Signed Distance Functions for Shape Representation

Xiyao LI

April 19, 2023

1 Introduction

Traditional explicit methods for representing 3D meshes include mesh, voxel, and point cloud representations. Instead of using an explicit method, we implement an implicit representation, the Signed Distance Function (SDF) representation, based on the paper by Park et al. [1], which was published in 2019. Our objective is to rebuild the auto-decoder network for a batch of similar shapes and reproduce the shape construction task for the ShapeNet dataset [2]. The implementation code can be found in our GitHub repository[1].

In our collaborative project, I am primarily focused on the SDF deep learning model, while Andrei BUNEA is focused on the PointCleanNet denoising model. In this report, I will provide a more detailed study of the SDF model used in our project.

2 Related work

The SDF is a learned continuous function used to represent a class of shapes based on a continuous volumetric field: the value 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. Therefore, this model is able to infer the zero-level-set of the learned function to explicitly represent the classification of space as being part of the shape's interior or not.

In our project, we train an auto-encoder to predict the SDF values and then find the zero-level set for shape reconstruction. In this section, we will provide a brief summary of the relevant parts from the DeepSDF paper [1] that form the basis of our implementation.

2.1 DeepSDF

DeepSDF is a continuous shape learning approach that represents the surface of a shape as the zero-level decision boundary with a learned continuous Signed Distance Function (SDF). The authors propose to directly regress the SDF from point samples using neural networks. Once the network is trained, it can predict the SDF value at a query position, from which the zero-level-set can be extracted and an explicit surface can be reconstructed using marching cubes. This continuous representation can be understood as a binary classifier, for which the decision boundary is the surface of the shape itself. DeepSDF is used for shape reconstruction, interpolation, and completion from partial and noisy 3D input data.

Data Preparation The network is trained on the ShapeNet dataset [2], which provides complete 3D shape meshes organised by class. To prepare the training data, Park et al. in their paper first normalise each mesh into a unit sphere and then sample 16,384 spatial points, mostly near the surface. They then compute the SDF value for each sampled point with respect to the mesh surface. The sampling is uneven in the unit sphere and is done more aggressively

[1] `https://github.com/Lixiyao-meow/DeepSDF.git`

near the surface to capture more details. This creates a set of position-SDF pairs which is then used for training the DeepSDF model.

Auto-decoder-based DeepSDF To use the computational resources more efficiently, they use an auto-decoder for learning a shape embedding without an encoder. Different from an auto-encoder whose latent code is generated by the encoder, an auto-decoder receives directly a latent vector as an input. They initialise the latent vector randomly to each model in the beginning of training, and the latent vectors are optimised along with the decoder weights through standard backpropagation. During inference, decoder weights are fixed, and an optimal latent vector is estimated. It involves using gradient descent to find a latent vector that locally minimises prediction error for a set of observed SDF values.

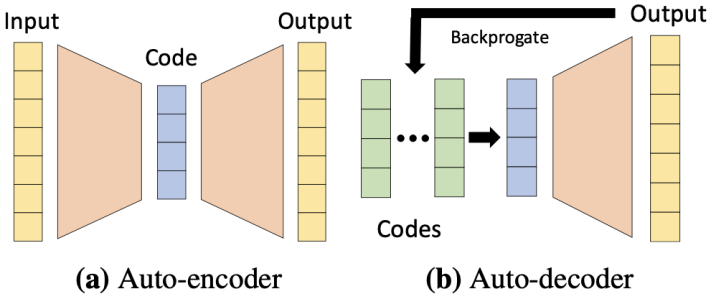

Figure 1: SDF Auto-decoder [1]

Shape reconstruction To rebuild the mesh from the DeepSDF auto-decoder, the authors first create a 3D grid by dividing the unit sphere into a set of small cubes. For each cube in the grid, they infer the SDF value using the trained DeepSDF model. Once they have the SDF value for each cube, they can render the mesh with the marching cubes algorithm. This algorithm extracts the zero-level-set boundary of the SDF, which represents the surface of the reconstructed shape. Finally, the extracted surface is smoothed using a Laplacian smoothing technique to produce a smooth mesh.

We will review the paper and discuss the theory later after our own experiments in the Evaluation section 5.

2.2 PointCleanNet

Rakotosaona et al. [3] published a deep learning-based denoising model for point clouds, which is capable of removing outliers and fine-tuning noisy point clouds. The model is able to discover and preserve high-curvature features without requiring additional information about the surface detail. By incorporating this pre-trained denoising model into our DeepSDF implementation, we can ensure that the sampled points are close enough to the mesh surface, which is crucial for capturing intricate details. This involves applying the denoising model after the sampling step in DeepSDF, thereby improving the quality of the sampled points and ultimately enhancing the accuracy of the reconstructed shape. The denoising model complements the DeepSDF auto-decoder by enabling more robust and precise shape reconstruction, especially for complex 3D objects with intricate features.

3 Implementation

Our implementation follows the methodology outlined in the DeepSDF paper [1] closely. However, due to computational constraints, we trained our model on a smaller dataset from one category and for fewer epochs compared to the original paper.

3.1 Preprocessing

We train our model on the ShapeNet dataset [2]. Due to limited access to powerful GPUs, we decide to train it on a single class, the `sofa` class. We used 50 normalised and aligned samples for training and 15 samples for validation. The meshes used for training are illustrated in Figure 2.

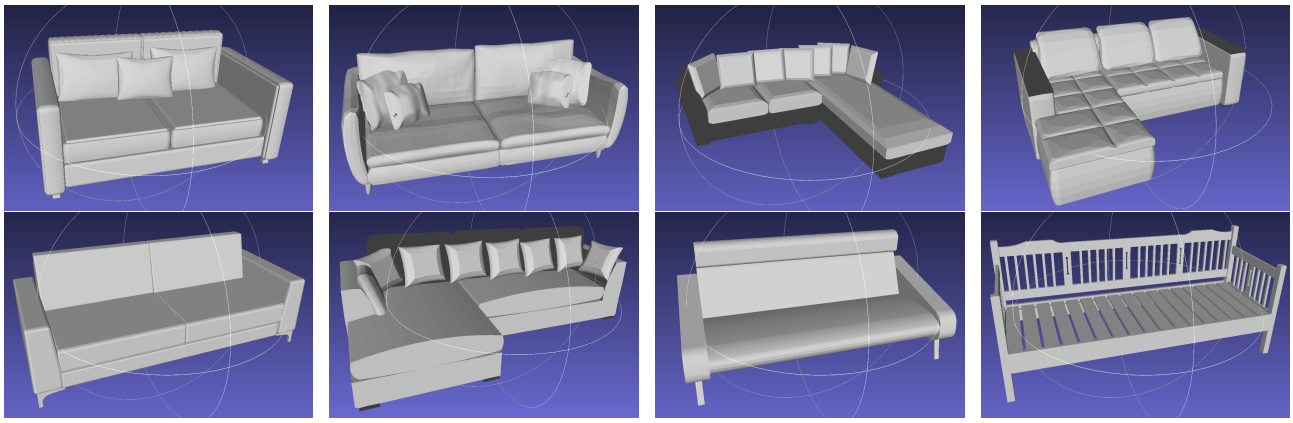

Figure 2: Examples of `sofa` from ShapeNet [2]

To calculate the approximate SDF for each model, we non-uniformly sample $15,000$ points near the surface. To accomplish this, we use the Python library `mesh_to_sdf` [4], which is based on the DeepSDF paper [1] and can handle non-watertight meshes, self-intersecting meshes, and meshes with non-manifold geometry. Within the `sample_sdf_near_surface` function, the algorithm first sets up equally spaced virtual cameras around the object and densely samples surface points with surface normals oriented towards the camera. Next, each point is perturbed along all three axes with zero-mean Gaussian noise, with variances of 0.0025 and 0.00025, to generate two spatial samples per surface point. To compute the SDF value, the algorithm finds the nearest surface point, measures the distance, and decides the sign based on the dot product between the normal and the vector difference.

We show some of the sampled point clouds in Figure 3. The points are coloured with their SDF values with a linear scale from red (+) to blue (-). Most of the sampled points are close to the surface, while only a few points are outside in the unit sphere.

3.2 Network structure

Our neural network adopts the primary architecture of DeepSDF but with fewer features, such as the regularisation component in the loss function. We load data for each model in the form of a matrix of size $15,000 \times 4$, where 4 contains the position vector (x, y, z) used in the first layer and the SDF ground truth value used for the loss function. We initialise one 256-dimensional latent vector $\mathbf{z}_i$ per model. Then, we concatenate the corresponding latent vector to the query position (x, y, z) to create a 259-length vector, which serves as the first layer of the neural network.

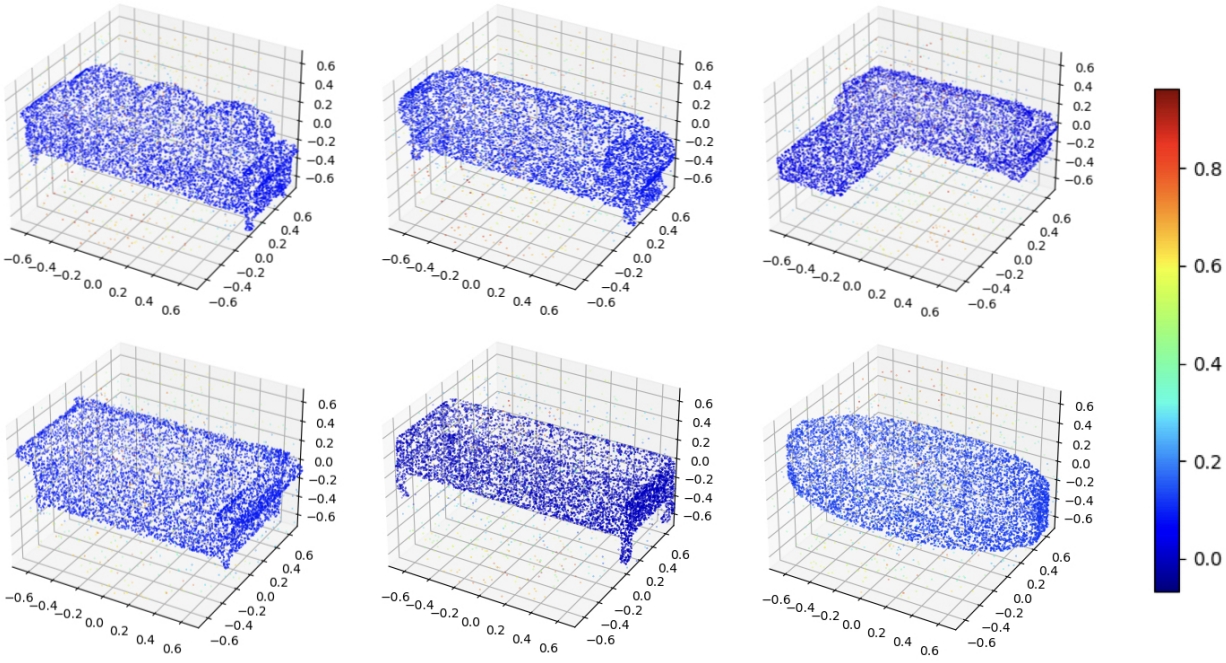

Figure 3: Examples of sampled point clouds

The network consists of eight fully connected layers. After each FC layer, we apply a ReLU activation and a dropout probability of 0.2 to avoid overfitting. The authors of DeepSDF found that inserting the latent vector again in the middle layers significantly improves learning, so we concatenate the 259-dimensional vector with the output of the fourth fully connected layer to create a 512-dimensional vector. The final SDF value is predicted using a hyperbolic tangent non-linear activation function. The network structure is shown in Figure 4.

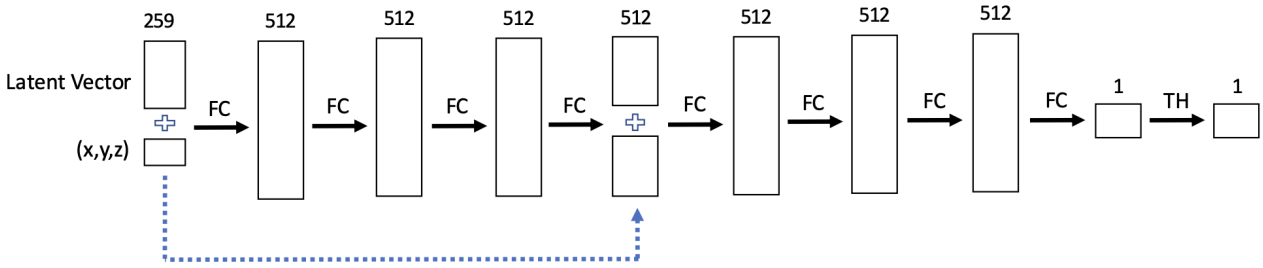

Figure 4: Network Structure [1]

3.3 Training process

For a given model, we define its latent vector $\mathbf{z}_i$ and its input data X_i as follows:

$$X_i = \{(\mathbf{x}_j, s_j) \quad | \quad s_j = \text{SDF}^i(\mathbf{x}_j)\}$$

where $\mathbf{x}_j$ is the query position.

The SDF value prediction $\tilde{s}_j$ from the decoder is based on the latent vector corresponding to the model and the position of the sample point on the unit sphere, noted as:

$$\tilde{s}_j = f_\theta(\mathbf{z}_j, \mathbf{x}_j)$$

And we use the clamped L1 loss for backpropagation:

$$\mathcal{L}_1(\tilde{\mathbf{s}}, \mathbf{s}) = \sum_j |\tilde{s}_j - s_j|$$

The latent vectors are initialises with zero-mean Gaussian noise with a standard deviation of $\sigma = 0.01$. The loss function is used to optimise both the latent vectors and the parameters in the auto-decoder. After the training step, we will obtain a trained model and 50 latent vectors, each of which corresponds to one model.

3.4 Reconstruction

After the training step, we assume that the model is well-functional. The first step in reconstruction is to find the latent vector of a given model. For a given validation model, we input the point positions with ground truth SDF values to the trained auto-decoder and use gradient descent to find its latent vector $\tilde{\mathbf{z}}_i$ that locally minimises the prediction loss.

Using the optimal latent vector $\tilde{\mathbf{z}}_i$ and query positions, we can predict the SDF value and render the zero-level-set boundary using the marching cubes algorithm. Specifically, we partition a 3D grid into small cubes and use our trained model to infer the SDF value for each cell. With the SDF value for each cell, we can render the mesh using the marching cubes algorithm. We adapted the code for rebuilding the mesh from the DeepSDF code [5].

4 Results

4.1 Training

We use the Adam optimiser for training with a learning rate of 5×10^{-4} for the auto-encoder and a learning rate of 10^{-3} for latent vector embedding. Due to limited GPU resources, we train the network on only 50 samples from the ShapeNet [2] `sofa` subset. The auto-decoder is trained for 200 epochs, and the loss function is shown below.

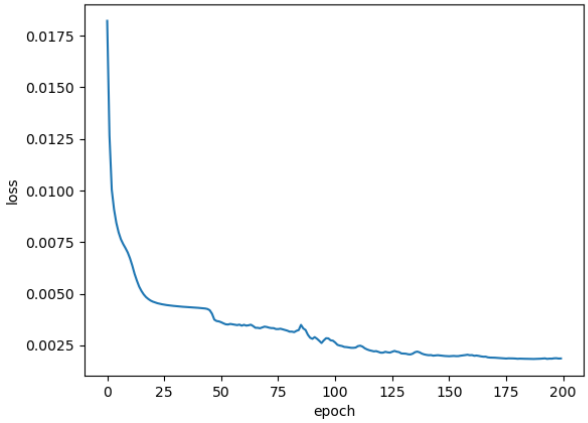

Figure 5: Training loss curve

4.2 Reconstruction

We present several examples of the reconstructed shapes using our trained auto-decoder in Figure 6. Although our model is able to learn the general shape of the class and some variation can be observed depending on the input model's shape, it fails to capture more intricate details.

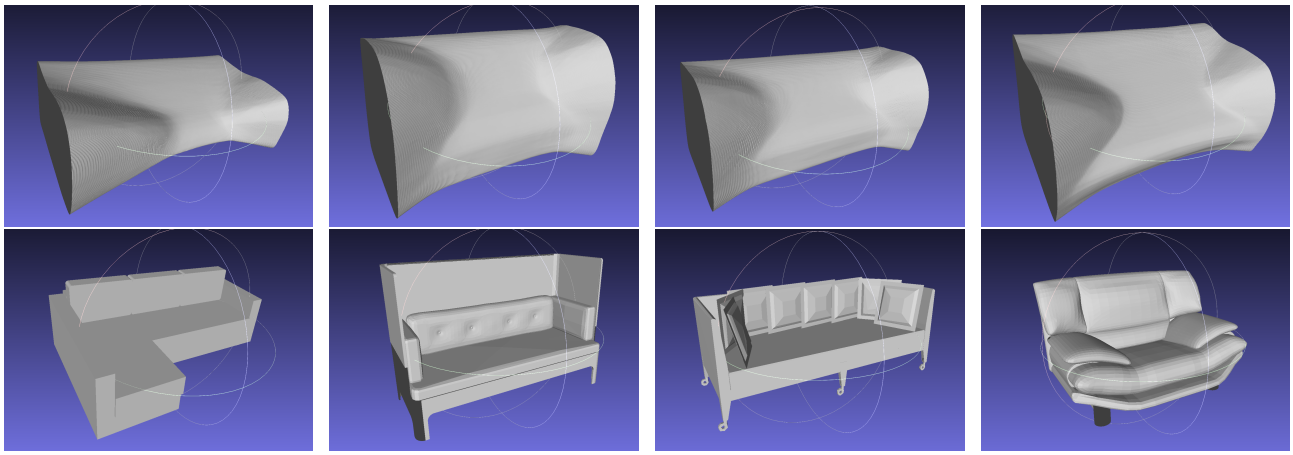

Figure 6: Shape reconstruction

Another issue that is worth mentioning is that the reconstructed shape is not closed. The example shown in Figure 7 illustrates the difference in scale between the reconstructed mesh and the original mesh. The predicted mesh is much larger than the original one, and the side of the sofa is not closed. This is because the SDF of points located near the side of the sofa is predicted to be negative even when close to the unit grid's boundary.

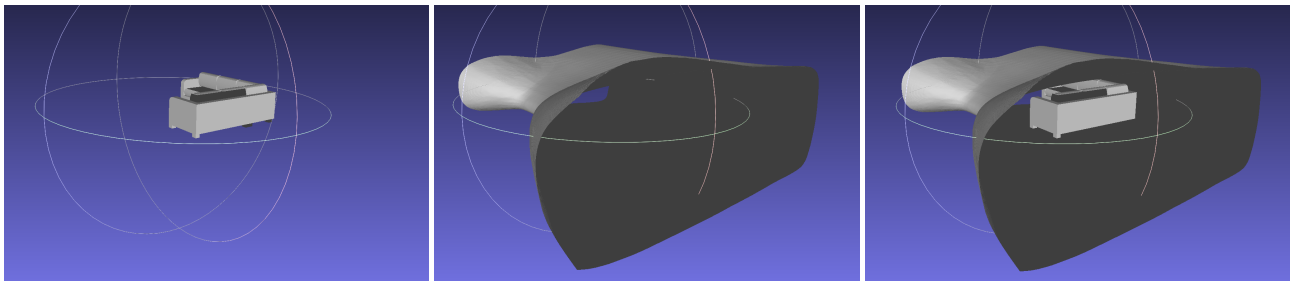

Figure 7: Comparison between reconstructed and original mesh scale

To identify the reason for the scale error, we perform a confusion matrix analysis on our trained network. We focus only on the sign of the predicted SDF value and compare it to the sign of the ground truth SDF value. This analysis is carried out on the average of 50 models.

	Actual Positive	Actual Negative
Predicted Positive	795	405
Predicted Negative	7095	6705

Table 1: Confusion matrix of trained auto-decoder

The result of this analysis is presented below in Table 1. We notice that the model tends to predict all values to negative to minimise the loss during the training process. However, it still manages to learn some very general information from the input point clouds. Unfortunately,

we are not able to identify exactly the reason for this behaviour, but further possibilities will be discussed in the following section.

5 Evaluation

In contrast to traditional explicit representations, the SDF representation is a continuous method that can handle complex topologies and geometries, and even capture some sophisticated details after sufficient training. However, there are still concerns regarding the stability and reproducibility of the SDF auto-decoder.

5.1 Stability

The authors of DeepSDF have made their code available on GitHub [5]. Some individuals have attempted to reproduce the results using the authors' code but have encountered several issues, which they have documented in the **Issues** section of the repository. We have also encountered some of these issues and will discuss possible reasons for them.

Training loss An issue on GitHub [6] mentioned that the training loss does not decrease after about 40 epochs, and the model predictions give only positive or negative SDF values for a single model. We encountered the same problem during the training process. However, after repeating the training several times, we observed that this issue does not occur every time. We believe that it may be due to the initialisation of the model, especially the initial values of the latent vector embedding.

Latent vector Another issue, as mentioned in [7], is that during inference, the latent vector must be initialised very close to the known and perfect latent vector for the Adam optimiser to converge to the correct optimum. We have also experienced similar issues during inference with the latent vector. The inference only seems to work when the latent vector is initialised to very small values, such as $\sim \mathcal{N}(0, 0.01^2)$.

Moreover, the only solution to this issue seems to be sampling the latent space and running inference for many latent vectors to find the correct initialisation, which is a brute force approach. This also suggests that the model is heavily dependent on the initialisation and cannot be considered a stable model.

Reconstruction This individual [8] attempted to overfit the network to a single mesh but obtained very poor reconstruction quality. The training loss appeared to plateau after 750 epochs, while the authors of the paper trained the model for 2000 epochs until convergence. The individual's solution was to generate several point clouds from one model and train on a larger batch size instead of just one. They believed that this was either a batch size issue or simply not enough information in a single sampled point cloud to properly learn and reconstruct the mesh.

Another individual encountered a similar issue [9]. They found that when training with small batch sizes (3 or 4), they encountered errors during reconstruction because the network was unable to predict negative SDF values. This suggests that the model's performance is also influenced by the batch size used during training.

5.2 Limitations

Based on our experiments and analysis of the DeepSDF model, we have concerns about its feasibility and reproducibility, particularly regarding the idea of replacing the encoder with randomly initialised latent vectors. During the training process, the latent vectors and the auto-decoder's weights are optimised together. And during inference, the auto-decoder is expected to work well to find the optimal latent vector for a given shape. However, the optimisation of the latent vector only works under the assumption that the auto-decoder is well-functional, which depends on the randomly initialised noise at the beginning. Therefore, this idea seems unfeasible in theory, and it may explain why others attempting to reproduce it encountered many issues.

It seems that the SDF representation is promising, but the approach of replacing the encoder with random noise may not be the most stable or reliable method. It appears that the auto-decoder and the latent vector initialisation are highly dependent on each other, which may explain why the model is unstable and sensitive to the initial conditions. In our opinion, the encoder is still necessary in this case for stability and reliability.

Another point is that during preprocessing, the points are essentially sampled near the surface. As a result, the model has limited ability to predict values for positions far from the model. Moreover, it can only learn shapes from the training models, so any variations in the validation data can lead to unpredictable behaviour from the auto-decoder.

In addition, during the training process, we evaluate the model only based on the training loss. To prevent overfitting, we can introduce a validation step after each epoch and save the checkpoint that has the lowest loss on the validation dataset as the optimal model. This approach enables us to assess the performance of the model on unseen data and ensure that it generalises well.

6 Conclusion

In conclusion, we attempted to reproduce the DeepSDF auto-decoder and apply it to the shape reconstruction task. Due to limited computational resources and the model's instability, our model could only learn very general information from the shape and failed to capture more details. Based on our experiments, we believe that the SDF is a promising representation, but the idea of removing the encoder does not seem reliable. The traditional encoder-decoder approach should produce more stable results in theory.

References

[1] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019.

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

[3] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J. Mitra, and Maks Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds, 2019.

[4] Marian Kleineberg. Calculate signed distance fields for arbitrary meshes. `https://pypi.org/project/mesh-to-sdf/`.

[5] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf. `https://github.com/facebookresearch/DeepSDF`.

[6] training loss does not decrease. `https://github.com/facebookresearch/DeepSDF/issues/56`.

[7] Latent code optimization during inference. `https://github.com/facebookresearch/DeepSDF/issues/71`.

[8] Improving reconstruciton quality. `https://github.com/facebookresearch/DeepSDF/issues/94`.

[9] error during reconstruction using small batch sizes. `https://github.com/facebookresearch/DeepSDF/issues/31`.