Appendices

This supplementary material includes a benchmark evaluation on normal maps in Sec. A, our implementation details in Sec. B, and more visualization of our captured objects in Sec. C.

A. Benchmark evaluation

In addition to mesh quality, Section A.1 compares the rendered normal map quality of different methods. Section A.2 compares different strategies for rendering the normal maps of a trained neural SDF.

A.1. Normal evaluation

Normal maps are critical to the quality of physics-based rendering. Therefore, the quality of the normal maps rendered from a trained SDF or a recovered mesh is another indicator of the performance of the 3D reconstruction approach. To evaluate normal accuracy, we use 15 views of DiLiGenT-MV [16] for training and the remaining 5 views for testing. Note that PS-NeRF [31] and MVAS [4] use the same training strategy, while other methods use all 20 views for reconstruction.

Table 4 reports the mean angular error averaged over all foreground pixels from the 5 test views. Since volume or surface rendering does not guarantee unit normal vectors, we normalize the rendered normal maps before evaluation. On average, our method outperforms all compared methods in terms of normal accuracy. Compared to neural rendering methods, our method achieves a mean angular error 24% lower than PS-NeRF [31] and 21% lower than MVAS [4].

Figure 9 shows the rendered normal maps and the corresponding angular error maps. Our method produces normal maps with the best high frequency detail, especially the eyes of *Buddha* and the flower pattern on *Pot2*.

A.2. Normal map rendering at inference time

At training time, we use DFD to compute the SDF gradients and accumulate the SDF gradients on the rays to obtain the normal vectors for the pixels. At inference time, however, we are not restricted to DFD; both FD and AD can be used. Instead of volume rendering, we can also use surface rendering, *i.e.*, we only compute the SDF gradient at the zero level set points.

Table 5 compares normal accuracy in terms of mean angular error and rendering time in terms of frames per second (FPS) using different strategies for rendering the normal map. From Tab. 5, all volume rendering based methods produce close results. Finite difference performs slightly better, but at the cost of rendering time. It takes 64% more time to produce 1.1% better results. Surface rendering, on the other hand, consistently performs worse than volume rendering at inference time, even though it is almost three times faster than volume rendering. There are two possible reasons for this: 1) the high frequency noise introduced by multi-resolution hash coding, and 2) the loss function is defined based on volume rendering, which only encourages volume rendering results to match the input normal maps.

Figure 10 shows the normal maps rendered from the same neural SDFs using different rendering strategies and corresponding angular error maps. All volume rendering strategies produce visually similar normal maps. This also validates the approximation accuracy of the DFD. In contrast, surface rendering introduces noise into the rendered normal map.

Table 4. Quantitative evaluation of rerendered normal maps. Mean angular error [deg.] averaged on 5 test views are reported. Darker colors indicate better results.

Methods	Bear	Buddha	Cow	Pot2	Reading	Average
R-MVPS [23]	12.80	13.67	10.81	14.99	11.71	12.80
B-MVPS [16]	3.80	10.57	2.83	5.76	6.90	5.97
PS-NeRF [31]	3.45	10.25	4.35	5.94	9.36	6.67
MVAS [4]	3.08	9.90	3.72	5.07	10.02	6.36
Ours	2.56	7.64	3.10	4.49	7.40	5.04

Table 5. Normal map rendering accuracy and time using different strategies to render the normal maps. **VR**: Volume rendering. **SR**: Surface rendering.

		Mean angular error [deg.] (↓)						FPS
		Bear	Buddha	Cow	Pot2	Reading	Average	
VR	DFD	2.56	7.64	3.10	4.49	7.40	5.04	1.8
	AD	2.63	7.50	3.16	4.61	7.48	5.08	1.5
	FD	2.57	7.29	3.10	4.49	7.43	4.98	1.1
SR	AD	3.45	9.75	4.08	5.89	8.80	6.39	3.3

B. Implementation Details

B.1. Network architecture

Figure 11 shows our neural SDF architecture. The input 3D coordinate $\mathbf{x}$ is transformed by multi-resolution hash encoding into a 28-dim feature vector $h(\mathbf{x};\phi)$. The feature vector is concatenated with the 3D vector $\mathbf{x}$ and input to a one-layer MLP. We use 64 units and ReLU activation for the only hidden layer and no activation for the output layer. Overall, the neural SDF can be written as

$$f(\mathbf{x}) = \mathbf{w}_2 \max\left(\mathbf{W}_1 \begin{bmatrix} h(\mathbf{x};\phi) \\ \mathbf{x} \end{bmatrix} + \mathbf{b}_1, \mathbf{0}\right) + b_2, \quad (14)$$

where $\max(\cdot,\mathbf{0})$ is the element-wise ReLU function. We apply a geometric initialization [8] to $\mathbf{W}_1$ and $\mathbf{w}_2$ so that the initial zero level set of the neural SDF approximates a sphere with radius b_2. We set $b_2 = 0.7$ as the initial value. Despite its simplicity, the neural SDF can represent highly complex geometry, as shown in our reconstruction results.

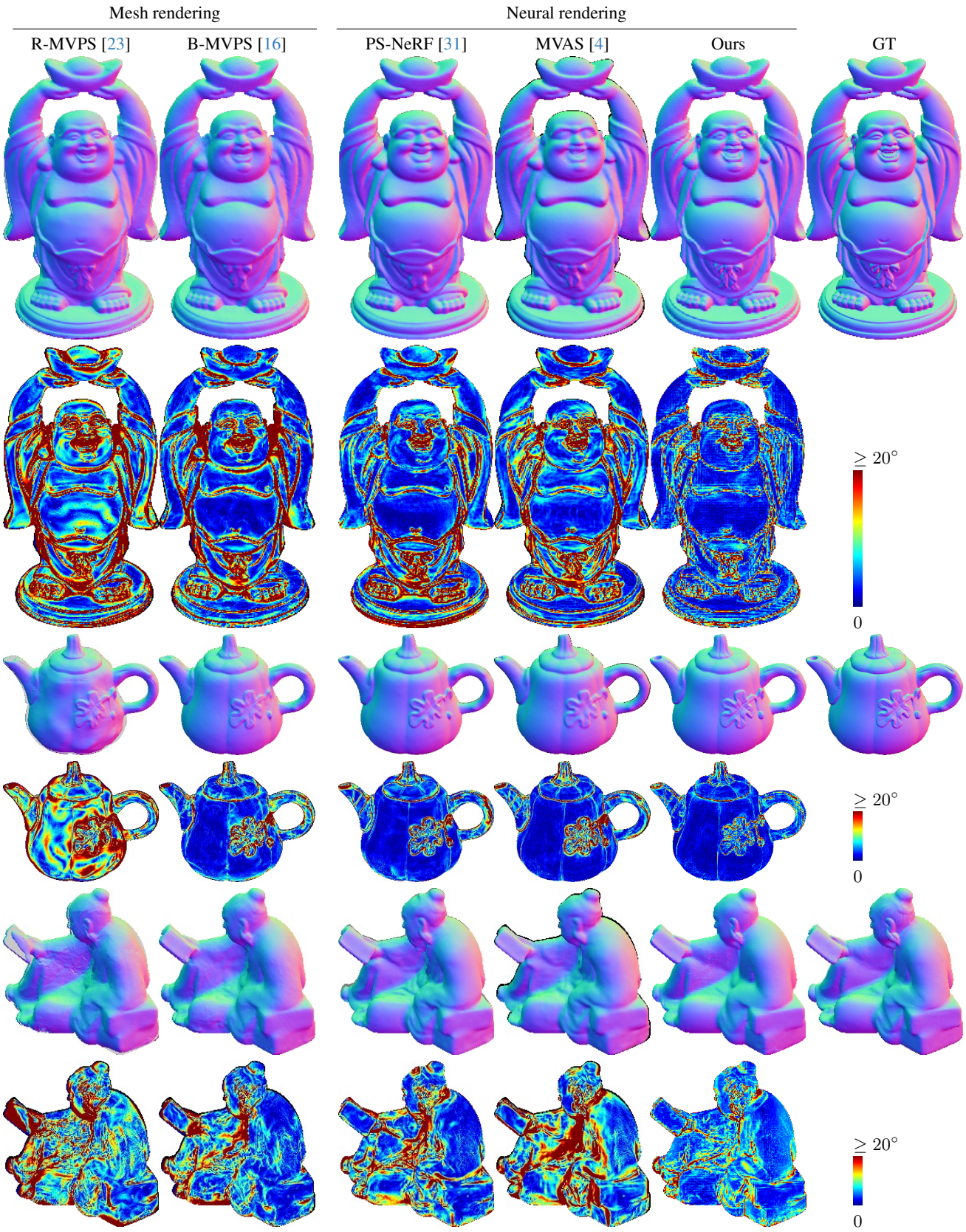

Figure 9. Qualitative comparison of normal maps rendered from reconstructed meshes or neural representations and angular error maps.

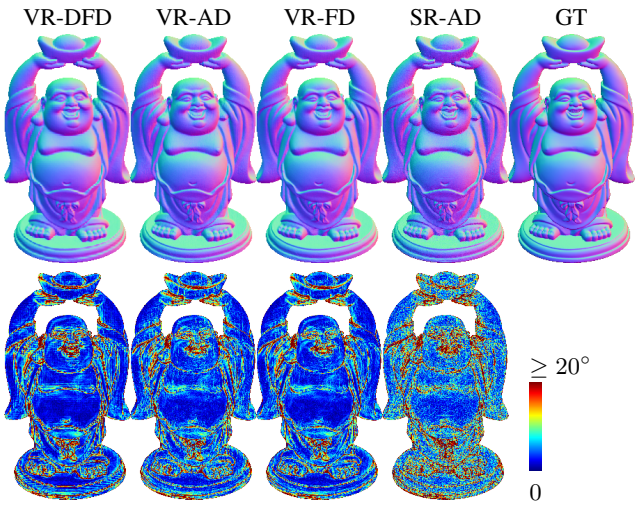

Figure 10. **(First row)** Normal maps rendered from the trained neural SDF using different rendering strategies. **(Second row)** Angular error maps.

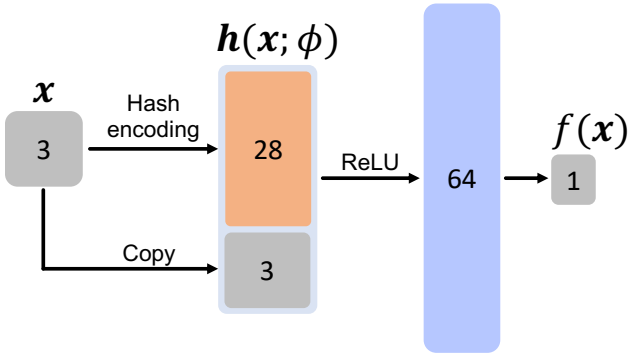

Figure 11. Neural SDF architecture.

B.2. Patch-based ray marching details

Given a patch of pixels, we perform ray marching for the center ray using NerfAcc [17] package and compute the points on the remaining rays by finding the ray-plane intersections. We maintain an occupancy grid so that the empty regions on the rays are skipped for efficient training.

Denote $\mathbf{o}$ the ray origin of a patch of rays, $\mathbf{v}_i$ the center ray's unit direction, and $\mathbf{v}_j$ the unit direction of any remaining rays in the same patch. Suppose we have sampled a point from the center ray at a distance t_i from the ray origin. According to the geometry, the corresponding point to be sampled on ray $\mathbf{v}_j$ should satisfy

$$(\mathbf{o} + t_i \mathbf{v}_i)^\top \mathbf{m} = (\mathbf{o} + t_j \mathbf{v}_j)^\top \mathbf{m}. \tag{15}$$

where $\mathbf{m}$ is the marching plane normal. In our case, $\mathbf{m}$ is perpendicular to the image plane where the patch is located. Rearrange Eq. (15) yields

$$t_j = \frac{t_i \mathbf{v}_i^\top \mathbf{m}}{\mathbf{v}_j^\top \mathbf{m}}, \tag{16}$$

After patch-based ray marching, we obtain the sampled points for a batch of patches of pixels, stored as a tensor in the shape of `(num_samples, patch_height, patch_width, 3)`, alone with a 1D tensor in the shape of `(num_samples,)` indicating to which *patch* each sample belong to. `num_samples` is the total number of points sampled from the center rays of all the patches. We then compute the SDF gradients using DFD and obtain a tensor in the shape of `(num_samples, patch_height, patch_width, 3)`.

However, NerfAcc [17] does not support patch-based volume rendering, *i.e.*, accumulating the SDF gradients of shape `(num_samples, patch_height, patch_width, 3)` into a tensor of shape `(num_patches, patch_height, patch_width, 3)`. Using a for-loop to process each patch significantly slows down the volume rendering procedure. To address this, we modify NerfAcc [17]'s CUDA code to handle patch-based volume rendering in parallel. As a result, patch-based volume rendering is as fast as pixel-based volume rendering, assuming the same amount of pixels.

B.3. Occupancy grid

For efficient training, we periodically update a binary occupancy grid. Specifically, we update the 128^3 occupancy grid for every 8 batch. The value of each grid is determined by the SDF value $f(\mathbf{x})$ at that grid:

$$occ(\mathbf{x}) = \begin{cases} 1, & \text{if } \frac{1}{1+\exp(-kf(\mathbf{x}))} < \tau_o; \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

We empirically set $k = 80$ and use a $\tau_o = 0.1$ threshold.

B.4. Evaluation metrics

L2 Chamfer distance measures the distance from one set of points to another. Given two sets of points χ_1 and χ_2, we first define the distance from one point to another set of points as

$$\begin{aligned} d_{\mathbf{x}_1 \to \chi_2} &= \min_{\mathbf{x}_2 \in \chi_2} \|\mathbf{x}_1 - \mathbf{x}_2\|_2 \quad \text{and} \\ d_{\mathbf{x}_2 \to \chi_1} &= \min_{\mathbf{x}_1 \in \chi_1} \|\mathbf{x}_1 - \mathbf{x}_2\|_2. \end{aligned} \tag{18}$$

The Chamfer distance $d(\chi_1, \chi_2)$ is then defined as

$$d(\chi_1, \chi_2) = \frac{1}{2|\chi_1|} \sum_{\mathbf{x}_1 \in \chi_1} d_{\mathbf{x}_1 \to \chi_2} + \frac{1}{2|\chi_2|} \sum_{\mathbf{x}_2 \in \chi_2} d_{\mathbf{x}_2 \to \chi_1}. \tag{19}$$

F-score is the geometric mean of the precision and recall of the recovered surfaces to the GT surfaces. Precision and recall are defined based on the distances from a point to a set of points as

$$\mathcal{P} = \frac{1}{|\chi_1|} \sum_{\mathbf{x}_1 \in \chi_1} [d_{\mathbf{x}_1 \to \chi_2} < \tau] \quad \text{and}$$
$$\mathcal{R} = \frac{1}{|\chi_2|} \sum_{\mathbf{x}_2 \in \chi_2} [d_{\mathbf{x}_2 \to \chi_1} < \tau]. \tag{20}$$

Here, $[\cdot]$ is the Iverson bracket, and τ is the distance threshold for a point to be considered close enough to a point set. F-score is then

$$\mathcal{F} = \frac{2\mathcal{P}\mathcal{R}}{\mathcal{P} + \mathcal{R}}. \tag{21}$$

We set $\tau = 0.5\,\mathrm{mm}$ in our evaluations.

To compute Chamfer distance and F-score, we find the points from the resulting mesh that are visible to the input views. Specifically, we cast rays for pixels within the mask and find their first intersection with the mesh. This strategy avoids the randomness [31] by sampling points randomly from the mesh and eliminates the effect of invisible regions.

C. More visualization on our objects

Figures 12 to 14 show more visualization results on our captured objects. Overall, our method produces better surface detail than the MVS method NeuS2 [30], and comparable results to the structured light based scanner. The scanner struggles with dark or concave regions. As shown in Fig. 14, the details of the dark region of *dog* are almost missing in the mesh reconstructed by the scanner. Our method can sometimes produce fault artifacts, as seen in the hair part of Fig. 12 and the neck part in Fig. 14.

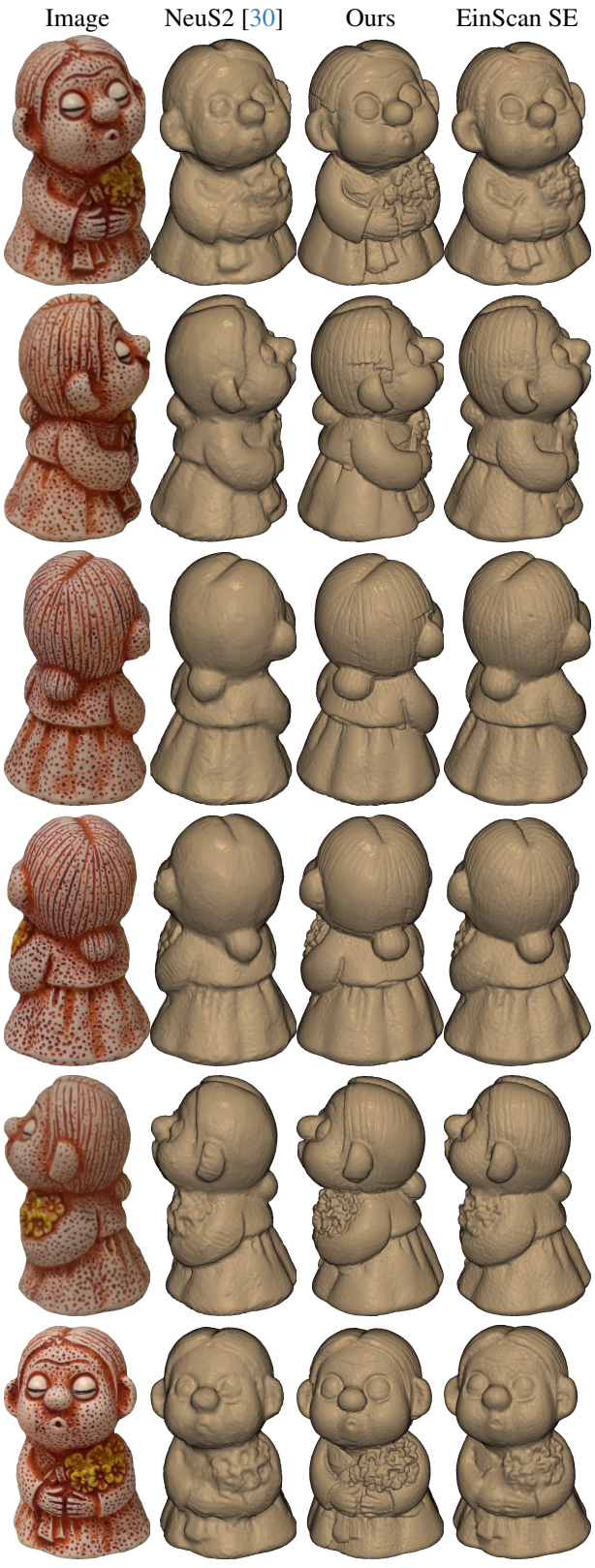

Figure 12. Qualitative comparison on our captured object *Girl*.

Figure 13. Qualitative comparison on our captured object *Lion*.

Figure 14. Qualitative comparison on our captured object *Dog*.