

Neural Kernel Surface Reconstruction

Jiahui Huang¹ Zan Gojcic¹ Matan Atzmon¹ Or Litany¹ Sanja Fidler^{1,2,3} Francis Williams¹

¹NVIDIA ²University of Toronto ³Vector Institute

Project page: <https://research.nvidia.com/labs/toronto-ai/NKSR/>

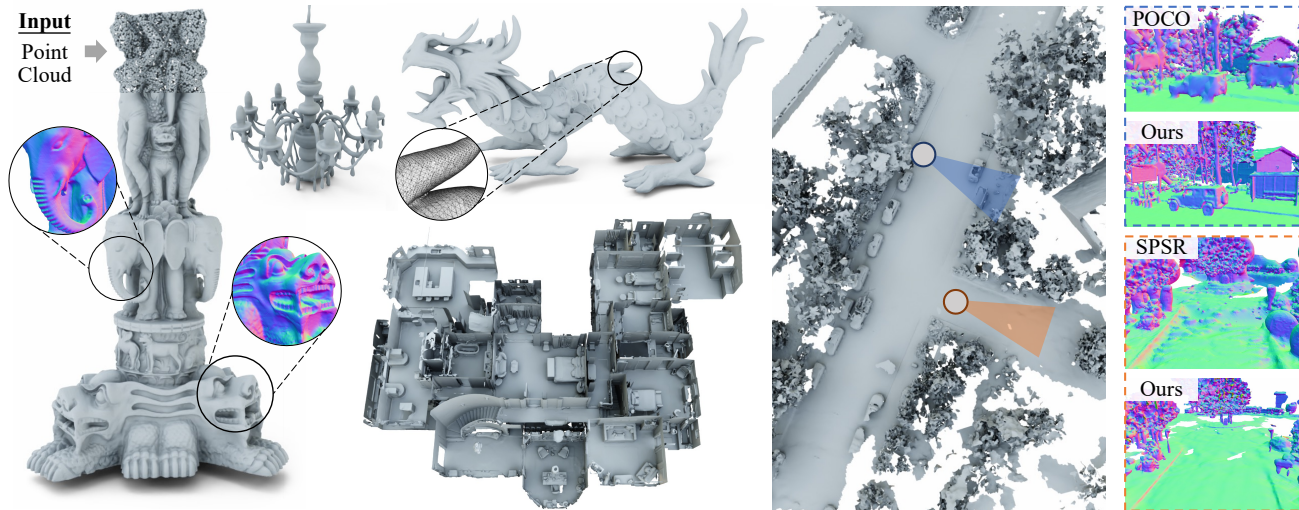


Figure 1: We present Neural Kernel Surface Reconstruction (NKSR) for recovering a 3D surface from an input point cloud. Trained directly from dense points, our method reaches state-of-the-art reconstruction quality and scalability. NKSR is also highly generalizable: All the meshes in this figure are reconstructed using a single trained model.

Abstract

We present a novel method for reconstructing a 3D implicit surface from a large-scale, sparse, and noisy point cloud. Our approach builds upon the recently introduced Neural Kernel Fields (NKF) [65] representation. It enjoys similar generalization capabilities to NKF, while simultaneously addressing its main limitations: (a) We can scale to large scenes through compactly supported kernel functions, which enable the use of memory-efficient sparse linear solvers. (b) We are robust to noise, through a gradient fitting solve. (c) We minimize training requirements, enabling us to learn from any dataset of dense oriented points, and even mix training data consisting of objects and scenes at different scales. Our method is capable of reconstructing millions of points in a few seconds, and handling very large scenes in an out-of-core fashion. We achieve state-of-the-art results on reconstruction benchmarks consisting of single objects (ShapeNet [6], ABC [36]), indoor scenes (ScanNet [12], Matterport3D [5]), and outdoor scenes (CARLA [17], Waymo [55]).

1. Introduction

The goal of 3D reconstruction is to recover geometry from partial measurements of a shape. In this work, we aim

to map a sparse set of oriented points sampled from the surface of a shape to a 3D implicit field. This is a challenging inverse problem since point clouds acquired from real-world sensors are often very large (millions or billions of points), vary in sampling density, and are corrupted with sensor noise. Furthermore, since surfaces are continuous but points are discrete, there are many valid solutions which can explain a given input. To address these issues, past approaches aim to recover surfaces that agree with the input points while satisfying some prior everywhere else in space. Classical methods use an explicit prior (e.g. smoothness), while more recent learning-based approaches promote a likely reconstruction under a data-driven prior.

There are, however, key limitations to both types of techniques that inhibit their application in practical situations. Since classical methods are fast, scalable, and able to handle diverse inputs, they have become an industry standard (e.g. [35, 68]). Yet, they suffer from quality issues in the presence of high noise or sparse inputs, often failing to reconstruct even simple geometry such as a ground plane (see the ground in Fig. 1). On the other hand, learning-based approaches were shown to handle large noise [46], and sparse inputs [43, 4], yet they often struggle to generalize to out-of-distribution shapes and sampling densities as was highlighted

in [65]. These generalization issues can be attributed to the fact that current learning-based methods struggle to exploit large and diverse amounts of data for training. One cause of this is that a single forward pass can take minutes for even moderately sized inputs (*e.g.* [4]), limiting training to collections consisting of small point clouds. Furthermore, many existing methods rely on a preprocessing step to extract supervision in the form of occupancy or signed distance function [47, 41, 44, 4, 65]. In practice, this preprocessing step hinders the ability to easily use diverse datasets for training since most shape datasets (including synthetic ones such as the popular ShapeNet [6]) consist of non-watertight shapes, open surfaces, or contain ghost geometry from which extracting supervision is hard.

Recently, [65] proposed Neural Kernel Fields (NKF), a new paradigm to address the problem of generalization in 3D reconstruction. NKF learns a data-dependent kernel, and predicts a continuous occupancy field as a linear combination of this kernel supported on the input points. The key insights of NKF are that a kernel explicitly encodes inductive bias, and that solving a kernel linear interpolation problem at test time always produces solutions that adhere to the inputs. Thus, by training on diverse shapes, NKF can learn a good inductive bias for the general 3D reconstruction problem rather than for a specific dataset. While NKF achieves impressive generalization results, it suffers from two major limitations that restrict its practical application. First, since it uses a globally supported kernel, it requires solving a dense linear system and cannot reconstruct inputs with more than ten thousand input points. Second, it degrades poorly in the presence of noise due to its interpolation of exact positional occupancy constraints.

In this work, we build upon the excellent generalization capability of NKF and tackle its main limitations to achieve a *practical* learning-based reconstruction method that is scalable, fast, and robust to noise. Like NKF, our work leverages the idea of a learned kernel for generalization, but we (1) develop a novel, gradient-based kernel formulation which is robust to noise, and (2) use an explicit voxel hierarchy structure and compactly supported kernels to make our interpolation problem sparse, multi-scale, and capable of handling large inputs while still producing high fidelity outputs. The result is a learning-based yet out-of-the-box reconstruction method that can be applied to point clouds in the wild. In particular, it enjoys all of the following properties:

- It can generalize to out-of-distribution inputs, producing high-fidelity reconstructions, even in the presence of sparsity and noise.
- It can be trained on the union of diverse datasets while only requiring dense oriented points as supervision, unlocking a new level of training data scale.
- It can reconstruct point clouds consisting of millions of points in seconds, and scale to extremely large inputs

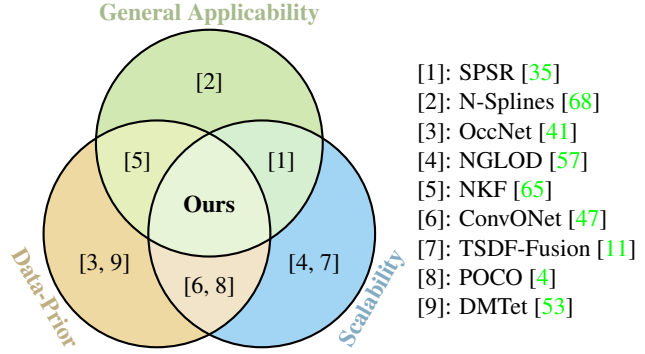


Figure 2: **Comparison to related works.**

in an out-of-core fashion.

We illustrate other methods in the context of these points visually in Fig. 2.

2. Related Work

We now give a brief overview of prior works that are relevant to our approach. Learned kernels were investigated in [69, 32, 45, 27] for tasks such as few-shot transfer learning, classification of images, or robot mapping. In the context of 3D reconstruction, Chu *et al.* [10] encoded the inductive bias intrinsically in a 3D CNN structure without training data. NKF [65] proposed a novel *data-dependent* kernel, which improved upon the non-learned kernel method derived from infinitely wide ReLU networks in Neural Splines [68]. Comparably, we use a data-dependent kernel but restrict its spatial support to increase computational efficiency and use a gradient-based fitting formulation to increase noise robustness. Mapping 3D points to a feature grid via a convolutional architecture was proposed in ConvONet [47] and CIRCLE [7] for predicting an occupancy field. POCO [4] improved the quality and performance of ConvONet by using a transformer architecture instead of convolutions. Both methods, however, take a long time to reconstruct even a small scene. Differently, our feature mapping is made efficient through a hierarchical sparse data structure. Non-dense data structures were studied in ASR [61] and DOGNN [63] which proposed octree-based convolutional architectures for reconstructing large scenes. Generalization to novel scenes was addressed by LIG [22, 34] using local patches which have smaller variability, but are very sensitive to the choice of patch size and relies on test-time optimization with unknown convergence properties. Similarly, our kernel weights are fitted to the scene at prediction but the fitting is done via a linear solver in a form of meta-learning [54]. Shape as Points [46] learns to upsample the input points followed by differentiable Poisson reconstruction, and this idea is further extended by NGSolver [33] to incorporate learnable basis functions. However, the representation power of the surface is still bounded by the chosen family of basis functions where

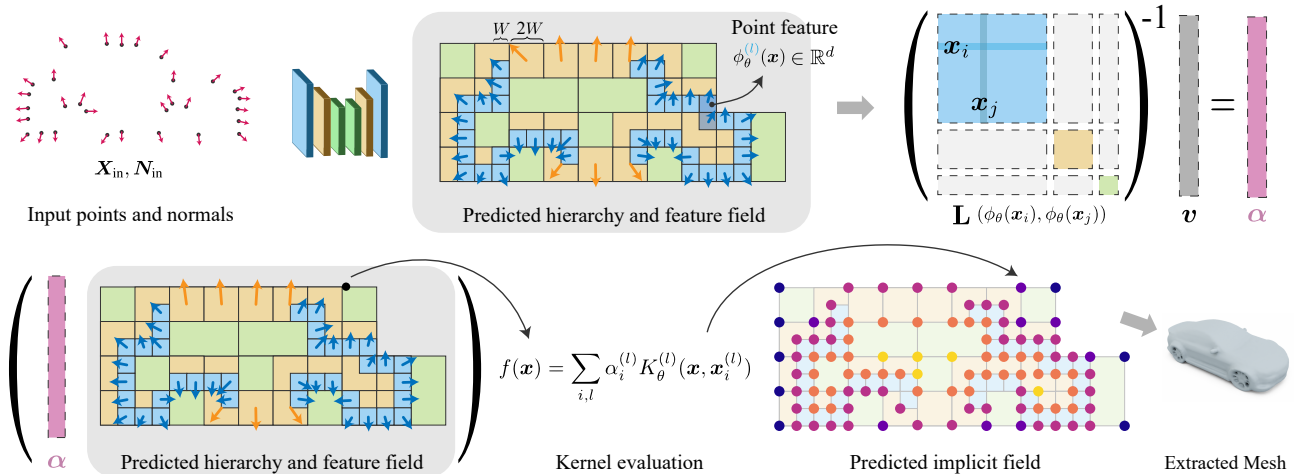


Figure 3: **Pipeline.** Our method accepts an oriented point cloud and predicts a sparse hierarchy of voxel grids containing features as well as normals in each voxel. We then construct a sparse linear system and solve for a set of per-voxel coefficients α . The linear system corresponds to the gram matrix arising from a kernel which depends on the predicted features, illustrated as \mathbf{L} and \mathbf{v} above but mathematically defined in Eq (4). To extract the predicted surface, we evaluate the function values at the voxel corners using a linear combination of the learned kernel basis functions, followed by dual marching cubes.

non-trivial integrations have to be applied. Beyond methods based on implicit surfaces, other shape reconstruction techniques exist which leverage different output representations. These representations include dense point clouds [50, 40, 76, 48, 49, 75, 56, 71, 72, 19, 37], polygonal meshes [31, 8, 21, 30, 24, 66, 13, 38, 28, 53], manifold atlases [67, 16, 26, 20, 3], and voxel grids [9, 59, 29, 70, 60, 23]. While our method uses a neural field for reconstruction, past work has used neural fields to perform a variety of point cloud tasks such as shape compression [57, 68], voxel grid upsampling [47, 41], reconstruction from rotated inputs [15, 2] and articulated poses [14, 73].

3. Method

Our method predicts a 3D surface given a point cloud with normals. We encode this predicted surface as the zero level set of a Neural Kernel Field, *i.e.* an implicit function represented as a weighted sum of learned, positive-definite basis functions which are conditioned on the input, and whose weights are computed using a linear optimization in the forward pass. These basis functions are supported on a sparse voxel hierarchy which we predict from the input point cloud using a sparse convolutional network, and depend on interpolated features at each voxel corner. In the following sections, we describe the key steps of our method, and summarize it pictorially in Fig. 3. We additionally provide rigorous derivations for each step in the Appendix.

3.1. Predicting a 3D Shape from Points

Given points and normals, the forward pass of our model predicts an implicit surface as a weighted sum of learned

kernels in two steps: First, we feed the input to a sparse convolutional network that predicts a voxel hierarchy with features at each corner (Fig. 4). These features define a collection of learned basis functions, which are centered at each voxel in the hierarchy. Second, we find a set of weights for these basis functions by solving a linear system that encourages the predicted implicit field to have a zero value near the input points, and to have gradients which agree with the input normals. Optionally, we can also predict a geometric mask, which outputs where in space to extract the final surface, trimming away spurious geometry.

Predicting a Sparse Voxel Hierarchy. Given input points $\mathbf{X}_{\text{in}} = \{\mathbf{x}_j^{\text{in}} \in \mathbb{R}^3\}_{j=1}^{n_{\text{in}}}$, input normals $N_{\text{in}} = \{\mathbf{n}_j^{\text{in}} \in \mathbb{R}^3\}_{j=1}^{n_{\text{in}}}$, and a voxel size W , we first predict a hierarchy of L voxel grids using a convolutional backbone digesting the point cloud with concatenated normal $[\mathbf{x}_j^{\text{in}}; \mathbf{n}_j^{\text{in}}] \in \mathbb{R}^6$ for each point. Each of the predicted voxel grid has $n^{(1)}, \dots, n^{(L)}$ voxels with widths $W, 2W, \dots, 2^L W$ respectively and any voxel with center $\mathbf{x}_i^{(l-1)}$ at level $l-1$ is contained within some voxel with center $\mathbf{x}_j^{(l)}$ at level l . The design of such a backbone network is inspired by [64] and is described in detail in the Appendix. We additionally predict features $\mathbf{z}_i^{(l)} \in \mathbb{R}^d$ and normals $\mathbf{n}_i^{(l)} \in \mathbb{R}^3$ for each voxel in the hierarchy. The features $\mathbf{z}_i^{(l)}$ are employed to predict a feature field $\phi_\theta^{(l)}(\mathbf{x} | \mathbf{X}_{\text{in}}, N_{\text{in}})$ which lifts the coordinates $\mathbf{x} \in \mathbb{R}^3$ to d -dimensional vectors via Bézier interpolation followed by an MLP. Fig. 4 shows a 2D illustration of our predicted hierarchy and features.

Sparse Neural Kernel Field Hierarchy. We encode our reconstructed shape as the zero level set of a 3D implicit

field $f_\theta : \mathbb{R}^3 \rightarrow \mathbb{R}$ defined as a hierarchical *Neural Kernel Field*, i.e., a weighted combination of *positive definite kernels* which are conditioned on the inputs and centered at the midpoints $\mathbf{x}_i^{(l)} \in \mathbb{R}^3$ of voxels in the predicted hierarchy:

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

Here, $\alpha_i^{(l)} \in \mathbb{R}$ 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

$$K_\theta^{(l)}(\mathbf{x}, \mathbf{x}') = \langle \phi_\theta^{(l)}(\mathbf{x}; \mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}), \phi_\theta^{(l)}(\mathbf{x}'; \mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}) \rangle \cdot K_b^{(l)}(\mathbf{x}, \mathbf{x}'), \quad (2)$$

where $\langle \cdot, \cdot \rangle$ is the dot product, $\phi_\theta^{(l)} : \mathbb{R}^3 \rightarrow \mathbb{R}^d$ is the feature field extracted from the hierarchy, and $K_b^{(l)} : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ is the *Bézier Kernel*, which decays to zero in a one-voxel (at level- l) neighborhood around its origin (See Appendix for definition).

Computing a 3D implicit surface from points. Given our predicted voxel hierarchy, learned kernels $K_\theta^{(l)}$, and predicted normals $\mathbf{n}_j^{(l)}$, we compute an implicit surface by finding optimal coefficients $\alpha^* = \{\{\alpha_i^{(l)}\}_{l=1}^L\}_{i=1}^{n^{(l)}}$ for the kernel field (1). We find these coefficients by exactly minimizing the following loss in the forward pass of our model (omitting the conditioning of f_θ on $\mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}$ for brevity):

$$\alpha^* = \underset{\alpha_i^{(l)}}{\operatorname{argmin}} \sum_{l=1}^{L'} \sum_{i=1}^{n^{(l)}} \|\nabla_{\mathbf{x}} f_\theta(\mathbf{x}_i^{(l)}) - \mathbf{n}_i^{(l)}\|_2^2 + \sum_{j=1}^{n_{\text{in}}} |f_\theta(\mathbf{x}_j^{\text{in}})|^2, \quad (3)$$

where $L' \leq L$ is a hyper-parameter for the hierarchy. By minimizing this loss, we want our Neural Kernel Field f_θ to have a gradient which agrees with the normals at the voxel centers (hence regions around the surface), and to have a scalar value near zero at all the input points \mathbf{X}_{in} . Since f_θ is linear in the parameters $\alpha_i^{(l)}$, we find the optimal coefficients α^* by solving the linear system:

$$(\mathbf{Q}^\top \mathbf{Q} + \mathbf{G}^\top \mathbf{G})\alpha = \mathbf{Q}^\top \mathbf{n}, \quad (4)$$

where \mathbf{n} are the predicted normal vectors $\mathbf{n}_i^{(l)}$ stacked into a single vector, α is the vector of coefficients $\alpha_i^{(l)}$, and

$$\mathbf{G} = [\mathbf{G}^{(1)} \quad \dots \quad \mathbf{G}^{(L)}], \quad (5)$$

$$\mathbf{Q} = [\mathbf{Q}^{(1)} \quad \dots \quad \mathbf{Q}^{(L)}],$$

$$\mathbf{G}_{i,j}^{(l)} = K_\theta(\mathbf{x}_i^{\text{in}}, \mathbf{x}_j^{(l)}), \quad \mathbf{Q}_{i,j}^{(l)} = \partial_{\mathbf{x}_i^{(l)}} K_\theta(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}) \quad (6)$$

are the gram matrix and partial derivatives of the gram matrix at the voxel centers where normals are defined, respectively.

We remark that the linear system (4) is sparse due to modulation with the compactly supported $K_b^{(l)}$, and positive definite by construction since it is a Gram matrix. As a result, (4) can be solved very efficiently on a GPU.

Masking module. The predicted Neural Kernel Field f_θ is defined on the entire voxel hierarchy, however at coarse levels far from the surface, it may contain unwanted geometry. To discard such geometry away from the predicted surface, we add an additional branch to our backbone as $\varphi : \mathbb{R}^3 \rightarrow \{0, 1\}$ which determines if a point \mathbf{x} should be trimmed ($\varphi(\mathbf{x}) = 0$) or kept ($\varphi(\mathbf{x}) = 1$). The branch originates from the immediate features of the backbone network and consists of a few linear layers with ReLU activations followed by a sigmoid. When we extract the final surface, we only consider vertices in regions where $\varphi(\mathbf{x}) > 0.5$.

3.2. Supervision

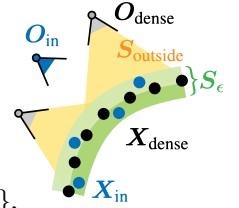
To train our model, we require pairs $(\mathbf{X}_{\text{in}} = \{\mathbf{x}_i \in \mathbb{R}^3\}, \mathbf{O}_{\text{in}} = \{\mathbf{o}_i \in \mathbb{R}^3\})$ and $(\mathbf{X}_{\text{dense}} = \{\mathbf{x}_j\}, \mathbf{O}_{\text{dense}} = \{\mathbf{o}_j\})$. Here \mathbf{X}_{in} and $\mathbf{X}_{\text{dense}}$ are noisy input points and dense supervision points respectively, and \mathbf{O}_{in} and $\mathbf{O}_{\text{dense}}$ are sensor origin for each input and supervision point (i.e. a position in 3D space from which each point was acquired). We additionally compute input and supervision normals $\mathbf{N}_{\text{in}}, \mathbf{N}_{\text{dense}}$ by fitting planes to points in a local neighborhood and orienting the normals to align with the directions from points to sensors. We remark that our training requirements impose no restrictions on the shapes being trained on. For example, one could use a single LiDAR frame as input and an accumulated LiDAR scan of a scene as supervision, alongside a noisy scan of a synthetic object as input and a dense noiseless scan of the same object as supervision.

In order to define the loss terms used to supervise our model, we first define two regions of space around the dense points $\mathbf{X}_{\text{dense}}$:

- \mathcal{S}_ϵ : points which are ϵ distance or less from $\mathbf{X}_{\text{dense}}$ i.e. $\{\mathbf{x} | \min_{\mathbf{x}_j \in \mathbf{X}_{\text{dense}}} \|\mathbf{x} - \mathbf{x}_j\|_2 < \epsilon\}$,
- $\mathcal{S}_{\text{outside}}$: points which lie within the region enclosing points in $\mathbf{X}_{\text{dense}}$ and their sensor origin in $\mathbf{O}_{\text{dense}}$.

Then, given a predicted Neural Kernel Field $f_\theta(\mathbf{x})$, we backpropagate through the following loss functions:

- $\mathcal{L}_{\text{surf}}(f) = \mathbb{E}_{\mathbf{x} \in \mathbf{X}_{\text{dense}}} [\|f(\mathbf{x})\|_1]$;
- $\mathcal{L}_{\text{tsdf}}(f) = \mathbb{E}_{\mathbf{x} \in \mathcal{S}_\epsilon} [\|f(\mathbf{x}) - \text{tsdf}(\mathbf{x}, \mathbf{X}_{\text{dense}})\|_1]$ where $\text{tsdf}(\mathbf{x}, \mathbf{X}_{\text{dense}})$ is the ground-truth truncated signed distance computed from $\mathbf{X}_{\text{dense}}$ using nearest neighbors;



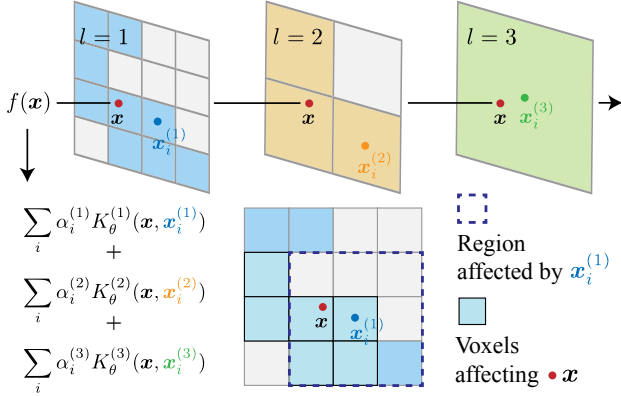



Figure 4: **Our implicit field** $f(\mathbf{x})$ is represented as a sum of kernel basis functions on a sparse voxel hierarchy. Each voxel with center $\mathbf{x}_i^{(l)}$ contributes one kernel basis function $K_\theta^{(i)}(\mathbf{x}, \mathbf{x}_i^{(l)})$ with support in the one-ring around $\mathbf{x}_i^{(l)}$.

- $\mathcal{L}_{\text{normal}}(f) = \mathbb{E}_{\mathbf{n} \in \mathcal{N}_{\text{dense}}} \left[1 - \left\langle \frac{\nabla_{\mathbf{x}} f(\mathbf{x})}{\|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2}, \mathbf{n} \right\rangle \right]$;
- $\mathcal{L}_{\text{outside}}(f) = \mathbb{E}_{\mathbf{x} \in \mathcal{S}_{\text{outside}}} e^{-\beta \|f(\mathbf{x})\|_1}$, where $\beta = 0.1$;
- $\mathcal{L}_{\text{min-surf}}(f) = \mathbb{E}_{\mathbf{x} \in \mathcal{S}_e} \left[\frac{\eta \pi^{-1}}{\eta^2 + f(\mathbf{x})^2} \right]$, where $\eta = 0.5$.

Here $\mathcal{L}_{\text{surf}}$ ensures that the implicit function is zero near the ground truth surface, $\mathcal{L}_{\text{tsdf}}$ ensures that the implicit field undergoes a sign change near the surface, $\mathcal{L}_{\text{normal}}$ ensures the gradient of the predicted implicit agrees with the dense normals, $\mathcal{L}_{\text{outside}}$ ensures there is no geometry far away from the surface, and $\mathcal{L}_{\text{min-surf}}$ acts as a regularizer encouraging the predicted implicit surface to have minimal area [74]. The latter two losses are omitted if no sensor-based information is available.

We additionally compute structure prediction and masking losses which we describe in the Appendix. We train our model in an end-to-end fashion using gradient descent by back-propagating through the sum of all these loss functions.

4. Experiments

Overview. In this section we demonstrate that NKSR fulfills the three main desired properties of practical surface reconstruction method as analyzed in § 1: (1) *Accuracy* (§ 4.1), by training and testing on object-level datasets [6, 36, 77] with varying noise settings. (2) *Scalability* (§ 4.2), by evaluating on large-scale outdoor driving dataset [17]. (3) *Generalizability* (§ 4.3), where we train on object-level/outdoor datasets and test on room-level datasets [12, 5] as well as scans with very low densities. Notably, to encourage the practical usage of NKSR, we present a *kitchen-sink-model* (denoted as ‘Ours - ’) trained on the union of various datasets [6, 36, 17, 5] and report its performance whenever

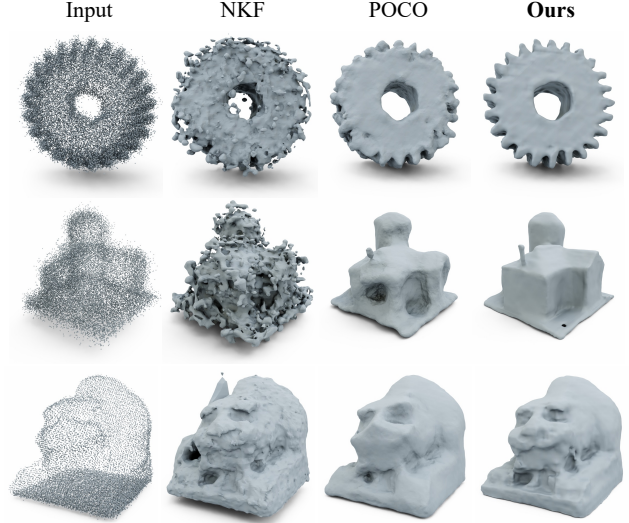


Figure 5: **ABC/Thing10K [36, 77] visualization.**

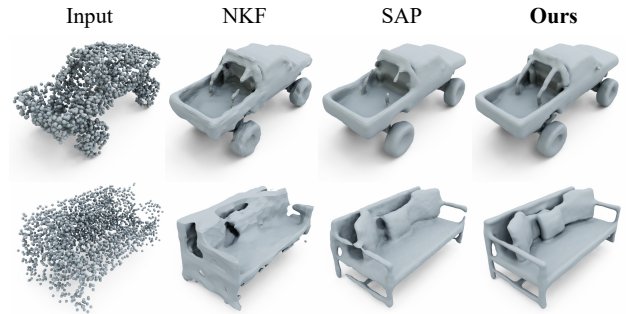


Figure 6: **ShapeNet [6] visualization.** The two shapes are with $\sigma = 0.005$ and $\sigma = 0.025$ Gaussian noise respectively.

applicable. While this model slightly underperforms domain-specific models, it still outperforms most baseline methods and can be used on a wide variety of inputs as shown in Fig. 1 and Fig. 9. We hope the kitchen-sink-model enables end-users to use NKSR in a plug-and-play manner.

Implementation Details. Our pipeline is fully accelerated using PyTorch and CUDA. The operations on our sparse hierarchy including convolution, neighborhood querying and interpolations are based on a customized tree structure that is highly efficient and scalable. Our sparse linear solver uses a Jacobi-preconditioned conjugate gradient method and works jointly with the sparse hierarchy for fast inference. Unless otherwise specified, our experiments are run on a single V100 GPU with 8 CPU cores. Hyperparameter details are given in the Appendix.

4.1. Accuracy: Object-level Reconstruction

Settings. We follow two common evaluation settings from the literature. One is the manifold ShapeNet [6] dataset prepared by [41]. The dataset contains man-made geome-

Table 1: **ABC/Thing10K [36, 77] comparison.** d_C is multiplied by 10^3 . σ is the Gaussian noise added to the sensor depth and L is the largest box length. 10 scans are used to accumulate the point cloud unless specified.


	ABC (100 shapes)						Thing10K (100 shapes)									
	$\sigma = 0$		$\sigma \in [0, 0.05L]$		$\sigma = 0.05L$		$\sigma = 0$		$\sigma = 0.01L$		$\sigma = 0.05L$		5 scans		30 scans	
	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow	$d_C \downarrow$	F-S. \uparrow
SPSR [35]	7.02	87.5	11.2	72.8	18.8	47.9	4.23	91.9	5.44	90.3	16.5	52.5	12.4	77.6	3.07	96.7
POCO [4]	5.34	88.3	8.23	75.7	12.0	58.9	4.42	92.5	5.10	89.7	11.2	58.8	6.95	84.4	3.69	95.0
SAP [46]	6.83	85.0	8.00	79.5	10.4	68.7	4.30	92.7	4.54	91.8	7.82	74.7	6.73	84.7	3.95	93.8
NKF [65]	6.10	88.1	13.8	62.3	24.0	35.1	3.48	94.2	4.78	90.8	24.7	34.3	7.05	84.8	4.36	93.2
NGSolver [33]	3.92	92.7	6.35	83.1	9.68	66.4	2.96	95.9	3.51	95.0	8.70	69.1	5.65	89.2	2.80	97.1
Ours	3.68	93.2	6.00	85.4	8.70	73.2	2.36	97.3	3.19	95.9	7.66	74.7	5.10	89.9	2.48	98.0
Ours - 	4.10	92.2	6.44	83.6	9.97	68.1	2.92	96.3	3.34	95.6	8.55	72.7	5.60	89.1	2.54	97.7


Table 2: **ShapeNet [6] comparison.** ‘N.’ denotes whether normals N_{in} are used as input. d_C is multiplied by 10^3 .

	N.	1000 Pts. $\sigma = 0.0$		3000 Pts. $\sigma = 0.005$		3000 Pts. $\sigma = 0.025$	
		$d_C \downarrow$	IoU \uparrow	$d_C \downarrow$	IoU \uparrow	$d_C \downarrow$	IoU \uparrow
ConvONet [47]		6.07	82.3	4.35	88.0	7.31	78.7
IMLSNet [39]		3.15	91.2	3.03	91.3	6.58	76.0
SAP [46]	-	3.44	90.8	3.30	91.1	5.34	82.9
POCO [4]		3.03	92.7	2.93	92.2	5.82	81.7
NGSolver [33]		2.91	91.9	2.90	91.8	5.06	82.8
Ours		2.64	93.4	2.71	92.6	4.96	82.9
SPSR [35]		6.26	81.4	3.84	88.5	10.7	66.8
SAP [46]		3.21	92.1	3.16	92.3	4.44	87.1
NKF [65]	\checkmark	2.65	94.7	3.17	91.2	11.7	67.0
NGSolver [33]		2.47	95.0	2.51	94.1	3.93	87.5
Ours		2.34	95.6	2.45	94.3	3.87	87.6

tries from 13 categories, with $>30K$ shapes for training and $>8K$ shapes for testing. Gaussian noise of different standard deviations (denoted as σ) is added to the randomly-subsampled points from the ground truth as input. As many existing learning-based baselines do not need point normals N_{in} as input, we present a variant of our model that does not take N_{in} as extra input channels for a fair comparison. The other setting is from [18] where a random subset of $\sim 5K$ shapes from ABC [36] is picked for training and testing, and an additional 100 shapes from Thing10K [77] is used for testing generalization. The input is acquired by simulating ToF sensors with different levels of noise and densities. For the metrics we use the standard Chamfer distance (d_C), F-score (F-S.), normal consistency (N.C.), and intersection-over-union ratio (IoU) as benchmarks.

Results. The comparisons are quantitatively shown in Tab. 1 and Tab. 2, and selectively visualized in Fig. 5 and Fig. 6. Our model reaches state-of-the-art performance on all the datasets. Our baseline, NKF, works well on the noise-free setting but inelegantly degrades with higher noise due to its over-reliance on the raw input normals. On the other hand, SAP and NGSolver are more robust under noise, but the

Table 3: **CARLA [17] comparison.** d_C is the average of Acc. and Comp. (Unit is cm. The smaller the better.)

	Original			Novel			Time (sec.)
	Acc.	Comp.	F-S. \uparrow	Acc.	Comp.	F-S. \uparrow	
TSDF-Fusion [62]	8.2	8.0	80.2	8.6	6.6	80.7	0.5
POCO [4]	10.5	3.6	90.1	9.1	2.9	92.4	420
SPSR [35]	10.3	16.4	86.5	9.9	12.8	88.3	30
Ours	5.6	2.2	93.9	3.6	2.1	96.0	2.6
Ours - 	4.1	3.0	94.0	3.6	2.4	96.0	2.6

fitting tightness as reflected by d_C is higher than ours due to the lack of representation power. Our performance gain is mainly based on the gradient-based energy fitting formulation backed up by the natural inductive biases emerging from the learned kernel.

4.2. Scalability: Outdoor Driving Scenes

Dataset. The applicability of NKSR to large-scale datasets is demonstrated using the synthetic CARLA [17] dataset due to the lack of large-scale real-world datasets with ground-truth geometries. To generate such a dataset, we manually pick 3 towns and simulate 10 random drives using the engine. We call these drives the ‘Original’ subset. An additional town along with its 3 drives is used only during evaluation to test generalization, which we denote as the ‘Novel’ subset. For (X_{in}, O_{in}) , we use a sparse 32-beam LiDAR with 0-5cm ray distance noise and 0-3° pose noise. For (X_{dense}, O_{dense}) , we employ a noise-free highly-dense 256-beam LiDAR for ground-truth supervision. The accumulated LiDAR points are cropped into $51.2 \times 51.2m^2$ chunks for the ease of benchmarking. Please find more details and visualizations in the Appendix.

Results. We compare our results to TSDF-Fusion, SPSR and the learning-based POCO. While for the latter two baselines we use the same voxel sizes $W = 10cm$ as ours, for TSDF-Fusion we find it necessary to increase W to 30cm to reach decent surface completeness. The results are shown in Tab. 3 and visualized in Fig. 7. We compute single-sided

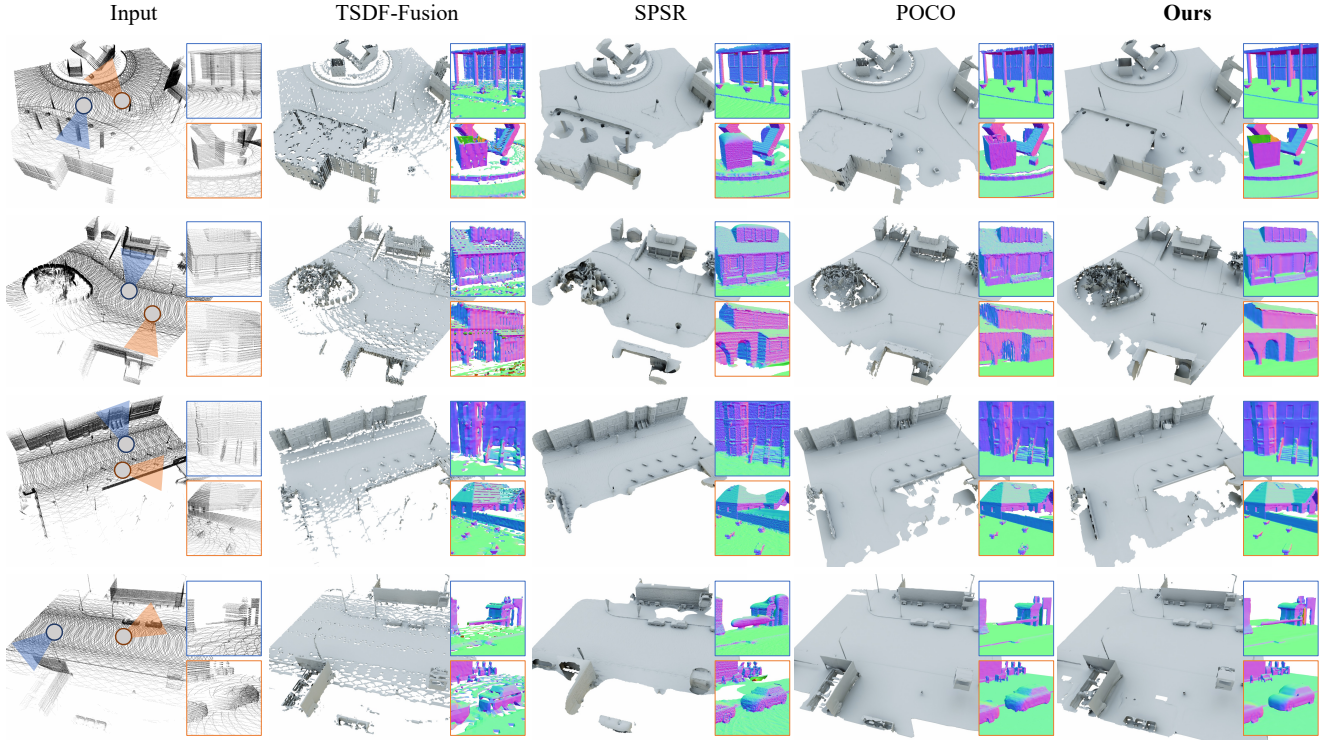


Figure 7: **CARLA [17] visualization.** The insets show zoom-ins captured by the cameras shown in the leftmost column with the corresponding color. The upper 2 rows are from the ‘Novel’ subset and the lower 2 rows are from the ‘Original’ subset.

Table 4: **Room-level dataset [12, 5] comparison.** d_C is multiplied by 10^3 .

	Training Set	ScanNet			Matterport3D		
		$d_C \downarrow$	F-S. \uparrow	N.C. \uparrow	$d_C \downarrow$	F-S. \uparrow	N.C. \uparrow
SPSR [35]	-	7.04	84.3	87.2	10.4	87.0	92.3
LIG [34]	Shape Net	6.19	83.8	83.7	5.13	90.1	90.1
POCO [4]		6.21	77.4	87.1	5.14	84.9	93.7
NKF [65]		6.50	80.9	84.2	6.48	84.2	90.4
DOGNN [63]		4.93	85.9	85.7	4.85	89.3	92.4
Ours		2.68	97.7	90.5	3.19	96.8	95.2
POCO [4]	Synth. Rooms	5.96	82.5	82.0	6.52	80.3	85.9
NKF [65]		9.15	66.5	83.4	9.87	69.3	86.2
Ours		5.38	86.6	86.4	5.01	90.5	91.8
Ours	CARLA	3.20	95.9	89.1	3.08	98.1	95.0
Ours -	Mixed	3.72	93.6	89.1	3.17	97.4	95.5

Chamfer distance that reflects reconstruction accuracy (Acc.) and completeness (Comp.). We additionally report average running times for each method on the datasets. The mean/min/max number of input points in this setting are 490k/290k/820k. Compared to ours, SPSR is quite sensitive to the noise and sparsity in the input, leaving bumpy and incomplete geometries. Although POCO could reach a similar completeness value, the fitted surfaces fail to faithfully respect the input. The long running time (161x slower than ours) also prohibits POCO from practical use.

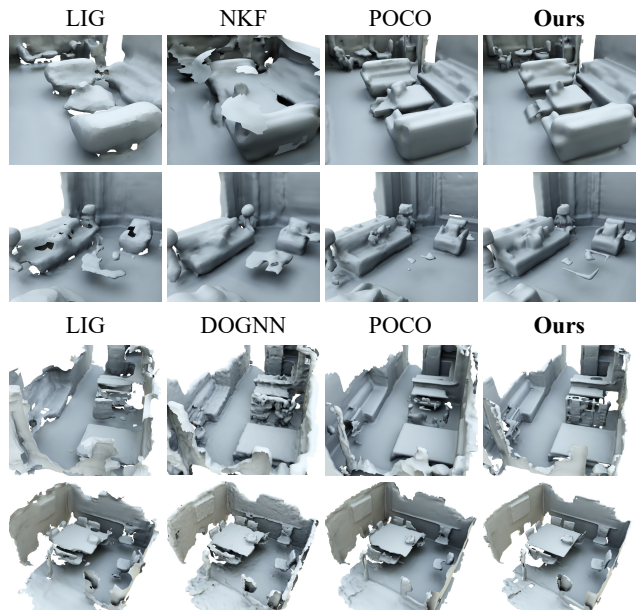


Figure 8: **Room-level datasets [5, 12] visualization.** All the models are trained only with ShapeNet.

4.3. Generalization across Domains and Densities

Across domains. We compare the generalizability of our method with others by directly applying the models trained on ShapeNet and Synthetic Room dataset (Synth. Rooms) [47] to room-level datasets, *i.e.*, ScanNet [12] and

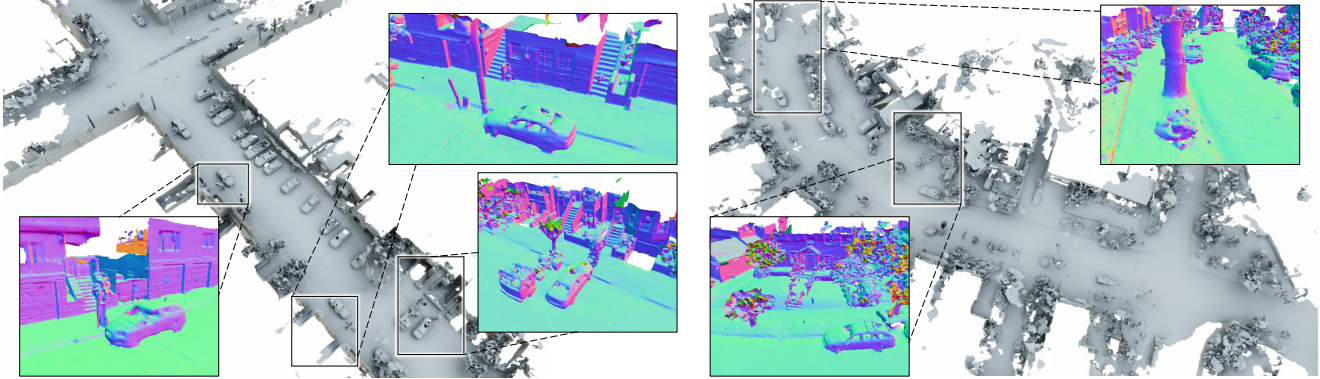


Figure 9: **Application to Waymo [55] dataset.** We run our kitchen-sink-model in an out-of-core manner (see Appendix for implementation details) to scale to very large scenes consisting of 10M / 11M (left / right) points, taking only 20s / 35s.

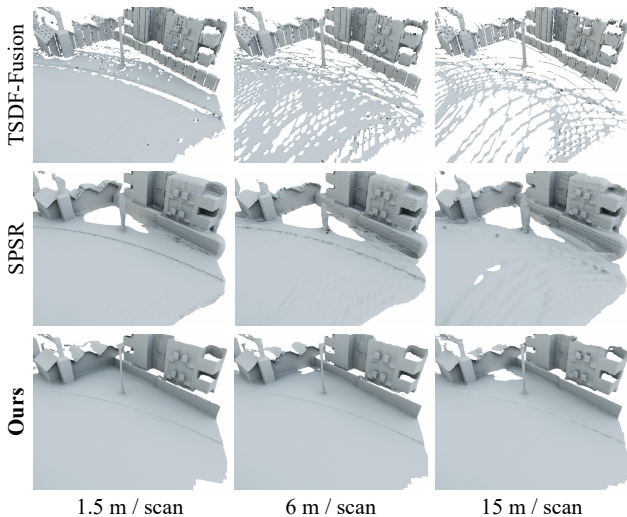


Figure 10: **Generalization to different densities.**

the test split of Matterport3D [5]. For both datasets we sample 10K points as input, and normalize the scale to roughly match the training set. As shown by the comparisons in Fig. 8 and Tab. 4, the generalization of our method is significantly better than the baselines, with ShapeNet training set reaching the highest accuracy possibly due to its diversity and similar geometric distributions.

Across densities. We test the robustness of our model under sparse input by keeping only one scan of LiDAR frame within a fixed driving distance in our CARLA dataset (‘Novel’ subset). The results are shown in Fig. 10 and Tab. 5. At the level of extreme sparsity our method is still able to reconstruct complete geometry (*e.g.* the ground) while the baselines start to degrade.

4.4. Ablation Study

We run our method with different feature dimensions d for kernel computation, as well as different voxel sizes W , and the results are shown in Fig. 11. While increasing the

Table 5: **Performance comparison using different input densities.** Here the F-Score \uparrow metric is shown.

Meters / scan	1.5	3	6	9	12	15
TSDF-Fusion [35]	80.0	80.7	78.4	74.8	70.6	67.8
SPSR [35]	88.2	88.3	87.9	86.4	83.3	79.5
Ours	96.1	96.0	95.4	94.1	92.6	92.0

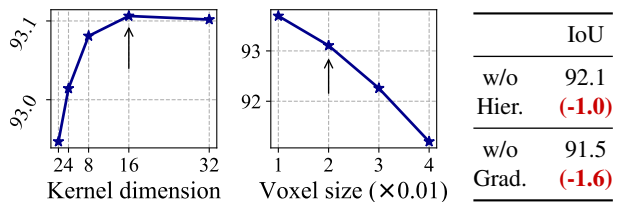


Figure 11: **Ablation study.** IoU metric is shown. The back arrows indicate the setting we use to obtain Tab. 2.

feature dimension helps reach a slightly better performance, the influence of voxel sizes is more prominent. We try to remove the hierarchies from the linear solver by setting $\{\alpha_i^{(l)} \mid l > 0\}$ to 0 (‘w/o Hier.’), or remove the gradient-based matrices $\mathbf{Q}^\top \mathbf{Q}$ (‘w/o Grad.’). Both of the settings lead to a degraded performance, showing the effectiveness of our design choices.

5. Conclusion

In this paper we present NKSR, an accurate and scalable surface reconstruction algorithm using the neural kernel field representation. We show by extensive experiments that our method reaches state-of-the-art quality and efficiency, while enjoying good generalization to unseen data. We believe our method further pushes the boundary of the field of 3D reconstruction and makes deep-learning-based surface reconstruction more practical for general use. For future work we will try further improving the reconstruction quality using more expressive kernel models, as well as reducing memory footprint to allow for even larger-scale reconstructions.

References

- [1] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950. [14](#)
- [2] Matan Atzmon, Koki Nagano, Sanja Fidler, Sameh Khamis, and Yaron Lipman. Frame averaging for equivariant shape space learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 631–641, 2022. [3](#)
- [3] Abhishek Badki, Orazio Gallo, Jan Kautz, and Pradeep Sen. Meshlet priors for 3d mesh reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2849–2858, 2020. [3](#)
- [4] 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. [1](#), [2](#), [6](#), [7](#), [16](#)
- [5] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 667–676, 2017. [1](#), [5](#), [7](#), [8](#)
- [6] 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. [1](#), [2](#), [5](#), [6](#), [17](#)
- [7] Hao-Xiang Chen, Jiahui Huang, Tai-Jiang Mu, and Shi-Min Hu. Circle: Convolutional implicit reconstruction and completion for large-scale indoor scene. In *European Conference on Computer Vision (ECCV)*, pages 506–522. Springer, 2022. [2](#)
- [8] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 45–54, 2020. [3](#)
- [9] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision (ECCV)*, pages 628–644. Springer, 2016. [3](#)
- [10] Lei Chu, Hao Pan, and Wenping Wang. Unsupervised shape completion via deep prior in the neural tangent kernel perspective. *ACM Transactions on Graphics (TOG)*, 40(3):32:1–32:17, 2021. [2](#)
- [11] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996. [2](#)
- [12] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2432–2443, 2017. [1](#), [5](#), [7](#)
- [13] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 31–44, 2020. [3](#)
- [14] Boyang Deng, John P Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa neural articulated shape approximation. In *European Conference on Computer Vision (ECCV)*, pages 612–628. Springer, 2020. [3](#)
- [15] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulernard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 12200–12209, 2021. [3](#)
- [16] Theo Deprelle, Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Learning elementary structures for 3d shape generation and matching. *arXiv preprint arXiv:1908.04725*, 2019. [3](#)
- [17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. [1](#), [5](#), [6](#), [7](#)
- [18] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J Mitra, and Michael Wimmer. Points2surf learning implicit surfaces from point clouds. In *European Conference on Computer Vision (ECCV)*, pages 108–124. Springer, 2020. [6](#)
- [19] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613, 2017. [3](#)
- [20] Matheus Gadelha, Rui Wang, and Subhransu Maji. Deep manifold prior. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1107–1116, 2021. [3](#)
- [21] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *arXiv preprint arXiv:2011.01437*, 2020. [3](#)
- [22] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4857–4866, 2020. [2](#)
- [23] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *European Conference on Computer Vision (ECCV)*, pages 484–499. Springer, 2016. [3](#)
- [24] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 9785–9795, 2019. [3](#)
- [25] Benjamin Graham and Laurens van der Maaten. Sub-manifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. [12](#)
- [26] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 216–224, 2018. [3](#)

- [27] Vitor Guizilini and Fabio Ramos. Learning to reconstruct 3d structures for occupancy mapping from depth and color information. *The International Journal of Robotics Research*, 37(13-14):1595–1609, 2018. **2**
- [28] Oshri Halimi, Ido Imanuel, Or Litany, Giovanni Trappolini, Emanuele Rodolà, Leonidas Guibas, and Ron Kimmel. Towards precise completion of deformable shapes. In *European Conference on Computer Vision (ECCV)*, pages 359–377. Springer, 2020. **3**
- [29] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 412–420. IEEE, 2017. **3**
- [30] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90:1–90:12, 2019. **3**
- [31] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084*, 2020. **3**
- [32] Geoffrey E Hinton and Russ R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. **2**
- [33] Jiahui Huang, Hao-Xiang Chen, and Shi-Min Hu. A neural galerkin solver for accurate surface reconstruction. *ACM Transactions on Graphics (TOG)*, 41(6):229:1–229:16, 2022. **2, 6, 16**
- [34] 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. **2, 7, 16**
- [35] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29:1–29:13, 2013. **1, 2, 6, 7, 8, 15**
- [36] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9601–9611, June 2019. **1, 5, 6**
- [37] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018. **3**
- [38] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1886–1895, June 2018. **3**
- [39] 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. **6, 16**
- [40] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2837–2845, 2021. **3**
- [41] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470, 2019. **2, 3, 5**
- [42] 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. **17**
- [43] Junyi Pan, Xiaoguang Han, Weikai Chen, Jiapeng Tang, and Kui Jia. Deep mesh reconstruction from single rgb images via topology modification networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 9964–9973, 2019. **1**
- [44] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. **2**
- [45] Massimiliano Patacchiola, Jack Turner, Elliot J Crowley, Michael O’Boyle, and Amos Storkey. Bayesian meta-learning for the few-shot setting via deep kernels. *arXiv preprint arXiv:1910.05199*, 2019. **2**
- [46] 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. **1, 2, 6, 16**
- [47] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, pages 523–540, 2020. **2, 3, 6, 7, 16**
- [48] 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. **3, 12**
- [49] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. **3**
- [50] Davis Rempe, Tolga Birdal, Yongheng Zhao, Zan Gojcic, Srinath Sridhar, and Leonidas J Guibas. Caspr: Learning canonical spatiotemporal point cloud representations. *arXiv preprint arXiv:2008.02792*, 2020. **3**
- [51] 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. **15**
- [52] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004. **14**
- [53] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in*

- Neural Information Processing Systems*, pages 6087–6101, 2021. [2](#), [3](#)
- [54] Vincent Sitzmann, Eric R Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. *arXiv preprint arXiv:2006.09662*, 2020. [2](#)
- [55] Pei Sun, Henrik Kretschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, June 2020. [1](#), [8](#)
- [56] Weiwei Sun, Andrea Tagliasacchi, Boyang Deng, Sara Sabour, Soroosh Yazdani, Geoffrey Hinton, and Kwang Moo Yi. Canonical capsules: Unsupervised capsules in canonical pose. *arXiv preprint arXiv:2012.04718*, 2020. [3](#)
- [57] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11358–11367, 2021. [2](#), [3](#)
- [58] 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. [15](#)
- [59] Shubham Tulsiani, Or Litany, Charles R Qi, He Wang, and Leonidas J Guibas. Object-centric multi-view aggregation. *arXiv preprint arXiv:2007.10300*, 2020. [3](#)
- [60] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2626–2634, 2017. [3](#)
- [61] Benjamin Ummerhofer and Vladlen Koltun. Adaptive surface reconstruction with multiscale convolutional kernels. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5651–5660, October 2021. [2](#)
- [62] Ignacio Vizzo, Tiziano Guadagnino, Jens Behley, and Cyrill Stachniss. Vdbfusion: Flexible and efficient tsdf integration of range sensor data. *Sensors*, 22(3), 2022. [6](#), [16](#)
- [63] Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (TOG)*, 41(4):103:1–103:15, 2022. [2](#), [7](#)
- [64] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(6):217:1–217:11, 2018. [3](#)
- [65] 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. [1](#), [2](#), [6](#), [7](#), [16](#)
- [66] Francis Williams, Jerome Parent-Levesque, Derek Nowrouzezahrai, Daniele Panozzo, Kwang Moo Yi, and Andrea Tagliasacchi. Voronoinet: General functional approximators with local support. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1069–1073, June 2020. [3](#)
- [67] Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10130–10139. IEEE, Jun 2019. [3](#)
- [68] Francis Williams, Matthew Trager, Joan Bruna, and Denis Zorin. Neural splines: Fitting 3d surfaces with infinitely-wide neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9949–9958, 2021. [1](#), [2](#), [3](#)
- [69] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016. [2](#)
- [70] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. MarrNet: 3D Shape Reconstruction via 2.5D Sketches. In *Advances in Neural Information Processing Systems*, pages 540–550, 2017. [3](#)
- [71] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Ec-net: an edge-aware point set consolidation network. In *European Conference on Computer Vision (ECCV)*, pages 386–402, 2018. [3](#)
- [72] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2799, 2018. [3](#)
- [73] Ge Zhang, Or Litany, Srinath Sridhar, and Leonidas Guibas. Strobenet: Category-level multiview reconstruction of articulated objects. *arXiv preprint arXiv:2105.08016*, 2021. [3](#)
- [74] Jingyang Zhang, Yao Yao, Shiwei Li, Tian Fang, David McInnon, Yanghai Tsin, and Long Quan. Critical regularizations for neural surface reconstruction in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6270–6279, 2022. [5](#)
- [75] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1009–1018, 2019. [3](#)
- [76] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. *arXiv preprint arXiv:2104.03670*, 2021. [3](#)
- [77] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016. [5](#), [6](#)

Appendix

In this appendix, we first provide more details of our method, including necessary network designs and derivations in Appendix A. Details related to experiments, including hyperparameters, metrics, and baselines are documented in Appendix B. The design of our pipeline allows for different extensions applicable to various scenarios, and these extensions are in Appendix C. Finally, more visualizations of our results are shown in Appendix 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 \mathbf{X}_{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[48]-like encoder network to pool all the points within it into a feature vector. The network is illustrated in Fig. 12. To allow for translational equivariance, we convert from the global coordinates of the input points to local coordinates within the voxels as input $\tilde{\mathbf{X}}_{in}$. For most of the datasets demonstrated in the main paper, per-point normal \mathbf{N}_{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 \mathbf{O}_{in} . $\tilde{\mathbf{X}}_{in}$ and \mathbf{N}_{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 [25] sandwiched by max pooling layers to coarsen the voxels, as shown in the upper part of Fig. 13. 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. 13. 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:

- 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 $\mathbf{n}_i^{(l)}$ that is 3-dimensional and used later in the linear system. Note that there is no direct supervision to this

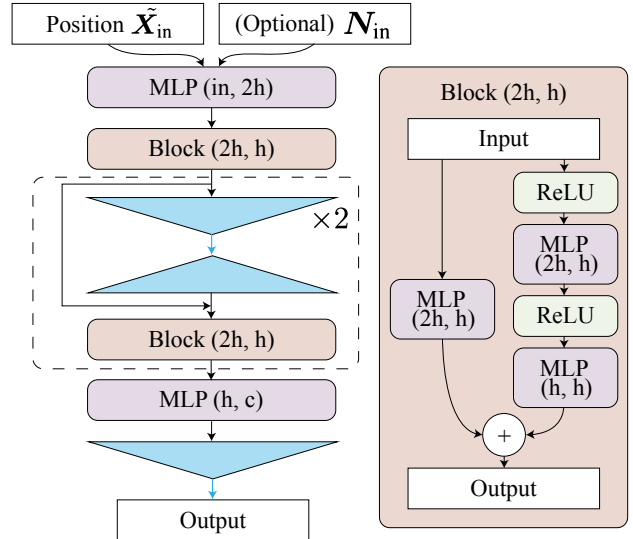


Figure 12: **Architecture for the point encoder.** Rounded rectangles show operations over each point, where h is the hidden dimension. Blue triangles denote max pooling and repeating operations.

branch and we find such a strategy provide better results due to the additional degrees of freedom introduced.

- Kernel prediction branch (★) outputs the features $\mathbf{z}_i^{(l)} \in \mathbb{R}^d$ that is defined in the main text. MLP followed by Bézier interpolation are used to obtain the kernel field at arbitrary position.
- Mask prediction branch (♠) outputs 16-dimensional 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-dimensional 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.

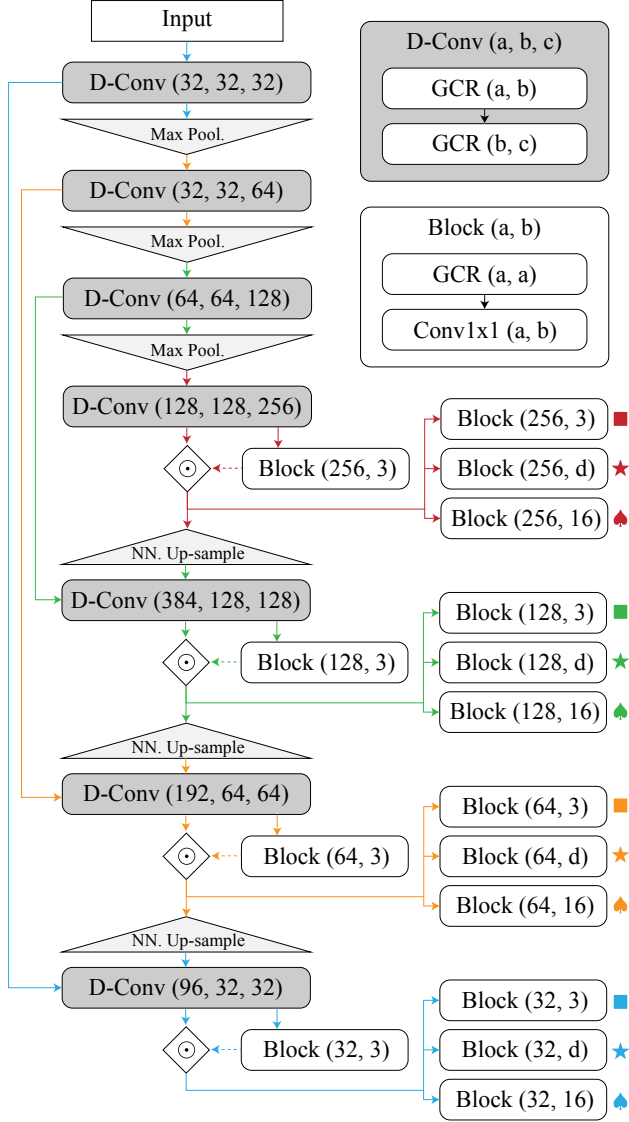


Figure 13: **Architecture for the U-Net.** ‘GCR’ means a sequential application of GroupNorm, Convolution and ReLU activation. ‘ \odot ’ denote voxel masking using the structure prediction results. Different colors of the arrows represent features at different layers in the hierarchy.

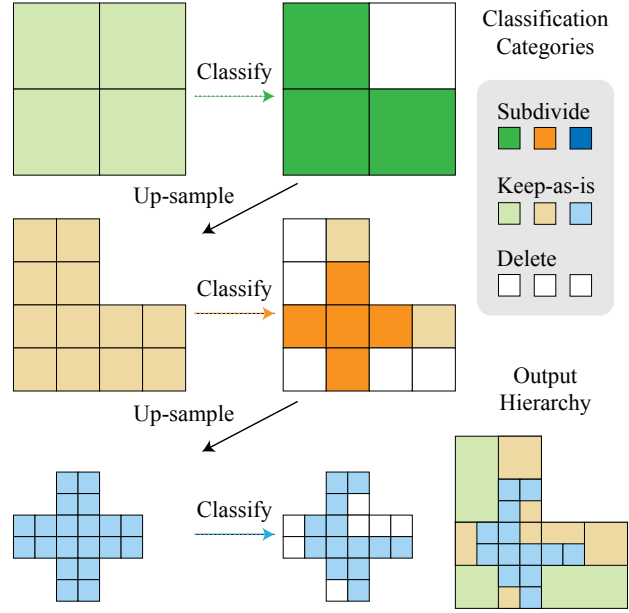


Figure 14: **Structure prediction.** We use a $L = 3$ hierarchy as an example here. According to different classification results, the voxels will be treated differently.

An illustration for the structure prediction is shown in Fig. 14. 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 hierarchical 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 $f_\theta : \mathbb{R}^3 \rightarrow \mathbb{R}$ defined as a weighted combination of *positive definite kernels* which are conditioned on the inputs and centered at the midpoints $\mathbf{x}_i^{(l)} \in \mathbb{R}^3$ of each voxel in the predicted hierarchy:

$$f_\theta(\mathbf{x} | \mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}) = \sum_{i,l} \alpha_i^{(l)} K_\theta^{(l)}(\mathbf{x}, \mathbf{x}_i^{(l)} | \mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}), \quad (7)$$

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

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

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

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

with $\psi^2 : \mathbb{R} \rightarrow \mathbb{R}$ 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 (7) used to construct our hierarchy are positive definite kernels.*

Proof. The kernel $K_{\theta}^{(l)}(\mathbf{x}, \mathbf{x}')$ at each level is defined as the dot product between features $\phi_{\theta}^{(l)}(\mathbf{x})$ and $\phi_{\theta}^{(l)}(\mathbf{x}')$ multiplied by the Bézier Kernel K_b . A kernel, by definition is a dot product of feature embeddings (Definition 2.8 in [52]), and the product of kernels is a kernel (Proposition 3.22 in [52]). Therefore each $K_{\theta}^{(l)}$ is a kernel. \square

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 $\alpha^* = \{\{\alpha_i^{(l)}\}_{l=1}^L\}_{i=1}^{n^{(l)}}$ for the kernel field (7). *i.e.*, given the predicted voxel hierarchy, learned kernels $K_{\theta}^{(l)}$, and predicted normals $\mathbf{n}_j^{(l)}$, we minimize the following loss in the forward pass of our model (omitting the conditioning of f_{θ} on $\mathbf{X}_{\text{in}}, \mathbf{N}_{\text{in}}$ for brevity):

$$\alpha^* = \underset{\alpha_i^{(l)}}{\operatorname{argmin}} \sum_{l=1}^{L'} \sum_{i=1}^{n^{(l)}} \|\nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}_i^{(l)}) - \mathbf{n}_i^{(l)}\|_2^2 + \sum_{j=1}^{n_{\text{in}}} |f_{\theta}(\mathbf{x}_j^{\text{in}})|^2, \quad (8)$$

where $L' \leq L$ is a hyper-parameter for the hierarchy. We can rewrite (8) in matrix form

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

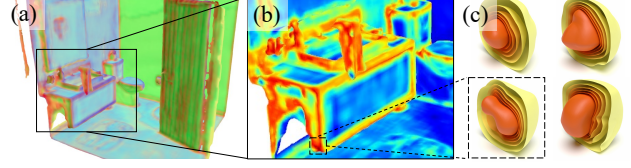


Figure 15: **Kernel visualization.** (a) PCA of the kernel features $\phi_{\theta}^{(l)}$. (b) Heatmap of kernel similarities w.r.t. one selected voxel in the dashed box. (c) Level sets of the kernel basis functions $K_{\theta}^{(l)}(\mathbf{x}, \mathbf{x}_j^{(l)})$.

where

$$\mathbf{G} = [\mathbf{G}^{(1)} \quad \dots \quad \mathbf{G}^{(L)}], \quad \mathbf{Q} = [\mathbf{Q}^{(1)} \quad \dots \quad \mathbf{Q}^{(L)}], \\ \mathbf{n} = [\mathbf{n}_{1,x} \quad \mathbf{n}_{1,y} \quad \mathbf{n}_{1,z} \quad \dots \quad \mathbf{n}_{n_v,x} \quad \mathbf{n}_{n_v,y} \quad \mathbf{n}_{n_v,z}]^{\top}, \\ \alpha = [\alpha_1^{(1)} \quad \dots \quad \alpha_{n^{(1)}}^{(1)} \quad \dots \quad \alpha_1^{(L)} \quad \dots \quad \alpha_{n^{(L)}}^{(L)}]^{\top}.$$

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

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

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

$$\mathbf{Q} = [\mathbf{Q}^{(1)} \quad \dots \quad \mathbf{Q}^{(L)}], \quad \mathbf{Q}^{(l)} = [\mathbf{Q}_x^{(l)} \quad \mathbf{Q}_y^{(l)} \quad \mathbf{Q}_z^{(l)}]$$

with

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

Setting the gradient with respect to α of (8) to $\mathbf{0}$, we find that the optimal α^* is the solution to the linear system:

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

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 $\forall \mathbf{x} \neq \mathbf{0}, \mathbf{x}^{\top} \mathbf{Q}^{\top} \mathbf{Q} \mathbf{x} = \|\mathbf{Q}\mathbf{x}\|_2^2 \geq 0$. Furthermore since \mathbf{Q} is constructed as a concatenation of Gram Matrices, it is full rank and thus $\|\mathbf{Q}\mathbf{x}\|_2^2 > 0$. 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. \square

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. 15, we perform PCA over the kernel features on the reconstructed surface and plot exemplar kernel basis functions $K_{\theta}^{(l)}(\mathbf{x}, \mathbf{x}_j^{(l)})$ 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:

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

where $\mathbf{c}_i^{(l)} \in \mathbb{R}^3$ refers to the output of the structure prediction branch, and $(\mathbf{c}_i^{(l)})_{\text{GT}}$ is its ground-truth counterpart. To compute the ground-truth hierarchy, we apply the approach in OctField [58] to $\mathbf{X}_{\text{dense}}$ and $\mathbf{N}_{\text{dense}}$. Specifically, we start by building a dense hierarchy of the coarsest 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:

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

where $\mathbf{x} \in \mathbf{X}_{\text{dense}}$ and $\text{std.}(\cdot)$ 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 $\varphi(\mathbf{x})$ 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 $\mathbf{X}_{\text{dense}}$ are 1 and 0 otherwise.

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. 16.

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

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

$$\varphi(\mathbf{x}) = \max_k \varphi_k(\mathbf{T}_k^{-1}\mathbf{x}),$$

where the index k refers to the chunks whose regions cover \mathbf{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 [51] over it.

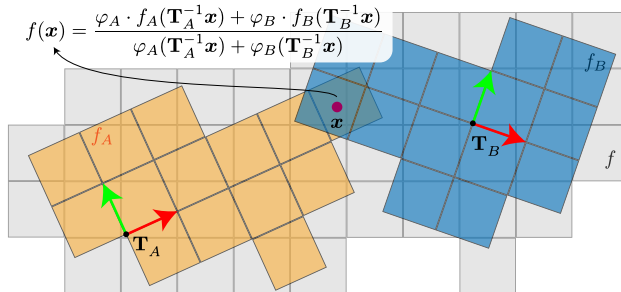


Figure 16: **Merging multiple reconstructions.** We demonstrate the merging operation with two chunks A and B . The final implicit value for point \mathbf{x} is defined as the average of the two chunks, weighted by their predicted masking values.

Table 6: **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

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 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. 6. 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 [35]. 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 [4]. 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 [33]. We use the official implementation from <https://github.com/huangjh-pub/neural-galerkin>, taking the default configurations provided by the repository.

SAP [46] and ConvONet [47]. 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 Appendix A.1.

NKF [65]. 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 [39]. 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 [62]. 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 $\mathbf{x} + \epsilon \mathbf{n}$ and stopping at \mathbf{x} as the input to their algorithm.

LIG [34]. We use the implementation from <https://github.com/huangjh-pub/di-fusion> with a pretrained 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 \mathbf{X}_{gt} and \mathbf{N}_{gt}) and the predicted mesh (denoted as \mathbf{X}_{pd} and \mathbf{N}_{pd}).

Table 7: **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

Chamfer Distance d_C . The Chamfer distance is computed using:

$$d_C = \frac{1}{2}(\text{Comp.} + \text{Acc.}),$$

$$\text{Comp.} = \frac{1}{|\mathbf{X}_{\text{gt}}|} \sum_{\mathbf{x}_{\text{gt}} \in \mathbf{X}_{\text{gt}}} \min_{\mathbf{x}_{\text{pd}} \in \mathbf{X}_{\text{pd}}} \|\mathbf{x}_{\text{gt}} - \mathbf{x}_{\text{pd}}\|,$$

$$\text{Acc.} = \frac{1}{|\mathbf{X}_{\text{pd}}|} \sum_{\mathbf{x}_{\text{pd}} \in \mathbf{X}_{\text{pd}}} \min_{\mathbf{x}_{\text{gt}} \in \mathbf{X}_{\text{gt}}} \|\mathbf{x}_{\text{pd}} - \mathbf{x}_{\text{gt}}\|.$$

Note that this is consistent with the one used in, *e.g.*, ConvONet [47] but different with the one used in POCO¹, hence the difference in the results.

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

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

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

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

where

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

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

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. 7. 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} .



Figure 17: **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 [6] cars as input and iterate 800 times for each shape.

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 [42]. 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:

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

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

$$\mathbf{G}_c^\top \mathbf{G}_c \boldsymbol{\gamma} = \mathbf{G}_c^\top \mathbf{t}, \quad (9)$$

where \mathbf{G}_c is the Gram matrix for the kernel K_ϕ and \mathbf{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 exemplar cars from ShapeNet [6] dataset with 10K colored input points and the results are shown in Fig. 17. We could accurately recover the textures along with the shape, showing a strong representation power for signals other than geometry.

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

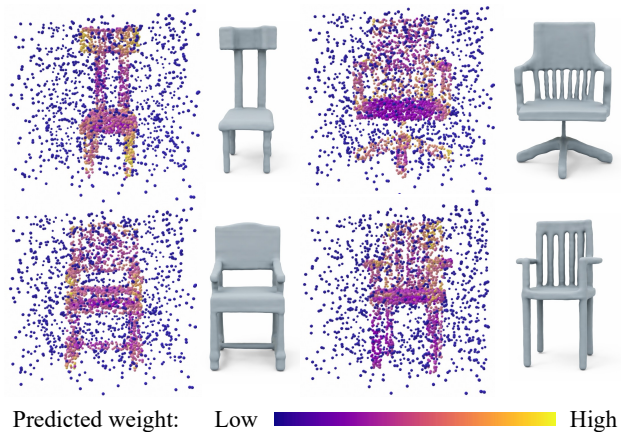


Figure 18: **Outlier detection and removal.** Given input points with extreme outliers (left), the model learns to automatically down-weight irrelevant points and reconstructs good geometry (right).

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 (8) as follows:

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

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.

¹<https://github.com/ErlerPhilipp/points2surf/issues/20>

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

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

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. 18 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. 19, Fig. 20, Fig. 21, Fig. 22, Fig. 23 and Fig. 24.

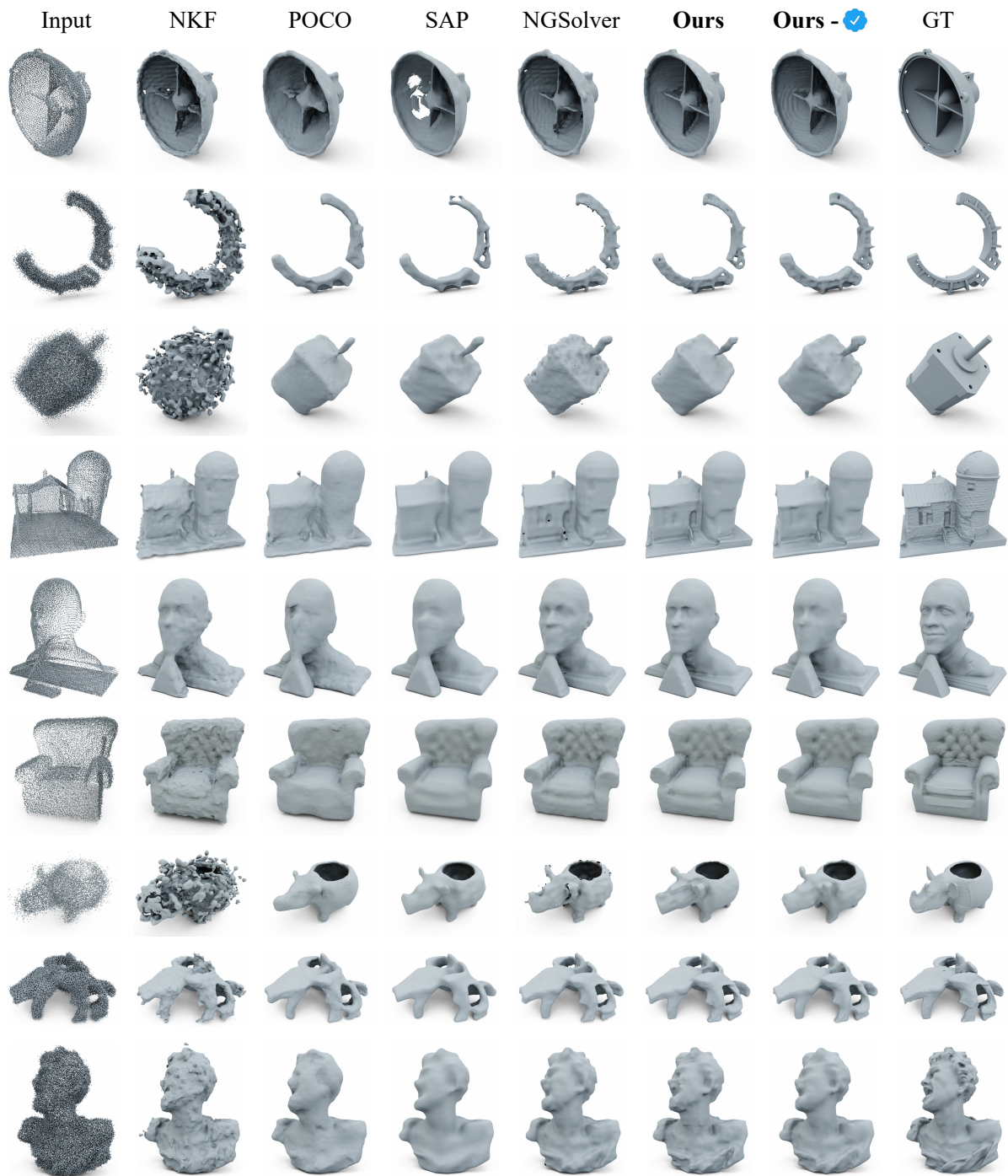


Figure 19: More results on ABC/Thing10K datasets. Best viewed with 2× zoom-in.

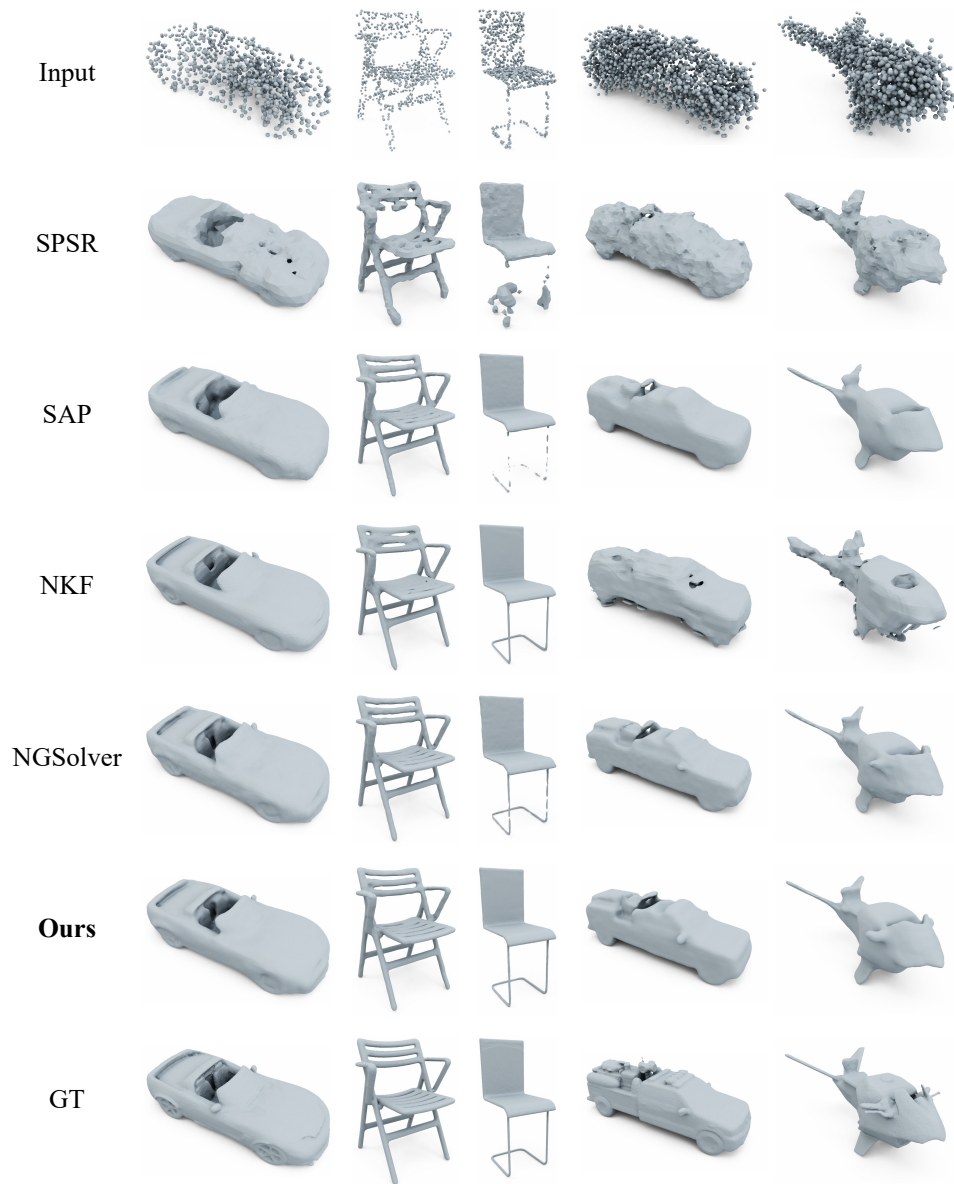


Figure 20: **More results on ShapeNet datasets (with normal).** Best viewed with 2× zoom-in.

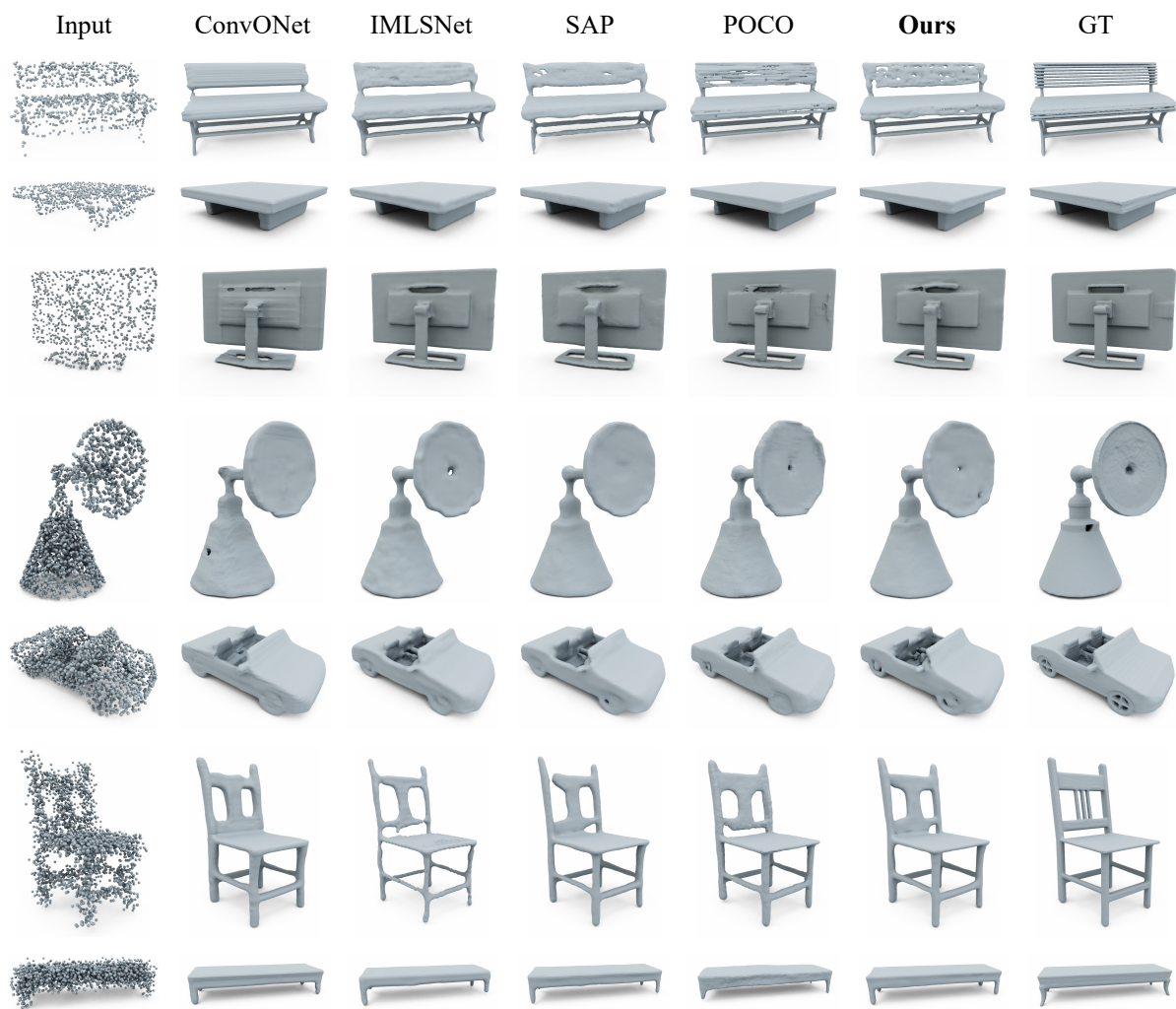


Figure 21: More results on ShapeNet datasets (without normal). Best viewed with $2\times$ zoom-in.

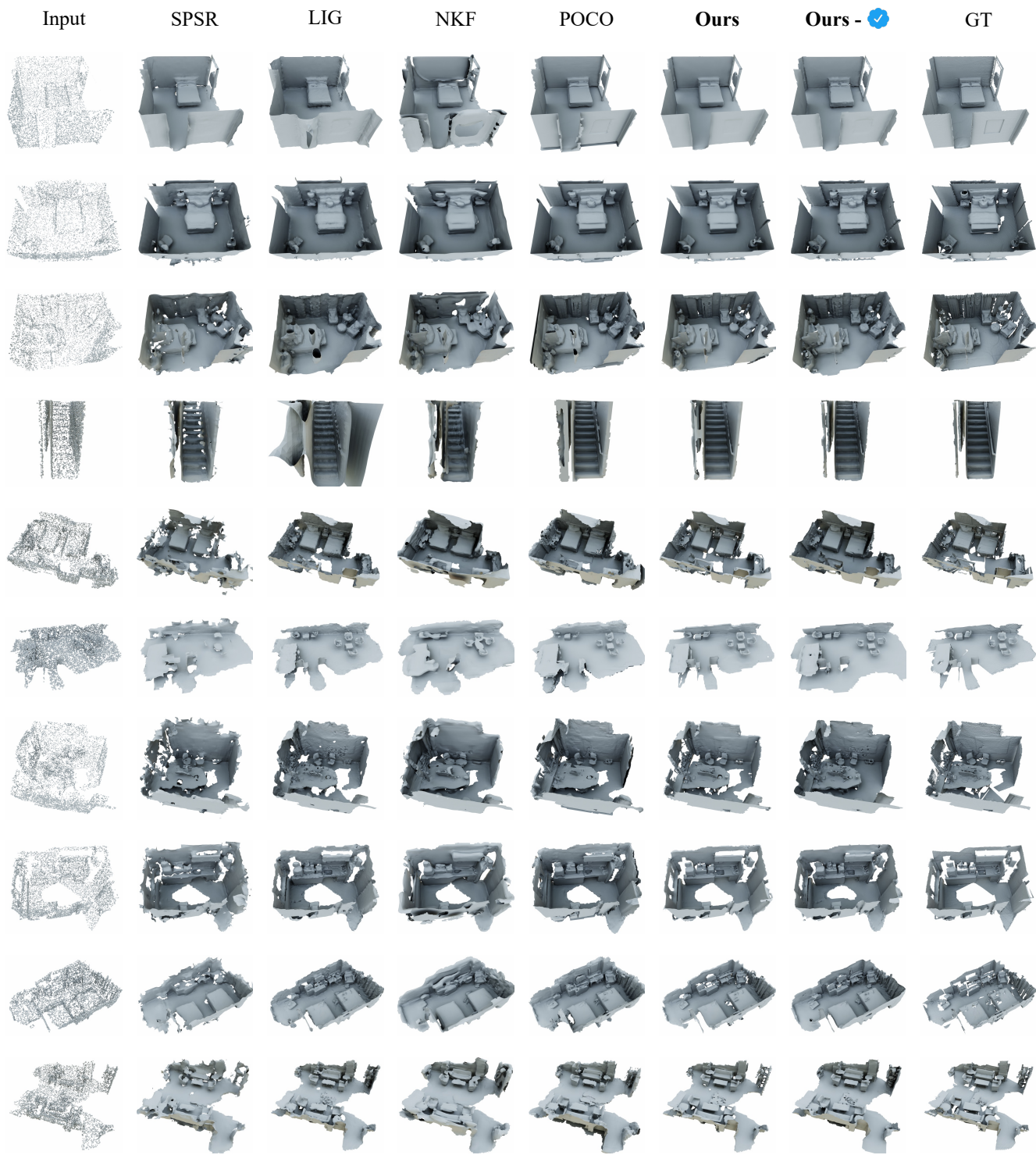


Figure 22: More results on Matterport/ScanNet datasets. Best viewed with 2× zoom-in.

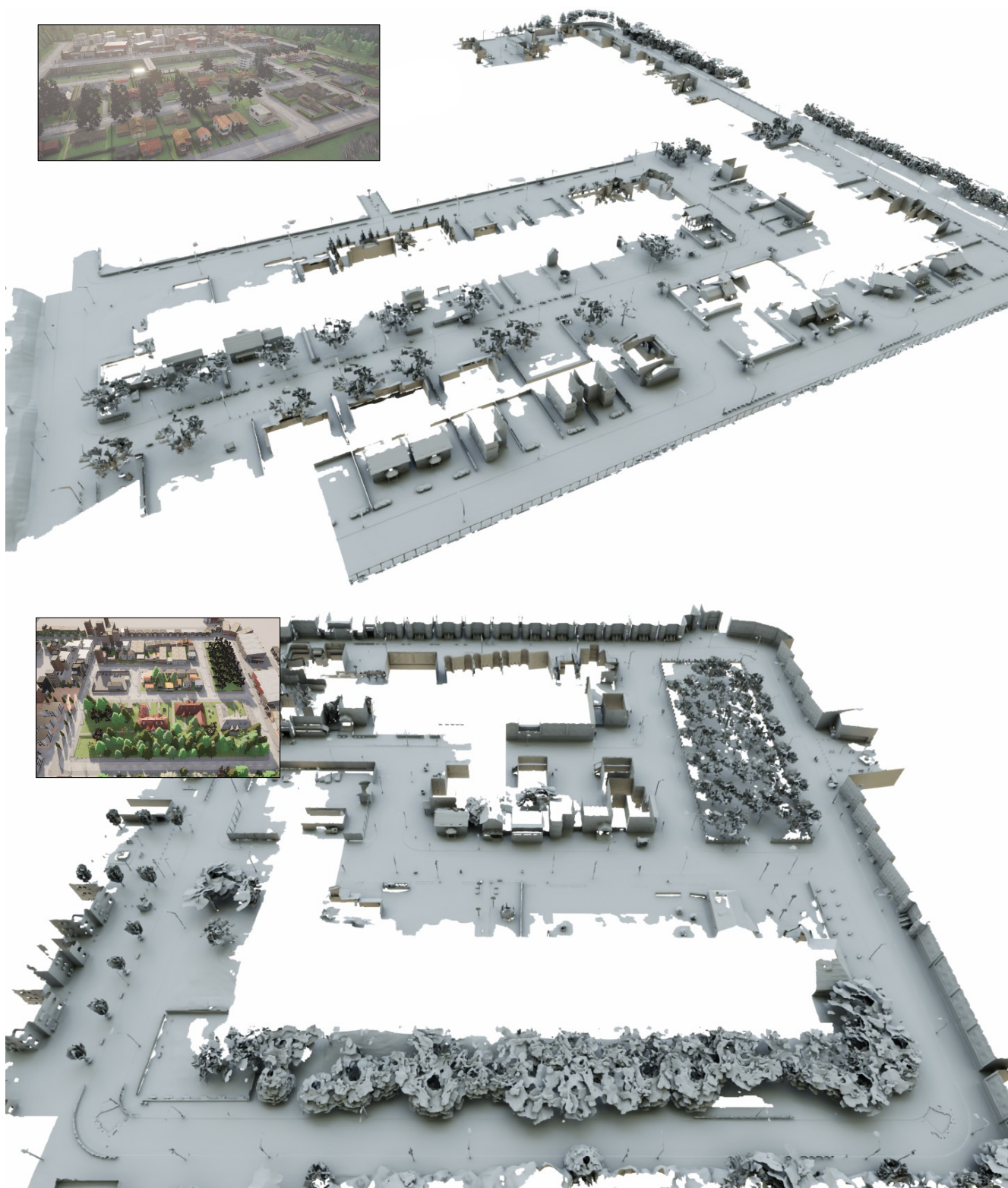


Figure 23: **Our reconstruction of the CARLA dataset.** The inset shows RGB rendering of the scene within the simulator.

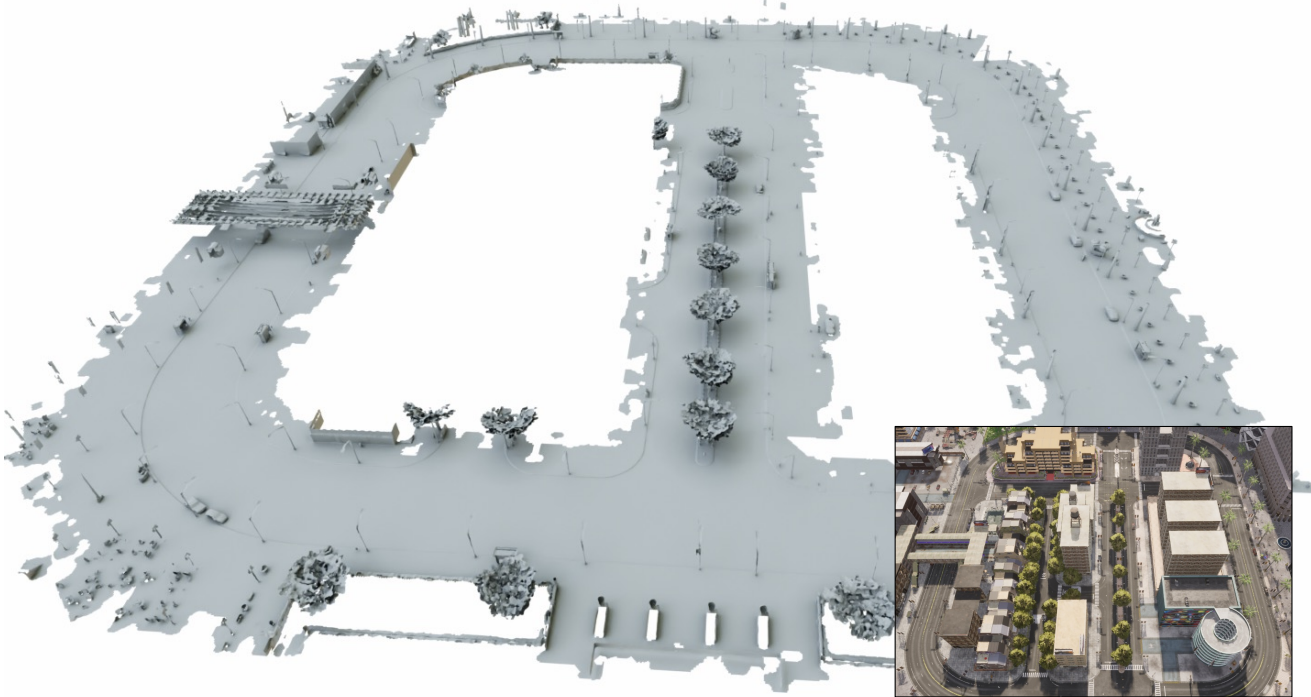
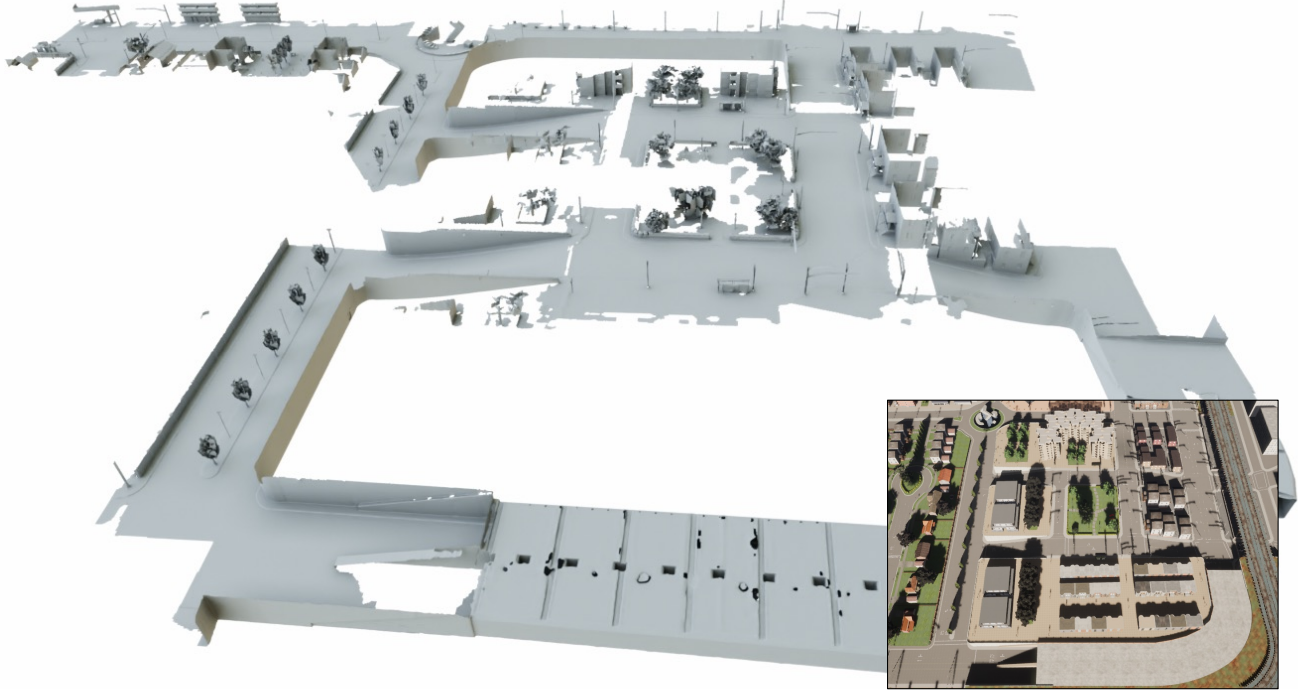


Figure 24: **Our reconstruction of the CARLA dataset.** The inset shows RGB rendering of the scene within the simulator.