SSRNet: Scalable 3D Surface Reconstruction Network

Zhenxing Mi † Yiming Luo † Wenbing Tao *
National Key Laboratory of Science and Technology on Multi-spectral Information Processing
School of Artifical Intelligence and Automation, Huazhong University of Science and Technology, China
{m201772503, yiming_luo, wenbingtao}@hust.edu.cn

1. Data Preparation

Our SSRNet is trained in a supervised fashion. The input training data include points and labeled vertices from the finest-level of an octree. We propose a method to label octree vertices with respect to a ground-truth surface. This method takes a point cloud and a ground-truth surface with surface normals as input. It first normalizes the coordinates of the points and surfaces according to the bounding box. Then an octree is built from the point cloud. The octree building method is adapted from the open-source code of Poisson Surface Reconstruction (PSR) [1]. It makes sure that the finest level of the result octree contains not only cubes with points in them, but also their $3 \times 3 \times 3$ neighbor cubes. Therefore, the octree is dense enough to completely contain the implicit surface. In the meantime, the vertices of the finest-level octree cubes are guaranteed to be close to the surface, which makes front-back labeling feasible.

In order to label finest-level octree vertices, we compute the intersections of octree edges and surface triangles. Since octree edges are all aligned with axes, we adapt an efficient method in [3]. Figure 1 shows a 2D example of our method for the sake of simplicity. The method consists of three steps.

Step 1: We first find the intersections of surface triangles and the grid lines of the octree. It is accomplished by projecting each triangle in the input surface to the XY, YZ, ZX planes. Since we propose an example in 2D, in Figure 1, the segment t represents a triangle in 3D space and the axis ox represents a coordinate plane in 3D space. As the Figure 1 (a) shows, after projection, we are able to detect that the two grid lines l_1 and l_2 intersect t.

Step 2: After detecting the intersections of grid lines and triangles, we are able to determine whether an octree edge intersects a triangles because each octree edge lies on a grid line. As the Figure 1 (b) shows, the two edges e_1 and e_2 intersects a triangle t. we label the two vertices of an intersected edge according to the triangle normal direction $\mathbf{n}_t$.

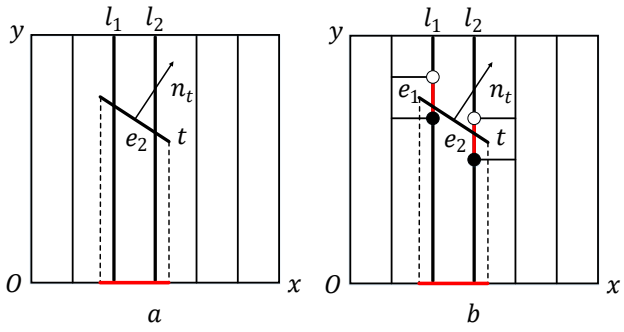

Figure 1. Example of our method for training data generation. (a) and (b) are 2D examples.

Let $\mathbf{n}_v$ be the vector from the intersection to the vertex v. In our front-back definition, if the angle between $\mathbf{n}_v$ and $\mathbf{n}_t$ is larger than $90°$, the vertex v is labeled as at back of the surface. Otherwise, it is labeled as in front of the surface. That is,

$$label_v = \begin{cases} 0 & \text{if } \mathbf{n}_v^\top \mathbf{n}_t \leq 0 \\ 1 & \text{if } \mathbf{n}_v^\top \mathbf{n}_t > 0 \end{cases} \quad (1)$$

In our computation, one edge may have multiple intersections with the surface and one vertex may belong to edges of different directions. Therefore, conflicts may occur on the label of one vertex. However, these situations rarely happen at a high resolution octree in practice. If these happen, we just choose one of the multiple labels.

Step 3: After Step 2, part of vertices near the surface are labeled. These vertices divide the region into two parts and serves as a decision boundary of front and back. In order to label other vertices, we just search their nearest neighbors in the labeled vertices. Then we give them the same label as the neighbors.

Our method for training data generation introduced above is for general purpose. In practice, we are lack of public available ground-truth surfaces for large-scale

†Equal contribution
*Corresponding author

datasets, so we use surfaces reconstructed by PSR to generate training data. If we have ground truth surfaces for large-scale datasets in the future, we can get more accurate training data directly from these surfaces using our training data generation method.

2. Surfaces on ShapeNet

We report more figures of the result surfaces reconstructed by our method on ShapeNet at octree depth 8. Figure 2, 3, 4, 5, 6, 7, 8 shows the reconstructed surfaces of our SSRNet and ONet [2].

3. Surfaces on DTU Testing Scans

We report more figures of the result surfaces reconstructed by our method on testing scans in DTU dataset at octree depth 9. Table 1, 2, 3, 4, 5, 6 show the detailed evaluation time. Figure 9, 10, 11, 12, 13, 14, 15 show the reconstructed surfaces of our SSRNet and PSR [1]. Surfaces reconstructed by PSR for comparison are reconstructed at octree depth 9 and trimmed at value 8 and 9.5.

References

[1] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):29, 2013. 1, 2

[2] 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*, pages 4460–4470, 2019. 2

[3] Claudio Rocchini, Paolo Cignoni, Fabio Ganovelli, Claudio Montani, Paolo Pingi, and Roberto Scopigno. Marching intersections: an efficient resampling algorithm for surface management. In *Proceedings International Conference on Shape Modeling and Applications*, pages 296–305. IEEE, 2001. 1

Table 1. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl001	stl002	stl003	stl004	stl005	stl006	stl007	stl008
Point /M	2.40	2.04	2.81	1.81	2.81	2.02	1.74	2.07
Vertex /M	0.958	0.753	1.64	0.883	1.24	0.829	0.855	0.978
Triangle /M	0.365	0.292	0.634	0.367	0.509	0.340	0.337	0.372
Batch	109	72	129	101	145	115	72	71
Time/s	stl001	stl002	stl003	stl004	stl005	stl006	stl007	stl008
Prep	25.466	20.491	34.875	20.555	33.021	22.844	19.361	22.201
Pred(1 GPU)	111.615	80.36	161.186	108.138	157.469	117.156	81.564	80.853
Pred(4 GPUs)	50.489	39.447	69.143	48.186	70.755	45.106	38.433	43.475
Total (1 GPU)	137.776	101.535	196.758	129.369	191.205	140.641	101.535	103.713
Total (4 GPUs)	76.650	60.622	104.715	69.417	104.491	68.591	58.404	66.335

Table 2. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl009	stl010	stl011	stl012	stl013	stl014	stl015	stl016
Point /M	2.35	2.04	2.22	1.29	2.61	2.94	2.55	3.02
Vertex /M	1.02	0.947	0.998	0.911	1.10	1.28	1.06	1.79
Triangle /M	0.444	0.368	0.440	0.361	0.431	0.564	0.469	0.775
Batch	97	82	94	39	86	144	132	96
Time/s	stl009	stl010	stl011	stl012	stl013	stl014	stl015	stl016
Prep	25.383	22.277	24.248	15.071	27.347	33.52	28.401	35.984
Pred(1 GPU)	113.115	90.115	114.557	51.906	101.602	164.933	141.376	131.807
Pred(4 GPUs)	52.864	43.256	63.047	28.020	49.640	69.795	60.871	57.434
Total (1 GPU)	139.253	113.163	139.432	67.675	129.59	199.169	170.491	168.566
Total (4 GPUs)	79.002	66.304	87.922	43.789	77.628	104.031	89.986	94.193

Table 3. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl018	stl019	stl020	stl021	stl024	stl029	stl035	stl038
Point /M	2.25	2.51	2.05	2.67	2.45	2.88	2.23	1.89
Vertex /M	1.08	1.17	0.993	1.31	0.877	1.14	0.901	0.787
Triangle /M	0.478	0.496	0.421	0.608	0.353	0.477	0.356	0.320
Batch	81	104	93	122	129	137	99	80
Time/s	stl018	stl019	stl020	stl021	stl024	stl029	stl035	stl038
Prep	24.767	28.693	22.966	30.732	26.443	31.451	24.234	20.607
Pred(1 GPU)	95.375	117.374	98.553	134.578	134.27	143.643	24.234	20.607
Pred(4 GPUs)	46.279	54.376	46.524	63.499	59.591	63.400	45.740	39.425
Total (1 GPU)	120.839	146.781	122.175	166.026	161.356	175.89	133.429	102.232
Total (4 GPUs)	71.743	83.783	70.146	94.947	86.677	95.647	70.671	60.765

Table 4. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl041	stl042	stl045	stl046	stl048	stl051	stl055	stl059
Point /M	1.70	1.97	1.81	2.17	1.88	1.75	2.72	2.11
Vertex /M	0.732	0.940	0.710	0.749	0.927	0.806	1.54	0.813
Triangle /M	0.287	0.374	0.293	0.322	0.368	0.314	0.612	0.328
Batch	57	65	75	105	58	58	93	72
Time/s	stl041	stl042	stl045	stl046	stl048	stl051	stl055	stl059
Prep	17.328	21.279	19.268	22.523	19.89	18.781	31.513	21.433
Pred(1 GPU)	17.328	21.279	19.268	22.523	19.890	18.781	31.513	21.433
Pred(4 GPUs)	32.284	40.075	38.097	51.272	33.651	33.520	56.488	40.812
Total (1 GPU)	78.452	96.718	102.634	136.631	91.202	89.168	149.613	109.549
Total (4 GPUs)	50.233	62.052	58.124	74.417	54.209	52.979	88.667	62.978

Table 5. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl060	stl061	stl063	stl065	stl069	stl084	stl093	stl094
Point /M	2.63	1.96	2.13	2.86	2.67	2.30	2.79	2.55
Vertex /M	1.13	0.794	0.886	1.67	1.41	1.06	1.14	1.02
Triangle /M	0.463	0.328	0.349	0.704	0.556	0.435	0.450	0.402
Batch	85	95	59	82	119	94	97	110
Time/s	stl060	stl061	stl063	stl065	stl069	stl084	stl093	stl094
Prep	27.597	21.052	21.553	33.728	31.514	25.254	29.804	26.563
Pred(1 GPU)	100.657	99.334	76.773	111.888	131.461	112.381	111.470	120.886
Pred(4 GPUs)	47.912	47.116	45.776	52.662	66.661	51.706	52.863	55.888
Total (1 GPU)	128.887	121.125	99.081	146.356	163.741	138.407	141.946	148.248
Total (4 GPUs)	76.142	68.907	68.084	87.130	98.941	77.732	83.339	83.250

Table 6. The time performance of our method. We report the first scale point number, first scale octree vertex number and triangle number in our results using million (M) as unit. The preprocessing time (Prep. time) includes octree construction time, downsampling time, tangent images precomputing time and batches computing time. The prediction time (Pred. time) is the time to load partitioned points and vertices and to predict labels of vertices using network.

Number	stl095	stl097	stl106	stl110	stl114	stl122	stl126	stl127
Point /M	2.75	2.60	3.34	2.12	2.96	2.60	4.05	4.21
Vertex /M	1.30	1.04	1.92	1.30	1.55	1.70	2.13	1.54
Triangle /M	0.517	0.424	0.775	0.513	0.588	0.702	0.836	0.601
Batch	98	92	116	69	135	114	118	133
Time/s	stl095	stl097	stl106	stl110	stl114	stl122	stl126	stl127
Prep	30.263	27.236	38.975	25.435	35.499	32.512	47.193	42.994
Pred(1 GPU)	122.669	109.600	145.663	91.904	168.743	147.693	169.057	168.511
Pred(4 GPUs)	53.797	52.231	68.101	58.003	67.392	61.119	85.317	75.161
Total (1 GPU)	153.57	137.594	185.458	117.974	205.004	180.949	217.141	212.206
Total (4 GPUs)	84.698	80.225	107.896	84.073	103.653	94.375	133.401	118.856

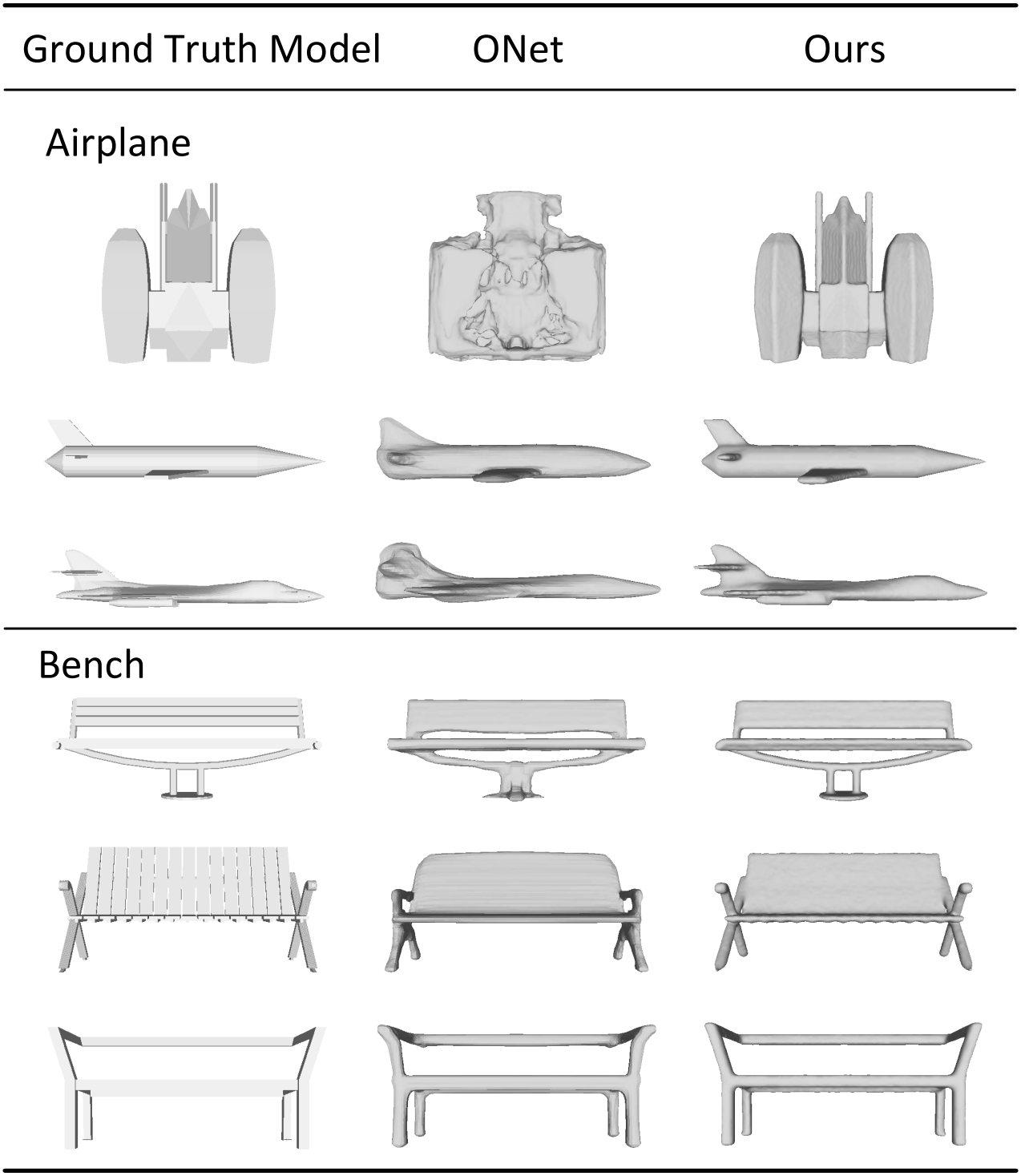

Figure 2. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

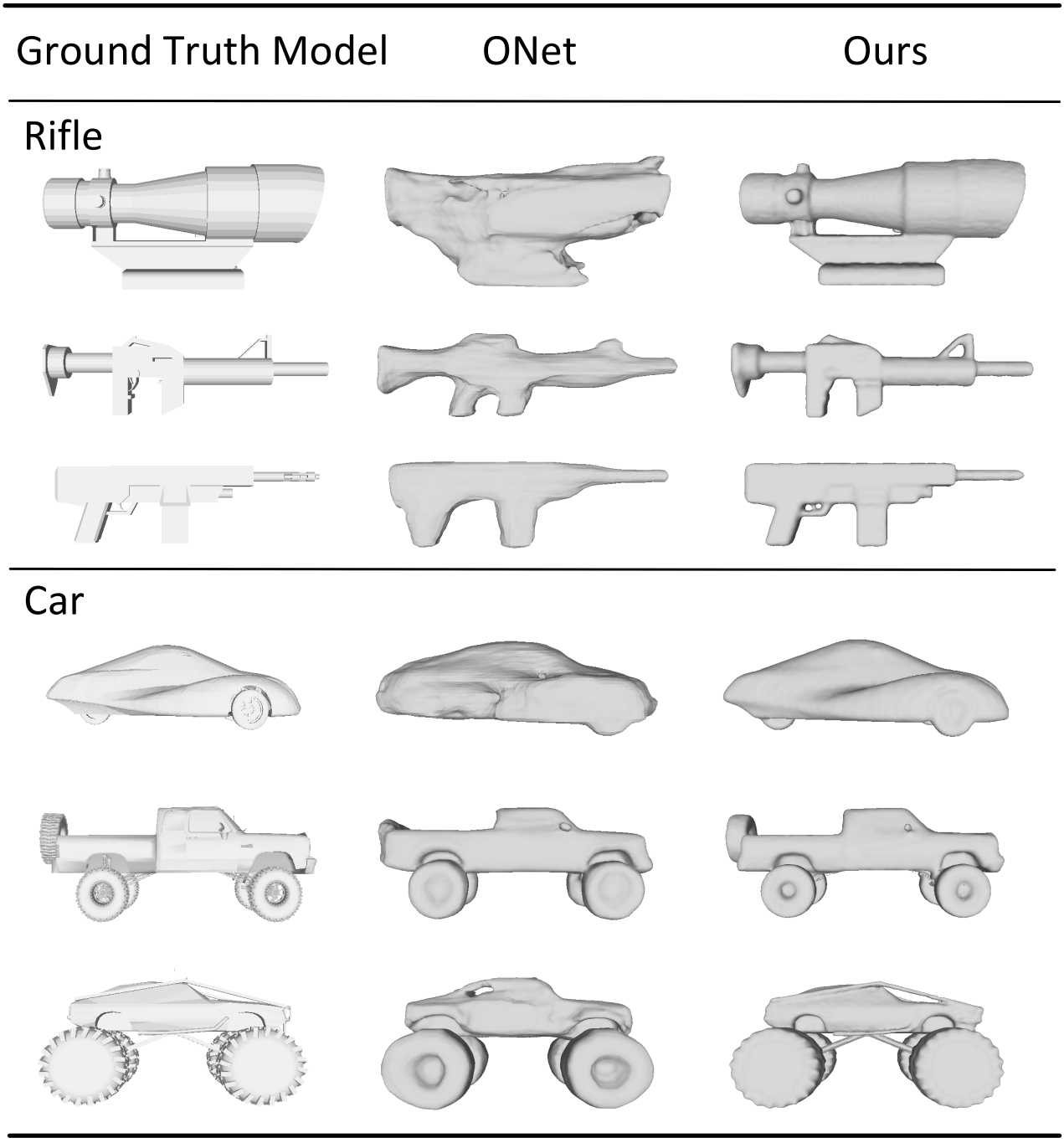

Figure 3. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

Figure 4. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

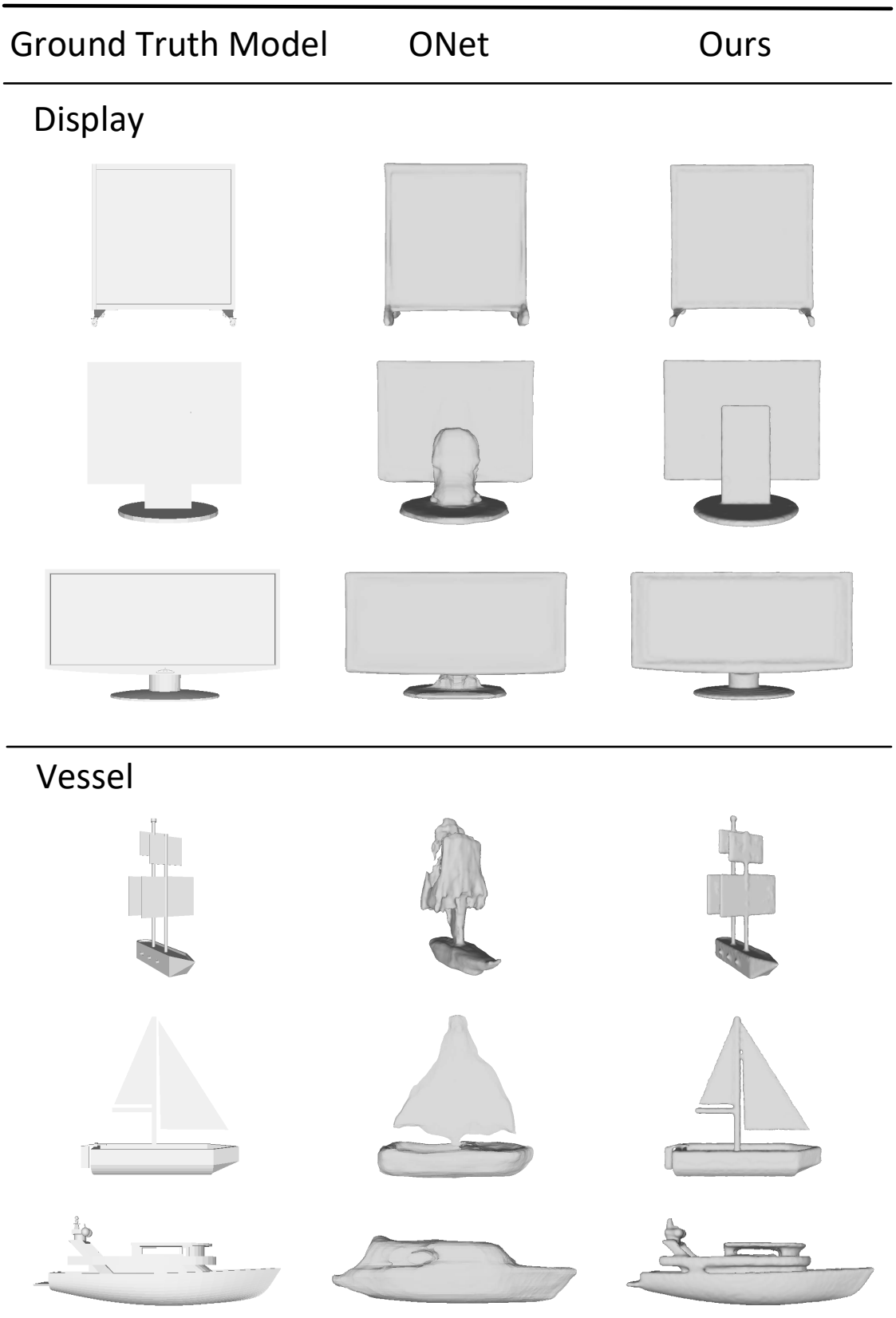

Figure 5. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

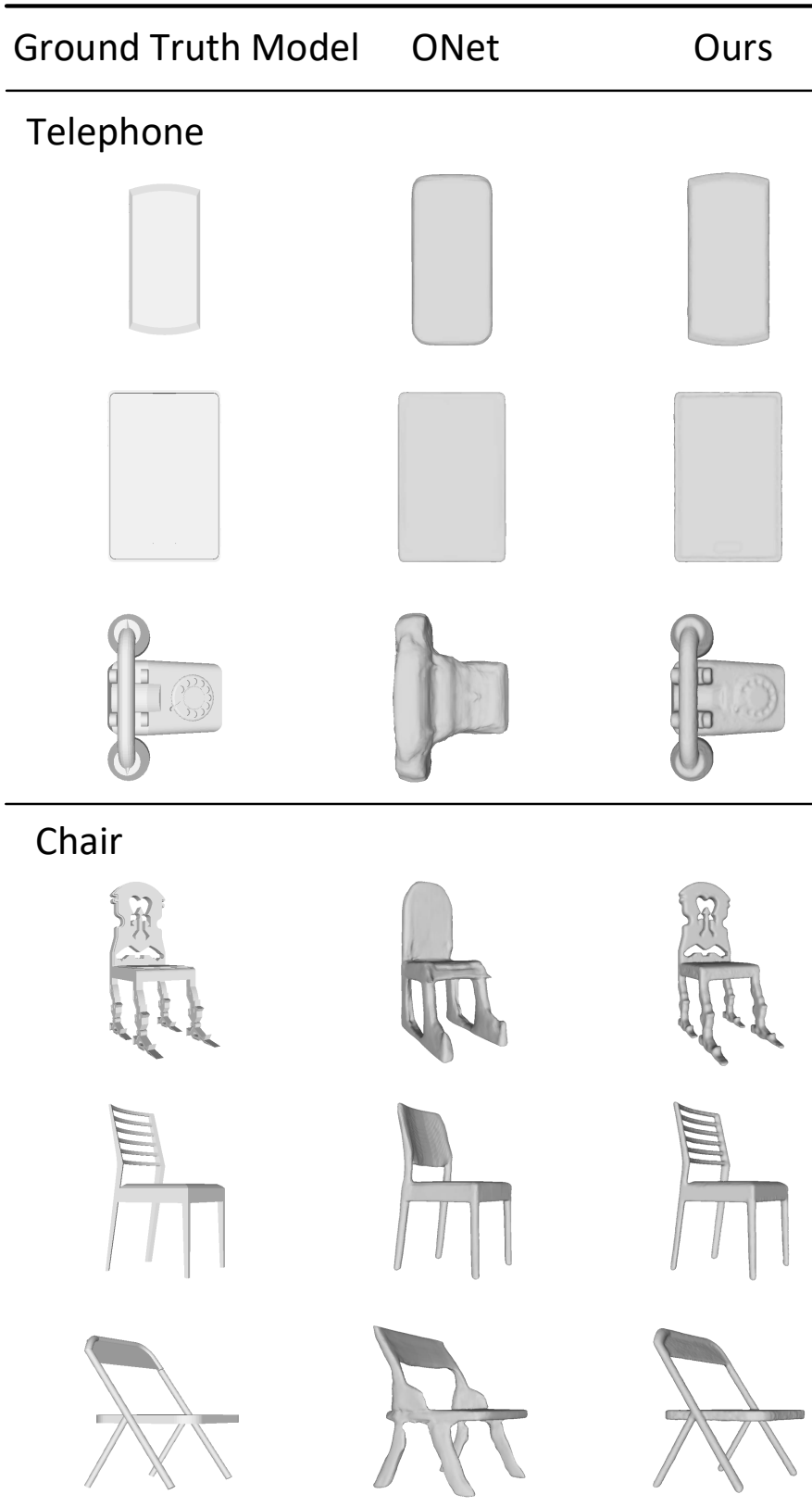

Figure 6. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

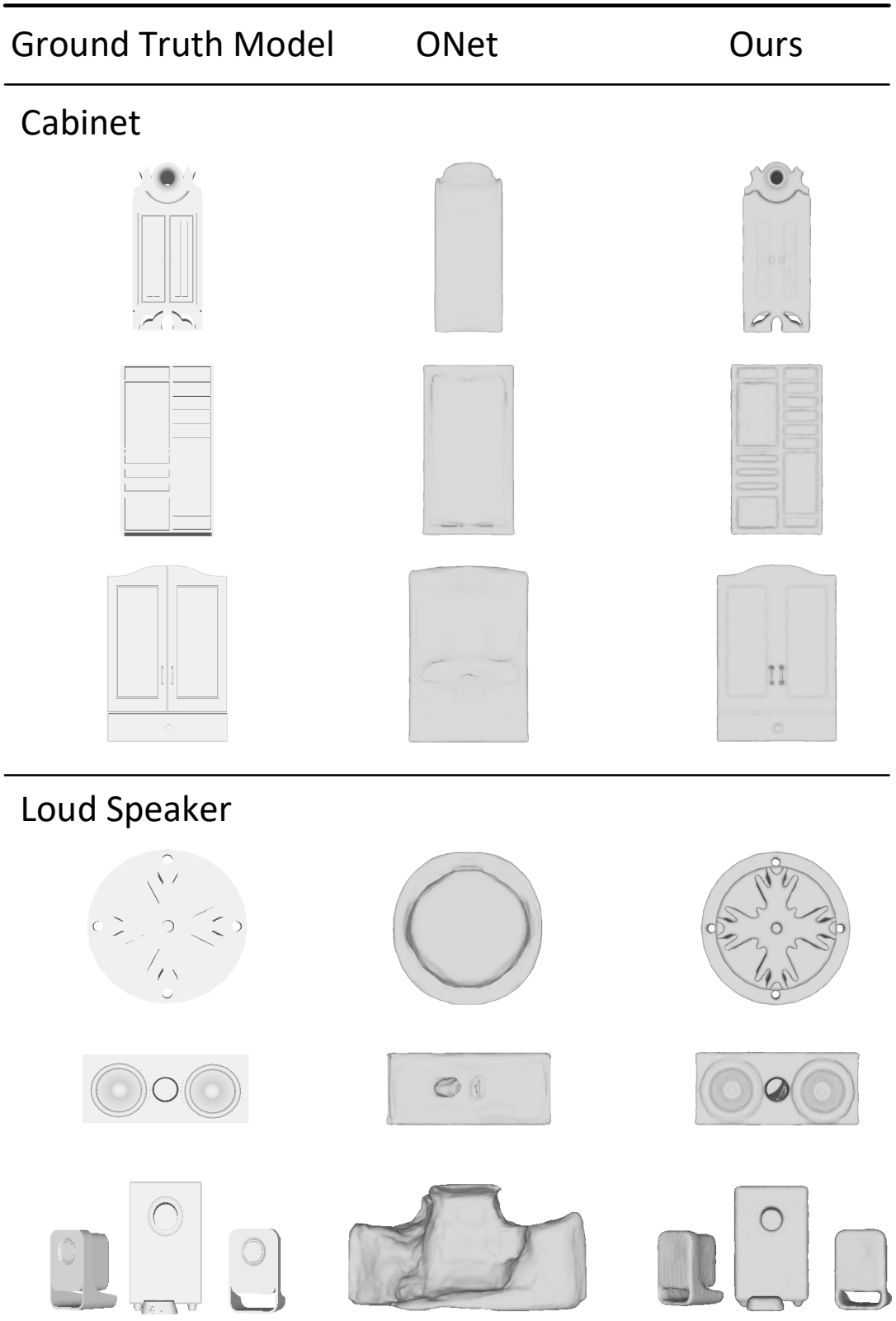

Figure 7. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

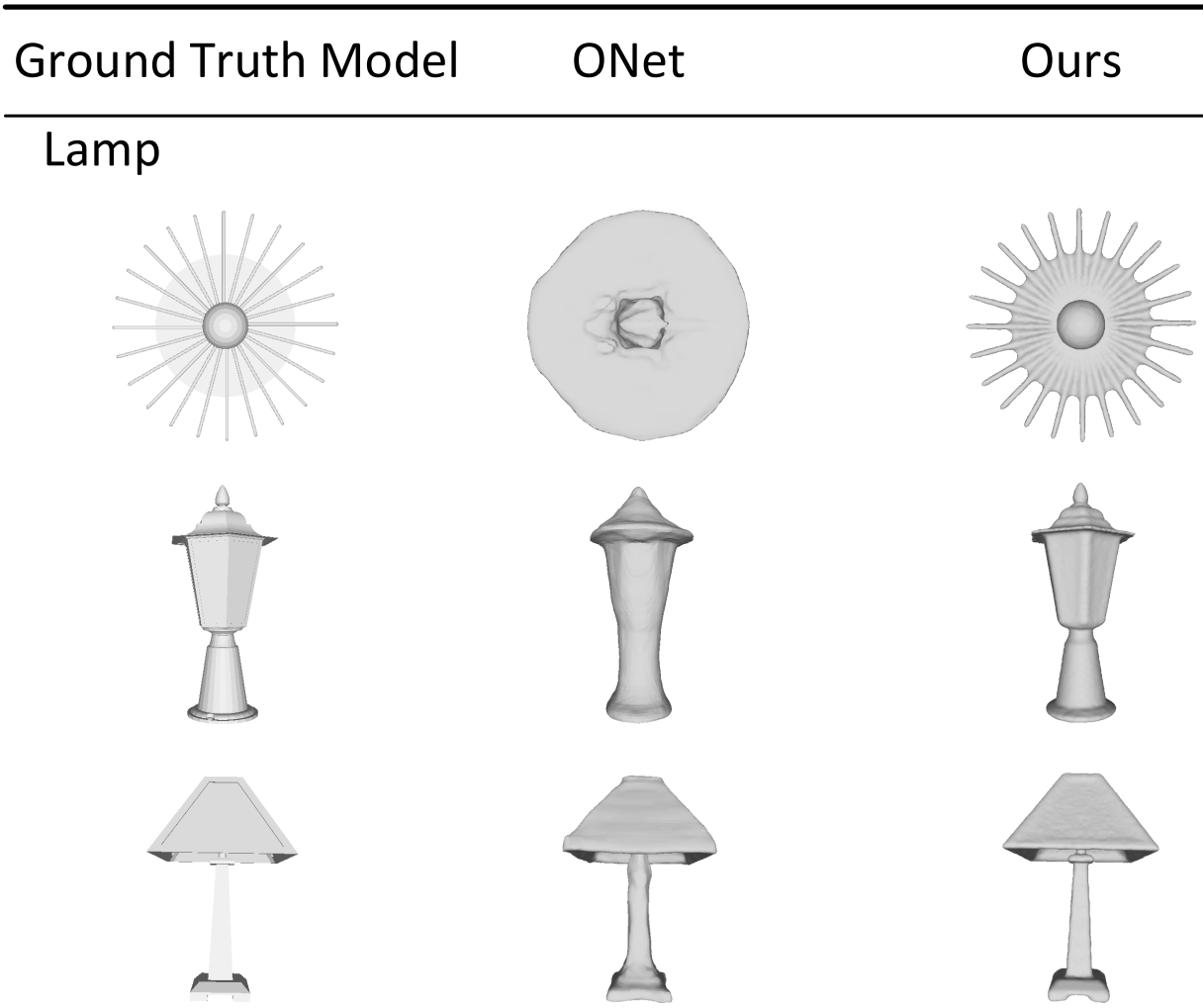

Figure 8. Reconstructed surfaces of our network and ONet on ShapeNet testing set.

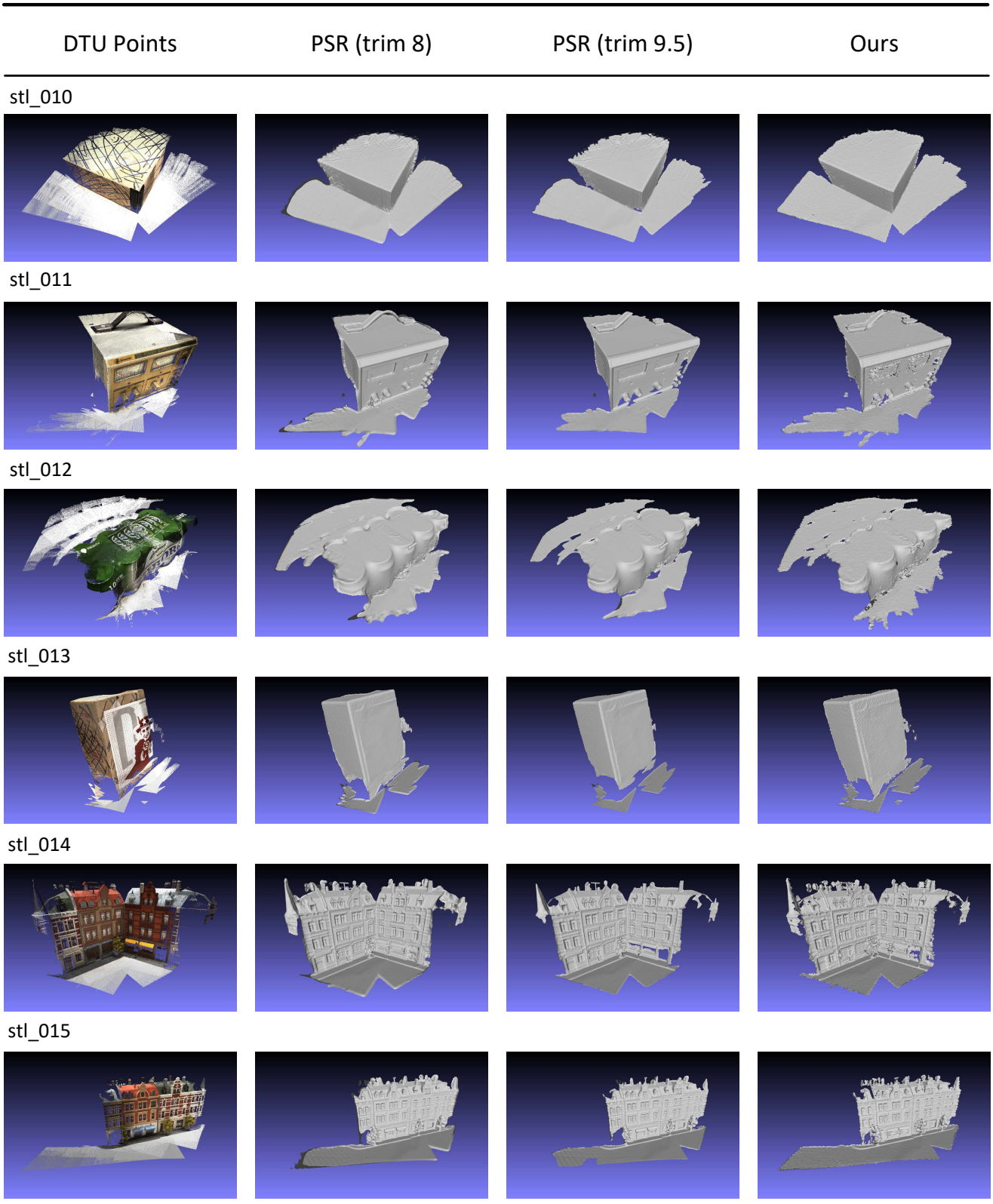

Figure 9. Reconstructed surfaces of our network and PSR on DTU testing scenes.

Figure 10. Reconstructed surfaces of our network and PSR on DTU testing scenes.

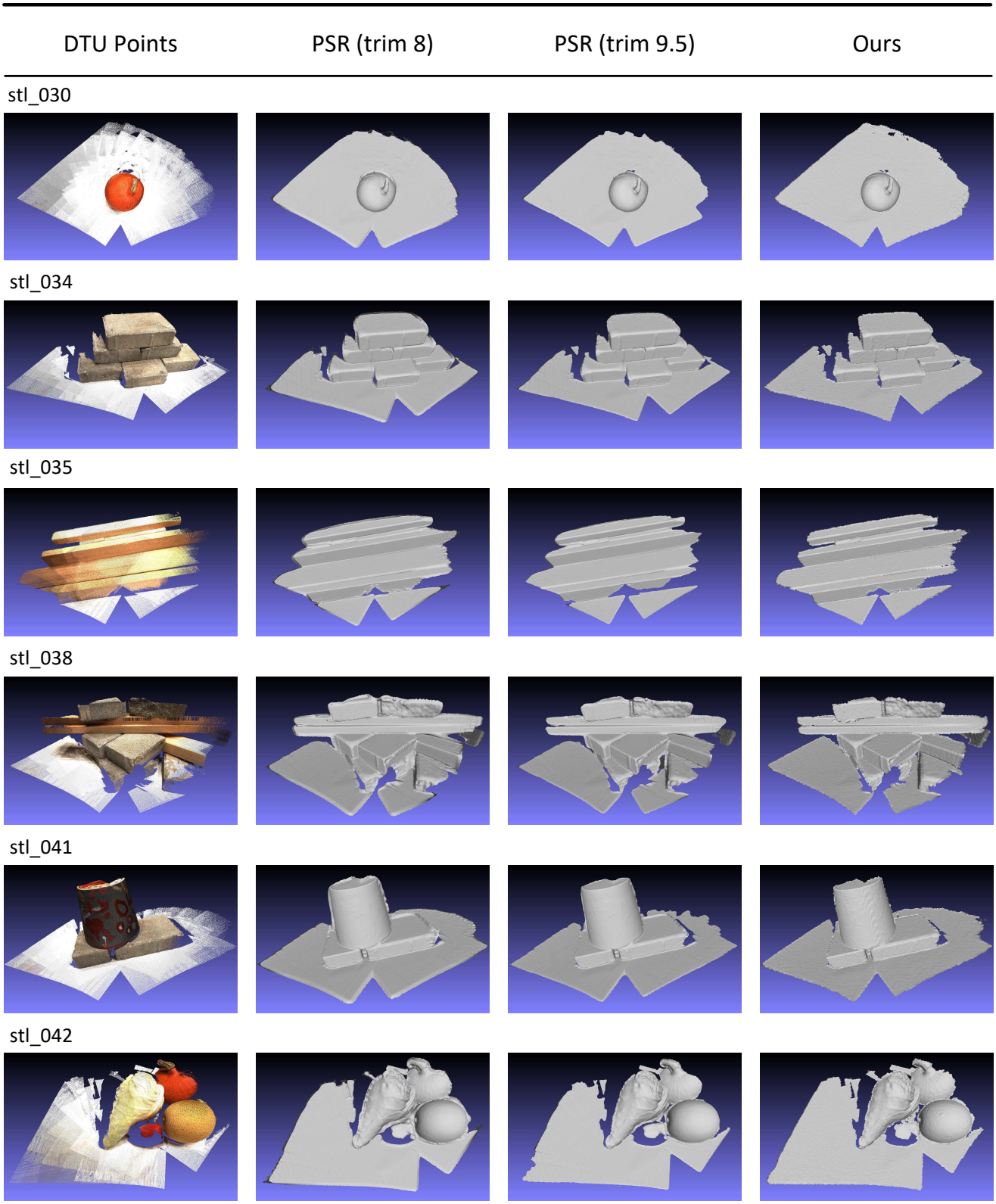

Figure 11. Reconstructed surfaces of our network and PSR on DTU testing scenes.

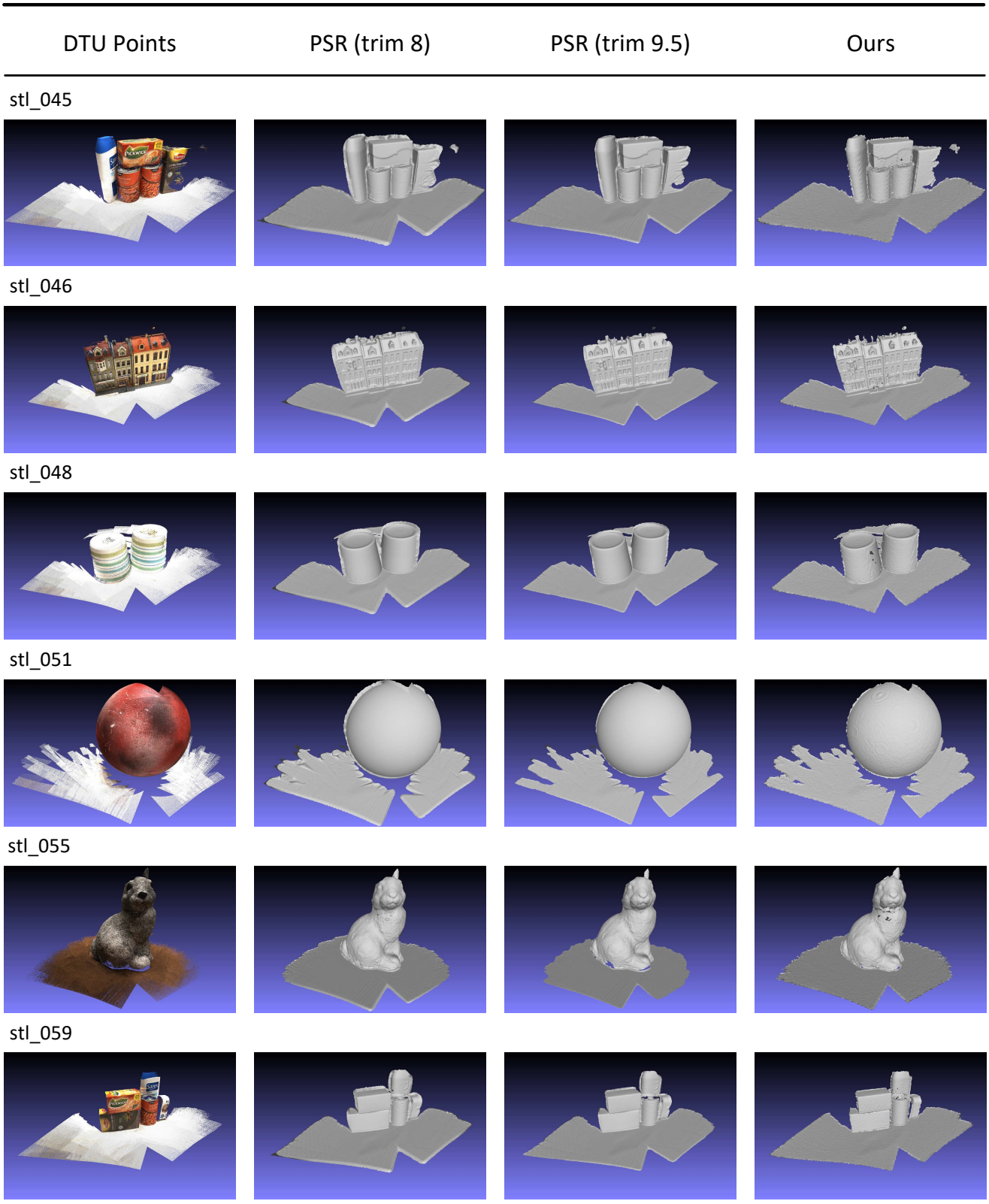

Figure 12. Reconstructed surfaces of our network and PSR on DTU testing scenes.

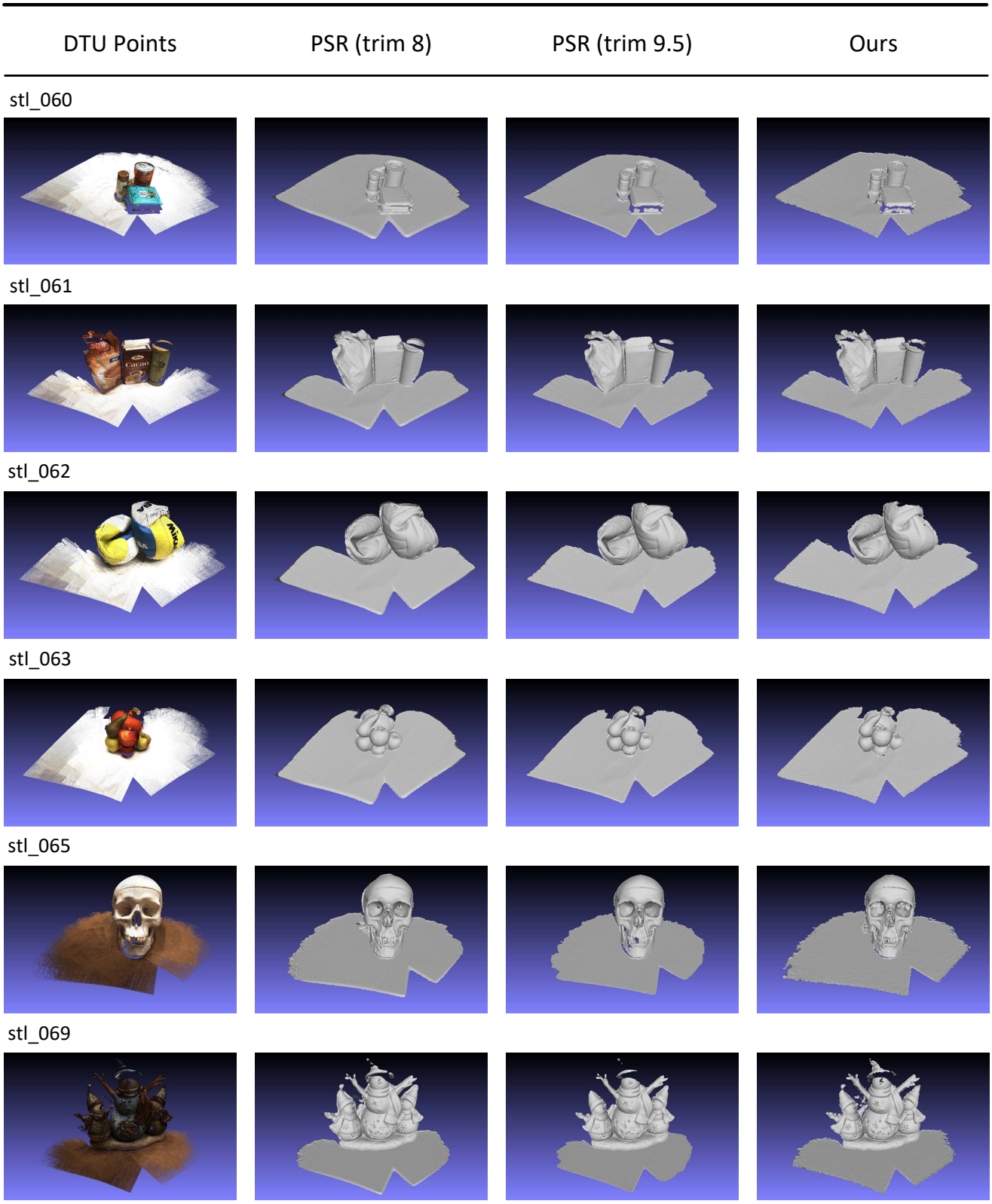

Figure 13. Reconstructed surfaces of our network and PSR on DTU testing scenes.

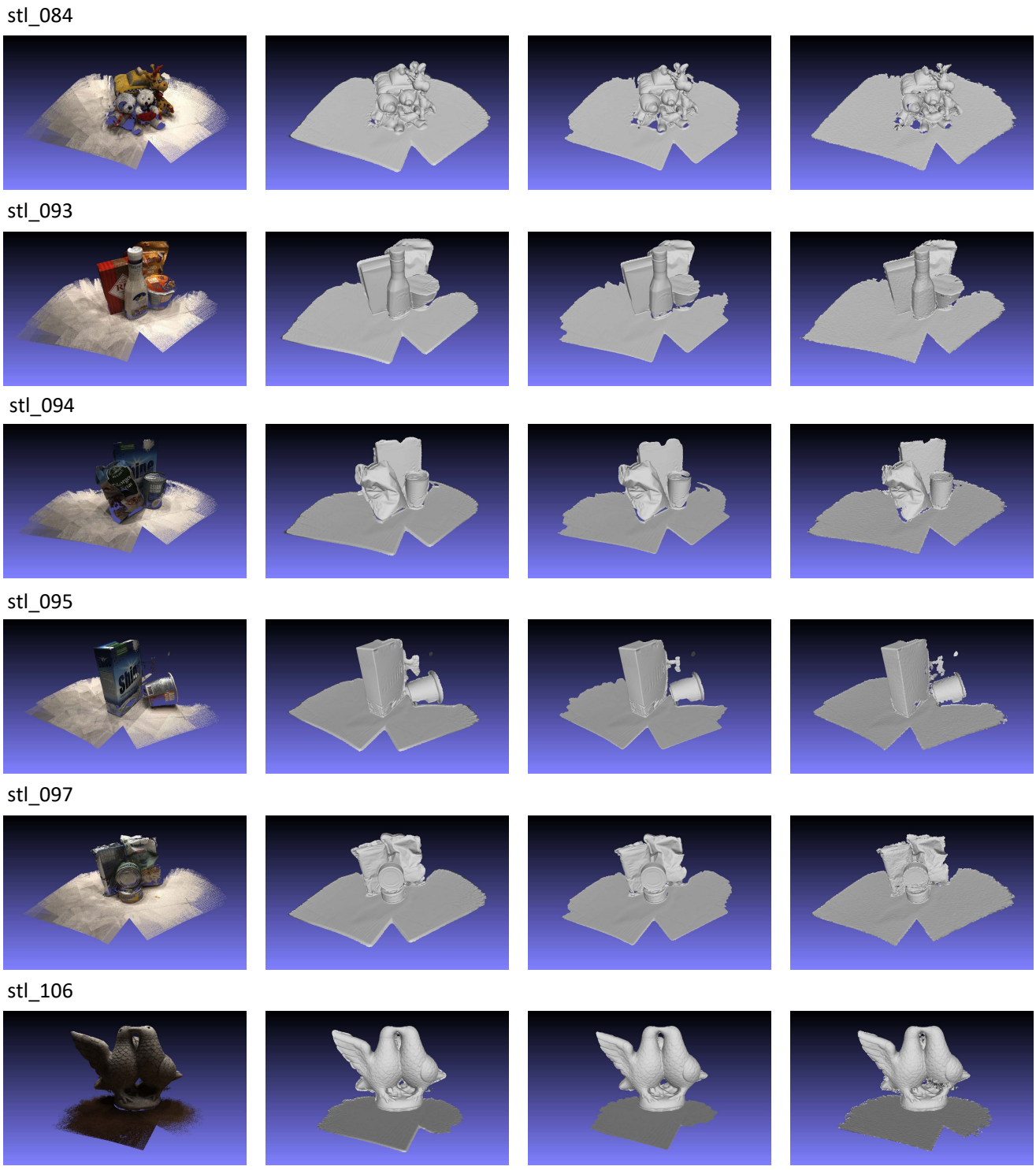

Figure 14. Reconstructed surfaces of our network and PSR on DTU testing scenes.

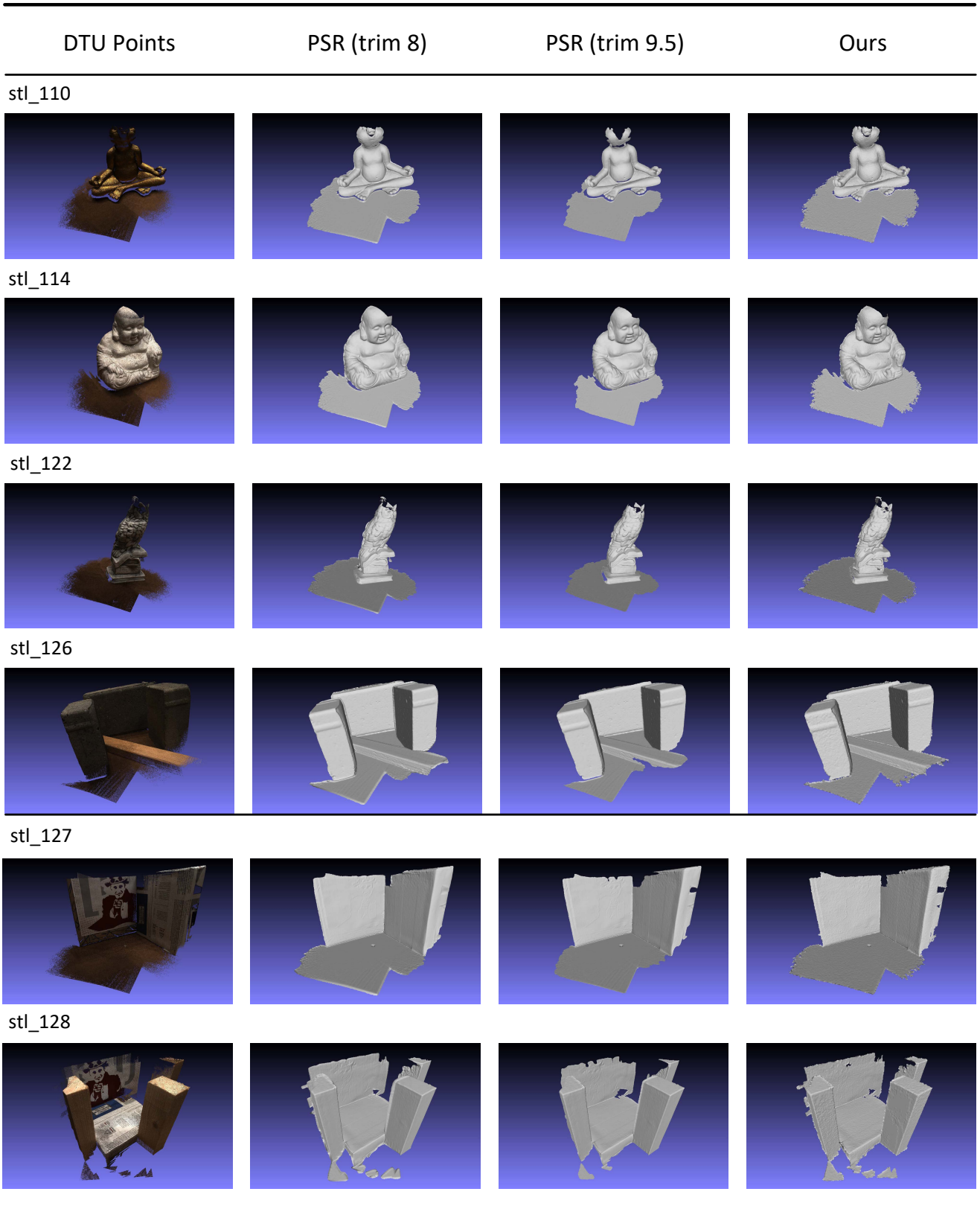

Figure 15. Reconstructed surfaces of our network and PSR on DTU testing scenes.