PermutoSDF: Fast Multi-View Reconstruction with Implicit Surfaces using Permutohedral Lattices Supplementary Material

Radu Alexandru Rosu Sven Behnke
University of Bonn, Germany
{rosu, behnke}@ais.uni-bonn.de

S1. Training Details

For the first 100 k iterations, we train using the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{rgb}} + \lambda_1 \mathcal{L}_{\text{eik}} + \lambda_2 \mathcal{L}_{\text{curv}}, \quad (1)$$

where $\lambda_1 = 0.05$, $\lambda_2 = 1.5$. For the remaining 100 k iteration, we remove $\lambda_2 \mathcal{L}_{\text{curv}}$ and replace it with $\lambda_3 \mathcal{L}_{\text{Lipschitz}}$, where $\lambda_3 = 1\text{e}{-5}$.

For 3D point sampling, we first create 64 uniform samples along each ray. We restrict the samples to be within the region that is defined as occupied by the occupancy grid. Afterwards, we run two iterations of importance sampling, each creating an additional 16 samples in the regions that are close to the surface. Concentrating samples close to the surface is crucial for recovering detail.

S2. Synthetic Data Comparison

We train also NeuS [4] and INGP [2] on the synthetically rendered head dataset described in Sec 7.3. The recovered meshes are shown in Fig. 1.

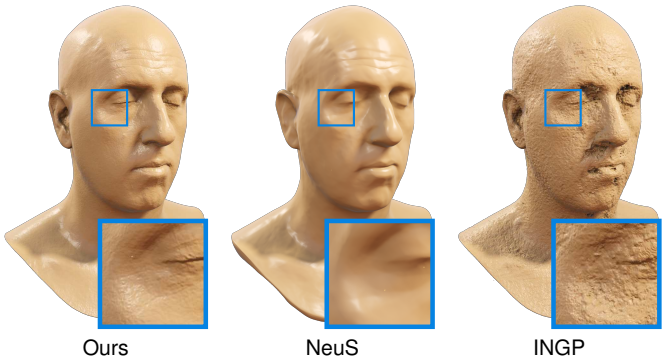

Figure 1. We train our method, NeuS [4], and INGP [2] on the synthetically rendered dataset, described in Sec 7.3. We recover significantly more small detail than the other two methods. Best viewed zoomed-in.

S3. Rendering Strategy

We compare images rendered through volumetric integration to the ones using sphere tracing. We observe that sphere tracing has the advantage of being significantly faster, as most rays converge towards the surface in few iterations. However, grazing surfaces require an arbitrary number of iterations and since we use a maximum of 20 iterations, these grazing surfaces may exhibit artifacts. A comparison between volumetric rendering and sphere tracing is shown in Fig. 2.

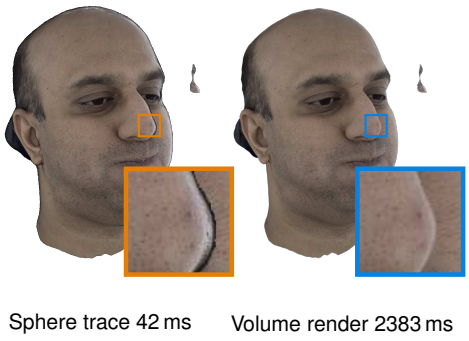

Figure 2. Sphere tracing is significantly faster than volumetric rendering, but it suffers from artifacts at surfaces with a grazing angle.

S4. Tetrahedron vs Cube

Apart from the speed improvements of using a permutohedral lattice instead of a hyper-cubical one, we are also interested on maintaining the encoding quality and therefore the reconstruction details. We reconstruct the same scene with both permutohedral encoding and cubical encoding as described in INGP [2]. We set the hash maps of both approaches to the same number of parameters, features per layer, and levels. We also extended the cubical lattice with the coarse-to-fine optimization in order match the optimiza-

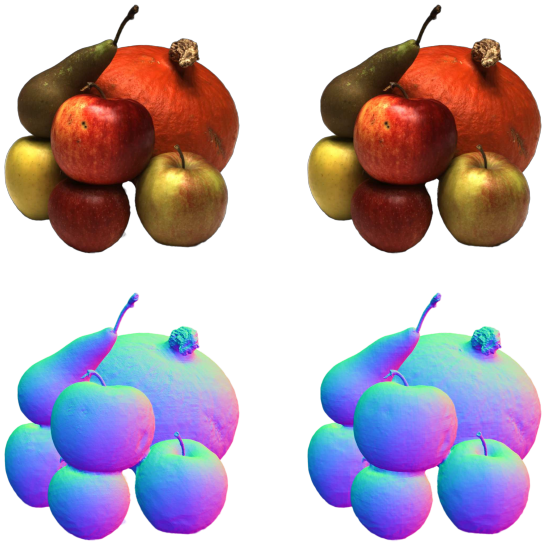

Figure 3. We reconstruct the same scene using cube encoding and permutohedral encoding. We did not observe significant differences in the reconstruction quality.

tion behavior of the permutohedral lattice. In Fig. 3, we show both reconstructions and compare their Chamfer distance and PSNR values for novel-view synthesis. We did not observe a significant difference in the reconstruction quality.

S5. Occupancy Grid Update

We initialize an occupancy grid with all the voxels being occupied and with an initial SDF that is constant zero. This ensures that we sample everywhere at the beginning of training.

For updating the occupancy grid, we use the following steps:

- Every 8th iteration of training, we sample 2^{18} random points within the bounding box that contains the scene.
- We obtain the SDF value $s_{\mathbf{x}}$ for each point $\mathbf{x}$ by running a forward pass through the model.
- We obtain the old SDF value s_{old} stored for the voxel in which the point falls into.
- We compute a new SDF value for this voxel s_{new} as the exponential average of the old SDF for the voxel and the SDF for the point: $s_{new} = s_{old} + 0.3(s_{\mathbf{x}} - s_{old})$.
- Since we are discretizing the SDF to a grid, and we don't want to miss any possible low SDF values that we would want to sample, we compute the minimum possible SDF that can be reached within this voxel under the assumption of perfect Eikonal loss. For this, we use: $s_{min} = max(0, |s_{new}| - d)$, where d is the length of the voxel diagonal.
- Using the logistic density distribution as described in NeuS [4], we compute the weight that this sample would contribute to the volumetric render—assuming no obstruction from other samples: $w = a \cdot e^{-a \cdot s_{min}} / (1 + e^{-a \cdot s_{min}})^2$.
- If the weight w falls bellow a specified threshold, we set the voxel to unoccupied and therefore don't create samples within it anymore.

S6. Color Calibration

We observe that some datasets exhibit images with different exposure times. This discrepancy between images can influence both the reconstruction and the obtained color field as the network would try to explain the variability with view-dependent effects. We circumvent this by learning a per-camera gain $g = (1 + \Delta g)$ and bias b so that the reconstructed color for each camera is $c = \sigma(\hat{c} \cdot g + b)$, where $\hat{c}$ is the raw color output from the network and σ is a sigmoid function that restricts the color to the correct range. We set a selected camera (usually the first one from the dataset) to have $g = 1$ and $b = 0$ and apply weight decay to Δg and b to further ensure that the calibration doesn't alter the colors unnecessarily.

S7. 4D Spatial-temporal Surface

For fitting a 4D surface, we sample random points from animated 3D meshes. These 3D points are concatenated with a time dimension that ranges from 0 to 1, where 0 is the start time of the animation and 1 is the end. We define these 4D samples at the surface of the mesh as $\mathbf{x}_s$. We also compute the normal $\mathbf{n}_s$ for each on the surface samples. We additionally define random 4D samples in a bounded domain around the animated mesh which we denote with $\mathbf{x}_r$ Afterwards, we learn a model $g(\mathbf{h}; \Phi)$ together with an encoding $\mathbf{h} = \text{enc}(\mathbf{x}; \theta)$ that maps from the 4D coordinate to an SDF value. For this, we follow the approach of SIREN [3] and use a loss of the form:

$$\begin{aligned} \mathcal{L}_{\text{sdf}} = & \sum_{\mathbf{x}_s \cup \mathbf{x}_r} \left(\|\nabla g(\text{enc}(\mathbf{x}))\| - 1 \right)^2 \\ & + \sum_{\mathbf{x}_s} \|g(\text{enc}(\mathbf{x})\| \\ & + \sum_{\mathbf{x}_s} (\nabla g(\text{enc}(\mathbf{x})) \cdot \mathbf{n}_s - 1) \\ & + \sum_{\mathbf{x}_r} exp(-\alpha \cdot |g(\text{enc}(\mathbf{x}))|). \end{aligned} \tag{2}$$

Figure 4. We experiment with the number of input images for our method. We observe significant degradation at around 7 input images and a failure to converge at 3 images.

In this 4D experiment, no explicit smoothness was enforced in the temporal domain since we didn't find it necessary. We sample from an animation of 100 frames so the temporal resolution is relatively high. At lower temporal resolution, smoothness might again become a concern.

Nevertheless, this approach shows that our model can deal with 4D representations onto which further ideas, like dynamic deformation fields, can be built upon.

S8. Number of Cameras

In order to study the robustness of our method to the number of input images, we vary the number of images used for reconstruction as shown in Fig. 4. Due to the curvature loss, our method can recover smooth but plausible surfaces even with as low as seven input images. However, for a smaller number of input images we observe a high likelihood of not converging to the correct surface.

S9. Training schedule

We follow a fixed training schedule over 200k iterations. This includes a phase where we train with curvature loss in order to recover the rough shape, and another phase with RGB regularization to recover detail. In order to study the robustness of our method to this schedule, we expand and contract the schedule to be as long as 300k iteration or as short as 50k iterations and show the results in Fig. 5. By modifying the schedule, we proportionally expand or contract the time that is spent optimizing the sphere, training with high curvature, and training with RGB regularization. We observe that the model is quite robust to different schedules and only for the very short ones it fails to recover some of the geometry. In general, we found that view-dependent effects like the highlight on the apple are the parts that take the longest to converge to good geometry. Most objects are reconstructed well with shorter schedules but our default schedule of 200k iteration is a good trade-off between optimization speed and accuracy.

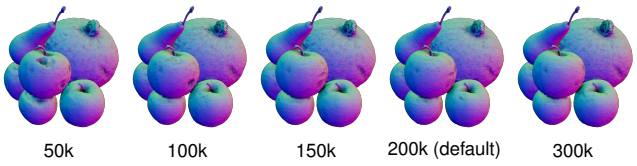

Figure 5. We follow a fixed training schedule that finishes after 200k iteration. We expand and contract this fixed schedule to be shorter or longer in order to test robustness. We see that for a schedule of 50k the method fails to reconstruct the geometry for the highlight of the apple. Our default of 200k can recover good geometry in reasonable time. A longer schedule results in better reconstructions at the cost of more optimization time.

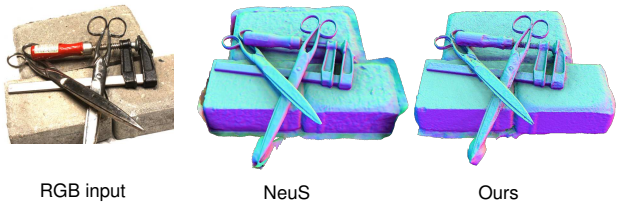

Figure 6. We observe that our model sometimes struggle with very reflective surface like the metal on the scissors. It tends to add noisy geometry to these surfaces in order to explain the view-dependent effects. Object priors or a higher curvature loss for this kind of objects could alleviate the issue.

S10. Reflective Surfaces

Our model tries to explain large color changes with changes in geometry. This behavior can be detrimental in the case of mirror-like surfaces. As we observe in Fig. 6, NeuS recovers a smooth surface on the metal scissors while our method exhibits more noise. This can be seen as a general limitation of RGB reconstruction methods for which it is difficult to know if the changes in color are from view-dependent effects or from high-frequency geometry. A model that learns object priors might perform better in these cases.

S11. Thin Structures

An interesting case to test for our SDF-based method is reconstructing thin structures. For this, we capture 14 images of a plant with relatively complex geometry with many leafs and self occlusions. Fig. 7 shows a rendered novel view and surface normals. Our method reconstructs accurate color and plausible geometry. Despite some errors that are to be expected given the low image count, it can recover thin steams and leaves which shows that our method is robust to this kind of data.

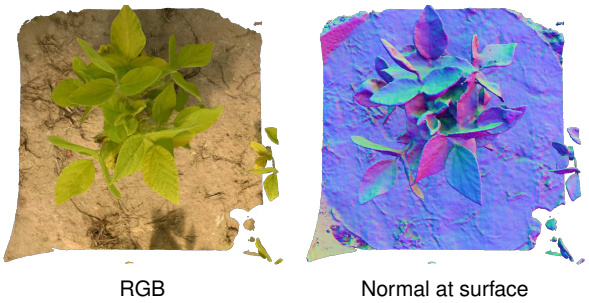

Figure 7. Plant reconstruction is an especially difficult case since it features many self-occlusions and thin structures. We observe that our method can deal well with this kind of data despite using only 14 images as input.

S12. Qualitative DTU Results

In Fig. 8 – Fig. 10, we show additional qualitative results from the DTU dataset [1]. We show extracted meshes and error maps which represent the distance from each mesh vertex towards the nearest point from the ground-truth. Please note that the ground-truth can have holes in areas of high reflectance or self-occlusion and this shows as a bright yellow color in the error map.

References

[1] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 406–413, 2014. 4

[2] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (SIGGRAPH)*, 41(4):102:1–102:15, July 2022. 1

[3] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pages 7462–7473, 2020. 2

[4] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 27171–27183, 2021. 1, 2

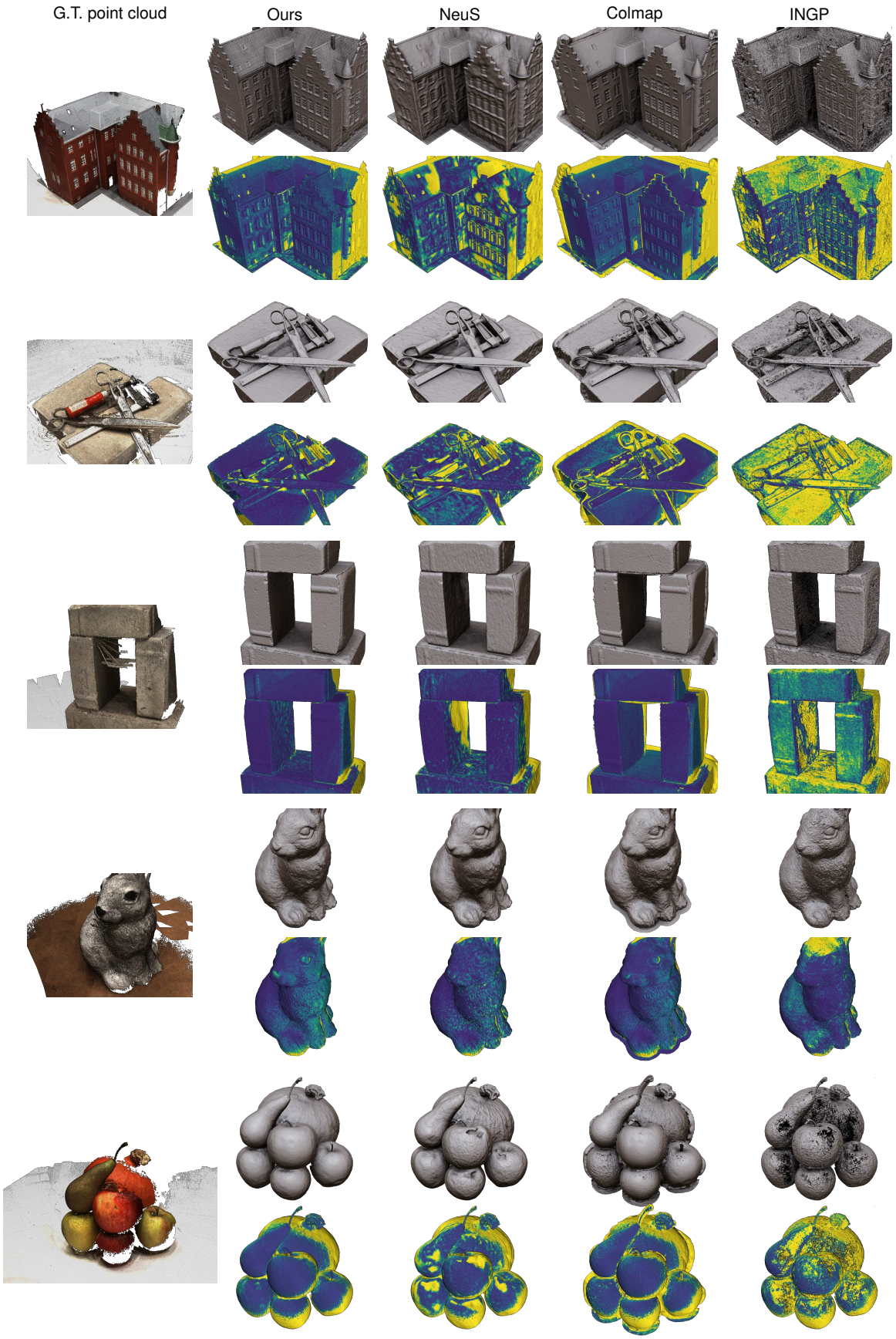

Figure 8. DTU qualitative comparison of extracted meshes and error maps.

Figure 9. DTU qualitative comparison of extracted meshes and error maps.

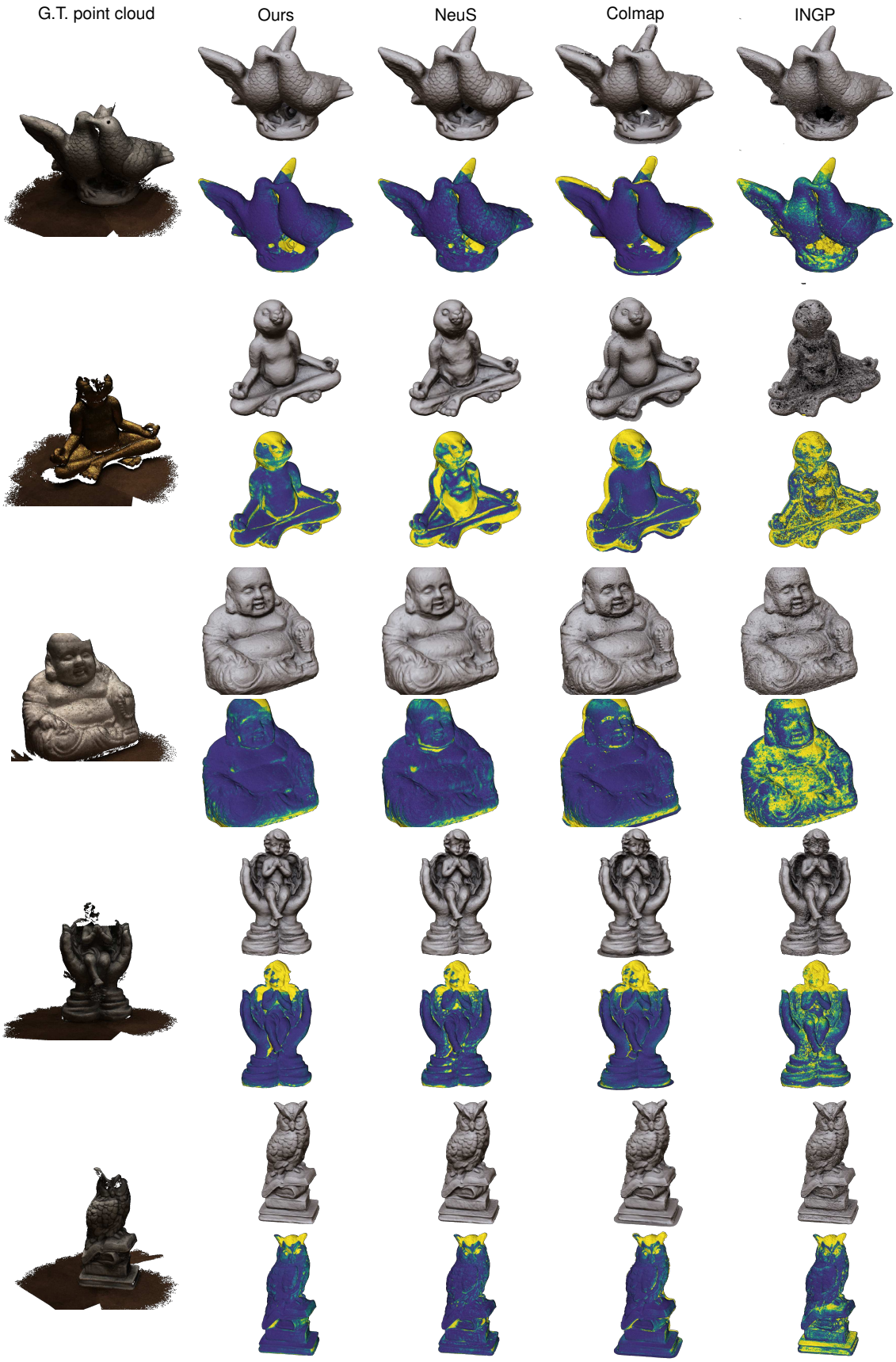

Figure 10. DTU qualitative comparison of extracted meshes and error maps.