*, pages 1955–1964, 2018.

[50] S. Tan and M. L. Mayrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6):1471–1480, 1995.

[51] M. Tarini, K. Hormann, P. Cignoni, and C. Montani. Polycube-maps. In *ACM TOG*, volume 23, pages 853–860. ACM, 2004.

[52] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *ICCV*, 2017.

[53] N. Verma, E. Boyer, and J. Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis.

[54] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, pages 82–90, 2016.

[55] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum. Learning shape priors for single-view 3d completion and reconstruction. *arXiv preprint arXiv:1809.05068*, 2018.

[56] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015.

[57] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Interpretable unsupervised learning on 3d point clouds. *arXiv preprint arXiv:1712.07262*, 2017.

[58] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *3DV*, 2018.

[59] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, pages 199–208. IEEE, 2017.