

# Latent Uncertainty-Aware Multi-View SDF Scan Completion

## Supplementary Material

### 7. Data Generation

We have performed several steps of preprocessing on Objaverse 1.0 [15] to generate training and evaluation data for our model. We first joined the scenes into one mesh and excluded all the unnecessary data such as animations, textures etc. We then filtered the meshes by size, excluding those above 32MiB and below 2kiB. This allowed us to avoid both computationally heavily demanding meshes and overly simplistic geometries. To compute SDFs from meshes, we first center their bounding box at the origin and then uniformly scale them to fit within a  $[-1, 1]^3$  range. We generated ground truth meshes for SDF computation by rendering and combining 100 partial views at image resolution of  $256 \times 256$ . This ground truth SDF is used for training and when computing the IoU.

**Partial View Generation.** For generating partial views, we first initialize a set of camera perspectives following a Spherical Fibonacci [28] point set for view directions. For each view, the original mesh is rasterized into an image (e.g.,  $256 \times 256$ ), from which we extract the depth values.

We pre-compute a dense triangle mesh matching the image grid with vertices at the pixel centers and two triangles connecting the space between all sets of 4 neighboring pixels. This mesh is projected into the scene space, given the depth values. For pixels which did not observe any surface, there is no valid depth value, so we exclude the corresponding vertices and connected faces. We also remove all faces, whose surface normal deviates from the camera view-direction by more than  $75^\circ$ , since they are likely to produce non-existent surfaces between disjoint parts of the original geometry. This results in a refined, clean partial mesh derived from the original mesh.

### 8. Detailed Results

We present the full shape completion results on ShapeNet [4], containing per-class results, in Table 3.

### 9. Training and Evaluation Details

**Main Model.** We have trained our main model on Objaverse 1.0 [15], pre-processed and filtered as explained in Appendix 7. The train split contains 722399 training samples, the validation split consists of 1904 samples, and, finally, we evaluate our results on our test split consisting of 4991 many samples.

We train our model for 14 epochs, taking roughly 10 days on 3 RTX 4090 GPUs. During the first 10 epochs,

we use a cosine scheduler with a warmup ratio of 0.02 and a starting learning rate of  $2 \cdot 10^{-5}$ . For the 4 remaining epochs, we use a constant learning rate of  $10^{-6}$ . During these last 4 epochs, we randomly select between only 1 to 4 random views as input to our model to fine-tune towards low-view cases. We use a batch size of 6 for training and 1 for validation.

To train the seam removal model, we train only for 2k many steps on Objaverse 1.0 [15] with a constant learning rate of  $10^{-4}$ . Here, training only took place for a few minutes. To generate inputs for seam removal, we run the pre-trained main model on partial scans.

To save on GPU memory, we trained our model in the bfloat16 mixed-precision mode. Our preliminary testing did not show any loss in performance compared to using regular 32-bit floats.

**Fine-tuning on ShapeNet [4].** For ShapeNet [4], we use the official split provided by PatchComplete [38] and we fine tune our model with data from 18 categories (table, chair, sofa, cabinet, bookshelf, piano, microwave, stove, file cabinet, trash bin, bowl, display, keyboard, dishwasher, washing machine, pots, faucet, and guitar) and subsequently test it on data from the other 8 unseen categories (bathtub, lamp, bed, bag, printer, laptop, bench, and basket). The training set and test set consist of 3202 and 1325 objects, respectively, each of which consist of 4 partial shapes, scanned from random viewpoints.

We fine-tuned our model on ShapeNet [4], using a constant learning rate of  $10^{-6}$ . Due to a lack of sufficient training data from the official split published by PatchComplete [38], we performed early stopping after 3.8k steps to prevent overfitting. We used the same training strategy as we explained in Section 3.3.4 with the only difference of using only 1 to 4 randomly generated views for training.

**Evaluation on Objaverse 1.0 [15].** We predefined 20 camera perspectives from a Spherical Fibonacci [28] point set. Out of these, per object, we randomly select 7 views, shuffle them, and store them into our test split description. During evaluation, we fetch the corresponding view sets for each object and we extract the first  $N$  many views, as many as we want to evaluate on. In this way, all of our evaluations are consistent each time and the views remain the same for each and every object. For calculating metrics on the test split, we use the actual Objaverse 1.0 [15] meshes as ground truth to calculate our metrics, rather than the ground truth SDF data used during training (Appendix 7).

Table 3. Single-view shape completion on unseen object categories on ShapeNet [4]. Data of other methods by DiffComplete [10].

Method	Metric	Avg	Bag	Lamp	Bathtub	Bed	Basket	Printer	Laptop	Bench
3D-EPN [13]	IoU $\uparrow$	0.594	0.738	0.472	0.579	0.584	0.540	0.736	0.620	0.483
Few-Shot [30]		0.403	0.561	0.254	0.457	0.396	0.406	0.567	0.313	0.272
IF-Nets [8]		0.581	0.698	0.508	0.550	0.607	0.502	0.705	0.583	0.497
AutoSDF [31]		0.452	0.563	0.391	0.410	0.446	0.398	0.499	0.511	0.395
ConvONet [35]		0.601	0.708	0.526	0.604	0.632	0.546	0.721	0.573	0.496
PatchComplete [38]		0.654	0.776	0.564	0.663	0.668	0.610	0.776	0.638	0.539
DiffComplete [10]		0.675	0.783	0.579	<b>0.689</b>	0.671	<b>0.655</b>	0.768	0.674	<b>0.582</b>
Ours (zero shot)		0.637	0.904	0.554	0.513	0.684	0.539	0.867	0.660	0.373
Ours		<b>0.684</b>	<b>0.909</b>	<b>0.596</b>	0.556	<b>0.716</b>	0.574	<b>0.878</b>	<b>0.766</b>	0.473
Ours+SR		0.649	<b>0.909</b>	0.567	0.523	0.703	0.541	0.875	0.692	0.383
3D-EPN [13]	CD <sub>L1</sub> $\downarrow$	5.58	5.01	8.07	4.21	5.84	7.90	5.15	3.90	4.54
Few-Shot [30]		9.58	8.00	15.10	7.05	10.03	8.72	9.26	10.35	8.11
IF-Nets [8]		5.29	4.77	5.70	4.72	5.34	4.44	5.83	6.47	5.03
AutoSDF [31]		5.86	5.81	6.57	5.17	6.01	6.70	7.52	4.81	4.31
ConvONet [35]		5.26	5.10	5.42	4.96	5.42	6.16	5.56	4.78	4.69
PatchComplete [38]		4.27	3.94	4.68	3.78	4.49	5.15	4.63	3.77	3.70
DiffComplete [10]		4.10	3.86	4.80	<b>3.52</b>	4.16	4.94	4.40	3.52	3.56
Ours (zero shot)		3.233	<b>1.713</b>	4.355	4.113	<b>3.041</b>	<b>3.965</b>	<b>2.450</b>	3.057	3.171
Ours		<b>3.040</b>	1.999	<b>3.560</b>	3.747	3.088	4.041	2.756	<b>2.626</b>	<b>2.502</b>
Ours+SR		4.336	2.357	5.564	5.166	3.750	5.653	3.058	4.946	4.192

Table 4. Shape completion on Objaverse [15] given imperfect camera poses. Each partial mesh generated per view is randomly rotated by, on average, 1°, 3°, or 5°, before being processed by our pipeline. The baseline without any rotation can be found in Table 1.

Model	Task	Offset	IoU $\uparrow$	$F_1\uparrow$	CD <sub>L2</sub> $\downarrow$	HD $\downarrow$	CD <sub>L1</sub> $\downarrow$	NC $\uparrow$	IN $\downarrow$	CMP $\uparrow$
Ours	1 view	1°	0.081	0.128	26.665	1.027	39.913	0.661	0.937	0.614
Ours	2 views	1°	0.352	0.366	10.693	0.658	20.385	0.734	0.855	0.692
Ours	3 views	1°	0.508	0.489	7.270	0.521	14.720	0.774	0.796	0.762
Ours	4 views	1°	<b>0.575</b>	<b>0.548</b>	6.673	0.474	12.861	<b>0.793</b>	<b>0.767</b>	<b>0.805</b>
Ours	1 view	3°	0.076	0.113	26.829	1.035	40.421	0.654	0.942	0.589
Ours	2 views	3°	0.334	0.333	10.659	0.656	20.616	0.727	0.867	0.662
Ours	3 views	3°	0.479	0.443	7.360	0.516	14.910	0.767	0.816	0.728
Ours	4 views	3°	0.543	0.502	5.616	0.453	<b>12.342</b>	0.785	0.792	0.768
Ours	1 view	5°	0.069	0.102	27.138	1.046	41.170	0.643	0.949	0.555
Ours	2 views	5°	0.314	0.297	10.958	0.658	21.338	0.716	0.883	0.611
Ours	3 views	5°	0.451	0.396	6.792	0.509	15.013	0.754	0.841	0.666
Ours	4 views	5°	0.509	0.447	<b>5.397</b>	<b>0.447</b>	12.521	0.772	0.826	0.701

**Evaluation on ShapeNet [4].** Our evaluation scheme for ShapeNet [4] is almost the same as our main model evaluation scheme. For calculating the metrics on the ShapeNet [4] test split published by PatchComplete [38], we use our generated ground truth created from 100 combined views at an image resolution of  $256 \times 256$ , rather than the original dataset. This ensures a consistent, watertight mesh, removing any self-intersections and other topological problems with the original meshes.

## 10. Imperfect Poses

We analyze the impact of imperfect pose estimates on shape completion in Table 4, where we evaluate our shape completion model on Objaverse 1.0 [15], while rotating each input view randomly by sampling a von Mises-Fisher distribution, parameterized by  $\kappa \in \{6566.9, 729.6, 262.8\}$ , which, on average, produces rotations by 1°, 3°, and 5°. Note that individual samples roughly follow a normal distri-

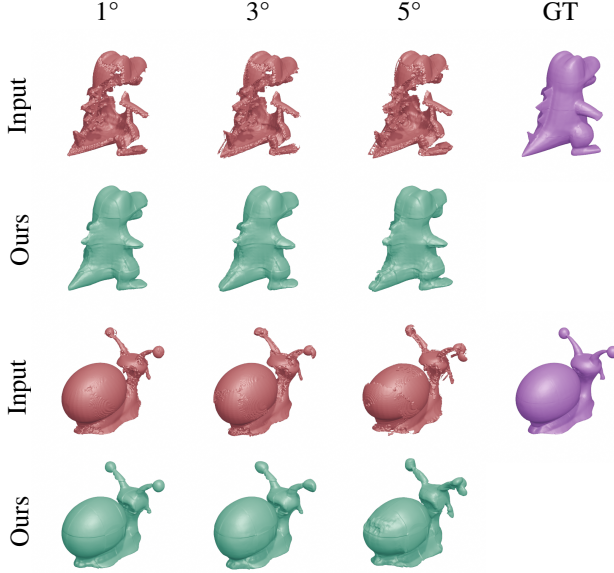


Figure 7. Partial scan completion on Objaverse 1.0 [15], given 4 views with imperfect poses with average rotations of  $1^\circ$ ,  $3^\circ$ , and  $5^\circ$ . Note how the stronger rotations deform individual parts of the objects and create artifacts on overlapping regions. In spite of the rotated inputs, shape completion produces plausible results but they no longer fully line up with the ground truth mesh. Best viewed zoomed-in.

bution. Therefore, some samples will be rotated less, while others will be rotated much, much more, but less frequently. Qualitative examples, given 4 views, are shown in Figure 7, to illustrate the distortions caused by imperfect alignment.

While all cases suffer from the reduced pose accuracy of individual meshes, compared to the rotation-free baseline in Table 1, especially the single-view case suffers heavily and, as expected, larger rotations lead to worse performance. While in the multi-view case, it is highly unlikely to sample a large rotation twice, for a single view, some samples will undergo a large rotation and no longer match up with the ground truth, leading to worse numeric results.

Also, our model, which has not been (re-)trained for this task, occasionally struggles to understand the slightly off-axis geometric cues.

Somewhat curiously, given 3 and 4 views, larger rotations produce better results in terms of HD,  $CD_{L1}$  and  $CD_{L2}$ . Here, the large misalignment leads to high uncertainty on overlapping parts, allowing our model to detect and fix the inconsistencies. In contrast, the 2 view case generally does not provide enough overlap between the individual views to flag the differences as uncertain.

## 11. Failure Cases

To show the limitations of our method, we demonstrate the cases where our method struggles to complete objects from

both Objaverse [15] and ScanNet [12].

**Objaverse [15]** As is shown in Figure 8, our method struggles with completing very thin structures such as the ladder and hands, as well as layered geometries, such as the objects in the last two columns.

**ScanNet [12]** As is explained in the Limitations Section, our model is only trained with clean objects. It does not attempt to remove geometry from neighboring or overlaid objects, which can negatively impact the shape completion process. In Figure 9, we show limitations of our model using inputs extracted from ScanNet [12]. In the case of a backpack on a chair, our model produces a chair, whereas the ground truth expects a backpack. In other cases, e.g., the sink in the first column, the mesh selected as ground truth by the dataset does not match the input, resulting in an excessively large distance to our reconstruction.

Please note that the training split defined by previous work for comparison on ScanNet (7.5k objects) is too small to train our model. Our model is not designed to be trained on such small sets of data and will inevitably overfit on them.



Figure 8. Failure cases on Objaverse [15] objects.

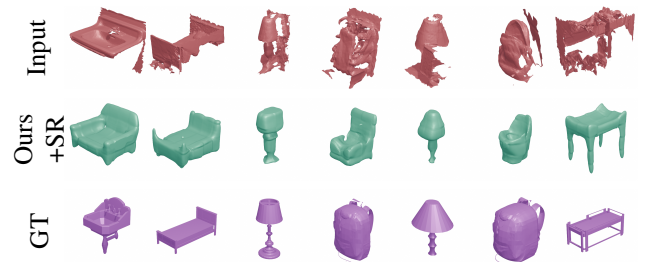


Figure 9. Failure cases on ScanNet [12] objects. The annotations provided by Scan2CAD [2] allow for extracting inputs from the scans for evaluation. The ground truth is defined as the most similar ShapeNet [4] object.

## 12. Using Semantics

The sensor data captured by 3D scanning can not only be used to extract depth information but also allows for seman-

tic segmentation of image patches, especially if one also captures RGB data. This additional information could be used to aid shape completion, e.g., by first isolating the geometry of individual meshes before shape completion. Also, the completion process can make use of semantic part labels to infer missing pieces and typical symmetries based on a learned understanding of the composition of different classes of objects.

To make use of this information, one could first run any semantic or panoptic segmentation model on the RGB image and transform the detected labels into the camera frame of the depth sensor. There, one can isolate the pixels of the main object, exclude everything else, and potentially project part labels into the 3D space as an additional signal fed into the shape completion model, as well as the main class label. This should give a strong signal to the shape completion model, further enhancing shape completion, by allowing the model to explicitly reason about object classes and parts, while still focusing completion primarily on the observed 3D geometry.

This entire process could, e.g., be applied to the raw frame-by-frame ScanNet [12] data in order to meaningfully preprocess it for performing object-level shape completion. For datasets, where frame-by-frame data is not available, one could synthesize it from the reconstructed scene geometry using virtual cameras.

This goes far beyond the object detection and classification, as introduced by Scan2CAD [2], where one could only crop out entire bounding boxes from the full scene geometry. Due to the scope of work necessary to do this, we leave this task for future work.

### 13. Metrics

To evaluate the following metrics reported in the paper, we evaluate our results on several volume-level and point-level metrics. For volume-level metrics, we simply evaluate on the reconstructed SDF versus ground truth SDF. On the other hand, for the point-level metrics, we first apply Marching Cubes [27] to create a mesh from the generated SDF, and then uniformly sample 1M points  $X$  and  $Y$  from the resulting meshes.

Using the evaluation approach of AutoSDF [31], we independently assess reconstruction artifacts, with F-score @1% ( $F_1$ ) [42], normal consistency (NC) and inaccurate normals (IN). F-score @1% ( $F_1$ ) applies a threshold of 1% of the reconstructed volume’s side length, requiring a nearby ground truth point within this range to calculate precision, and similarly for recall. Given precision and recall,  $F_1$  score is computed in a standard way. We also compute Intersection over Union (IoU) using the sign of the  $128^3$  SDFs. Normal Consistency (NC) calculates the mean absolute dot product between the normals on the reconstructed

surface and the nearest ground-truth surface point:

$$NC(X, Y) = \frac{1}{|X|} \sum_{x \in X} \left\{ |n_x \cdot n_y| : \right. \\ \left. y = \operatorname{argmin}_{y \in Y} d(x, y) \right\} \quad (7)$$

The absolute value is taken to allow for flipped normals in the ground truth data. We use the geometric face normals  $n_x$  and  $n_y$  of the faces containing the sampled points  $x$  and  $y$ .

While the NC is sensitive towards large errors, Inaccurate Normals (IN) complements the NC measure by counting the percentage of normals which are outside of a 5-degree threshold of the normal of the closest ground-truth point. In another way, IN captures the little errors.

Completeness (CMP) measures the recall within a threshold of 1.5% of the side length, based on the implementation of Wu et al. [48].

We use the several point-based metrics, where  $d(x, y)$  measures the  $L^2$  distance between two points  $x, y \in \mathbb{R}^3$ . Typically, we compute them from 1M uniformly sampled point clouds  $X$  and  $Y$  sampled from the ground truth meshes. When comparing with prior work, we reduce the amount of samples for a fair comparison.

Hausdorff Distance (HD) measures the largest gap between the original and the reconstructed geometry:

$$HD(X, Y) = \max \left\{ \max_{x \in X} \left\{ \min_{y \in Y} d(x, y) \right\}, \right. \\ \left. \max_{y \in Y} \left\{ \min_{x \in X} d(x, y) \right\} \right\}. \quad (8)$$

L2 Chamfer Distance [17] (based on squared  $L^2$  distances), multiplied by 100, measures the accuracy of the reconstruction via the mean squared distance between the original and reconstructed geometry:

$$CD_{L2}(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \{d^2(x, y)\} \\ + \frac{1}{|Y|} \sum_{y \in Y} \min_{x \in X} \{d^2(y, x)\}. \quad (9)$$

L1 Chamfer Distance, multiplied by 100 measures the mean distance between the original and reconstructed geometry:

$$CD_{L1}(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \{d(x, y)\} \\ + \frac{1}{|Y|} \sum_{y \in Y} \min_{x \in X} \{d(y, x)\}. \quad (10)$$

Despite its name, its computation does not involve the computation of any  $L^1$  distances, but rather describes its

relation towards the L2 Chamfer Distance: It uses non-squared  $L^2$  distances instead of squared ones.

To the best of our knowledge, the L1 Chamfer Distance was first defined in the supplemental of Occupancy Networks [29], where it is further multiplied by a factor of  $\frac{1}{2}$ , i.e., computing the mean instead of the sum of the two parts. This factor seems to be consistently omitted in the evaluation code of recent works. We similarly omit this extra factor for a fair comparison.