# NoKSR: Kernel-Free Neural Surface Reconstruction via Point Cloud Serialization

## Supplementary Material

This appendix provides additional ablation studies, experimental analyses and more qualitative results.

## A. Additional ablation studies

**Extending Table 4 with more levels – Table 7.** A large non-linear aggregation module is essential to our method due to the false negatives in the fast approximate neighbors. However, more layers in the aggregation module degrades time efficiency. We show that $\mathcal{A}$ with 2 non-linear layers suffices to achieve the best trade-off between accuracy and time efficiency.

**Different ways to fuse per-scale features – Table 8.** We show the different ways to fuse the per-scale features and observe that they have similar accuracy. Attentive pooling achieves slightly better performance at the cost of degraded time efficiency. Note that we have an additional linear layer to predict the attention from the concatenated features of all levels to perform the attentive pooling. To realize a learnable gate where we multiply per-level weights with features before fusing levels, we train an additional learnable per-level weight followed by a Sigmoid function for the multiplication.

| **Num. of hidden layers in $\mathcal{A}$** | CD ($10^{-2}$) ↓ | F-score ↑ | Latency (s) ↓ |
|---|---|---|---|
| 0 | 0.264 | 99.16 | 130 |
| 1 | 0.262 | 99.22 | 133 |
| 2 | 0.257 | 99.33 | 152 |
| 3 | 0.258 | 99.34 | 158 |
| 4 | 0.256 | 99.32 | 166 |
| 5 | 0.256 | 99.37 | 167 |

Table 7. **Impact of capacity of $\mathcal{A}$** – the extension of Table 4 with more levels. The larger aggregation module achieves better performance with decreased time efficiency. We show that 2 layers achieves the best trade-off between accuracy and time efficiency.

| **Fusion method** | CD ($10^{-2}$) ↓ | F-score ↑ | Latency (s) ↓ |
|---|---|---|---|
| Sum | 0.257 | 99.33 | 152 |
| Average | 0.257 | 99.33 | 151 |
| Concatenation | 0.256 | 99.37 | 151 |
| Learnable Gate | 0.257 | 99.33 | 152 |
| Attentive Pooling | 0.255 | 99.36 | 156 |

Table 8. **Scales fusion** – we investigate different ways to fuse per-scale features. Attentive pooling achieves marginal improvement at the cost of noticeable increased latency.

| **Method** | CD ($10^{-2}$) ↓ | Peak Memory ($GB$) ↓ | Latency/Iter. (s) ↓ |
|---|---|---|---|
| NKSR [22] | 0.246 | 41.3 | 1.44 |
| Ours | 0.257 | 4.6 | 0.59 |
| Ours(w/KNN) | 0.243 | 8.7 | 0.64 |
| Ours (Minkowski) | 0.301 | 3.4 | 0.27 |

Table 9. **Overhead during training** We report the overhead during training in terms of GPU peak memory and latency required for each training iteration. We show that our method achieves more efficient training than the current SOTA [22].

| **Methods** | Feature Backbone(s) | Decoder(s) | Dual Marching Cube(s) | Total (s) | CD ($10^{-2}$) ↓ |
|---|---|---|---|---|---|
| NKSR [22] | 83 | 313 | 78 | 480 | 0.246 |
| Ours | 10 | 70 | 68 | 152 | 0.243 |
| Ours (w/ KNN) | 10 | 72 | 68 | 151 | 0.257 |
| Ours (Minkowski) | 6 | 30 | 56 | 97 | 0.301 |

Table 10. **Latency distribution.** We report the latency distribution during inference steps for the feature backbone $\mathcal{F}$, decoder and marching cubes. Our method outperforms the SOTA [22] in all steps, particularly in the decoder step where [22] needs to solve a large differentiable linear system.

## B. More Experimental Analysis

**Overhead during training – Table 9.** We report our method's overhead during training in terms of GPU peak memory and latency required per each training iteration. Additionally, we profile training overhead (GPU peak memory and latency per iteration) on a single NVIDIA A6000 Ada with the PyTorch Lighting API. In all cases, we use a batch size of 1 for a fair comparison with the SOTA [22] that has the batch size of 1 in one backward pass.

**Latency distribution in the steps of inference – Table 10.** We show that our method achieves better time efficiency than the SOTA [22] in all different steps whilst having better accuracy, even without the time-consuming decoder as in SOTA.

**The impact of point cloud size on time efficiency of KNN vs serialization encoding – Figure 6.** We report how the point cloud size impacts the time efficiency of KNN and neighbors based the serialization encoding. Theoretically, serialization encoding should be more efficient. However, we observe that when the point cloud size is small such as `SyntheticRoom` [36] and `ScanNet` [12], KNN is more efficient then serialization encoding. We suspect this is because KNN is highly engineered, with a CUDA implementation while the serialization encoding is purely im-

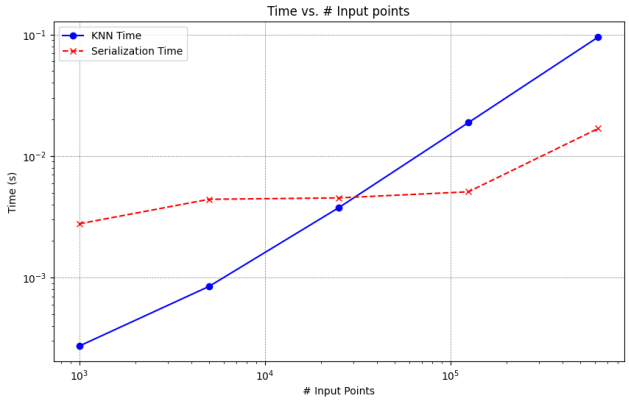



Figure 6. **Impact of point cloud size on time efficiency** We observe that K-nearest-neighbor (KNN) is more efficient than neighbors based on serialization encoding when the number of points is smaller than 25000. We suspect this is because KNN is highly optimized with a CUDA implementation, while the serialization encoding is purely based on Python.

| **Num. of segments in training and reconstruction** | CD ($10^{-2}$) ↓ | F-score ↑ | Latency (s) ↓ | Peak memory (GB) ↓ |
|---|---|---|---|---|
| 1 | 3.3 | 97.4 | 1.7 | 20.4 |
| 10 | 3.4 | 96.7 | 3.0 | 7.1 |
| 50 | 3.4 | 96.6 | 6.2 | 5.0 |

Table 11. **Handling large scenes via partition –** Simply with serialization codes, we partition a large scene into smaller segmentation to avoid GPU memory. We show that our method reduce the peak memory with a negligible decrease in reconstruction quality.

plemented in python. To estimate the time efficiency, we randomly generate 25000 query points and record the execution times of methods based on KNN and serialization encoding across varying numbers of input points.

**More metrics: completeness and accuracy – Table 15 and Table 14.** Following the state of the art method by Huang et al. [22], we further report additional metrics below. We observe that the performance is consistent with other metrics we report in main paper.

**Handling infinitely large scenes – Table 11.** We show that our method is capable of handling the infinitely large scenes. With serialization codes, we partition a large scene into segments and extract feature of segments individually, avoiding exploding the GPU memory. Note the partition stops message passing between segments, which harms the reconstruction accuracy. Even though, as shown in Table 11, our method achieves the good trade-off between reconstruction quality and the peak memory usage.

**Smoother surfaces with Laplacian loss – Table 12 and Figure 7.** We show that our method achieves smoother surfaces by regularizing the distance field $\mathcal{D}$ with Laplacian

| **weight. of $\mathcal{L}_{\text{laplacian}}$** | CD ($10^{-2}$) ↓ | F-score ↑ |
|---|---|---|
| 0 | 3.3 | 97.4 |
| 1e-4 | 3.5 | 96.3 |
| 1e-3 | 3.6 | 95.8 |

Table 12. **Impact of the weights on Laplacian loss during training.**

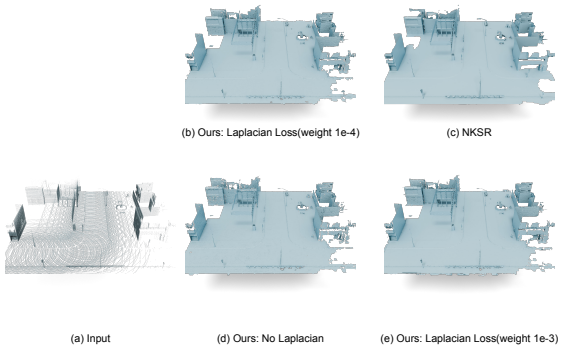



Figure 7. **Smoother surface on CARLA [13] by Laplacian Loss**

| Methods | Primitive | CD ($10^{-2}$) ↓ | IoU ↑ |
|---|---|---|---|
| NKSR [22] | Voxels | 2.34 | 95.6 |
| Ours (Minkowski w/ KNN) [11] (w/ KNN) | Voxels | 4.36 | 87.5 |
| Ours (w/ KNN) | Points | 3.91 | 89.9 |
| Ours (w/ KNN, w/ similar DMC grid number) | Points | 2.88 | 94.6 |

Table 13. **Evaluation on ShapeNet [5]**

loss from [2]. We define the loss as

$$\mathcal{L}_{\text{Laplacian}} = \mathbb{E}_{\mathbf{x}\sim\mathcal{Q}}\left[\nabla^2\mathcal{D}(\mathbf{x})\right]. \tag{8}$$

As shown in Figure 7, the larger weight of $\mathcal{L}_{\text{Laplacian}}$ leads to the smoother surface. However, as a downside, the reconstruction accuracy is degraded as shown in Table 12. Nevertheless, with the weight of $1 \times 10^{-4}$, our method achieves the better reconstruction accuracy than NKSR [22] while having the similar surface smoothness.

**Performance on synthetic object-level dataset – Table 13.** We evaluate the reconstruction quality on ShapeNet[5], a synthetic object-level dataset. Note we use the data prepared by NKSR [22], and the smaller grid size (0.005) during serialization to avoid collisions. As shown in Table 13, our method outperforms voxel-based methods, while performs worse than NKSR [22]. We suspect that the "voxel-growing" strategy in NKSR [22] is crucial to the synthetic object-level dataset, and we leave the integration of this strategy into our method for future work.

## C. More qualitative results

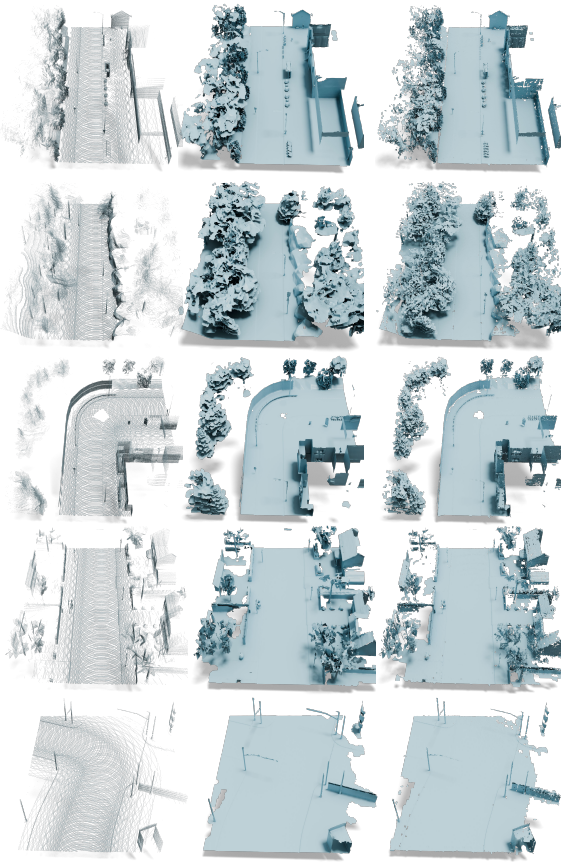
We provide more qualitative results in Figure 8, Figure 9 and Figure 10.

| Methods | Primitive | SyntheticRoom [36] | | | | | ScanNet [12] | | | | | CARLA [13] (Original) | | | | | CARLA [13] (Novel) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CD ($10^{-2}$) ↓ | completeness ($10^{-2}$)↓ | accuracy ($10^{-2}$)↓ | F-Score ↑ | Latency (s) ↓ | CD ($10^{-2}$) ↓ | completeness ($10^{-2}$)↓ | accuracy ($10^{-2}$)↓ | F-Score ↑ | Latency (s) ↓ | CD (cm) ↓ | completeness(cm)↓ | accuracy(cm)↓ | F-Score ↑ | Latency (s) ↓ | CD (cm) ↓ | completeness(cm)↓ | accuracy(cm)↓ | F-Score ↑ | Latency (s) ↓ |
| SA-CONet [39] | Voxels | 0.496 | - | - | 93.60 | - | - | - | - | - | - | - | - | - | - | - | | | | | |
| ConvOcc [36] | Voxels | 0.420 | - | - | 96.40 | - | - | - | - | - | - | - | - | - | - | - | | | | | |
| NDF [10] | Voxels | 0.408 | - | - | 95.20 | - | 0.385 | - | - | 96.40 | - | - | - | - | - | - | | | | | |
| RangeUDF [3] | Voxels | 0.348 | - | - | 97.80 | - | 0.286 | - | - | 98.80 | - | - | - | - | - | - | | | | | |
| TSDF-Fusion [49] | - | - | - | - | - | - | - | - | - | - | - | 8.1 | 8.0 | 8.2 | 80.2 | - | 7.6 | 6.6 | 8.6 | 80.7 | - |
| POCO [4] | - | - | - | - | - | - | - | - | - | - | - | 7.0 | 3.6 | 10.5 | 90.1 | - | 12.0 | 2.9 | 9.1 | 92.4 | - |
| SPSR [23] | - | - | - | - | - | - | - | - | - | - | - | 13.3 | 16.4 | 10.3 | 86.5 | - | 11.3 | 12.8 | 9.9 | 88.3 | - |
| NKSR [22] | Voxels | 0.345 | 0.304 | 0.387 | 97.26 | 0.40 | 0.246 | 0.221 | 0.27 | 99.51 | 1.54 | 3.9 | 2.2 | 5.6 | 93.9 | 2.0 | 2.8 | 2.1 | 3.6 | 96.0 | 1.8 |
| NKSR [22] (more data) | Voxels | - | - | - | - | - | - | - | - | - | - | 3.5 | 3.0 | 4.1 | 94.1 | 2.0 | 3.0 | 2.4 | 3.6 | 96.0 | 1.8 |
| Ours (Minkowski) [11] (w/ KNN) | Voxels | - | | | | | 0.254 | 0.234 | 0.273 | 99.41 | 0.46 | 3.4 | 4.1 | 2.7 | 97.2 | 1.9 | 2.7 | 3.1 | 2.4 | 98.1 | 2.0 |
| Ours (Minkowski) [11] | Voxels | - | | | | | 0.301 | 0.327 | 0.275 | 98.48 | 0.31 | 3.8 | 4.4 | 3.2 | 96.2 | 1.5 | 3.0 | 3.3 | 2.8 | 97.4 | 1.5 |
| Ours (w/ KNN) | Points | 0.322 | 0.270 | 0.374 | 98.25 | 0.13 | 0.243 | 0.230 | 0.256 | 99.61 | 0.48 | 3.2 | 3.6 | 2.8 | 97.5 | 3.2 | 2.6 | 2.7 | 2.4 | 98.3 | 3.4 |
| Ours | Points | 0.358 | 0.318 | 0.399 | 96.43 | 0.14 | 0.257 | 0.243 | 0.270 | 99.33 | 0.49 | 3.3 | 3.9 | 2.6 | 97.4 | 1.7 | 2.7 | 3.0 | 2.4 | 98.2 | 1.7 |

Table 14. **Additional metrics from NKSR [22] for in-domain evaluation**

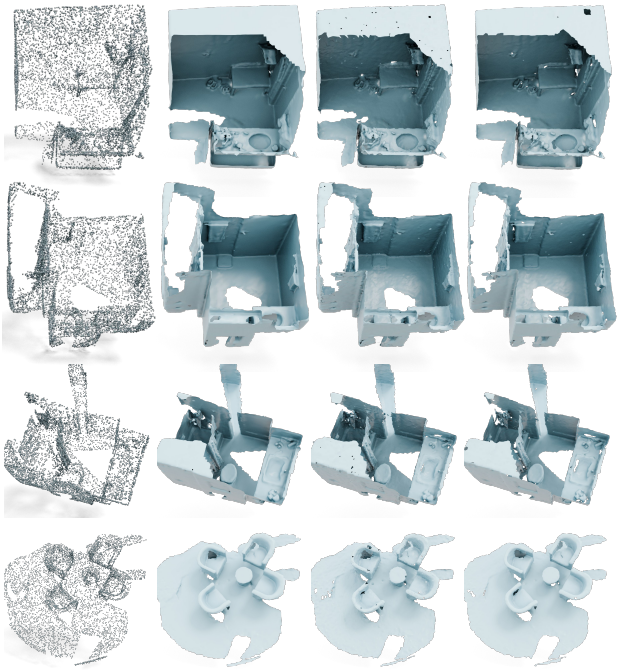
| Methods | Primitive | SyntheticRoom [36] → ScanNet [12] | | | | | ScanNet [12] → SyntheticRoom [36] | | | | | ScanNet [12] → SceneNN [20] | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CD ($10^{-2}$) ↓ | completeness ($10^{-2}$)↓ | accuracy ($10^{-2}$)↓ | F-Score ↑ | Latency (s) ↓ | CD ($10^{-2}$) ↓ | completeness ($10^{-2}$)↓ | accuracy ($10^{-2}$)↓ | F-Score ↑ | Latency (s) ↓ | CD ($10^{-2}$) ↓ | completeness ($10^{-2}$)↓ | accuracy ($10^{-2}$)↓ | F-Score ↑ | Latency (s) ↓ |
| SA-CONet [39] | Voxels | 0.845 | - | - | 77.80 | - | - | - | - | - | - | - | - | - | - | - |
| ConvOcc [36] | Voxels | 0.776 | - | - | 83.30 | - | - | - | - | - | - | - | - | - | - | - |
| NDF [10] | Voxels | 0.452 | - | - | 96.00 | - | 0.568 | - | - | 88.10 | - | 0.425 | - | - | 94.80 | - |
| RangeUDF [3] | Voxels | 0.303 | - | - | 98.60 | - | 0.481 | - | - | 91.50 | - | 0.324 | - | - | 97.80 | - |
| NKSR [22] | Voxels | 0.329 | 0.296 | 0.362 | 97.37 | 2.02 | 0.351 | 0.301 | 0.401 | 97.41 | 0.46 | 0.268 | 0.253 | 0.283 | 99.18 | 1.95 |
| Ours (w/ KNN) | Points | 0.284 | 0.266 | 0.302 | 98.65 | 0.54 | 0.327 | 0.263 | 0.391 | 98.37 | 0.13 | 0.277 | 0.277 | 0.277 | 99.00 | 0.50 |

Table 15. **Additional metrics from NKSR [22] for cross-domain evaluation**



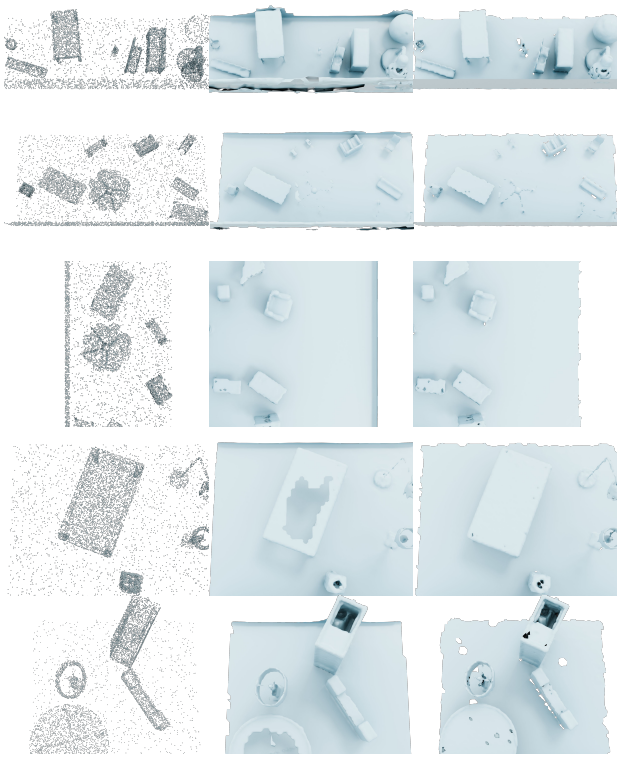
(a) Input point cloud (b) NKSR [22] (c) Ours

Figure 8. **More qualitative results on CARLA [13]** – Zoom in for better view.



(a) Input point cloud (b) NKSR [22] (c) Ours(Minkowski) (d) Ours

Figure 9. **More qualitative results on ScanNet [12]** – Zoom in for better view.



(a) Input point cloud (b) NKSR [22] (c) Ours

Figure 10. **More qualitative results on `SyntheticRoom` [36]** – Zoom in for better view.