

Supplementary Material: Deep Implicit Surface Point Prediction Networks

Rahul Venkatesh¹ Tejan Karmali² Sarthak Sharma³ Aurobrata Ghosh³
R. Venkatesh Babu² László A. Jeni^{1*} Maneesh Singh^{3*}

¹Carnegie Mellon University, Pittsburgh, PA, USA ²Indian Institute of Science, Bengaluru, India

³Verisk Analytics, Jersey City, NJ, USA

Supplementary Material

In the paper, we presented results from the shape-agnostic CSP network (a single function for all shapes) which for a given encoded shape provided at the input, produced the closest surface point for the queried input point. Here, we first supplement those results with a single-shape CSP network. This is presented in (Sec. 1) below.

Subsequent sections present additional details for the experimental evaluation in the main paper as follows:

- The network architecture and training details pertaining to the shape representation trained in Section 4.1 in the main paper are presented in Sec. 2.
- Details for the Jacobian computed in Section 4.2 of the main paper and its computational performance are presented in Sec. 3.
- The experimental setup used for rendering and meshing (Section 4.3 of the main paper) is described next in (Sec. 4).
- Sec. 5 presents additional qualitative results for rendering via sphere tracing, supplementing those presented in Section 4.3 of the main paper.
- Sec. 6 share the details of the various off-the-shelf tools used in our implementations and experimental evaluation.

1. Single-shape CSP

While the primary focus of this work was to build a single, shape-agnostic CSP model, we present here a model for a single-shape CSP implemented as follows: For any input point (or query point) in the 3D space, p , a 10-layer MLP estimates the closest point on the surface, \hat{p} . Let fc_i denote a fully-connected layer with i output dimensions. Then the MLP is given by

$$fc_{120}, fc_{512}, fc_{1024}, fc_{2048}, fc_{2048}, \\ fc_{1024}, fc_{512}, fc_{256}, fc_{128}, fc_3$$

where the input dimension of fc_i is determined by the output dimension of the layer prior to it and every fc layer is followed by *ReLU* non-linearity, except the final layer. The architecture of the single-shape CSP is presented in Fig. 1.

We present qualitative results for single shape reconstruction for a few complex shapes in Figures 2 and 3, illustrating the ability of CSP to model complex shapes with high fidelity having either an open or a closed topology. It can be clearly seen that CSP is able to preserve surface details and accurately represent the surface orientations. In Figures 2 and 3, we present results on complex shapes like (a) a dried rose, and, (b) a lion statute having an intricate design and regions of varying curvature (c) a bathtub, that has high levels of detail and complex sub-structures, (d) the seifert surface [11], that has complex topology (multiple holes and knots).

2. Training and architecture details

This section shares the network architecture modeling the shape representation in Section 4.3 of the main paper and details for training it.

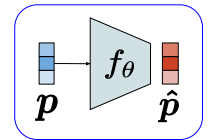


Figure 1: Single shape CSP

¹indicates two authors equally advised

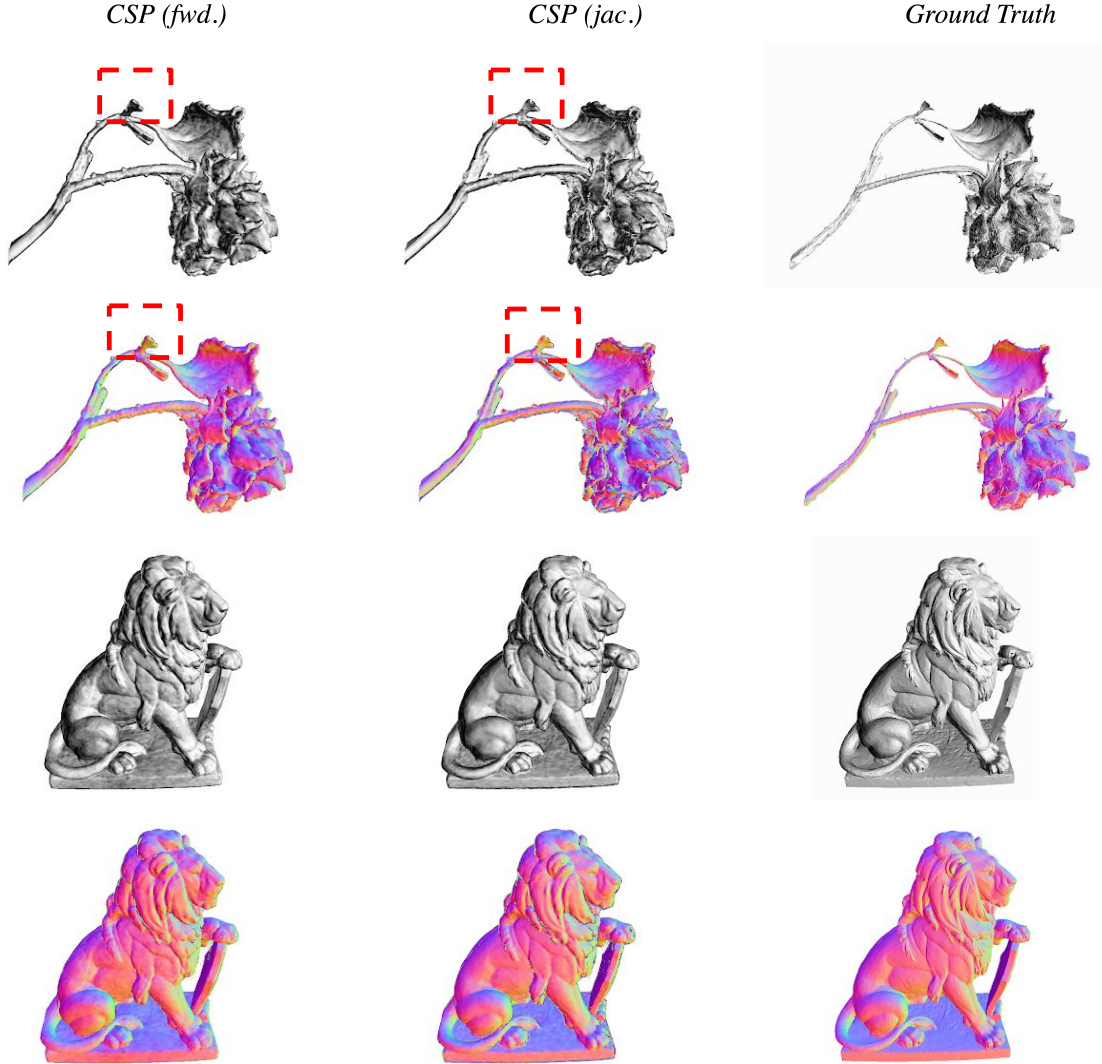


Figure 2: **Single Shape reconstructions:** Renderings from single shape architecture described in Sec. 1. Here, we evaluate *CSP* independently on two shapes with complex structures. We show lighted normals (row 1 of each shape) as well as the raw normal map (row 2 of each shape) using both normal estimation methods (see Sec. 3.2 in main paper) and compare against the ground truth for the same. The *CSP* (jac.) results in higher quality normals compared to *CSP* (fwd.), which are reasonably comparable, but provide us with faster estimates (Highlighted in Red). More examples on next page.

We use the 3D volumetric encoder architecture proposed in [9] with a *feature volume of resolution 64*. Since our point estimation task is arguably more complex than binary occupancy prediction, we use a larger *decoder, with 512 hidden units* (with the same architecture as in [9]).

We train with a *batch size of 32* on different shapes, with an input point cloud of size 3000 (we follow the setup in NDF [2]). For each shape in the batch, we use 10K points sampled from the training points, \mathcal{P} (See Sec. 4.1.2 of the main paper). We train on an NVIDIA GeForce RTX 2080Ti GPU using an ADAM [8] optimizer and a *learning rate of 1e-4*. It takes ≈ 5 days to train on the full ShapeNet dataset.

3. Jacobian Computation: Implementation Details

We now share the implementation details for the Jacobian computation as described in Sec. 3.2.1 of the main paper and discuss implications on its computational performance.

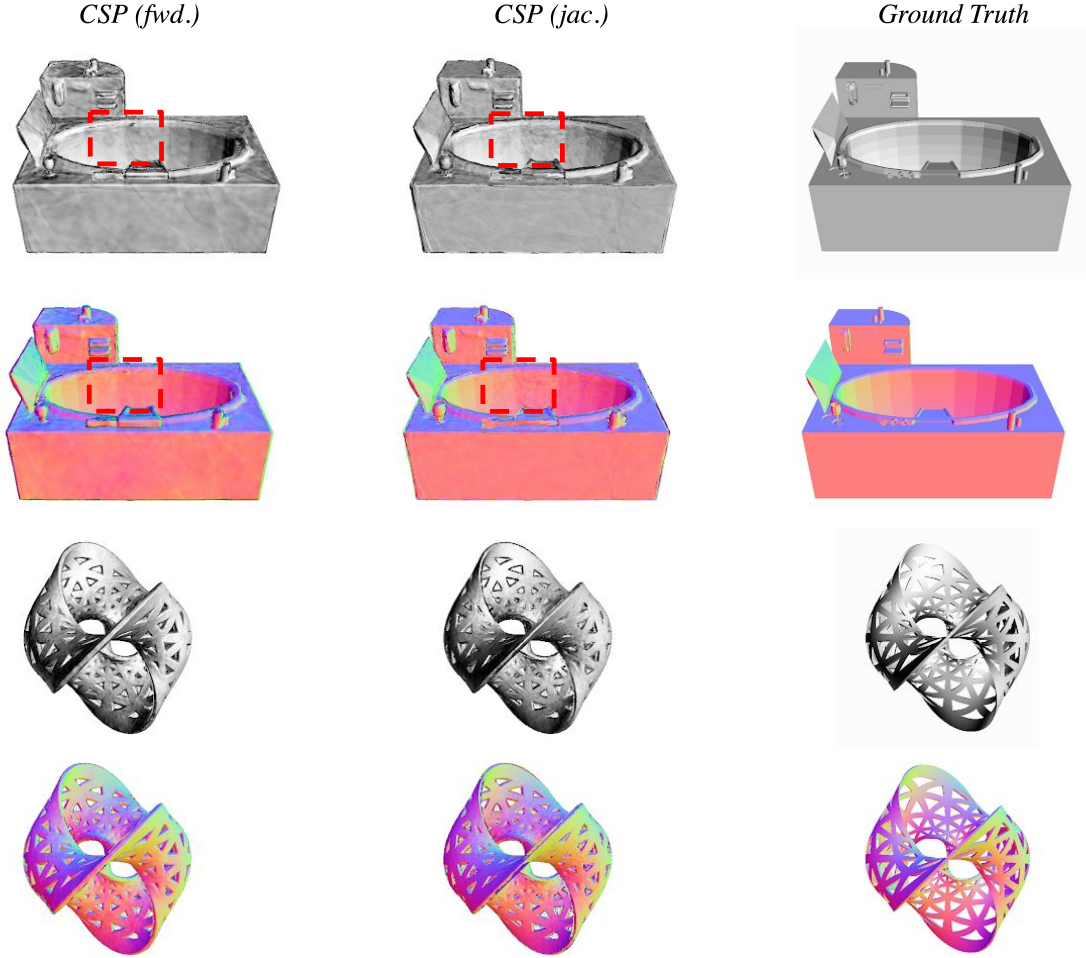


Figure 3: **Single Shape reconstructions:** Here, we show some results on a bathtub which has a high level of detail, with complex sub-structures, and a seifert surface which has complex topology (knots and holes).

The Jacobian is computed using 1 forward pass and 3 backward passes (one for each row of the Jacobian) through the same network. For this, we use the `autograd` package in PyTorch and set `retain_graph=True` when computing the first row of the Jacobian. This caches the activations in the graph and makes them readily available for computing the subsequent rows, speeding up the computation of the Jacobian.

We logged the time taken to estimate the Jacobian matrix for the experiments described in Sec. 4.3 of the main paper for *CSP (jac.)* and find that it takes on an average 0.08s for a 512×512 image. In comparison, NDF is faster and takes 0.063s. This is to be expected as NDF just needs 1 forward and 1 backward pass. However, given that the computational graph needs to be obtained only once, we only incur an additional 25% overhead (0.017s). Therefore, this is a reasonable trade-off for extracting high-fidelity surface normals.

On the other hand, we also proposed an extremely fast method, *CSP (fwd.)*, which computes surface normals in a forward-mode taking only 0.003s for a 512×512 image and is of a quality surpassing that of NDF (See Table 4 of main paper).

4. Meshing and Rendering: Experimental setup

Results for setup used for rendering and meshing are presented in Section 4.3 of the main paper. Here we provide details of the experimental setup.

Rendering. For a given input point cloud, we first compute the 3D feature volume from the encoder. We then render the learnt *CSP* representation (modeled using the decoder) from 3 different views. For doing so, we create a batch of rays from each viewpoint (3 views give us a total of $512 \times 512 \times 3 = 0.79\text{M}$ rays), and begin the sphere-tracing process (batched/parallel)

for these set of rays. At the termination of sphere-tracing, we compute the surface normals for each ray (using gradients in case of NDF, and *NVF* in case of *CSP*). Since *NVF* does not require a backward pass, it can accommodate a batch of 0.5M rays on a 8GiB GPU. The corresponding batch size for NDF is much lower at 0.15M since it requires the computation of gradients. As reported in Table 5 of the main paper, the increased batch size leads to a significant improvement in the rendering speed (i.e. time taken to compute the surface normals).

Meshing. We present here additional details for meshing *CSPs* using the novel coarse-to-fine meshing strategy outlined in Sec. 3.3.2 of the main paper. We compute a 3D distance grid (of resolution = 256) using the proposed hierarchical space subdivision strategy, and perform meshing using Marching Cubes (using *libmcubes* [10]) with a small positive threshold of $\epsilon = 0.006$. For NDF, we use the code provided by authors to generate a dense point cloud (of 1M points) and mesh it using the ball-pivoting [1] tool in meshlab [3], using a ball-radius of 0.01.

In our experiments, we have found the ball-pivoting process to be very sensitive to this threshold, and in many cases it had to be tuned per-shape. On the other hand, our method uses a single threshold for all shapes, and generates high-fidelity meshes. Moreover, as reported in Sec. 4.3 of the main paper, our coarse-to-fine meshing strategy is significantly faster than that of NDF.

5. Additional qualitative results

To supplement the qualitative results on the various sphere-tracing strategies (Fig. 5 of main paper), in Fig. 4, 5, we show additional results which compare *depth maps* generated using our novel sphere-tracing algorithm for *CSP*, against a vanilla sphere-tracing technique for unsigned distance functions. Further, in Fig. 6, 7, 8, 9, 10 we show additional examples of shape reconstruction which bolster the results shown in Fig. 1 of the main paper, and demonstrate the capability of our class-agnostic model to reconstruct shapes from any class of ShapeNet. All results are shown on a test-set of shapes (ShapeNet test-set used in [6]) not seen in training. Additionally, to reiterate the utility of meshes generated by our novel meshing algorithm for *CSPs* (Sec. 3.3.2 of main paper), we also show some *representative meshes* (compared against GT meshes) generated in Fig. 11.

6. Off The Shelf Tools and Packages Used

In this work, we make use of a variety of off-the-shelf packages to run our experiments. For generating data, we use *faiss* [7], which is a library for performing fast nearest neighbour search on GPU. We compute GT normal and depth maps using the *trimesh* [5] with *pyembree* bindings viz. `trimesh.ray.pyembree.RayMeshIntersector`. *torch-scatter*¹ is used for trilinear interpolation of the 3D Feature Volume (See Fig. 2 of the main paper). For sphere-tracing *CSP*, we provision a custom implementation in *PyTorch*, which renders multiple images efficiently by batching rays across different views.

¹https://github.com/rusty1s/pytorch_scatter

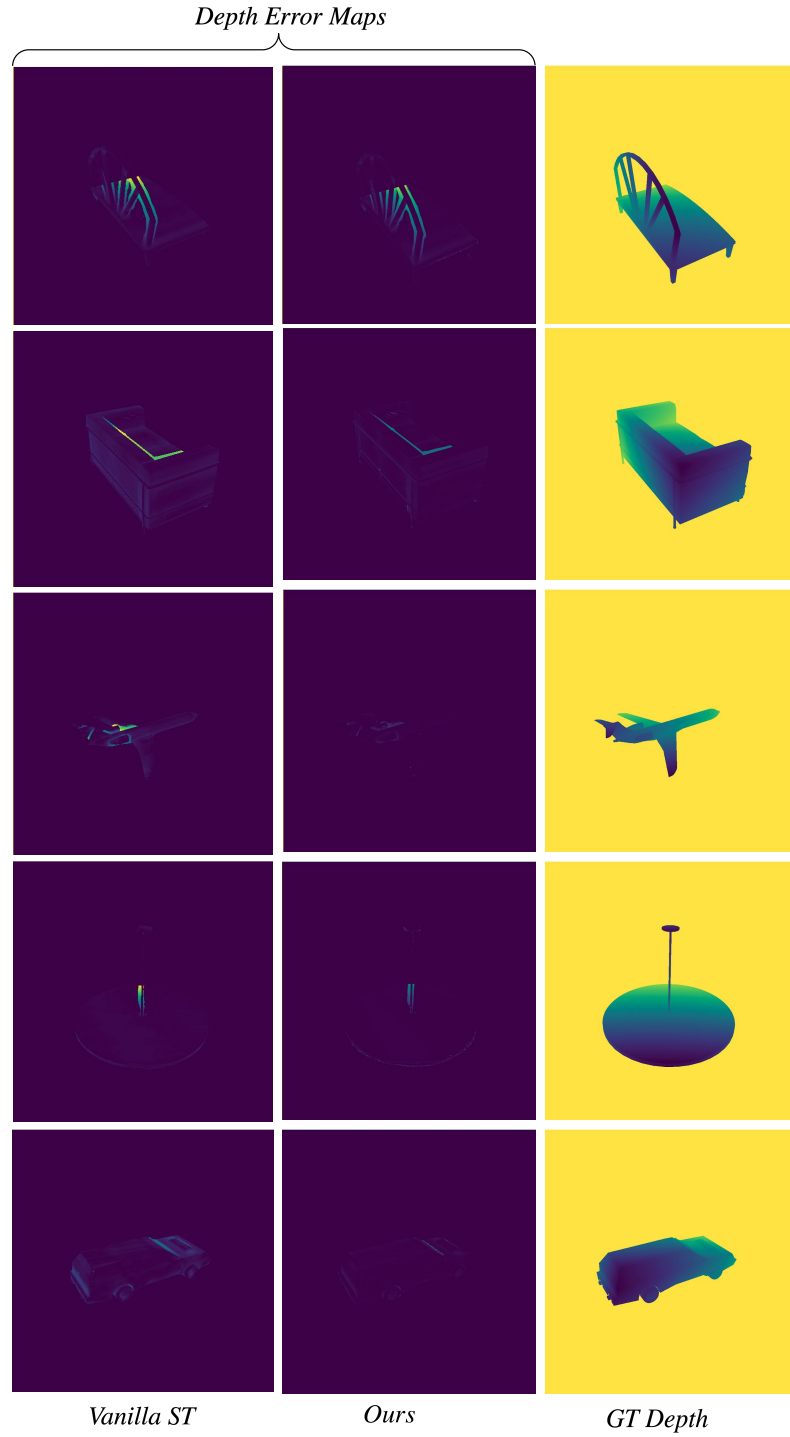


Figure 4: Comparison of depth maps generated by Vanilla Sphere Tracing (ST) and our novel projection-based algorithm outlined in Sec. 3.3.1 of the main paper. We find that our method generates much lesser error when compared to the conventional sphere-tracing strategy.

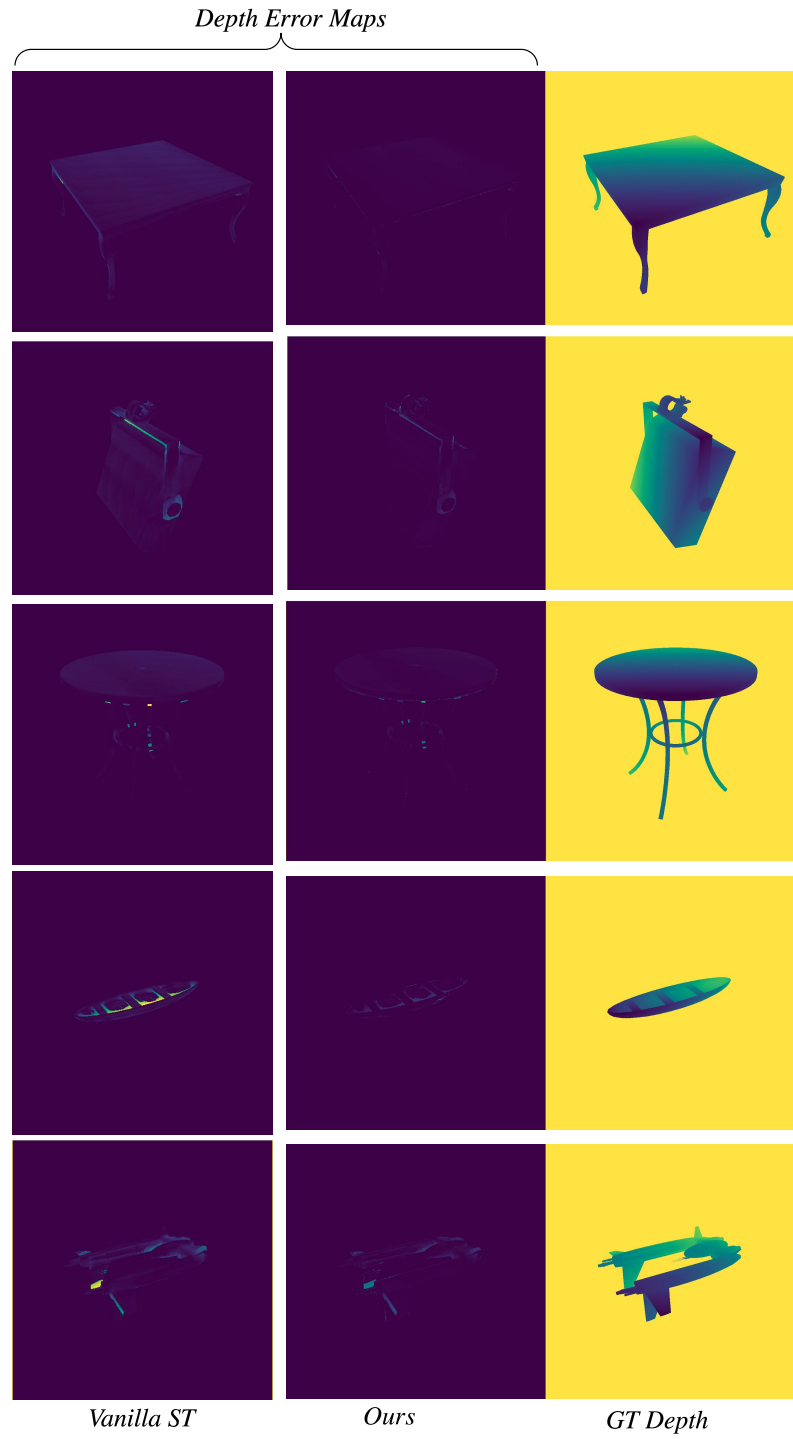


Figure 5: Additional results showing depth error maps.

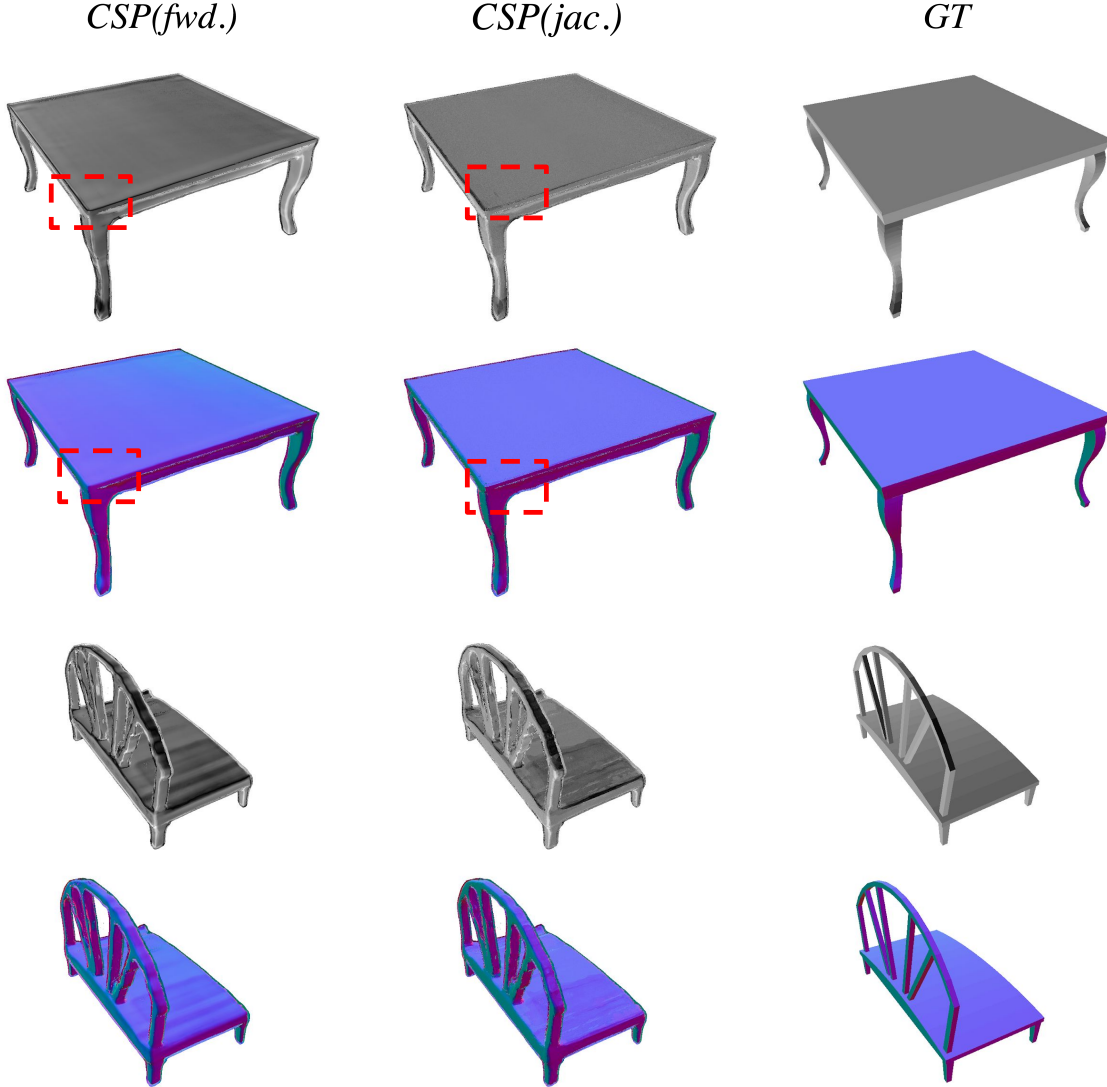


Figure 6: Surface reconstruction results on exemplar shapes from ShapeNet test set. Here, we show both *CSP (jac.)* and *CSP (fwd.)* ($\alpha = 0.005$) side-by-side, with the first row of each shape depicting a rendering of the sphere-traced surface normal map (shown in the second row) with directional light. We find that both methods (see Sec. 3.2 for a description of these methods, and Sec. 4.2 for some initial results reported in main paper) yield high-quality surface normals (with *CSP (fwd.)*) providing efficient forward-mode normal estimates. Note also that *CSP (jac.)* is marginally better in some regions (Highlighted in red.).

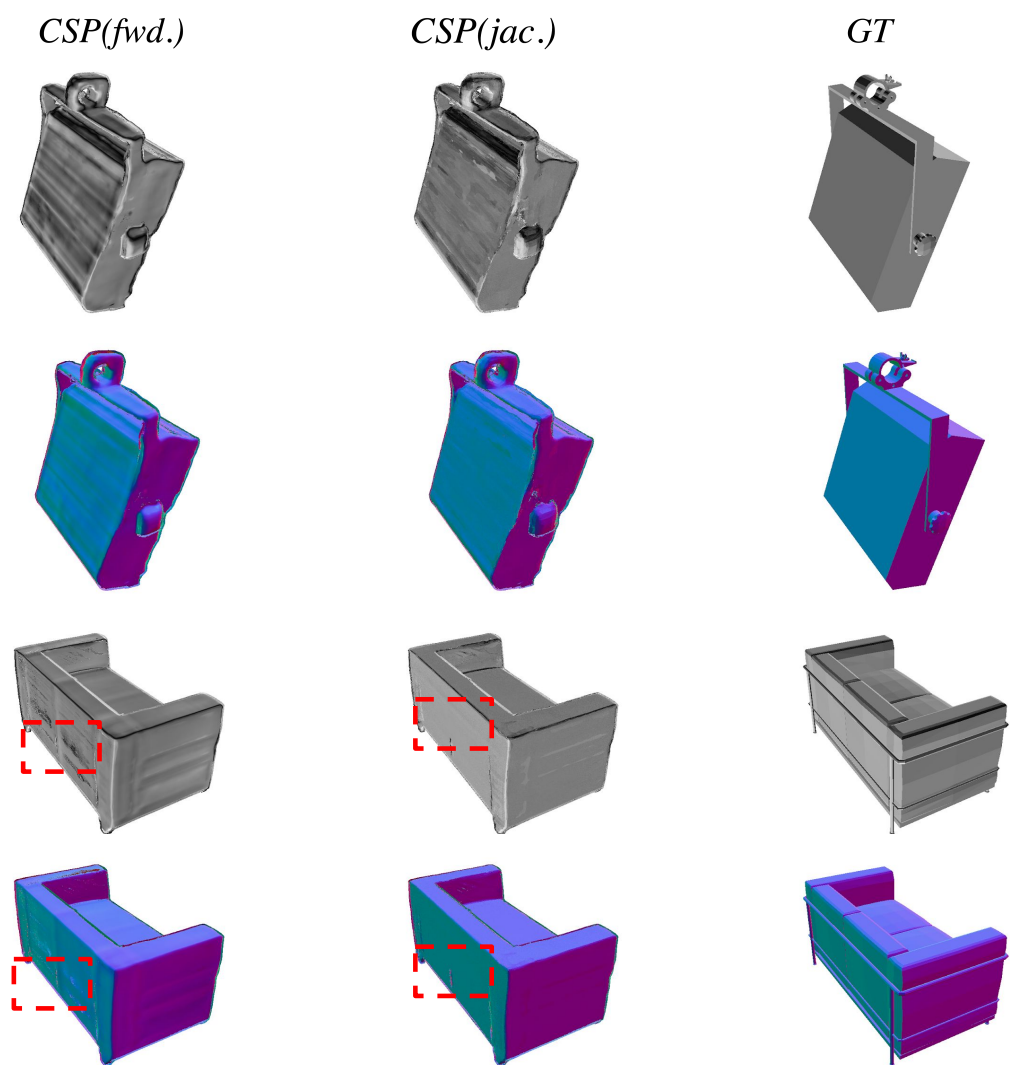


Figure 7: Additional surface reconstruction results from ShapeNet test set.

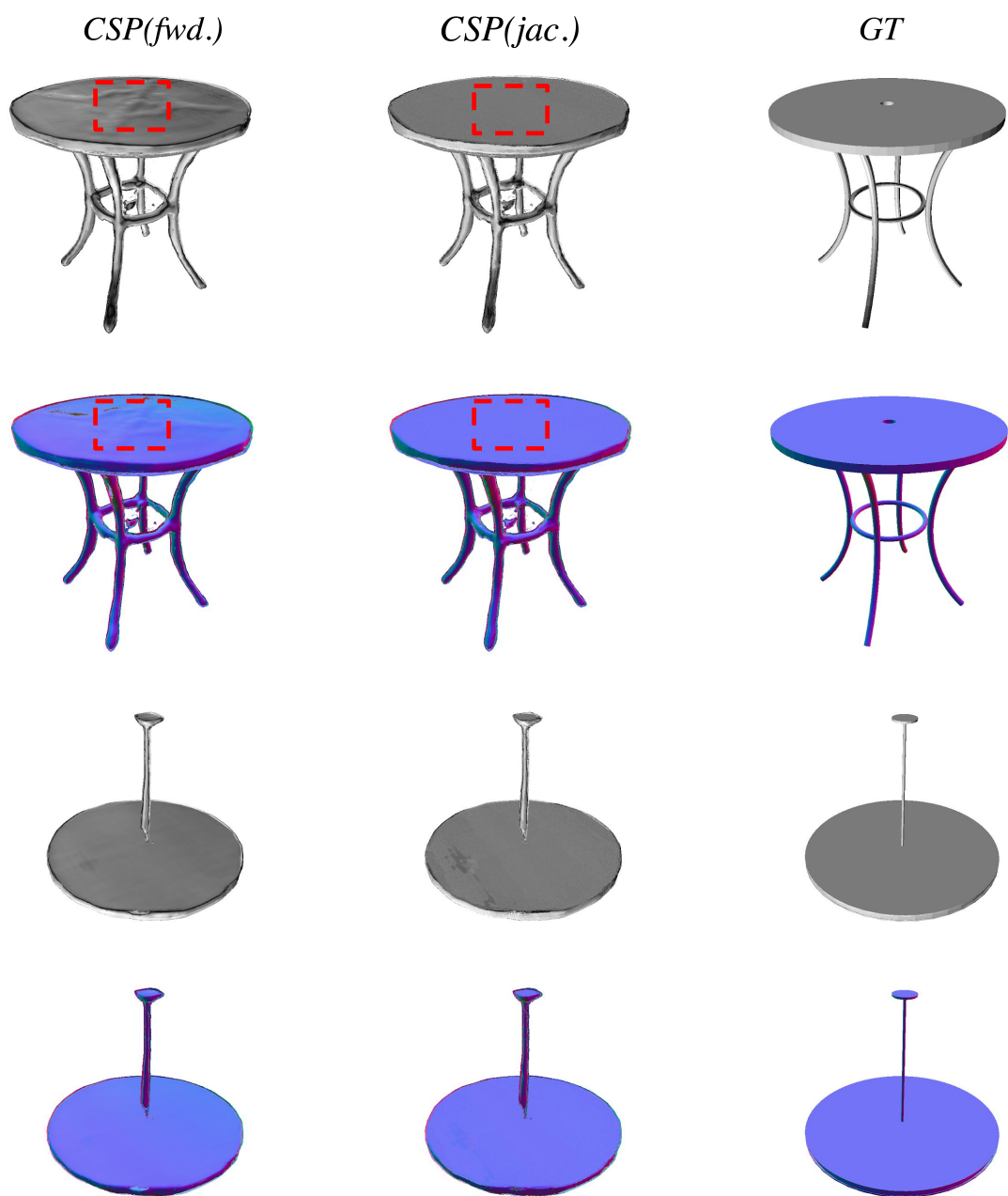


Figure 8: Additional surface reconstruction results from ShapeNet test set.

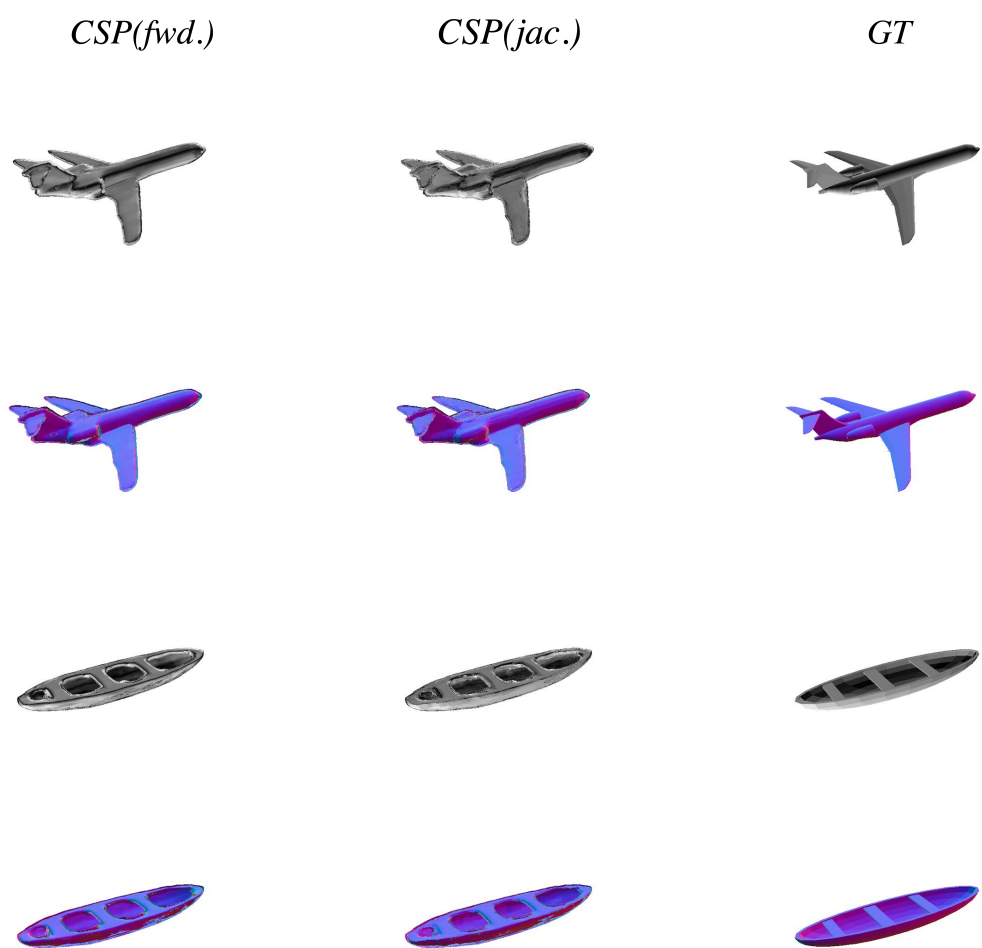


Figure 9: Additional surface reconstruction results from ShapeNet test set.

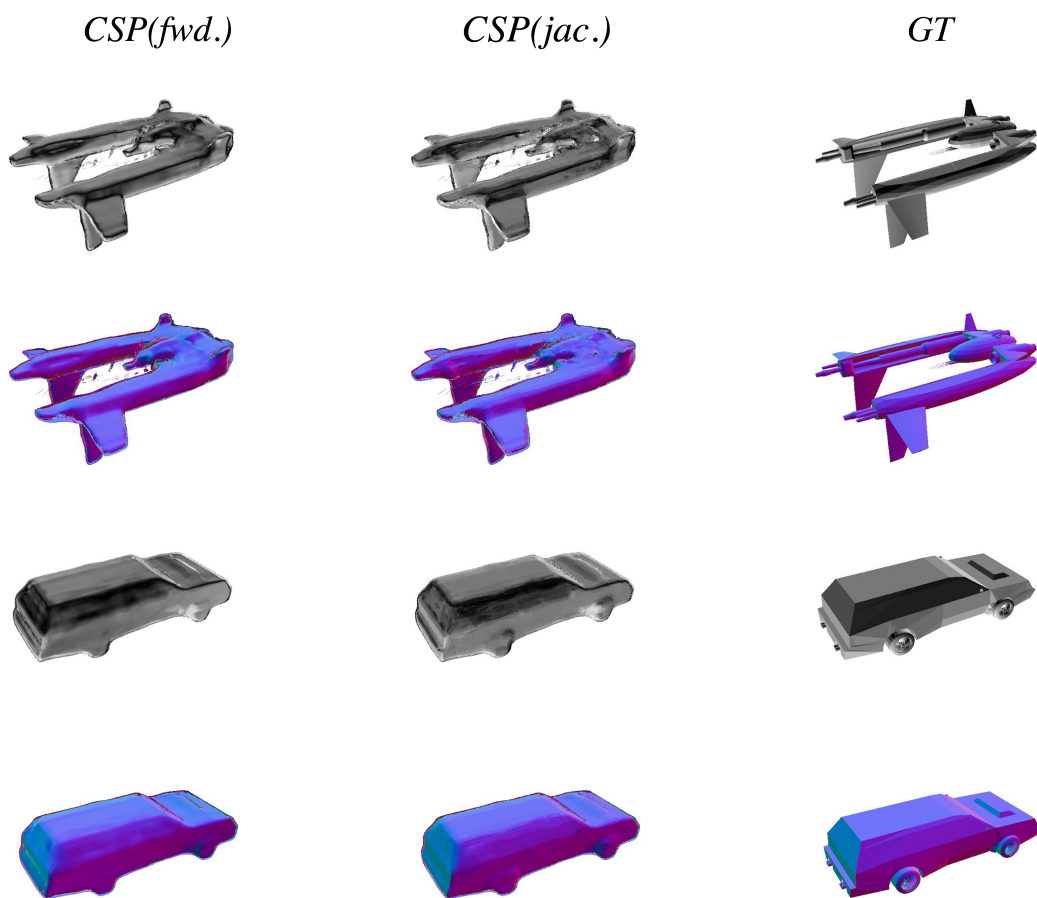


Figure 10: Additional surface reconstruction results from ShapeNet test set.

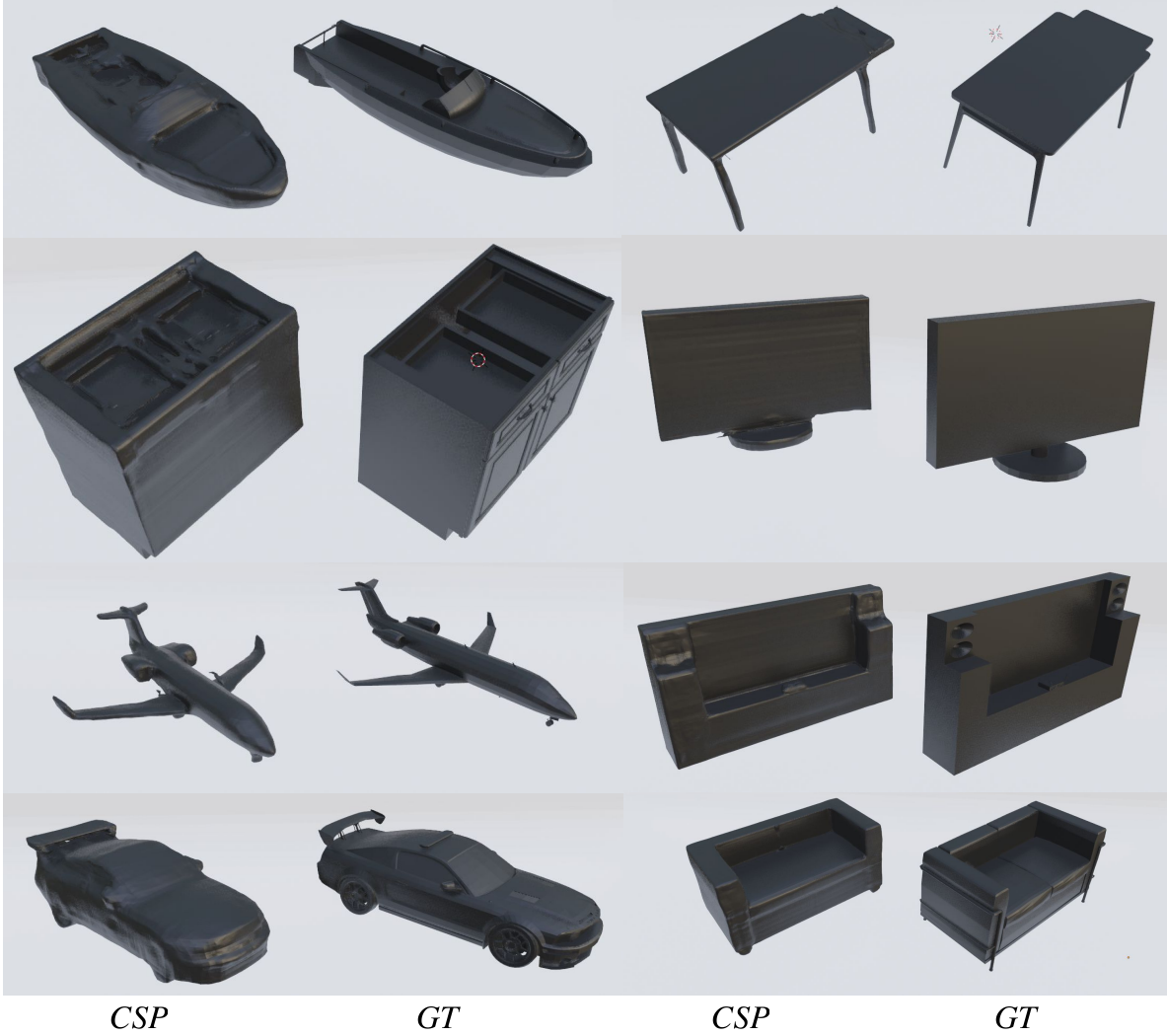


Figure 11: Meshes generated by our novel coarse-to-fine meshing algorithm for *CSPs* (see Sec. 3.3.2 of main paper). We also show the Ground Truth mesh on the right of each subfigure. Note that our algorithm generates structurally consistent meshes, which render visually pleasing images in Blender [4].

References

- [1] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999. 4
- [2] Julian Chibane, Mohamad Aymen mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21638–21652. Curran Associates, Inc., 2020. 2
- [3] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136. Salerno, Italy, 2008. 4
- [4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 12
- [5] Dawson-Haggerty et al. trimesh. 4
- [6] 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*, pages 605–613, 2017. 4
- [7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 4
- [8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2
- [9] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 523–540, Cham, 2020. Springer International Publishing. 2
- [10] David Stutz and Andreas Geiger. Learning 3D shape completion from laser scan data with weak supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1955–1964, 2018. 4
- [11] Jarke J Van Wijk and Arjeh M Cohen. Visualization of seifert surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):485–496, 2006. 1