Living Scenes: Multi-object Relocalization and Reconstruction in Changing 3D Environments

Supplementary Material

In this supplementary material, we provide:

1. A video that explains our method and shows our experimental results (Sec. A).
2. Additional ablation study results (Sec. B.1) and detailed per category results (Sec. B.2).
3. Implementation details on the network architecture (Sec. C) and algorithm for registration and joint optimization (Sec. E).
4. Details on data processing for training and evaluation (Sec. F).
5. Description of evaluation metrics (Sec. G).
6. Additional visualizations on multi-object relocalization and reconstruction (Sec. H).
7. A discussion of the limitations of our method (Sec. I).

A. Video

See the provided supplementary video for a summarized description of the method and results on creating living scenes.

B. Additional Experimental Results

In this section, we present additional experimental results. Specifically, we further validate our design choices with three ablation studies (Sec. B.1). In Sec. B.2, we provide detailed per category results. We provide more qualitative results in Sec. H.

B.1. Ablation Studies

We ablate MoRE^2 to justify our design choices for instance matching, joint optimization, and network architecture.

Embeddings for matching. We compare the performance of different combinations of embeddings to compute the score matrix used for instance matching on *3RScan* [14]. The results are tabulated in Tab. 1. When compared individually, the invariant embedding $\mathbf{F_{inv}}$ performs consistently better than the equivariant $\mathbf{F_{eqv}}$, signifying it plays a more important role in instance matching. This is not a surprise, since the shape details can be more decisive than pose, especially when there is a changing environment. However, when using both embeddings, the results are further boosted. Hence, we use both embeddings in MoRE^2.

Parameters to optimize. To find the best set of parameters for the joint optimization, we experiment with different optimization settings on *FlyingShape*. Tab. 2 shows the optimization setting on the left and the corresponding performance on the right. The best performance is achieved when we optimize all four parameters. The optimization on equivariant embeddings significantly reduces the rotation error. Pure optimization on the pose graph gives the best rotation error but provides poor performance on reconstruction. This also shows that a single forward pass from the VN-encoder cannot provide accurate pose and shape information and verifies the importance of jointly optimizing pose and shape together. We use different learning rates (step size) during optimization for each parameter (Tab. 3). We mainly optimize $\mathbf{F_{eqv}}$ and $\mathbf{G}_0$ and only apply small adjustments to $\mathbf{F_{inv}}$ and $\mathbf{F_t}$.

Matching Setting		Evaluation Metrics					
$\mathbf{E}$ ($\mathbf{F_{eqv}}$)	$\mathbf{\Lambda}$ ($\mathbf{F_{inv}}$)	Instance-level Recall ↑			Scene-level Recall ↑		
		Static	Dynamic	All	@25%	@50%	@75%
	✓	58.20	78.60	69.38	89.77	79.55	45.45
✓		58.20	73.80	66.75	85.23	72.73	43.18
✓	✓	60.32	87.50	71.77	87.50	78.41	50.00

Table 1. **Ablation study of instance matching on *3RScan* [14].** ✓ denotes using the embedding for instance matching.

Optimization Setting				Evaluation Metrics			
$\mathbf{G}$	$\mathbf{F_{eqv}}$	$\mathbf{F_{inv}}$	$\mathbf{F_c}$	RE [°]		$L1$-Chamfer [$\times 10^{-3}$]	
				Mean ↓	Median ↓	Mean ↓	Median ↓
✗	✗	✗	✗	11.85	5.28	5.37	2.31
✓	✗	✗	✗	7.88	3.45	5.38	2.30
✓	✗	✗	✓	7.81	3.41	5.67	2.30
✓	✓	✗	✓	8.38	1.81	3.95	2.06
✓	✓	✓	✓	8.70	1.71	3.81	1.86

Table 2. **Ablation study on joint optimization.** ✗ denotes that the parameters are fixed during optimization and ✓denotes learnable parameters.

Parameter	$\mathbf{F_{inv}}$	$\mathbf{F_t}$	$\mathbf{F_{eqv}}$	$\mathbf{G}$
Step Size [$\times 10^{-5}$]	1	1	10	5

Table 3. **Step sizes of different parameters during joint optimization**.

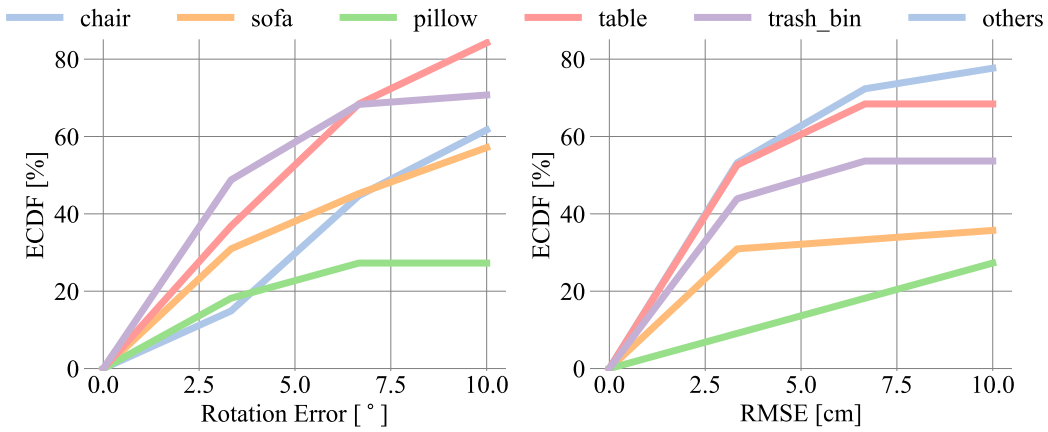

Figure 1. **Per category registration results on *3RScan* [14].** We report ECDF curves of rotation error (RE) and transformation error (RMSE).

Decoding strategy. We explore four decoding strategies using equivariant $\mathbf{F_{eqv}}$ and invariant $\mathbf{F_{inv}}$ embeddings on *FlyingShape*. The DeepSDF decoder takes as input the positional embedding of the coordinates to query and the global latent code of the shape representation. For a query point $\mathbf{p} = (x, y, z)$, there are two ways to compute the positional embedding:

- Direct concatenation (inv.): $\mathrm{cat}[\mathbf{F_{inv}}, \mathbf{p}]$
- Concatenating with the inner product (inner.) of equivariant embeddings and query point: $\mathrm{cat}[\mathbf{F_{inv}}, \langle \mathbf{F_{eqv}}, \mathbf{p} \rangle]$

where $\mathrm{cat}[\cdot, \cdot]$ denotes feature-wise concatenation

As decoding architectures, we consider DeepSDF [9] and a multi-layer perceptron (MLP) [4]. DeepSDF differs from regular MLPs, as shown in Fig. 4, by adding a skip connection of the query code. We evaluate the performance of combining each decoding strategy with each of the decoder architectures and present the results in Tab. 4. Inner.+DeepSDF shows the best performance. The inner product can map the query coordinates to a high dimensional space with more positional information as provided by the skip connection in DeepSDF [9].

B.2. Category-level Performance

We report the registration performance of each category on *3RScan* [14] in Fig. 1. The category of '*table*' has the overall best performance and we believe this is due to the distinct geometric features of tables and their relatively large size in the scene. The category of pillows has the worst performance, which is not surprising since it is not strictly rigid and has multiple symmetrical axes.

C. Network Architecture

In this section, we provide the implementation details of the encoder-decoder network used in the main paper.

PE	Decoder	$L1$-Chamfer ↓	$L1$-UNI ↓	$L1$-NSS ↓	IoU ↑
Inv.	MLP	7.04	0.037	0.021	0.325
Inv.	DeepSDF	7.34	0.041	0.019	0.343
Inner.	MLP	3.65	0.029	0.016	0.436
Inner.	DeepSDF	3.58	0.027	0.015	0.454

Table 4. **Ablation study on decoding strategies**. Positional Encoding (PE). Inner product (Inner.) and invariant feaures (Inv.). $L1$-Chamfer [$\times 10^{-3}$], UNI - Uniform SDF Sampling, NSS - Near Surface Sampling.

C.1. Encoder

Here, we follow recent progress [1, 7] in equivariant networks in vector neurons (VN) [3] and present the graphic illustration in Fig. 2. The encoder takes as input 3D point cloud coordinates and processes them through 2 VN-Linear blocks and 5 VN-Attention blocks. The VN-Linear block consists of computing the k-nearest neighbor graph feature [15], and the VNLA block, which comprises three layers: VN-Linear, VN-normalization, and VN-activation [3]. The VN-Attention block [1] utilizes three VNLA blocks to compute the $\boldsymbol{K}$, $\boldsymbol{Q}$, and $\boldsymbol{V}$, respectively, and applies the attention operation [13] and message passing. The dimensions of the features for the 7 blocks are: 32, 32, 64, 64, 128, 256, and 512. This means that the intermediate features are down-sampled after the 2nd (VN-Linear), 4th (VN-Attention), and 5th (VN-Attention) blocks. The final global feature of dimension 512 is then passed to compute the final output $\mathbf{F} = (\mathbf{F_{inv}} \in \mathbb{R}^{256}, \mathbf{F_{eqv}} \in \mathbb{R}^{3\times 256}, \mathbf{F_s} \in \mathbb{R}_+, \mathbf{F_c} \in \mathbb{R}^3)$. Here, $\mathbf{F_{inv}}, \mathbf{F_{eqv}}$, and $\mathbf{F_c}$ are computed by three VN-Linear prediction heads and $\mathbf{F_s}$ from channel-wise normalization.

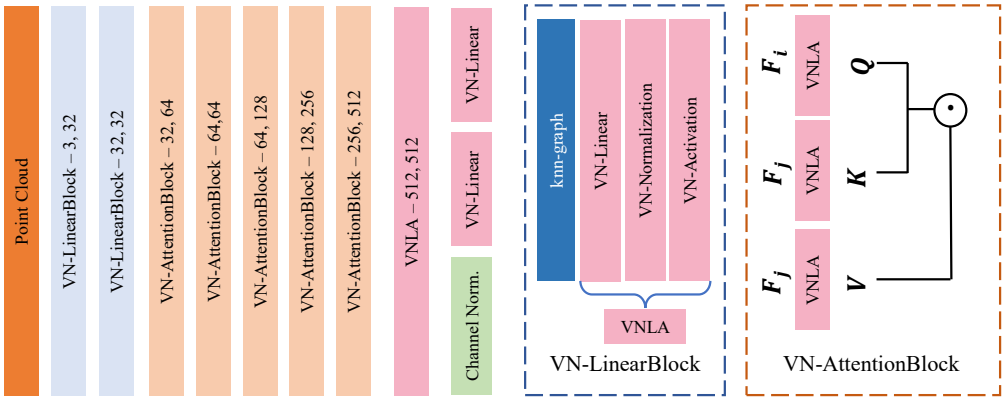

Figure 2. **VN-Encoder Architecture [1, 3].** On the left is the overall architecture from the input point cloud to the intermediate embeddings, which are subsequently fed to the DeepSDF decoder. $\odot$ denotes dividing the feature into multiple heads and computing score matrix S from Q and K, and message passing between S and V, lastly to the final output feature of the encoder. On the right, we show two important blocks of the encoder: VN-LinearBlock and VN-AttentionBlock.

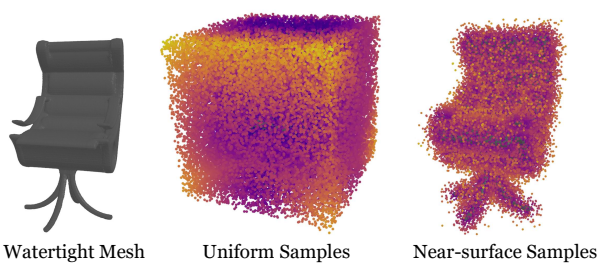

Figure 3. **Illustration of SDF samples for training.** Left: a watertight mesh. Middle: uniform SDF samples. Right: near-surface SDF samples.

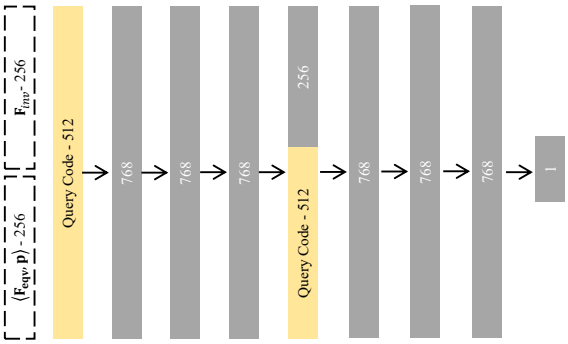

Figure 4. **DeepSDF Decoder.** $\rightarrow$ denotes residual connection. The query code is re-concatenated to the intermediate feature at the 4th layer.

C.2. Decoder

The architecture of the decoder is presented in Fig. 4. The number of fully connected layers is 8, the same as in the original DeepSDF [9]. In MoRE^2, the feature dimension is increased from 256 in DeepSDF to 768 for two reasons: (1) the dimension of the positional embedding is 256 and not 3; and (2) to increase the expressivity of the decoder when trained on multiple categories (category-agnostic). Same as in [9], the query code is computed by concatenating the shape code, *i.e.*, $\mathbf{F_{inv}}$, and the positional embedding $\langle \mathbf{F_{eqv}}, (\mathbf{p} - \mathbf{F_c})/\mathbf{F_s} \rangle$, which is further re-concatenated to the intermediate feature of the 4th layer (skip connection). The skip connection [9] can provide better performance and a more regularized reconstruction (*c.f.* Inner.+MLP vs. Inner.+DeepSDF in Tab. 4).

D. Training Details

We follow EFEM [7] and DISN [18] and train the network using SDF loss with regularization terms.

SDF Loss. The SDF samples are generated from a unit cube around the object. 50% are sampled near the surface and 50% uniformly in the space (*c.f.* Fig. 3). We encourage the network to learn more local details with:

$$\mathcal{L}_{\text{SDF}} = \frac{\lambda_{\text{near}} \sum_{x \in \mathcal{Q}_{\text{near}}} \mathcal{L}_{\text{recon}}(x) + \lambda_{\text{far}} \sum_{x \in \mathcal{Q}_{\text{far}}} \mathcal{L}_{\text{recon}}(x)}{|\mathcal{Q}_{\text{near}}| + |\mathcal{Q}_{\text{far}}|}. \tag{1}$$

where $\mathcal{Q}_{\text{near}}$ denotes the set of samples with $|\mathbf{SDF}| < 0.1$ and $\mathcal{Q}_{\text{far}}$ the set of samples with $|\mathbf{SDF}| \geq 0.1$. The weights of the two sets of samples are $\lambda_{\text{near}} = 1.0$ and $\lambda_{\text{far}} = 0.5$.

Regularization [7]. Two loss terms are used to regularize the training of scale and centroid prediction:

$$\mathcal{L}_{\text{scale}} = |1.0 - \mathbf{F_s}| \tag{2}$$

and

$$\mathcal{L}_{\text{center}} = \|\mathbf{F_c}\|_2 . \tag{3}$$

Algorithm 1: Registration

Input:
$\mathbf{X}^{t_1}$(source), $\mathbf{X}^{t_2}$(target), Φ(encoder), Ψ(decoder)
/* Initialization */
$\mathbf{F}^{t_1}, \mathbf{F}^{t_2} \leftarrow \Phi(\mathbf{X}^{t_1}), \Phi(\mathbf{X}^{t_2})$
$\mathbf{R}, \mathbf{t} \leftarrow \mathrm{Kabsch}(\mathbf{F}^{t_1}, \mathbf{F}^{t_2})$
$\eta \leftarrow 10^{-3}$: step size, $K \leftarrow 200$: number of steps;
/* Iterative Update */
for i $= 0, ..., K$ **do**
  $\mathbf{P}_\mathrm{i} \leftarrow \mathbf{R}_\mathrm{i}\mathbf{X}^{t_1} + \mathbf{t}_\mathrm{i}$
  $\mathbf{F}_q^{t_2} \leftarrow \mathrm{cat}[{\mathbf{F_{inv}}}^{t_2}, \langle {\mathbf{F_{eqv}}}^{t_2}, (\mathbf{P}_\mathrm{i} - {\mathbf{F_c}}^{t_2})/{\mathbf{F_s}}^{t_2} \rangle]$
  $\mathcal{L} \leftarrow \mathcal{L}_{reg}(\mathbf{X}^{t_2}, \mathbf{X}^{t_2})$
  $J(\mathbf{R}_\mathrm{i}, \mathbf{t}_\mathrm{i}) = \mathcal{L}_{reg}\left(\mathbf{X}^{t_2}, \mathbf{X}^{t_2}\right)$
  $\mathbf{R}_{\mathrm{i}+1} \leftarrow \mathbf{R}_\mathrm{i} - \eta \cdot \nabla_\mathbf{R} J(\mathbf{R}_\mathrm{i}, \mathbf{t}_\mathrm{i})$
  $\mathbf{t}_{\mathrm{i}+1} \leftarrow \mathbf{t}_\mathrm{i} - \eta \cdot \nabla_\mathbf{t} J(\mathbf{R}_\mathrm{i}, \mathbf{t}_\mathrm{i})$
end
/* Terminate Iteration */
Output: $\mathbf{R}, \mathbf{t} \leftarrow \arg\min_{\mathbf{R},\mathbf{t}} \mathcal{L}$

The regularization forces the scale to be one and the center of the shape to be at the origin because ShapeNet [2] meshes are placed at their canonical space. The centroid regularization provides signals to correct the center of gravity of a partial point cloud to its actual center in the canonical space during training.

The final loss is computed as follows:

$$\mathcal{L} = \omega_{\text{SDF}}\,\mathcal{L}_{\text{SDF}} + \omega_{\text{center}}\,\mathcal{L}_{\text{center}} + \omega_{\text{scale}}\,\mathcal{L}_{\text{scale}}\,, \quad (4)$$

We follow [7] and set $\omega_{\text{SDF}} = 1.0$, $\omega_{\text{center}} = 0.2$, and $\omega_{\text{scale}} = 0.01$.

E. Algorithms

Registration. We provide the details of our proposed registration in Algorithm 1. We set $\mathbf{X}^{t_1}$ as source and $\mathbf{X}^{t_2}$ as target in the registration and use the encoder Φ and decoder Ψ for initialization and optimization, respectively. $\mathrm{Kabsch}(\cdot,\cdot)$ denotes Kabsch algorithm [6]. $\mathbf{P}_\mathrm{i}$ denotes the source point cloud transformed by the estimated $(\mathbf{R}_\mathrm{i}, \mathbf{t}_\mathrm{i})$ at i^{th} iteration. J denotes the analytical Jacobians of $(\mathbf{R}_\mathrm{i}, \mathbf{t}_\mathrm{i})$. ∇ denotes the gradient of parameters.

Joint Optimization. We provide the algorithmic details of our proposed joint optimization in Algorithm 2. We take accumulated point clouds of each instance $\{\mathbf{X}^t | t \in \{1, ..\mathrm{T}\}\}$ as input. $\mathrm{REG}()$ denotes the registration algorithm in Algorithm 1 and we use it to initialize the pose graph $\mathbf{G}$. We compute the loss $\mathcal{L}_{joint}$ of the accumulated point cloud, which is the sum of the SDF loss $\mathcal{L}_{sdf}$ and the regularization loss $\mathcal{L}_z$.

Algorithm 2: Joint Optimization

Data: $\{\mathbf{X}^t | t \in \{0, 1, ..\mathrm{T}-1\}\}$
/* Initialization */
$\mathbf{F}^1, \mathbf{F}^1, ...\mathbf{F}^T \leftarrow \Phi(\mathbf{X}^0)$;
for $t = 0, ..., T-1$ **do**
  $\mathbf{T}_t \leftarrow \mathrm{REG}({\mathbf{F_{eqv}}}^t, {\mathbf{F_{eqv}}}^{t+1})$;
end
$\mathbf{G} \leftarrow \{T_t\}_{t=1}^T$;
$\mathbf{F}^* \leftarrow \arg\min_\mathbf{F} \{\mathcal{L}_{sdf}(\mathbf{F}^t)\}_{t=1}^T$;
ϵ: learning rate, $I \leftarrow 200$: number of steps;
/* Iteration */
while i $< I$ **do**
  $\mathcal{L}_{joint} \leftarrow 0$;
  for $t = 0, ..., T-1$ **do**
    $\mathcal{L}_{joint}$ += $\mathcal{L}_{sdf}(\mathbf{X}^t) + \mathcal{L}_z(\mathbf{F}_\mathrm{i})$;
  end
  /* Update using Adam */
  $[\mathbf{F}_\mathrm{i}, \mathbf{G}_\mathrm{i}] \leftarrow \mathrm{AdamUpdate}(\mathcal{L}_{joint}, \epsilon)$;
  i $\leftarrow$ i $+ 1$;
end
/* Terminate Iteration */
Output: $[\mathbf{F}, \mathbf{G}] = \arg\min_{\mathbf{F},\mathbf{G}} \mathcal{L}_{joint}$

F. Data Processing

F.1. Training Data

The network is trained under the supervision of Signed Distance Fields (SDFs). In order to generate the training samples, we compute the SDF for every shape in the training set of the ShapeNet [2] *subset*. We partly follow the processing pipelines of [7, 8], which include three steps: *(i)* making the mesh watertight, *(ii)* generating point clouds from partial mesh renderings, and *(iii)* sampling SDFs.

Making the mesh watertight. Watertight meshes usually describe meshes consisting of one closed surface. This means that they do not contain holes and have a clearly defined interior [11]. The CAD models in ShapeNet are non-watertight. We use MeshFusion[1] [12] to process raw meshes to watertight.

Generating point clouds from mesh renderings. To mimic partial observations as in real-world datasets, we render depth maps of meshes from multiple views. First, we construct a sphere around the mesh, fully covering it and placing it exactly at the center. We uniformly sample 24 points on the sphere as focal points of the depth camera. To ensure that the majority of the shape is within the field of view, we place the principal point on the line connecting

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

the focal point and the center of the sphere by solving the camera orientation $\mathbf{R}$:

$$z \begin{bmatrix} w/2 \\ h/2 \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{T}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{5}$$

where $\mathbf{K}$ denotes camera intrinsics and $[\mathbf{R}|\mathbf{T}]$ transformation from world frame to camera frame. w and h are the dimensions of the image and $[X, Y, Z, 1]^T$ are the coordinates of the focal point in the world frame. $\mathbf{R}$ is the unknown to solve. We render depth maps at the 24 sampled positions for every mesh using OpenGL [17] and back-project them to 3D space as point clouds.

Sampling SDFs [7]. We sample SDF values around meshes. We adopt two sampling strategies: uniform sampling and near-surface sampling (*c.f.* Fig. 3). Uniform sampling captures the global structure of the shape and near-surface sampling captures high-frequency (detailed) signals. For each mesh, we sample 10^5 SDF samples, 50% uniform and 50% near the surface.

F.2. FlyingShape Dataset

We synthesize the *FlyingShape* dataset based on the *subset* of ShapeNet's test set. It consists of scenes that have been captured repeatedly at irregular intervals (temporal scans), in between which consisting objects have been moved. The number of objects in each scene ranges from four to eight. As the number of objects per category in the *subset* are different, we balance their frequency when randomly drawing samples. To replicate the partial completeness of a scene in real captures, we generate the scan of each scene by rendering depth images from random viewpoints on the upper hemisphere within the scene and un-projecting the depth pixels into the 3D space. We combine the point clouds resulting from three rendered views. We create 100 scenes in total, with each scene containing five temporal scans. We generate annotations on instance segmentation, associations, and transformations directly from the ground truth. Sample scenes are shown in Fig. 8.

F.3. *3RScan* Dataset

The *3RScan* dataset [14] provides raw RGB-D scans with known poses. As our method reasons on point cloud, we downsample the RGB-D frames to reduce the point cloud density and back-project them to obtain the raw scan as our input. We use the semantic maps of 23 categories:

armchair, bed, bench, chair, coffee table, computer desk, couch, couch table, cushion, desk, dining chair, dining table, footstool, ottoman, pillow, rocking chair, round table, side table, sofa, sofa chair, stand, table and *trash can*.

We set the other semantic labels as background and do not process them. We use the official toolbox of [14] and generate the instance masks for our generated point cloud from the dataset's semantic mesh. As [14] only provides the mesh reconstruction on the level of a scene, we use the instance masks to extract the individual mesh (vertices and faces) of each instance in the scene as ground truth.

G. Evaluation Metrics

In this section, we provide the formulas for all the evaluation metrics we used in the main paper, in the order of the three sequential tasks.

G.1. Instance Matching

Instance-level matching recall. This metric measures the fraction of correctly matched instance pairs over ground truth instance pairs within the entire dataset.

$$\text{Instance Recall} = \frac{\#\text{ correct matches}}{\#\text{ total matches}} \tag{6}$$

Scene-level matching recall. To understand the performance on a scene level, since each scene has multiple instances, we compute instance recall only for instance pairs that exist in the scene and check what fraction of scenes have recall $> \tau$. In the main paper, we set τ to four values 25%, 50%, 75%, and 100%, of which the first three are used for *3RScan* [14] and the last three are used for *FlyingShape*, as *3RScan* is a real-world dataset and is more challenging than *FlyingShape*.

G.2. Point Cloud Registration.

Rotation error (RE). It measures the geodesic distance between two rotations:

$$\text{RE} = \arccos\left(\frac{\text{trace}\left(\mathbf{R}^{\mathrm{T}}\overline{\mathbf{R}}\right) - 1}{2}\right), \tag{7}$$

where $\mathbf{R}$ denotes the predicted rotation matrix and $\overline{\mathbf{R}}$ the ground truth. $\text{trace}()$ denotes the trace of a matrix.

Registration recall (RR). It is the fraction of rotation errors smaller than a threshold. We use as thresholds $\text{RE} < 5^\circ$ for *FlyingShape* and $\text{RE} < 10^\circ$ for *3RScan* [14]. We use a higher threshold for 3RScan because the accuracy of ground truth transformations is lower, since they were obtained using Procrustes on manually annotated 3D keypoint correspondences between two temporal scans [14]. *FlyingShape* is synthesized with no introduced errors.

Transformation error. It is the root mean square error of per-point distance between point clouds transformed by

prediction and ground truth poses.

$$\text{RMSE} = \sqrt{\frac{1}{M+N}\left(\sum \left\|\mathbf{T}_{\mathbf{P}}^{\mathbf{Q}}(\mathbf{p}) - \overline{\mathbf{T}}_{\mathbf{P}}^{\mathbf{Q}}(\mathbf{p})\right\|_2^2 + \sum \left\|\mathbf{T}_{\mathbf{Q}}^{\mathbf{P}}(\mathbf{q}) - \overline{\mathbf{T}}_{\mathbf{Q}}^{\mathbf{P}}(\mathbf{q})\right\|_2^2\right)} \tag{8}$$

where $\mathbf{T}_{\mathbf{P}}^{\mathbf{Q}}$ denotes the transformation from $\mathbf{P}$ to $\mathbf{Q}$ and vice versa. M, N denote the number of points in $\mathbf{P}, \mathbf{Q}$, respectively.

Chamfer distance (CD). It measures the quality of registration. Here we follow [5, 19] and use the modified chamfer distance metric:

$$\tilde{CD}(\mathbf{P}, \mathbf{Q}) = \frac{1}{|\mathbf{P}|}\sum_{\mathbf{p}\in\mathbf{P}} \min_{\mathbf{q}\in\mathbf{Q}_{\text{raw}}} \left\|\mathbf{T}_{\mathbf{P}}^{\mathbf{Q}}(\mathbf{p}) - \mathbf{q}\right\|_2^2 + \frac{1}{|\mathbf{Q}|}\sum_{\mathbf{q}\in\mathbf{Q}} \min_{\mathbf{p}\in\mathbf{P}_{\text{raw}}} \left\|\mathbf{q} - \mathbf{T}_{\mathbf{P}}^{\mathbf{Q}}(\mathbf{p})\right\|_2^2, \tag{9}$$

where $\mathbf{P}_{\text{raw}} \in \mathbb{R}^{M\times 3}$ and $\mathbf{Q}_{\text{raw}} \in \mathbb{R}^{N\times 3}$ are *raw* source and target point clouds, and $\mathbf{P} \in \mathbb{R}^{1024\times 3}$ and $\mathbf{Q} \in \mathbb{R}^{1024\times 3}$ are *input* source and target point clouds.

Empirical Cumulative Distribution Function (ECDF). This metric measures the distribution of a set of values

$$\text{ECDF}(x) = \frac{|\{o_i < x\}|}{|O|}, \tag{10}$$

where O is a set of values (RE and RMSE in our case) and $x \in [\min\{O\}, \max\{O\}]$ [5]. We compute the ECDF curves w.r.t. rotation error and transformation error in the main paper.

G.3. Instance Reconstruction

We follow the definition from [8, 10]. Let $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$ be the sets of all points that are inside or on the surface of the predicted and ground truth mesh, respectively.

Chamfer-$L1$. We follow [10] and define reconstruction accuracy and completeness:

$$\text{Accuracy}\left(\mathcal{M}_{\text{pred}} \mid \mathcal{M}_{\text{GT}}\right) \equiv \frac{1}{|\partial\mathcal{M}_{\text{pred}}|}\int_{\partial\mathcal{M}_{\text{pred}}} \min_{\mathbf{q}\in\partial\mathcal{M}_{\text{GT}}} \|\mathbf{p}-\mathbf{q}\| \mathrm{d}\mathbf{p} \tag{11}$$

$$\text{Complete.}\left(\mathcal{M}_{\text{pred}} \mid \mathcal{M}_{\text{GT}}\right) \equiv \frac{1}{|\partial\mathcal{M}_{\text{GT}}|}\int_{\partial\mathcal{M}_{\text{GT}}} \min_{\mathbf{p}\in\partial\mathcal{M}_{\text{pred}}} \|\mathbf{p}-\mathbf{q}\| \mathrm{d}\mathbf{q} \tag{12}$$

Here $\partial\mathcal{M}_{\text{GT}}$ and $\partial\mathcal{M}_{\text{GT}}$ are the surfaces of $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$, respectively. The Chamfer-L_1 can be defined as below:

$$\text{Chamfer-}L_1\left(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}\right) = \frac{1}{2}\left(\text{Accuracy}\left(\mathcal{M}_{\text{pred}} \mid \mathcal{M}_{\text{GT}}\right) + \text{Completeness}\left(\mathcal{M}_{\text{pred}} \mid \mathcal{M}_{\text{GT}}\right)\right) \tag{13}$$

Volumetric IoU. It is equal to the intersection divided by the union of two sets:

$$\text{IoU}\left(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}\right) \equiv \frac{|\mathcal{M}_{\text{pred}} \cap \mathcal{M}_{\text{GT}}|}{|\mathcal{M}_{\text{pred}} \cup \mathcal{M}_{\text{GT}}|}. \tag{14}$$

We follow the implementation of ConvONet [10] to compute IoU: randomly sample 100K points from the bounding boxes of each mesh and determine if the points lie inside or outside $\mathcal{M}_{\text{pred}}$ and $\mathcal{M}_{\text{GT}}$, respectively.

SDF Recall. It is designed by us to evaluate reconstruction quality when the watertight mesh of ground truth is not available:

$$\text{SDF Recall:} = \frac{1}{K}\sum_{\mathbf{v}\in\mathbf{V}} \mathbb{1}(\mathbf{SDF}(\mathbf{v})), \tag{15}$$

where $\mathbb{1}(\mathbf{SDF}(\mathbf{v})) = 1$ if $\mathbf{SDF}(\mathbf{v}) < 0.05$ else 0. $\mathbf{V}$ denotes the set of vertices of the ground truth mesh and $\mathbf{v}$ each vertex therein. We evaluate by computing the mean absolute SDF errors of vertices in the ground truth mesh (not necessarily watertight) w.r.t. the predicted mesh using library [16] and calculate the ratio of vertices with an SDF error smaller than the threshold. We set the threshold of SDF values to 0.05 because the threshold for near-surface samples during training is 0.1 and we divide it by two during evaluation.

H. Additional Qualitative Results

We provide the following qualitative results: *(1)* point cloud accumulation on *FlyingShape* with GT input (*c.f.* Fig. 7); *(2)* point cloud registration on *FlyingShape* at instance level with GT input (*c.f.* Fig. 6); *(3)* end-to-end multi-object relocalization on *3RScan* (*c.f.* Fig. 9); and *(4)* end-to-end multi-object relocalization and reconstruction (Fig. 10). In these additional visualizations, we provide more comprehensive qualitative evaluations, showing success and failure cases of our method and the baseline. The baseline method is the same per task as the one used in the main paper.

I. Discussion

In this paper, we introduce MoRE^2, a novel method to parse long-term evolving environments. Through our extensive experiments on two datasets, we demonstrate the superior performance and robustness of our method even on partial, noisy point clouds with pose variations.

Limitations *(1)* Our method includes test-time optimizations and hence cannot run end-to-end in real-time. We consider it as a post-processing algorithm for temporal scans to understand the instance-level change (rigid motion and geometry) between them. The capture of indoor environment does not happen in real-time but instead has relatively

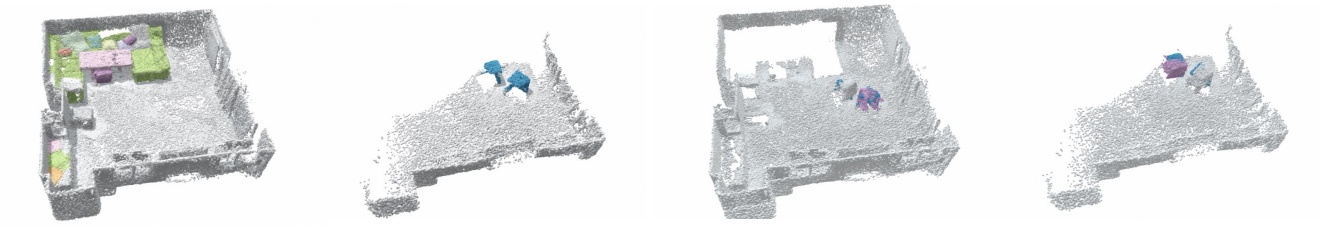

Figure 5. **A failure case of relocalization in *3RScan* [14].** The failure is due to the large number of similar objects in the scene *i.e.*, pillows, and the incomplete/low coverage of the scene in the rescan (target). As many instances in the source scan are no longer observed in the target scan, our method abandons unmatched ones as removed.

long intervals. Therefore, real-time execution is not a necessity in this task. *(2)* Our method faces challenges when dealing with multiple identical, similar, and/or symmetric shapes in the scene (*c.f*. Fig. 5). This can be alleviated in future work by incorporating RGB-values and global context in the scene into our method.

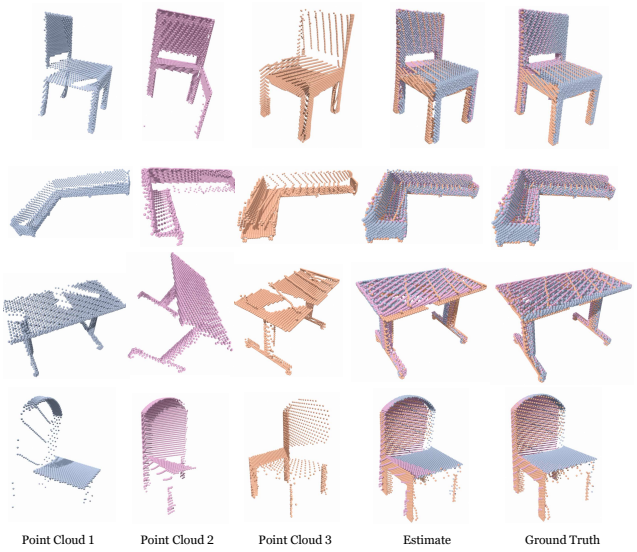

Figure 6. **Qualitative results of point cloud registration on *FlyingShape*.** Left three columns are the three input point clouds. Ground truth on the right. Our method does not look for correspondences between two point clouds, *e.g.*, the back and front of the chair in the last row: our method first tries to complete the instance surface based on partial observations and then registers the completed zero-level set.

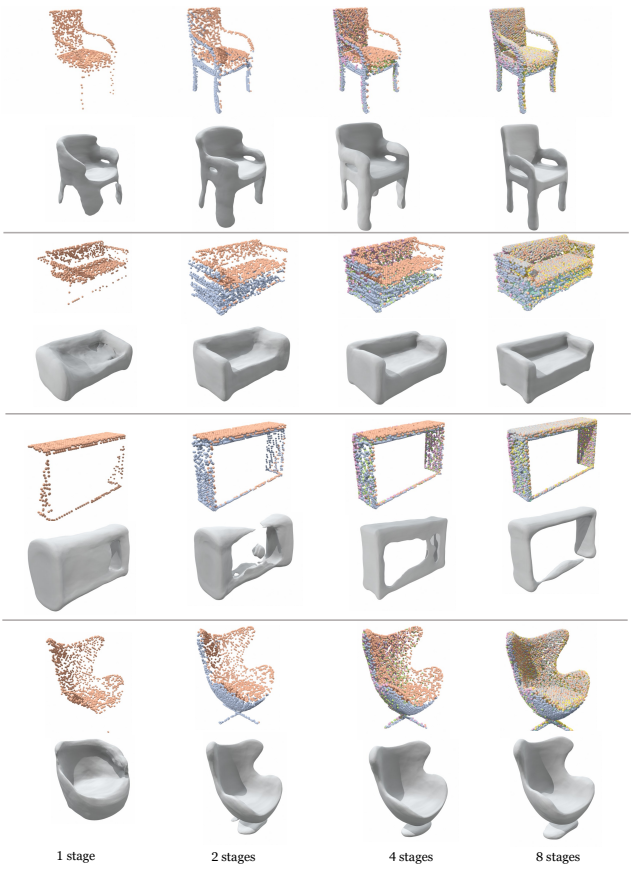

Figure 7. **Qualitative results of point cloud accumulation at instance level on *FlyingShape*.** This figure is corresponding to Figure. 7 (in the main paper), showing the ever-increasing quality and completeness of reconstruction with MoRE^2 when more data are accumulated. Each temporal point cloud is showcased in a different color. Per example, top row shows the registration and bottom row shows the reconstruction results.

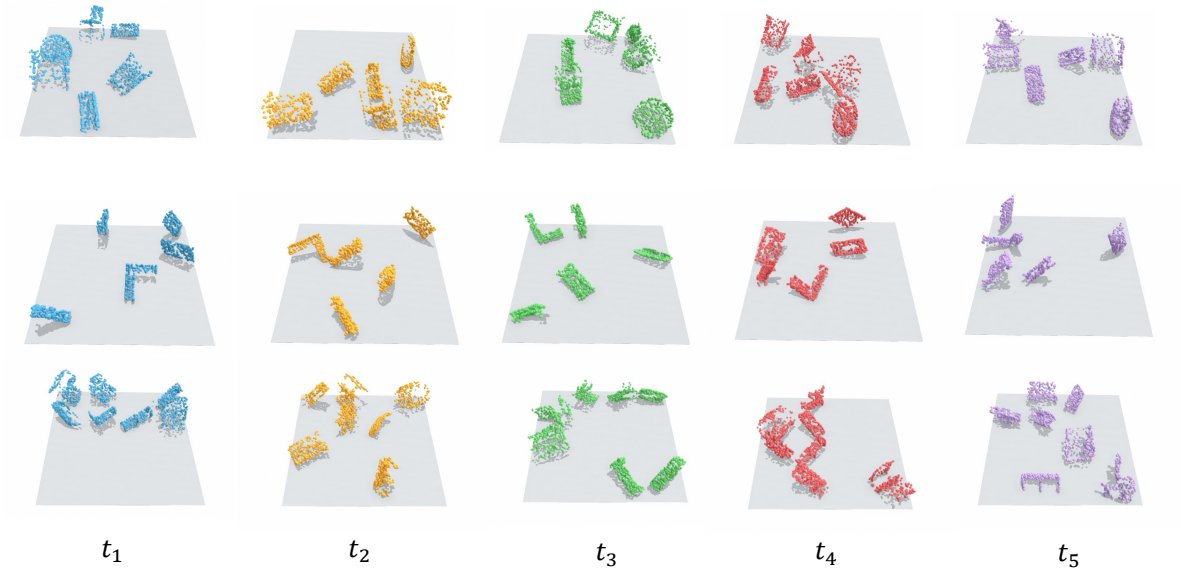

Figure 8. **Samples of changing scenes from *FlyingShape*.** Each scene has five temporal scans, shown in different colors.

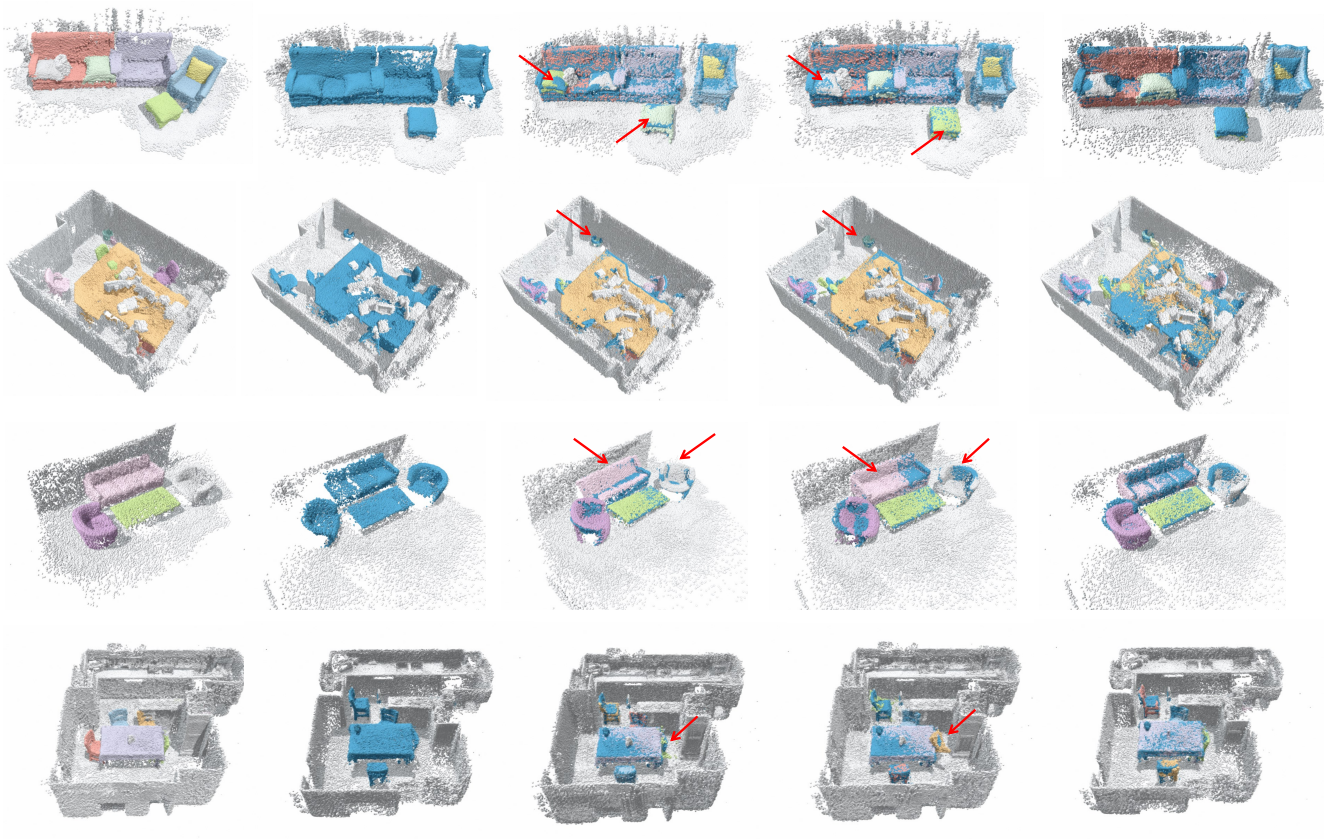

Figure 9. **Qualitative results of relocalization on *3RScan* [14].** MoRE^2 generates more correct matches thus better localization of moved instances. ↘ highlights the differences between baseline and ours. In the first row, the baseline mismatched pillows with ottoman, leading to wrong registration. In the second row, our method outperforms the baseline on the trash can. In the third row, the baseline wrongly flipped the sofa colored in pink. In the last row, we present one scene with multiple identical chairs, where both methods fail in relocalizing all of them. Ours relocalizes only one and the baseline none of them.

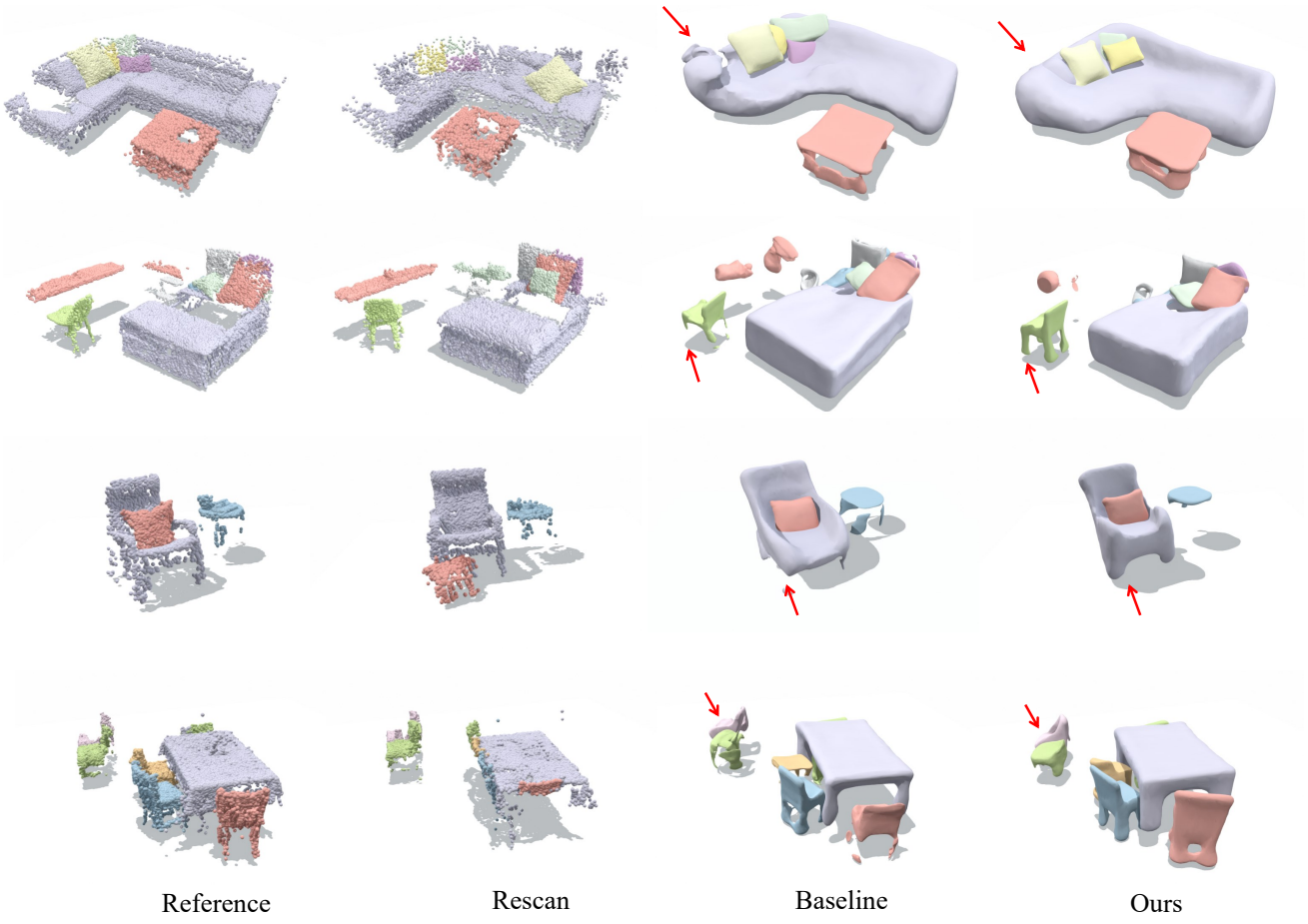

Figure 10. **Qualitative results of end-to-end performance on *3RScan* [14].** Rescans are accumulated to the reference scan. The reconstruction of the scene is based on the accumulation, including the errors and noises provided from performing the task end-to-end. Compared to the baseline, our method is able to reconstruct cleaner and more complete surfaces by accumulating partial observations.

References

[1] Serge Assaad, Carlton Downey, Rami Al-Rfou, Nigamaa Nayakanti, and Ben Sapp. Vn-transformer: Rotation-equivariant attention for vector neurons. *Transactions on Machine Learning Research*, 2023.

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.

[3] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *CVPR*, 2021.

[4] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[5] Shengyu Huang, Zan Gojcic, Mikhail Usvyatsov, Andreas Wieser, and Konrad Schindler. Predator: Registration of 3d point clouds with low overlap. In *CVPR*, 2021.

[6] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976.

[7] Jiahui Lei, Congyue Deng, Karl Schmeckpeper, Leonidas Guibas, and Kostas Daniilidis. EFEM: Equivariant neural field expectation maximization for 3d object segmentation without scene supervision. In *CVPR*, 2023.

[8] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019.

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

[10] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020.

[11] David Stutz. A formal definition of watertight meshes. https://davidstutz.de/a-formal-definition-of-watertight-meshes/, 2018.

[12] David Stutz and Andreas Geiger. Learning 3d shape completion under weak supervision. *arXiv:1805.07290*, 2018.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[14] Johanna Wald, Armen Avetisyan, Nassir Navab, Federico Tombari, and Matthias Nießner. Rio: 3d object instance re-localization in changing indoor environments. In *CVPR*, 2019.

[15] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (ToG)*, 38(5):1–12, 2019.

[16] Francis Williams. Point cloud utils, 2022. https://www.github.com/fwilliams/point-cloud-utils.

[17] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[18] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *NeurIPS*, 2019.

[19] Zi Jian Yew and Gim Hee Lee. Rpm-net: Robust point matching using learned features. In *CVPR*, 2020.