Neural Kernel Surface Reconstruction
— Appendix —

Jiahui Huang[1] Zan Gojcic[1] Matan Atzmon[1] Or Litany[1] Sanja Fidler[1,2,3] Francis Williams[1]

[1]NVIDIA [2]University of Toronto [3]Vector Institute

In this appendix, we first provide more details of our method, including necessary network designs and derivations in § A. Details related to experiments, including hyperparameters, metrics, and baselines are documented in § B. The design of our pipeline allows for different extensions applicable to various scenarios, and these extensions are in § C. Finally, more visualizations of our results are shown in § D and the accompanying video clips.

A. Detailed Method

A.1. Network Architecture

We use a customized version of a U-Net-like structure that operates fully over sparse voxels and outputs an adaptive hierarchy for kernel field computation.

Point Encoder. Given the input point cloud $\boldsymbol{X}_{\text{in}}$ and voxel size W for the finest level, we first identify the set of points that resides within each voxel. For each voxel, we then run a residual PointNet[12]-like encoder network to pool all the points within it into a feature vector. The network is illustrated in Fig. S1. To allow for translational equivariance, we convert from the global coordinates of the input points to local coordinates within the voxels as input $\tilde{\boldsymbol{X}}_{\text{in}}$. For most of the datasets demonstrated in the main paper, per-point normal $\boldsymbol{N}_{\text{in}}$ is required as an additional piece of information to disambiguate the orientations. This information does not have to be very accurate and usually can be easily obtained from sensor positions $\boldsymbol{O}_{\text{in}}$. $\tilde{\boldsymbol{X}}_{\text{in}}$ and $\boldsymbol{N}_{\text{in}}$ are concatenated as a 6-dimensional input that is fed into the point encoder.

U-Net Encoder. After quantizing per-point information into voxel-level features, we obtain a sparse voxel grid. We then apply a sequential sparse convolution layers [4] sandwiched by max pooling layers to coarsen the voxels, as shown in the upper part of Fig. S2. Deeper layers have larger receptive field and conceptually cover the area of $2^{l-1}W$.

U-Net Decoder. As a reverse process of encoding, the decoder of the sparse U-Net also consists of several convolution layers with nearest-neighbour-based up-sampling, as illustrated in the lower part of Fig. S2. Skip connections are added to encourage fusion of low-level and high-level information. For each layer we append additional task-specific branches to the backbone features that regress the following attributes needed in the following procedure:

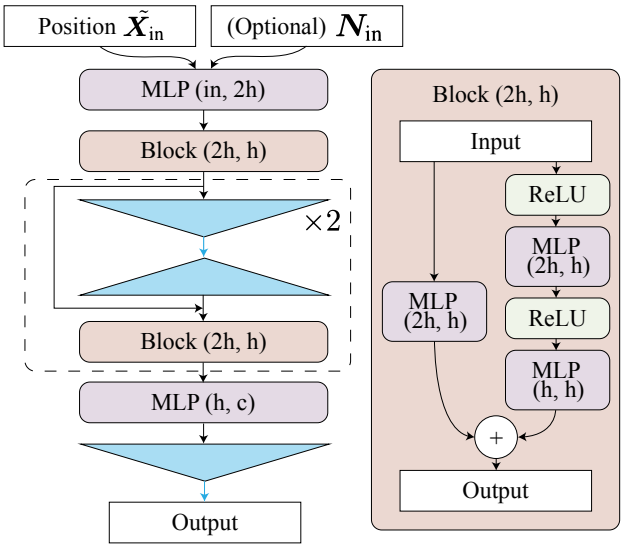

Figure S1: **Architecture for the point encoder.** Rounded rectangles show operations over each point, where h is the hidden dimension. Blue triangles denote max pooling and repeating operations.

- Structure prediction branch outputs 3-dimensional features to determine the structure of the output hierarchy. Details are presented in the next paragraph.
- Normal prediction branch (■) outputs the normals $\boldsymbol{n}_i^{(l)}$ that is 3-dimenensional and used later in the linear system. Note that there is no direct supervision to this branch and we find such a strategy provide better results due to the additional degrees of freedom introduced.
- Kernel prediction branch (★) outputs the features $\boldsymbol{z}_i^{(l)} \in \mathbb{R}^d$ that is defined in the main text. MLP followed by Bézier interpolation are used to obtain the

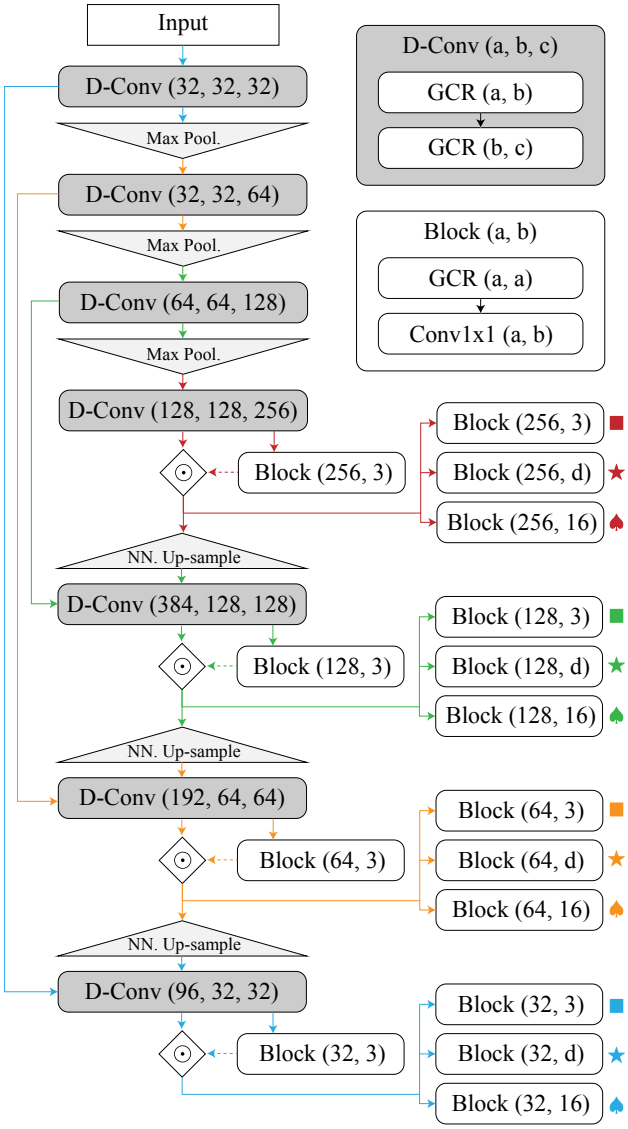

Figure S2: **Architecture for the U-Net.** 'GCR' means a sequential application of GroupNorm, Convolution and ReLU activation. '⊙' denote voxel masking using the structure prediction results. Different colors of the arrows represent features at different layers in the hierarchy.

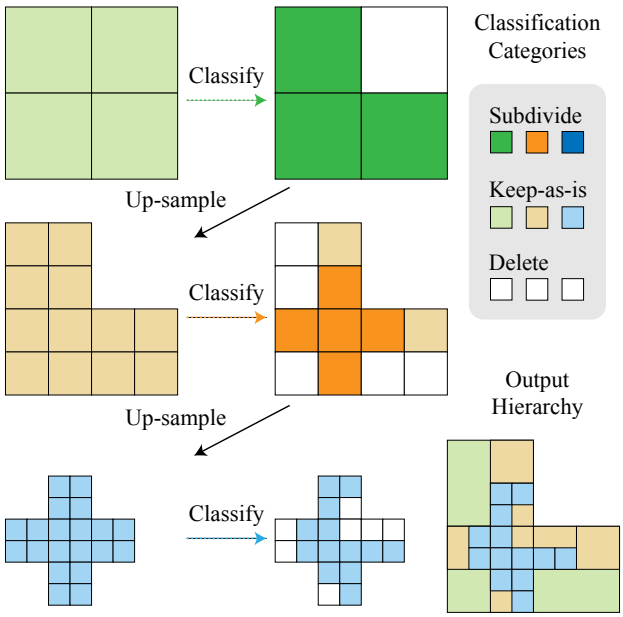

Figure S3: **Structure prediction.** We use a $L = 3$ hierarchy as an example here. According to different classification results, the voxels will be treated differently.

kernel field at arbitrary position.

- Mask prediction branch (♠) outputs 16-dimenensional features and are later transformed to a scalar value by MLP that determines whether the query position is far away from the real surface. In the main text the masking module is denoted as function $\varphi(\cdot)$.

Structure Prediction. We treat the 3-dimenensional features from the structure prediction branch as a 3-way classification score for each voxel. Based on the classification, the voxels will be treated differently and altogether form a predicted new hierarchy, with the guarantee that the region defined by finer voxel is always covered by coarser voxels. The semantics for these classifications are as follows:

1. `Subdivide`: the voxel should be subdivided into 8(= 2^3) voxels in the finer level.
2. `Keep-as-is`: the voxel should be treated as a leaf node in the hierarchy, *i.e.*, it is neither subdivided nor deleted.
3. `Delete`: the voxel should be deleted from the hierarchy.

An illustration for the structure prediction is shown in Fig. S3. Note that the hierarchy forms on the fly with the decoding process, and the other feature prediction branches are based only on the existing voxels (*i.e.*, not classified as `Delete`).

A.2. Hierarchical Kernel Formulation

We now give a detailed description of our hierachical neural kernel field formulation and procedure for solving for coefficients during inference. We then prove several facts about our formulation: namely that our learned kernel is indeed a kernel, that our predicted implicit belongs to an RKHS defined by that kernel, and that our linear system is symmetric and positive definite, and thus corresponds to a Gram matrix.

Defining the Neural Kernel Field. Recall that our shape is encoded as the zero level set of a Neural Kernel Field $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ defined as a weighted combination of *positive definite kernels* which are conditioned on the inputs and centered at the midpoints $\boldsymbol{x}_i^{(l)} \in \mathbb{R}^3$ of each voxel in the predicted hierarchy:

$$f_\theta(\boldsymbol{x}|\boldsymbol{X}_{\text{in}}, \boldsymbol{N}_{\text{in}}) = \sum_{i,l} \alpha_i^{(l)} K_\theta^{(l)}(\boldsymbol{x}, \boldsymbol{x}_i^{(l)}|\boldsymbol{X}_{\text{in}}, \boldsymbol{N}_{\text{in}}), \quad \text{(S.1)}$$

where $\alpha_i^{(l)} \in \mathbb{R}$ are scalar coefficients at the i^{th} voxel at level $l = 1, \dots L$ in the hierarchy, and $K_\theta^{(l)}$ is the predicted kernel for the l^{th} level defined as

$$K_\theta^{(l)}(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi_\theta^{(l)}(\boldsymbol{x}; \boldsymbol{X}_{\text{in}}, \boldsymbol{N}_{\text{in}}), \phi_\theta^{(l)}(\boldsymbol{x}'; \boldsymbol{X}_{\text{in}}, \boldsymbol{N}_{\text{in}}) \rangle \cdot K_{\text{b}}^{(l)}(\boldsymbol{x}, \boldsymbol{x}').$$

Here, $\langle \cdot, \cdot \rangle$ is the dot product, $\phi_\theta^{(l)} : \mathbb{R}^3 \to \mathbb{R}^d$ is the feature field extracted from the l^{th} level of the hierarchy, and $K_b^{(l)} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ is the *Bézier Kernel*:

$$K_b^{(l)}(\boldsymbol{x}, \boldsymbol{x}') = \psi^2\left(\frac{\boldsymbol{x}_x - \boldsymbol{x}'_x}{2^{l-1}W}\right) \cdot \psi^2\left(\frac{\boldsymbol{x}_y - \boldsymbol{x}'_y}{2^{l-1}W}\right) \cdot \psi^2\left(\frac{\boldsymbol{x}_z - \boldsymbol{x}'_z}{2^{l-1}W}\right),$$

with $\psi^2 : \mathbb{R} \to \mathbb{R}$ the univariate second order B-spline:

$$\psi^2(s) = \begin{cases} (s + \frac{3}{2})^2 & \text{if } s \in [-\frac{3}{2}, -\frac{1}{2}] \\ -2s^2 + \frac{3}{2} & \text{if } s \in [-\frac{1}{2}, \frac{1}{2}] \\ (s - \frac{3}{2})^2 & \text{if } s \in [\frac{1}{2}, \frac{3}{2}] \\ 0 & \text{otherwise} \end{cases}$$

which decays to zero in a one-voxel (at level-l) neighborhood around its origin.

Lemma 1. *The basis functions (S.1) used to construct our hierarchy are positive definite kernels.*

Proof. The kernel $K_\theta^{(l)}(\boldsymbol{x}, \boldsymbol{x}')$ at each level is defined as the dot product between features $\phi_\theta^{(l)}(\boldsymbol{x})$ and $\phi_\theta^{(l)}(\boldsymbol{x}')$ multiplied by the Bézier Kernel K_b. A kernel, by definition is a dot product of feature embeddings (Definition 2.8 in [14]), and the product of kernels is a kernel (Proposition 3.22 in [14]). Therefore each $K_\theta^{(l)}$ is a kernel. □

Remark 2. *Our functions f_θ defined on the hierarchy of kernels K_θ belong to an RKHS $\mathcal{H}$ induced by a kernel $\mathcal{K}$. This follows immediately from the Lemma 1 and the Moore–Aronszajn theorem [1].*

Computing a 3D Implicit Surface from Points. Recall that we compute an implicit surface by finding optimal coefficients $\boldsymbol{\alpha}^* = \{\{\alpha_i^{(l)}\}_{l=1}^{L}\}_{i=1}^{n^{(l)}}$ for the kernel field (S.1). *i.e.*, given the predicted voxel hierarchy, learned kernels $K_\theta^{(l)}$, and predicted normals $\boldsymbol{n}_j^{(l)}$, we minimize the following loss in the forward pass of our model (omitting the conditioning of f_θ on $\boldsymbol{X}_{\text{in}}, \boldsymbol{N}_{\text{in}}$ for brevity):

$$\boldsymbol{\alpha}^* = \underset{\alpha_i^{(l)}}{\text{argmin}} \sum_{l=1}^{L'} \sum_{i=1}^{n^{(l)}} \|\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x}_i^{(l)}) - \boldsymbol{n}_i^{(l)}\|_2^2 + \sum_{j=1}^{n_{\text{in}}} |f_\theta(\boldsymbol{x}_j^{\text{in}})|^2, \quad \text{(S.2)}$$

where $L' \leq L$ is a hyper-parameter for the hierarchy. We can rewrite (S.2) in matrix form

$$\underset{\boldsymbol{\alpha}}{\text{argmin}} \|\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{n}\|_2^2 + \|\mathbf{G}\boldsymbol{\alpha}\|_2^2,$$

where

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}^{(1)} & \dots & \mathbf{G}^{(L)} \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(1)} & \dots & \mathbf{Q}^{(L)} \end{bmatrix},$$
$$\boldsymbol{n} = \begin{bmatrix} \boldsymbol{n}_{1,x} & \boldsymbol{n}_{1,y} & \boldsymbol{n}_{1,z} & \dots & \boldsymbol{n}_{n_{\text{v}},x} & \boldsymbol{n}_{n_{\text{v}},y} & \boldsymbol{n}_{n_{\text{v}},z} \end{bmatrix}^\top,$$
$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1^{(1)} & \dots & \alpha_{n^{(1)}}^{(1)} & \dots & \alpha_1^{(L)} & \dots & \alpha_{n^{(L)}}^{(L)} \end{bmatrix}^\top.$$

Here $n_{\text{v}} = \sum_{l=1}^{L'} n^{(l)}$ and the matrix $\mathbf{G}$ is the Gram matrix of the kernel defined as:

$$\mathbf{G}_{i,j}^{(l)} = K_\theta^{(l)}(\boldsymbol{x}_i^{\text{in}}, \boldsymbol{x}_j^{(l)}),$$

and the matrix $\mathbf{Q}$ is the matrix of partial derivatives of $\mathbf{G}$ defined as:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(1)} & \dots & \mathbf{Q}^{(L)} \end{bmatrix}, \quad \mathbf{Q}^{(l)} = \begin{bmatrix} \mathbf{Q}_x^{(l)} & \mathbf{Q}_y^{(l)} & \mathbf{Q}_z^{(l)} \end{bmatrix}$$

with

$$\mathbf{Q}_{[x|y|z],i,j}^{(l)} = \partial_{[x|y|z]} K_\theta^{(l)}(\boldsymbol{x}_i^{\text{in}}, \boldsymbol{x}_j^{(l)}).$$

Setting the gradient with respect to $\boldsymbol{\alpha}$ of (S.2) to $\mathbf{0}$, we find that the optimal $\boldsymbol{\alpha}^*$ is the solution to the linear system:

$$(\mathbf{Q}^\top\mathbf{Q} + \mathbf{G}^\top\mathbf{G})\boldsymbol{\alpha} = \mathbf{Q}^\top\boldsymbol{n}.$$

Lemma 3. *The matrix $\mathbf{Q}^\top\mathbf{Q} + \mathbf{G}^\top\mathbf{G}$ used to solve for the coefficients $\alpha_i^{(l)}$ is symmetric and positive definite.*

Proof. The $n \times n$ matrix $\mathbf{Q}^\top\mathbf{Q}$ is symmetric and positive definite since $\forall \boldsymbol{x} \neq \mathbf{0}, x^\top\mathbf{Q}^\top\mathbf{Q}x = \|\mathbf{Q}\boldsymbol{x}\|_2^2 \geq 0$. Furthermore since $\mathbf{Q}$ is constructed as a concatenation of Gram Matrices, it is full rank and thus $\|\mathbf{Q}\boldsymbol{x}\|_2^2 > \mathbf{0}$. The same holds for $\mathbf{G}^\top\mathbf{G}$, and since the sum of positive definite matrices is positive definite, $\mathbf{Q}^\top\mathbf{Q} + \mathbf{G}^\top\mathbf{G}$ is positive definite. □

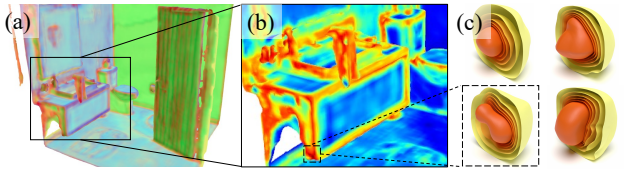

Figure S4: **Kernel visualization.** (a) PCA of the kernel features $\phi_\theta^{(l)}$. (b) Heatmap of kernel similarities w.r.t. one selected voxel in the dashed box. (c) Level sets of the kernel basis functions $K_\theta^{(l)}(\boldsymbol{x}, \boldsymbol{x}_j^{(l)})$.

Neural Kernel Visualization. The above learned kernel formulation makes the notion of inductive bias precise. Solutions to the ridge regression minimize the learned RKHS norm $\|\cdot\|_{\mathcal{H}}$ which controls the behavior of the fitted surfaces away from the input points. This norm tightly controls the inductive bias of solutions and is meta-learned to perform well on the reconstruction task. In Fig. S4, we perform PCA over the kernel features on the reconstructed surface and plot exemplar kernel basis functions $K_\theta^{(l)}(\boldsymbol{x}, \boldsymbol{x}_j^{(l)})$ in 3D. Note how similar geometries share similar learned kernels shown in the heatmap.

A.3. Additional Losses

Structure Loss. We compute a structure prediction loss on the predicted voxel hierarchy, written as:

$$\mathcal{L}_{\text{struct}} = \sum_{i,l} \text{Cross-Entropy}\left(\boldsymbol{c}_i^{(l)}, (\boldsymbol{c}_i^{(l)})_{\text{GT}}\right),$$

where $\boldsymbol{c}_i^{(l)} \in \mathbb{R}^3$ refers to the output of the structure prediction branch, and $(\boldsymbol{c}_i^{(l)})_{\text{GT}}$ is its ground-truth counterpart. To compute the ground-truth hierarchy, we apply the approach in OctField [15] to $\boldsymbol{X}_{\text{dense}}$ and $\boldsymbol{N}_{\text{dense}}$. Specifically, we start by building a dense hierarchy of the coarest level of voxels. Then we recursively subdivide a voxel (suppose the volume it takes is $R_i^{(l)}$) into 8 voxels when the following criterion is satisfied:

$$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{n})\in R_i^{(l)}}\left(\text{std.}(\boldsymbol{n}_x) + \text{std.}(\boldsymbol{n}_y) + \text{std.}(\boldsymbol{n}_z)\right) > 0.1,$$

where $\boldsymbol{x} \in \boldsymbol{X}_{\text{dense}}$ and std.$(\cdot)$ stands for standard deviation. Notably, we introduce another parameter L' for the hierarchy denoting the maximum adaptive depth. We run a second pass through the hierarchy to make sure that none of the voxels with depth $l > L'$ is a leaf node.

Masking Loss. To supervise $\varphi(\boldsymbol{x})$ for trimming spurious geometry from shapes with open surfaces, we apply a binary-cross-entropy loss, ensuring that points which are within the distance W from any point in $\boldsymbol{X}_{\text{dense}}$ are 1 and 0 otherwise.

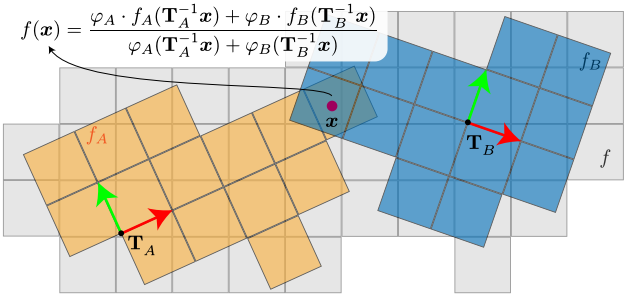

Figure S5: **Merging multiple reconstructions.** We demonstrate the merging operation with two chunks A and B. The final implicit value for point $\boldsymbol{x}$ is defined as the average of the two chunks, weighted by their predicted masking values.

A.4. Out-of-Core Reconstruction

When NKSR is applied to very large scenes with millions of points, the $\mathbf{G}$ and $\mathbf{Q}$ matrices become inevitably huge and could hardly fit into the GPU memory of a single video card. Hence, we opt to divide the large scenes into several chunks with overlap, run our full pipeline on each of the chunks and then merge the reconstructions in its implicit form. Due to the energy minimization nature of our algorithm, the overlapping regions of different chunks share the same constraints and are hence highly coherent. For outdoor scenes with open surfaces, we merge the implicit functions in a way that also considers the output of the masking module, as illustrated in Fig. S5.

Mathematically, the final merged implicit field f and the masking function φ are defined as:

$$f(\boldsymbol{x}) = \sum_k \varphi_k(\mathbf{T}_k^{-1}\boldsymbol{x}) f_k(\mathbf{T}_k^{-1}\boldsymbol{x}) / \sum_k \varphi_k(\mathbf{T}_k^{-1}\boldsymbol{x}),$$
$$\varphi(\boldsymbol{x}) = \max_k \varphi_k(\mathbf{T}_k^{-1}\boldsymbol{x}),$$

where the index k refers to the chunks whose regions cover $\boldsymbol{x}$, and $\mathbf{T}_k \in \mathbb{SE}(3)$ is the transformation of the chunk. To extract the triangular mesh, we build a new hierarchy encapsulating all the hierarchies of the chunks and run Dual Marching Cubes [13] over it.

B. Experimental Settings

B.1. Hyperparameters

Shared Parameters. To train the model we adopt a batch size of 4 using the technique of gradient accumulation. We use the Adam optimizer with an initial learning rate of 10^{-4}, and decay it to 70% every 50K iterations. The gradients are clipped with a norm threshold of 0.5 to protect the model under spurious gradients. To train the structure prediction branch along with other data branches, we use a warm-up

Table S1: **Dataset-specific hyperparameters.**

	ShapeNet	ABC	Room	CARLA
Scale	1.1^3	$\sim 1^3$	1^3	51.2m^2
Voxel size W	0.02	0.02	0.01	0.1
Adaptive depth L'	1	2	2	2
Kernel dim. d	16	4	4	4

strategy for the structures, where we start with the ground-truth structure and gradually increase the probability that the ground-truth structure is replaced with the predicted structure. We find this stabilizes training. We use the Jacobi-preconditioned Conjugate Gradient solver for both the forward and the backward passes of the linear solve, and set the convergence tolerance to 10^{-5}. The solver typically converges within several hundreds of iterations.

Dataset-Specific Parameters. Due to the different scales and attributes of the datasets we tested on, we empirically choose different parameters for them, as listed in Tab. S1. Notably, in the *kitchen-sink-model* (), we normalize all the training data to align with the scale of CARLA dataset during both training and testing. After normalization, the average number of points per voxel is around 5.

B.2. Baselines

SPSR [7]. We use the code from https://github.com/mkazhdan/PoissonRecon, and sets the voxel size (`width` parameter) to be the same as ours during comparison. For trimming we use the density values provided along with the mesh. We remove vertices with densities lower than a given quantile which we determine empirically for each dataset.

POCO [2]. We use the official implementation from https://github.com/valeoai/POCO. We tried our best to train a model with normal input (using the `normals` switch) but could not get a decently-performing model. *i.e.*, the quality of the generated meshes are consistently much worse than the version without normal input. Hence, for datasets where they do not provide a pretrained model, we train from scratch using our data without normals.

NGSolver [5]. We use the official implementation from https://github.com/huangjh-pub/neural-galerkin, taking the default configurations provided by the repository.

SAP [10] and ConvONet [11]. We use the implementation from https://github.com/autonomousvision/convolutional_occupancy_networks and https://github.com/autonomousvision/shape_as_points respectively and take the official configurations whenever possible. For comparisons with normal input, we modify their point encoder to accept an additional input of normal information through concatenation, similar to ours as in § A.1.

NKF [17]. We ask the original authors of the paper who kindly run all the comparisons for us because their code is not yet publicly available.

IMLSNet [8]. The implementation is taken from https://github.com/Andy97/DeepMLS and we use the default configurations to re-train their network for settings where pretrained models are not available.

TSDF-Fusion [16]. We choose to use the implementation from https://github.com/PRBonn/vdbfusion among all others due to its efficiency. As the algorithm requires sensor rays instead of points and normals, we generate pseudo-rays emitting from $\boldsymbol{x} + \epsilon \boldsymbol{n}$ and stopping at $\boldsymbol{x}$ as the input to their algorithm.

LIG [6]. We use the implementation from https://github.com/huangjh-pub/di-fusion with a pre-trained local implicit auto-encoder that takes normal input. Nearby local grids are blended with trilinear weights to ensure a smooth reconstruction.

B.3. Metrics

To compute the metrics, we densely sample points and the corresponding normals from both the ground-truth mesh (denoted as $\boldsymbol{X}_{\text{gt}}$ and $\boldsymbol{N}_{\text{gt}}$) and the predicted mesh (denoted as $\boldsymbol{X}_{\text{pd}}$ and $\boldsymbol{N}_{\text{pd}}$).

Chamfer Distance d_C. The Chamfer distance is computed using:

$$
\begin{aligned}
d_C &= \frac{1}{2}(\text{Comp.} + \text{Acc.}), \\
\text{Comp.} &= \frac{1}{|\boldsymbol{X}_{\text{gt}}|} \sum_{\boldsymbol{x}_{\text{gt}} \in \boldsymbol{X}_{\text{gt}}} \min_{\boldsymbol{x}_{\text{pd}} \in \boldsymbol{X}_{\text{pd}}} \|\boldsymbol{x}_{\text{gt}} - \boldsymbol{x}_{\text{pd}}\|, \\
\text{Acc.} &= \frac{1}{|\boldsymbol{X}_{\text{pd}}|} \sum_{\boldsymbol{x}_{\text{pd}} \in \boldsymbol{X}_{\text{pd}}} \min_{\boldsymbol{x}_{\text{gt}} \in \boldsymbol{X}_{\text{gt}}} \|\boldsymbol{x}_{\text{pd}} - \boldsymbol{x}_{\text{gt}}\|.
\end{aligned}
$$

Note that this is consistent with the one used in, *e.g.*, ConvONet [11] but different with the one used in POCO[1], hence the difference in the results.

Normal Consistency. The normal consistency score is defined as follows:

$$
\frac{1}{2}\left(\sum_{\boldsymbol{x}_{\text{gt}} \in \boldsymbol{X}_{\text{gt}}} |\langle \boldsymbol{n}_{\text{gt}}, \boldsymbol{n}_{\text{NN}(\boldsymbol{x}_{\text{gt}}, \boldsymbol{X}_{\text{pd}})} \rangle| + \sum_{\boldsymbol{x}_{\text{pd}} \in \boldsymbol{X}_{\text{pd}}} |\langle \boldsymbol{n}_{\text{pd}}, \boldsymbol{n}_{\text{NN}(\boldsymbol{x}, \boldsymbol{X}_{\text{gt}})} \rangle| \right).
$$

F-Score. The F-Score is defined as follows:

$$
\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},
$$

[1] https://github.com/ErlerPhilipp/points2surf/issues/20

Table S2: **Dataset specifications for CARLA.**

	Town1	Town2	Town3	Town10
Subset	Original	Original	Novel	Original
# Drives	3	3	3	4
# Chunks	93	93	90	124
# Avg. Points	510K	649K	546K	388K

where

$$\text{Precision} = \frac{|\{\boldsymbol{x}_{\text{pd}} \in \boldsymbol{X}_{\text{pd}} \mid \min_{\boldsymbol{x}_{\text{gt}} \in \boldsymbol{X}_{\text{gt}}} \|\boldsymbol{x}_{\text{gt}} - \boldsymbol{x}_{\text{pd}}\| < \xi\}|}{|\boldsymbol{X}_{\text{pd}}|},$$

$$\text{Recall} = \frac{|\{\boldsymbol{x}_{\text{gt}} \in \boldsymbol{X}_{\text{gt}} \mid \min_{\boldsymbol{x}_{\text{pd}} \in \boldsymbol{X}_{\text{pd}}} \|\boldsymbol{x}_{\text{pd}} - \boldsymbol{x}_{\text{gt}}\| < \xi\}|}{|\boldsymbol{X}_{\text{gt}}|}.$$

We use $\xi = 0.01$ for object-level and indoor datasets, and $\xi = 0.1$ for CARLA dataset.

B.4. Details on CARLA Dataset

We report detailed specifications of our generated CARLA dataset in Tab. S2. To obtain the input and ground-truth training pairs, we use a simulated LiDAR sensor that is mounted 1.8m above the ground, with a vertical field-of-view ranging from $-15°$ to $15°$ and an atmosphere attenuation rate of 4×10^{-3}.

C. Extensions

C.1. Texture Reconstruction

Our sparse neural kernel field representation defined over the hierarchy can be easily extended to represent other scene attributes, such as textures. The textures recovered can be defined continuously in the region covered by the hierarchy, similar to TextureField [9]. Specifically, we define 3 additional implicit functions $g_\phi^R, g_\phi^G, g_\phi^B$ for the red, green, and blue channel of the texture field as:

$$g_\phi^{[R|G|B]}(\boldsymbol{x}) = \sum_{i,l} \gamma_i^{(l),[R|G|B]} K_\phi^{(l),[R|G|B]}(\boldsymbol{x}, \boldsymbol{x}_i^{(l)}),$$

and the coefficients $\gamma_i^{(l)}$ can be obtained by solving the following linear system (using similar derivations as in § A.2, omitting R, G, B superscripts for brevity):

$$\mathbf{G}_c^\top \mathbf{G}_c \boldsymbol{\gamma} = \mathbf{G}_c^\top \boldsymbol{t}, \tag{S.3}$$

where $\mathbf{G}_c$ is the Gram matrix for the kernel K_ϕ and $\boldsymbol{t}$ is the input color vector.

To demonstrate our ability of texture reconstruction, we add 3 additional branches to our network backbone that predict kernel fields K_ϕ for the red, green and blue channel respectively. We overfit some examplar cars from ShapeNet [3] dataset with 10K colored input points and the results are shown in Fig. S6. We could accurately recover the textures along with the shape, showing a strong representation power for signals other than geomtry.

C.2. Outlier Detection

For input point clouds that are corrupted with outliers, the structure prediction branch can already prune many of them by not generating supporting voxels for regions that are faraway from the real surfaces. However, for outliers that are close to the surface, they act as false data constraints which should not be included in our linear system. To this end, we introduce a weighted version of our energy formulation (S.2) as follows:

$$\boldsymbol{\alpha}^* = \underset{\alpha_i^{(l)}}{\operatorname{argmin}} \sum_{l=1}^{L'} \sum_{i=1}^{n^{(l)}} \|\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x}_i^{(l)}) - \boldsymbol{n}_i^{(l)}\|_2^2 + \sum_{j=1}^{n_{\text{in}}} w_j^{\text{in}} |f_\theta(\boldsymbol{x}_j^{\text{in}})|^2,$$

where the highlighted variable $w_j^{\text{in}} \in [0, 1]$ is defined for each input point and predicted by an MLP (ended with Sigmoid) that relies on the trilinearly-interpolated backbone features of our U-Net.

The change in the energy formulation only requires a minor change in the linear system as:

$$(\mathbf{Q}^\top \mathbf{Q} + \mathbf{G}^\top \mathbf{W} \mathbf{G})\boldsymbol{\alpha} = \mathbf{Q}^\top \boldsymbol{n},$$

where $\mathbf{W} = \text{diag}(w_j^{\text{in}})$, and the gradients could also be propagated to the weights during training.

The model could then be trained without adding any extra supervision, and we show in Fig. S7 that the model could automatically learn the weights of the points in a meaningful way, where the model is trained and tested on 3K-point input with 50% of outliers.

D. More Visualizations

We provide more visualizations in Fig. S8, Fig. S9, Fig. S10, Fig. S11, Fig. S12 and Fig. S13.

References

[1] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950. 3

[2] Alexandre Boulch and Renaud Marlet. Poco: Point convolution for surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6302–6314, 2022. 5

[3] 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 preprint arXiv:1512.03012*, 2015. 6, 7

[4] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. 1

Figure S6: **Texture reconstruction.** Our sparse neural kernel hierarchy is expressive enough to faithfully represent the textures on the shape. We use 10K colored points sampled from ShapeNet [3] cars as input and iterate 800 times for each shape.

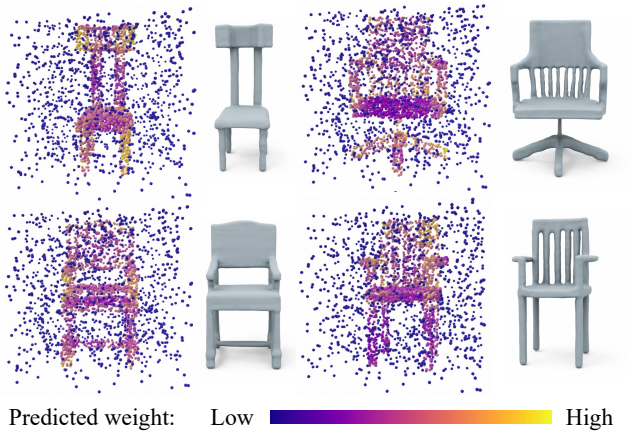

Figure S7: **Outlier detection and removal.** Given input points with extreme outliers (left), the model learns to automatically down-weigh irrelevant points and reconstructs good geometry (right).

[5] Jiahui Huang, Hao-Xiang Chen, and Shi-Min Hu. A neural galerkin solver for accurate surface reconstruction. *ACM Transactions on Graphics (TOG)*, 41(6), 2022. 5

[6] Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. Di-fusion: Online implicit 3d reconstruction with deep priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8932–8941, 2021. 5

[7] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3), 2013. 5

[8] Shi-Lin Liu, Hao-Xiang Guo, Hao Pan, Peng-Shuai Wang, Xin Tong, and Yang Liu. Deep implicit moving least-squares functions for 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1788–1797, 2021. 5

[9] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4531–4540, 2019. 6

[10] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, Andreas Geiger, et al. Shape as points: A differentiable poisson solver. *arXiv preprint arXiv:2106.03452*, 2021. 5

[11] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020. 5

[12] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017. 1

[13] Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, pages 70–76. IEEE, 2004. 4

[14] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004. 3

[15] Jia-Heng Tang, Weikai Chen, Jie Yang, Bo Wang, Songrun Liu, Bo Yang, and Lin Gao. Octfield: Hierarchical implicit functions for 3d modeling. *arXiv preprint arXiv:2111.01067*, 2021. 4

[16] Ignacio Vizzo, Tiziano Guadagnino, Jens Behley, and Cyrill Stachniss. Vdbfusion: Flexible and efficient tsdf integration of range sensor data. *Sensors*, 22(3), 2022. 5

[17] Francis Williams, Zan Gojcic, Sameh Khamis, Denis Zorin, Joan Bruna, Sanja Fidler, and Or Litany. Neural fields as learnable kernels for 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18500–18510, 2022. 5

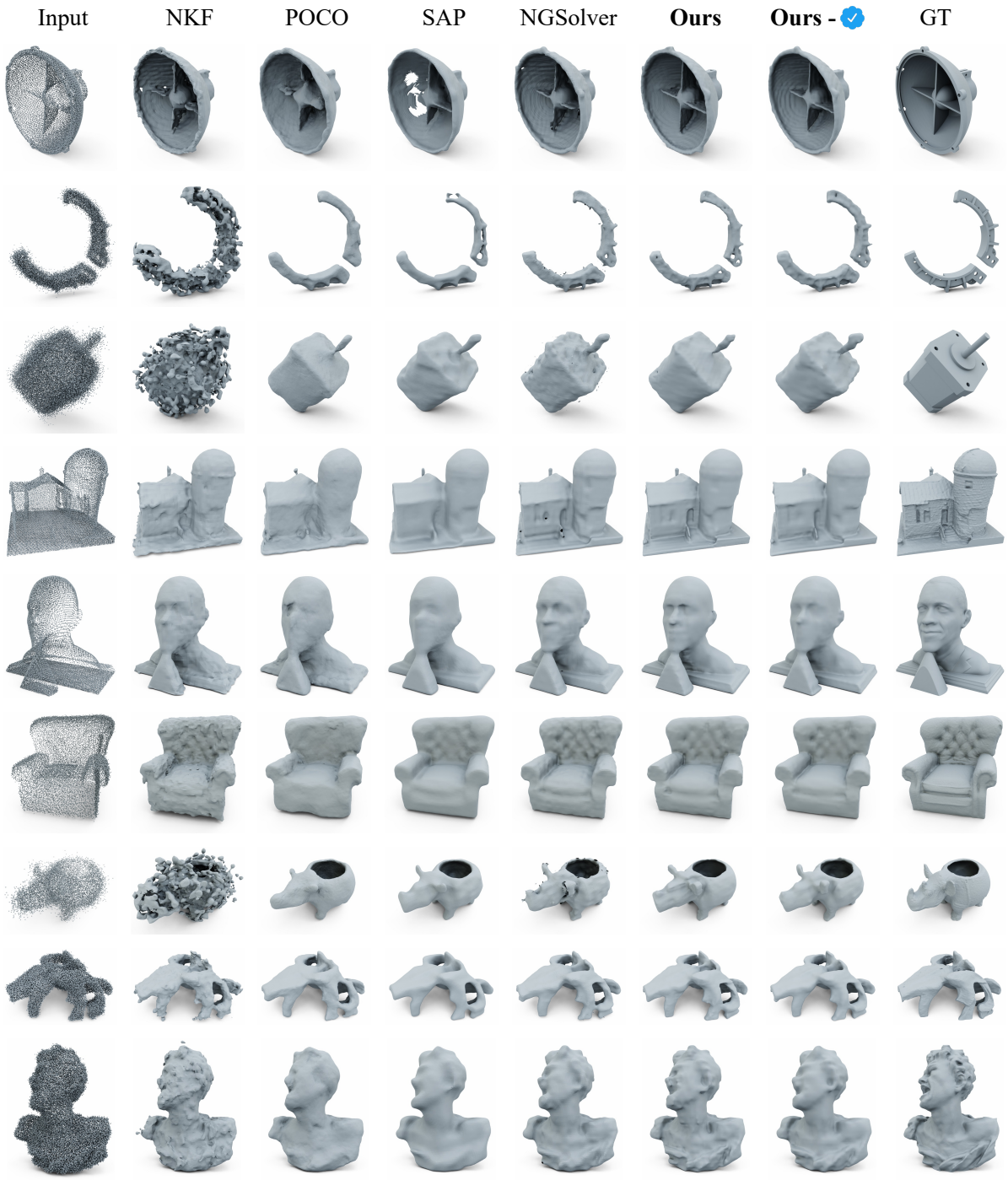

Figure S8: **More results on ABC/Thingi10K datasets.** Best viewed with 2× zoom-in.

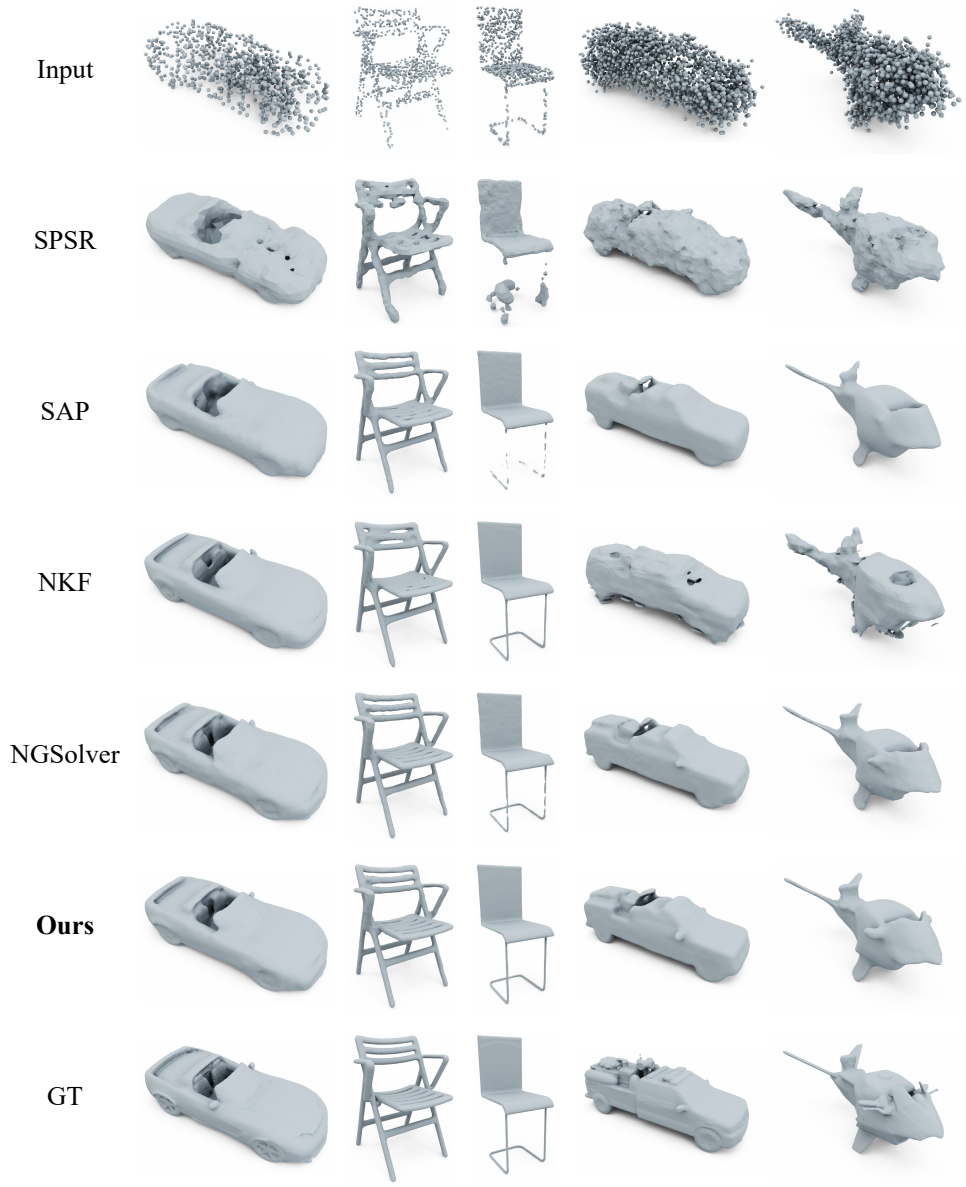

Figure S9: **More results on ShapeNet datasets (with normal).** Best viewed with 2× zoom-in.

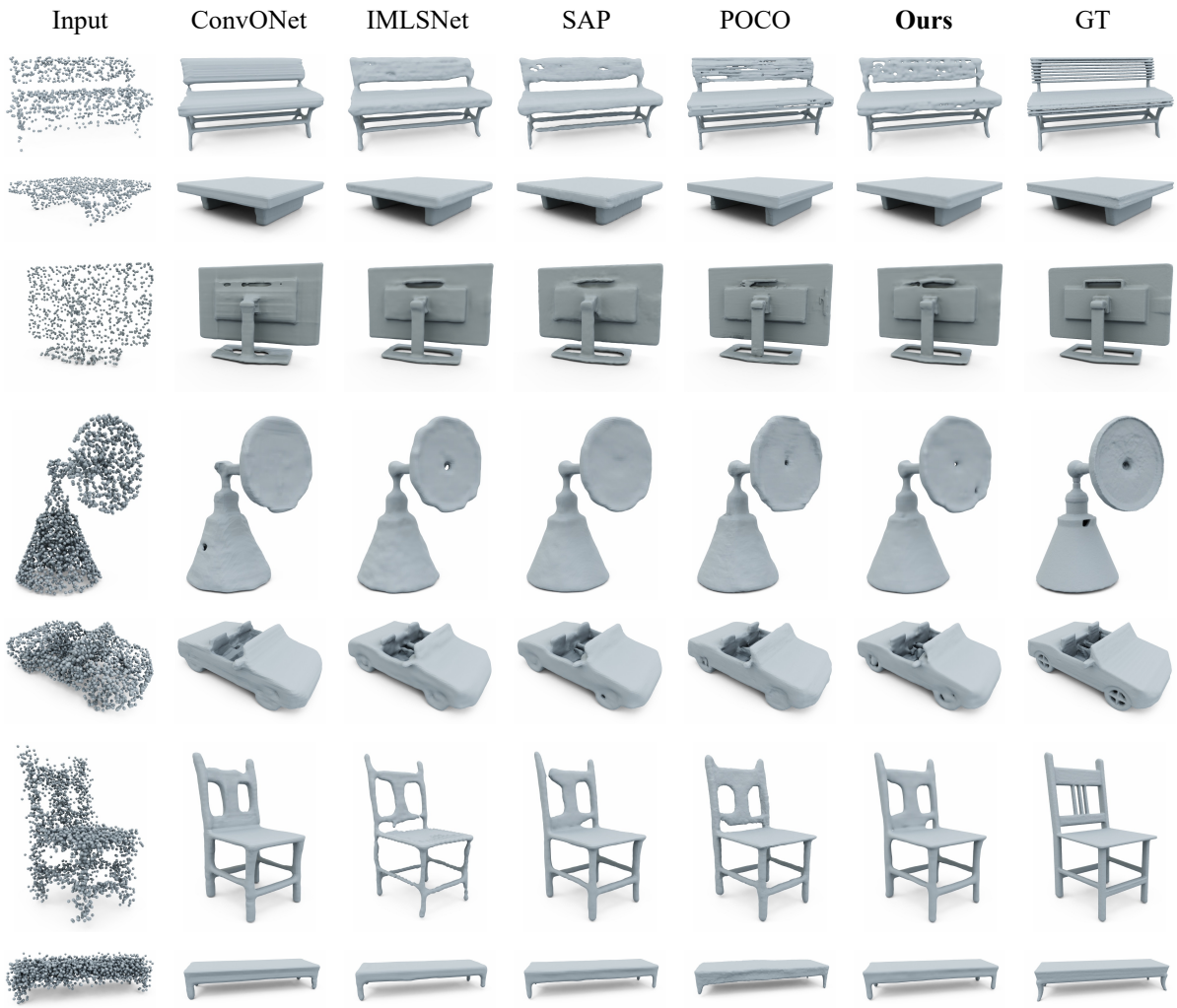

Figure S10: **More results on ShapeNet datasets (without normal).** Best viewed with 2× zoom-in.

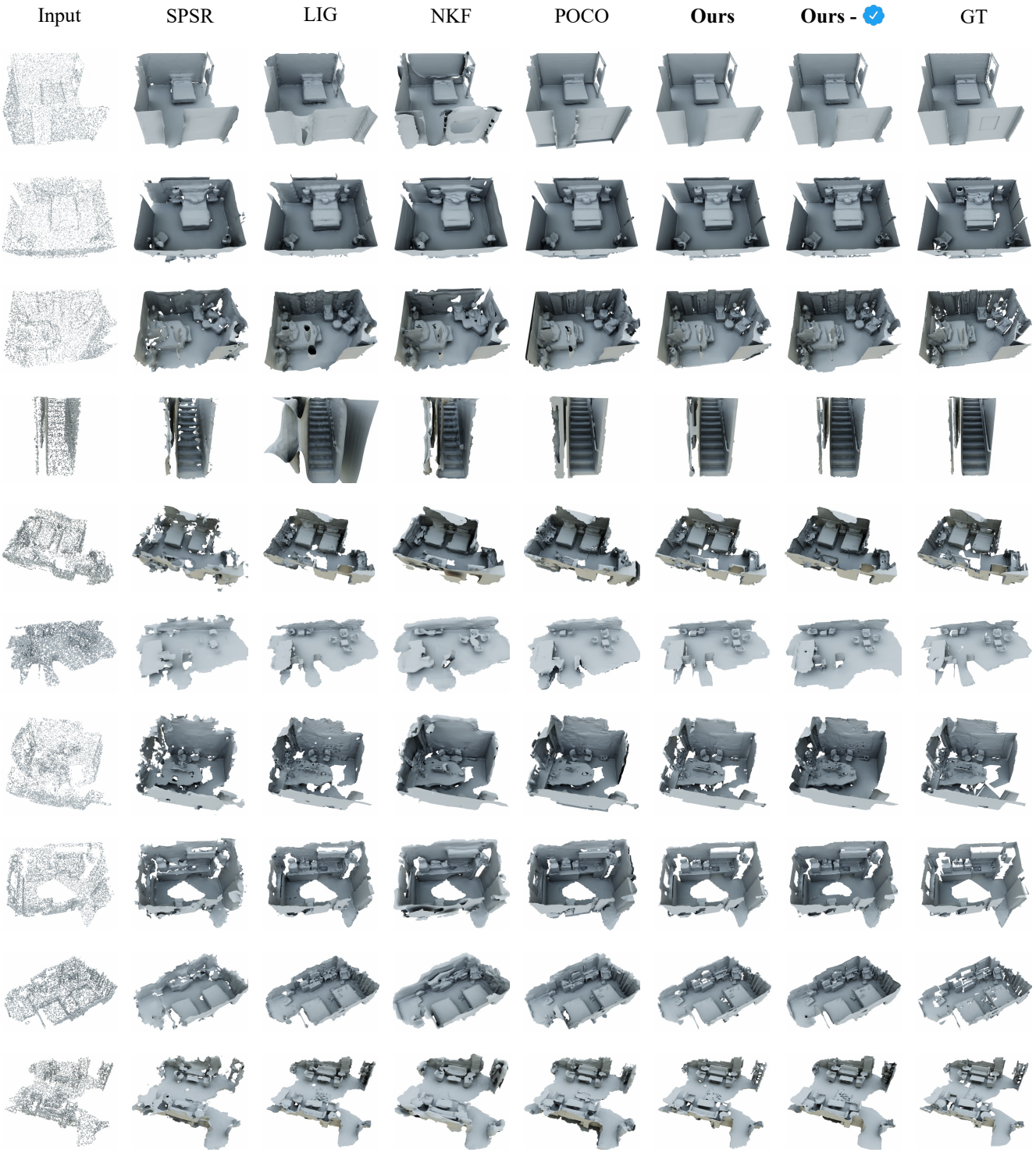

Figure S11: **More results on Matterport/ScanNet datasets.** Best viewed with 2× zoom-in.

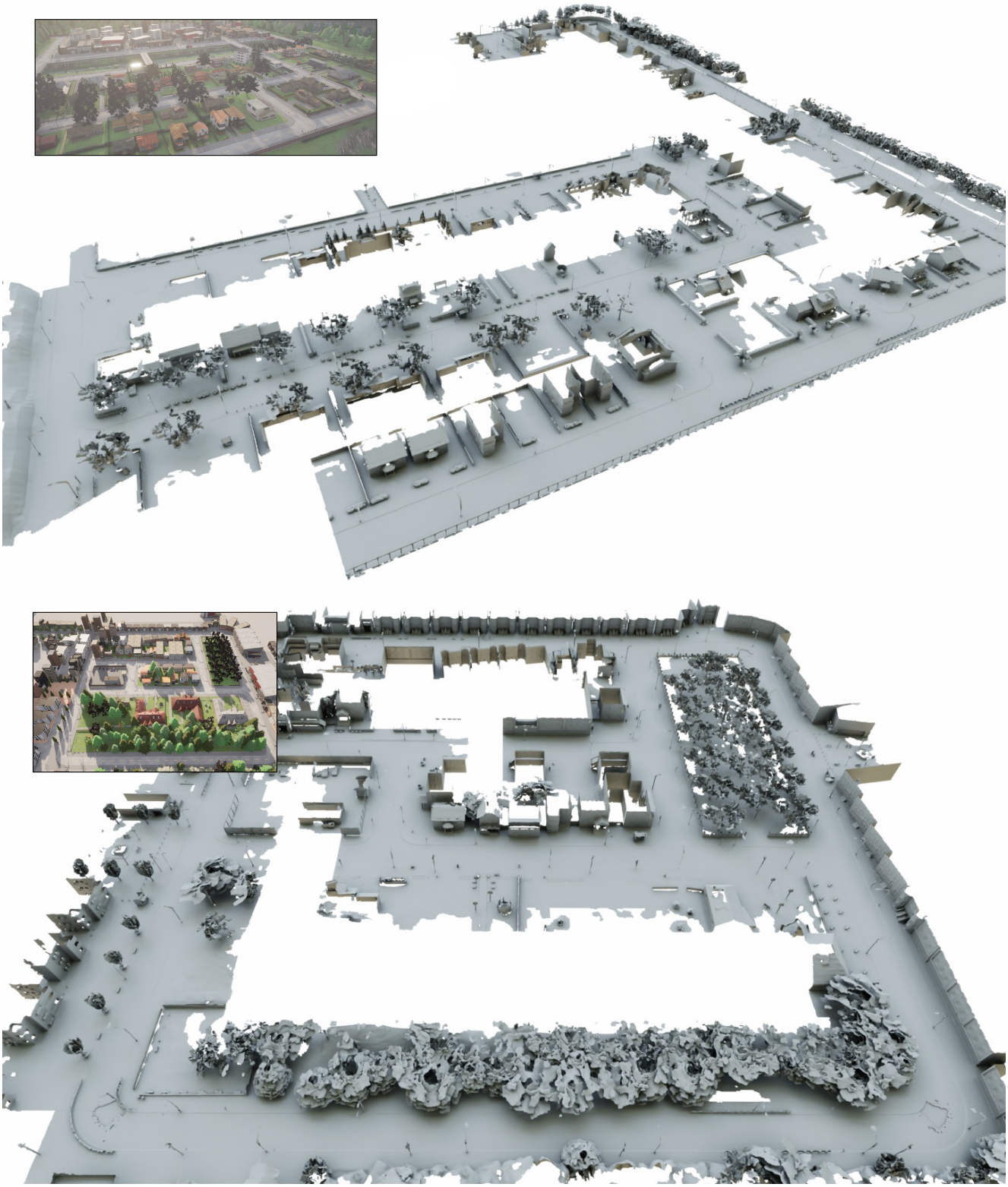

Figure S12: **Our reconstruction of the CARLA dataset.** The inset shows RGB rendering of the scene within the simulator.

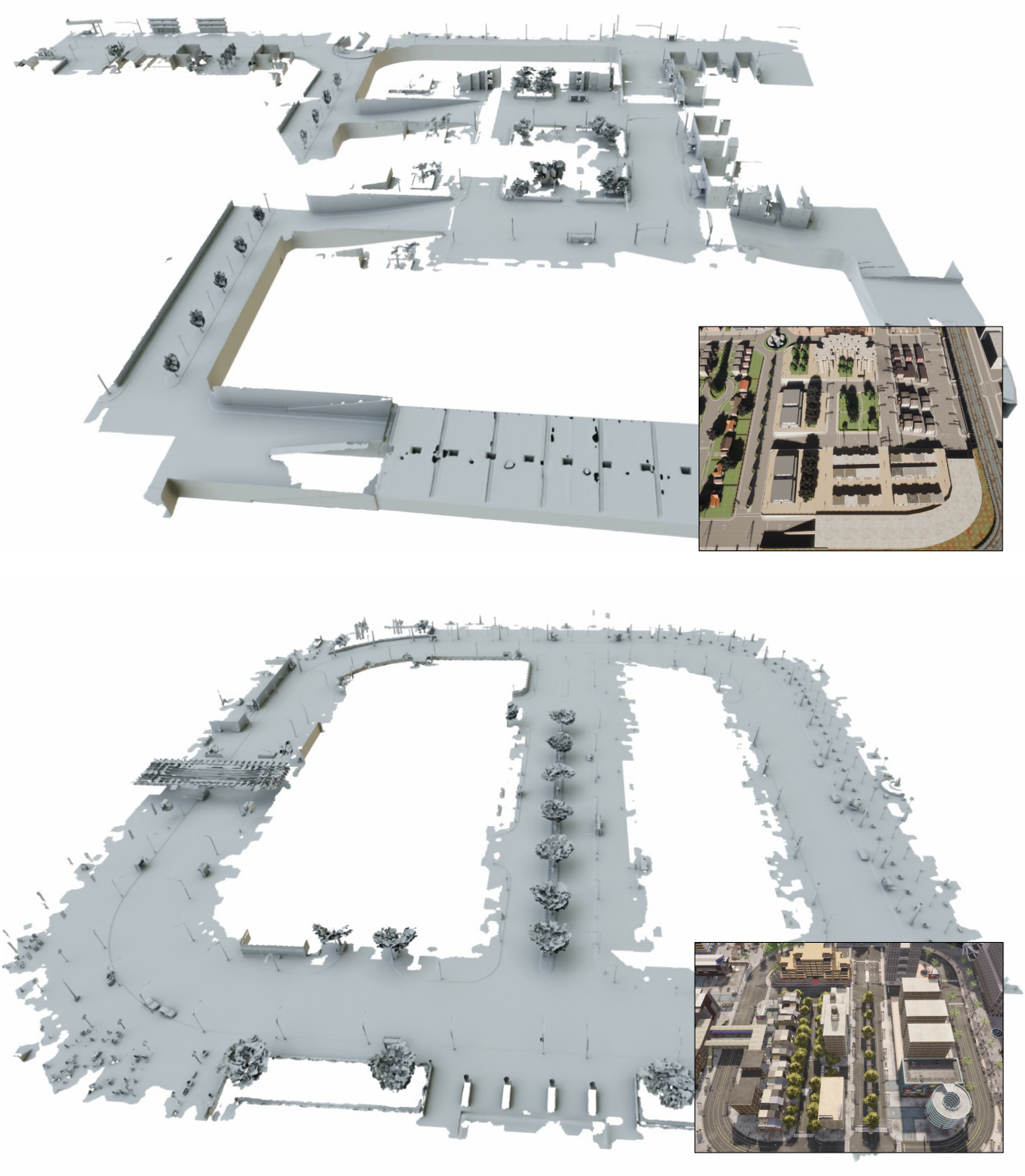

Figure S13: **Our reconstruction of the CARLA dataset.** The inset shows RGB rendering of the scene within the simulator.