DR-KFS: A Differentiable Visual Similarity Metric for 3D Shape Reconstruction

Jiongchao Jin[1], Akshay Gadi Patil[2], Zhang Xiong[1], and Hao Zhang[2]

[1] Beihang University
[2] Simon Fraser University

1 More Results

More evidence on the effectiveness of DR-KFS over pixel-wise MSE loss. In Figure1, we present additional qualitative comparisons of reconstruction using "visual similarity loss vs. image distortion loss" and reconstruction from "silhouettes vs. shaded images". Quantitative comparison are shown in Table 1. Figure 1 and Table 1 underscore our claim that shaded images are friendlier for the task of reconstruction and that using DR-KFS pipeline results in better reconstruction. This is also shown in the main paper (Figure 5 - Table 4).

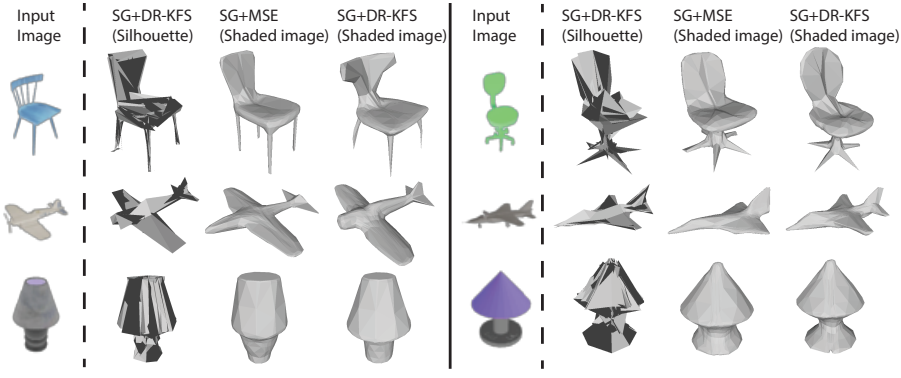

Fig. 1. Additional reconstructed models when the Shape Generator (SG) used in SoftRas[4] is trained using silhouettes vs. shaded images corresponding to an input RGB image. See Table 1 for the *average* similarity scores for the shapes shown above

Features capturing perceptual differences. In Figure 2 and 3, we present more results for lamps and chairs respectively, throwing light on the kind of image features that can efficiently capture distinguishable perceptual differences on the 2D projections of a 3D shape. A desired property of features making up a visual similarity metric is that they should be able to *robustly* capture the perceptual differences. Investigating this for the lamp shown in Figure 3, the stem manipulation operation (fourth column) is visually more apparent than the stem

	LFD	Shape Google
SG+DR-KFS (Silhouettes)	6468	7.66
SG+MSE (Shaded Images)	5089	6.35
SG+DR-KFS (Shaded Images)	**4036**	**4.83**

Table 1. More results on training a simple Shape Generator (SG) adopted in SoftRas [4] with both DR-KFS and per-pixel MSE loss. DR-KFS, which performs image feature matching, essentially incorporates visual similarity, while per-pixel MSE is merely an image distortion loss on the image pixels. The reconstructed 3D models are shown in Figure 1.

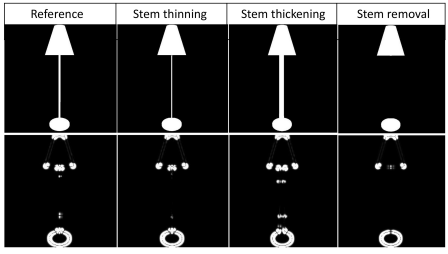

Fig. 2. Manipulating the thickness of the stem between the head and the bottom of the lamp, including its deletion: Binary images are shown on top and their corresponding PoI maps are on the bottom. Image similarity scores w.r.t the reference image for each operation using different image-level features are tabulated in Table 2.

Loss	Thin stem	thick stem	Removal
MSE (I)	0.00731	*0.0140*	0.0089
MSE (LIFT feats)	0.0591	*0.0899*	0.0892
MSE (PoI Maps)	0.0014	*0.0051*	0.0048
DR-KFS	0.00438	0.0128	*0.015*

Table 2. Image similarity scores for shape manipulation operations shown in Figure 2 (in the same order), using MSE loss, on four different image-level features: raw image pixels, image features using LIFT descriptors [10], PoI maps and DR-KFS feature descriptors.

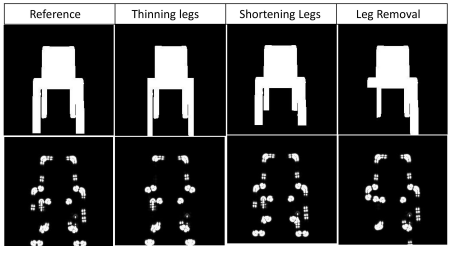

Fig. 3. Manipulations on the chair legs, including leg removal: Binary images are shown on top and their corresponding PoI maps are on the bottom. Image similarity scores w.r.t the reference image for each operation using different image-level features are tabulated in Table 3.

Loss	Thin legs	Short legs	Removal
MSE (I)	*0.371*	0.322	0.185
MSE (LIFT feats)	0.132	0.179	*0.183*
MSE (PoI Maps)	0.0129	*0.0489*	0.0144
DR-KFS	0.0171	0.0349	*0.0551*

Table 3. Image similarity scores for the image-level operations shown in Fig 3, using: MSE loss on image pixels (I), image features using LIFT descriptors[10], MSE loss on PoI map pixels, and DR-KFS framework. Italicized numbers (row-wise) indicate sensitivity to the respective image-level operation.

thinning and thickening operations (second and third columns, respectively). If we look at the rows of Table 2, we observe that only DR-KFS features produce the largest similarity scores (the smaller the score, the more similar the two images are; so a larger score indicates that the there is significant perceptual difference between the two images) for this stem-deletion operation, as indicated by the italicized numbers. This tells us that our pipeline is more efficient in capturing perceptual object level manipulations (2D projection case here) than incorporating pixel-wise MSE loss.

In addition, for the chair shown in Figure 3, the leg deletion operation (fourth column) is more visually apparent than the other leg operations (thinning and shortening, second and third columns, respectively). DR-KFS and LIFT features capture this large perceptual difference (see Table 3, second and fourth rows) over the other operations, while pixel-wise MSE is sensitive to manipulations that are less visually apparent. It is to be noted that LIFT features may capture the perceptual differences well (certainly not as efficiently as DR-KFS, as seen in Figure 2-Table 2), but employing them into a pipeline like ours would make the entire approach non-differentiable.

Additional reconstructed samples More results of reconstructions obtained using AtlasNet [2], Matryoshka Network [6], Pixel2Mesh [9] and 3D-R2N2 [1] are shown in Figure 4.

2 Perceptual Studies

For all the Perceptual Studies (PS) pointed out to in the main paper, we detail the experimental settings, including the model pairs used in PS-1,PS-2. Figure 8,9,10, 11 and 12 show the reconstructed models used in PS-1 and Figure 13,14,15,16 and 17 show the reconstructed models used in PS-2. For details on PS-1, PS-2, please refer to the main paper (Section 4.2).

User Interface In Figure 7, we show the user interface (UI) employed in PS-1 and PS-2. Turkers on AMT are forced to choose the best reconstructed model (A or B) for the given input image.

3 Components of DR-KFS

3.1 Differentiable Renderer, *SoftRas*

In DR-KFS, we replace discrete rasterization and z-buffering with the soft-rasterization and aggregation function introduced in *SoftRas* [4]. The input to this differentiable renderer (*softras*) is a scaled (vertices scaled to [-1,1]) and sphere-centered 3D model (sphere radius 5). Rendering viewpoints are obtained by placing a virtual camera on the sphere. The elevation angles at which the camera is placed are $\{0, \frac{pi}{4}, \frac{pi}{2}, \frac{3\pi}{2}, \frac{7\pi}{4}\}$, the azimuth angles are $\{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi\}$ and the direction of the light source is $[0, 1, 0]$. With this camera setup, we use a

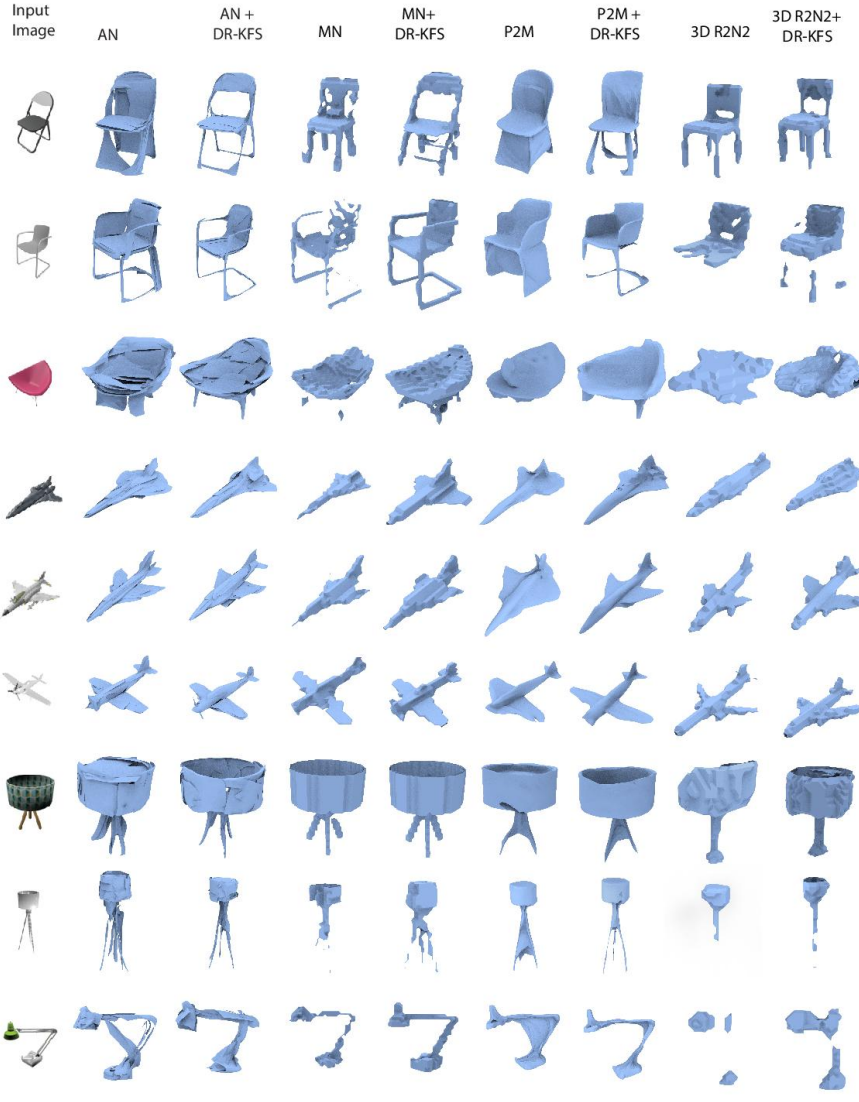

Fig. 4. An extended gallery (from the main paper) of results reconstructed when the networks are trained using the original loss and the DRKFD loss. Reconstructed results from representative networks such as AtlasNet (AN) [2], Matryoshka Net (MN) [6], Pixel2Mesh (PM) [9] and 3D-R2N2 [1] are shown in this gallery of additional results.

differentiable renderer as shown in Figure 5 to get the corresponding view images. In doing so, we first take the model mesh **M**, a light source **L** and the camera pose **P** as inputs, using which we calculate mesh normal **N**, image-coordinate **U** and the view depth **Z** via mesh transformation Next, surface information **S** is computed by the mesh normal **N** and the light input **L**. We then obtain a probability map **D** for each mesh from the input **U**, through *Soft Rasterization*. Finally, we aggregate all the probability maps together with surface information **S** using an aggregation function to get the final rendered image **I**.

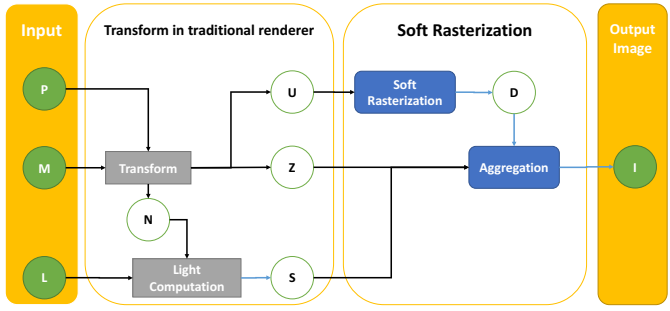

Fig. 5. Pipeline of the differentiable renderer, *SoftRas*, used in DR-KFS

3.2 Keypoint detector and PoI maps

We use LIFT [10] and TILDE [8] to obtain PoI maps for every view-image of the 3D model. A simplified pipeline of LIFT detector is show in Figure 6. LIFT detector consists of a convolution layer together with a Generalized Hinging Hyperplane (GHH) layer that implements piecewise linear activation function, and a softargmax layer that extracts keypoints from the input view-image. In GHH feature layer, we calculate the score map by:

$$PoI_{map} = f_{Net}(I) = \sum_{n}^{N} \delta_n \max_{m}^{M}(W_{mn} \circledast I + b_{mn}) \tag{1}$$

where $f_{Net}(I)$ is a non-linear function of the rendered view-image I, using a neural network Net, which is nothing but a CNN-based keypoint detector. N, M are hyperparameters controlling the complexity of the piece-wise linear activation function. δ is +1 if n is odd, and -1 otherwise. The parameters of the network Net to be learned are the convolution filter weights W_{mn} and biases b_{mn}, and $\circledast$ denotes the convolution operation. Keypoints x can be retrieved from softargmax by:

$$x = softargmax(PoI_{map}) \tag{2}$$

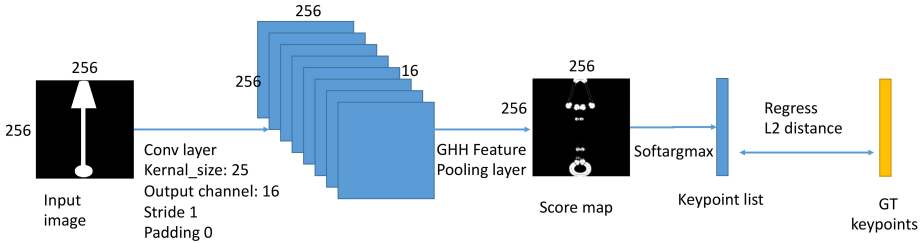

Fig. 6. Pipeline of simplified LIFT detector

where $softargmax$ is a function which computes the Center of Mass, with weights being the output of a standard softmax function. $softargmax$ can be written as:

$$softargmax(PoI_{map}) = \frac{\sum_y exp(\beta PoI_{map}(y))y}{\sum_y exp(\beta PoI_{map}(y))} \tag{3}$$

where y's are the pixel locations in PoI_{map}, and $\beta = 10$ is a hyper-parameter controlling the smoothness of the $softargmax$ function. This function is a *differentiable* version of non-maximum suppression. We finetune the pretrained LIFT detector on our rendered view-images.

3.3 Continuous patch-feature extractor

With patches of size 32×32 (obtained using a sliding window technique on the PoI maps) as input to the HardNet [5] module (see Figure 3 in the main paper), 128-D vectors are output from the DR-KFS feature extractor module for every patch within an image. The network architecture for HardNet is detailed in Table 4.

input	kernal size	stride	padding	activation layer	output
(1,32,32) image	3	1	1	BatchNorm + ReLU	(32,32,32)
(32,32,32)	3	1	1	BatchNorm + ReLU	(32,32,32)
(32,32,32)	3	2	1	BatchNorm + ReLU	(64,16,16)
(64,16,16)	3	1	1	BatchNorm + ReLU	(64,16,16)
(64,16,16)	3	2	1	BatchNorm + ReLU	(128,8,8)
(128,8,8)	3	1	1	BatchNorm + ReLU	(128,8,8)
(128,8,8)	8	1	1	BatchNorm	(128,1,1)

Table 4. Architectural details of HardNet used for patch based local feature extraction from PoI maps.

3.4 Localized Feature Matching

For every patch P_{recon} centered at (x, y) in the PoI maps corresponding to the reconstructed model, we consider a set of neighboring patches in the GT PoI maps, centered at $(x \pm \delta, y \pm \delta)$, and find a patch that yields the minimum MSE over the patch features (HardNet features). We do this for every patch in the PoI maps corresponding to the reconstructed model.We then perform a weighted addition of all such minimal loss values to get a similarity score as given below:

$$Sim_{recon} = \sum w_i |P_{recon_i} - P_{gt_k}| \tag{4}$$

where Sim_{recon} is the overall similarity score, $w_i = average(P_{recon_i})$ is the average value of the pixels in the i^{th} patch in the PoI map corresponding to the reconstructed model. P_{gt_k} is the k^{th} patch in the PoI map correponding to the GT model (which has the least patch-feature MSE for P_{recon_i}).

We essentially repeat the above procedure for the PoI maps corresponding to the GT model and get a similarity score Sim_{GT}. Our final training loss is the sum of Sim_{recon} and Sim_{GT}.

This two-way matching strategy helps our metric to be tolerant to small visual changes, yet not compromising on the discriminative capability on a global level. It also makes our approach pretty sensitive to part-deletion operations. See **??**

4 Implementation details of 3D reconstruction networks

In order to understand the quality of the reconstructed models using our metric, we train AtlasNet [2], Matryoshka Net [6], OGN [7], Pixel2Mesh [9] and 3D-R2N2 [1], with the original loss and DR-KFS loss, independently.

General training. During the training of AtlasNet [2] and Pixel2Mesh [9], reconstructed point clouds are corresponded to meshes directly using a pre-defined set of correspondences. Thus, we can straight away replace Chamfer Distance with DR-KFS metric when training such networks. When training voxel-based networks such as 3D-R2N2 [1], OGN [7] and Matryoshka Net [6], after obtaining the predicted occupancy fields, we use a deep marching cube layer (DMCL) proposed in [3] to convert an occupancy field to a mesh. The obtained mesh is refined via backpropagation using the DR-KFS loss. For such voxel-based networks, we train them with 32*32*32 ShapeNet voxel data.

OGN. Specifically, for OGN, due to the indifferentiability of converting an octree to a voxel, we only use one level of the octree and use a voxel of size 32*32*32. Thus, the octree-based models can be treated as a 32*32*32 voxel-based model and can be trained accordingly. For a fair comparison of the results, we also train the original OGN with a voxel input of size 32*32*32 and a one-level octree.

Matryoshka Net. As for Matryoshka Network, we adopted the training strategy in the original paper, encoding 32*32*32 voxel data with 6 shape layers $(x_0, x_1, y_0, y_1, z_0, z_1)$. In the original work, the network output voxels $voxel$ are determined by $voxel_x, voxel_y, voxel_z$, which are decoded from the aforementioned shape layers. Concretely,

$$voxel = voxel_x \cap voxel_y \cap voxel_z \tag{5}$$

And each $voxel_x, voxel_y, voxel_z$ is calculated using the formulation given in 6, as:

$$voxel_c = (indices_c \geq c_0) \cap (indices_c \leq c_1) \tag{6}$$

where c is x, y or z When training with DR-KFS loss, we feed the voxel probability to DMCL to decode the predicted shape into a voxel probability cube. For every shape layer index (x, y or z), with the associated sub-layer levels (x_0, x_1, y_0 etc.), we model the voxel probability p_x as a Normal distribution $N(\mu, \sigma)$, where $\mu = mean(x_0, x_1)$, $\sigma = 1$. The final voxel probability p, is then calculated as:

$$p = p_x * p_y * pz \tag{7}$$

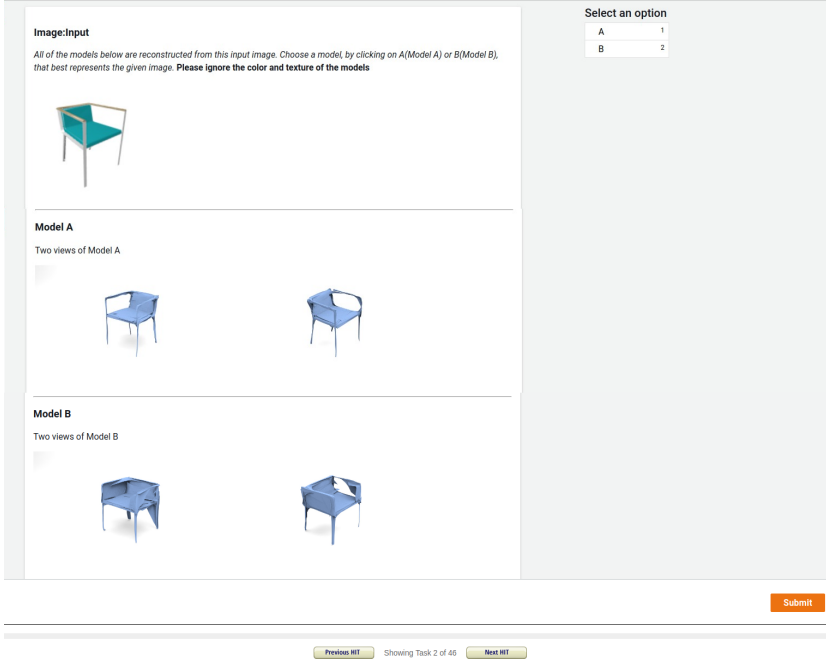

Fig. 7. User interface for PS1, PS2

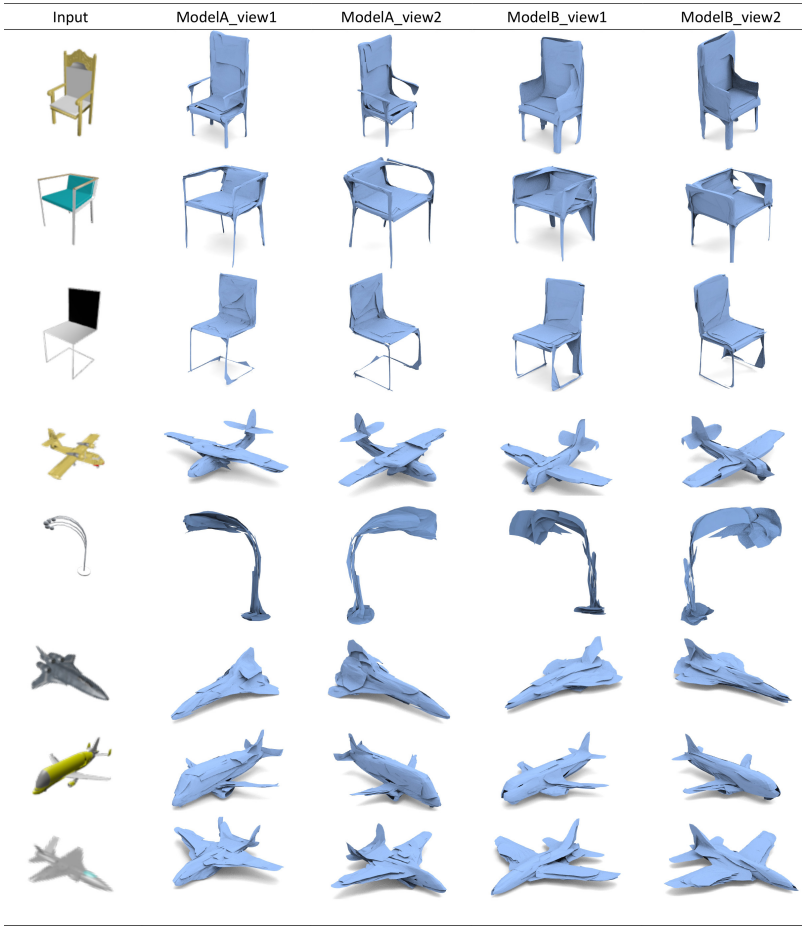

Fig. 8. Models used in PS-1 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

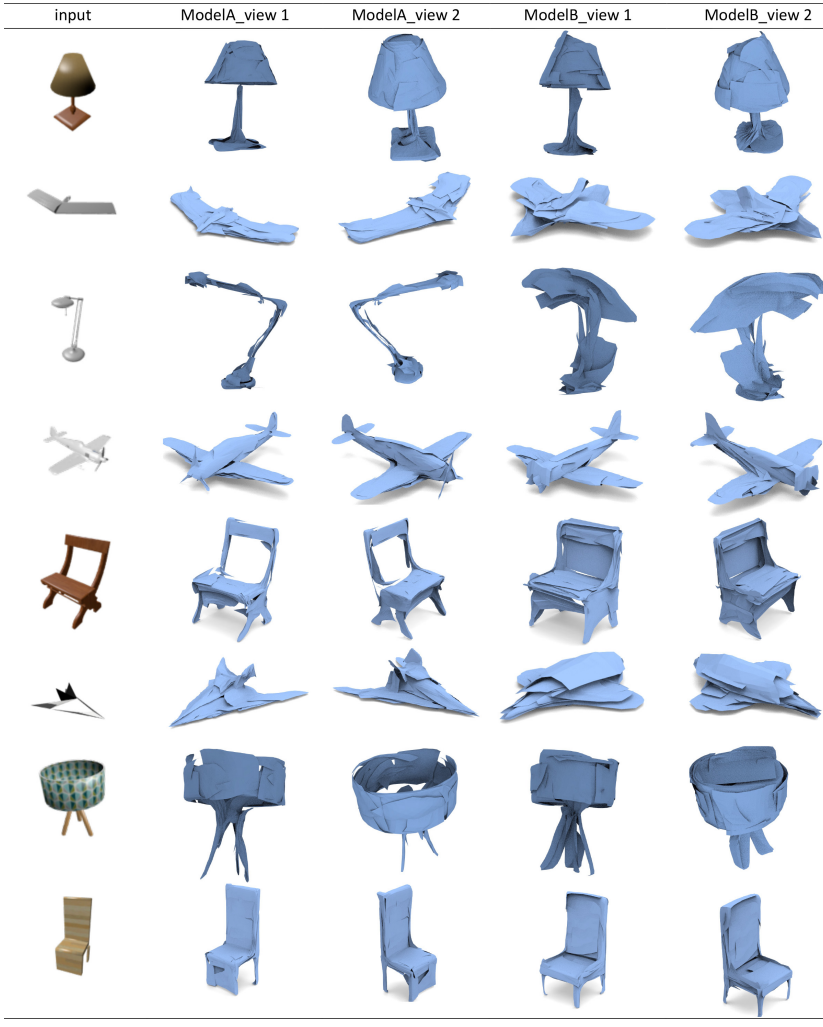

Fig. 9. Models used in PS-1 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

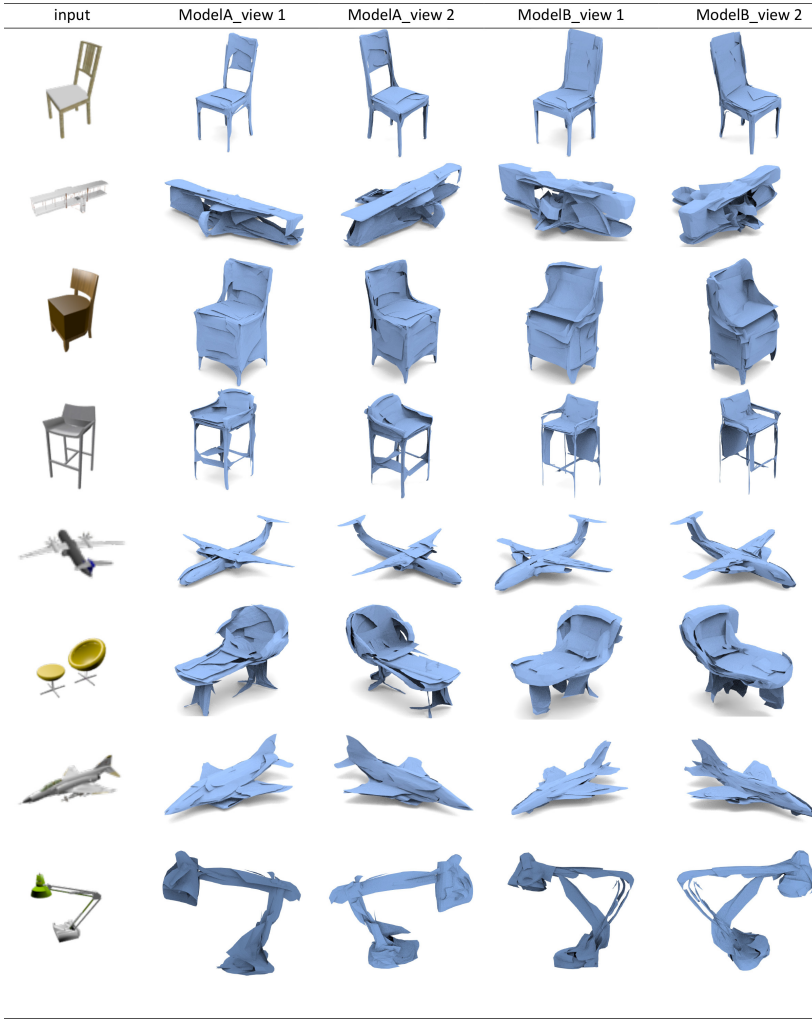

Fig. 10. Models used in PS-1 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

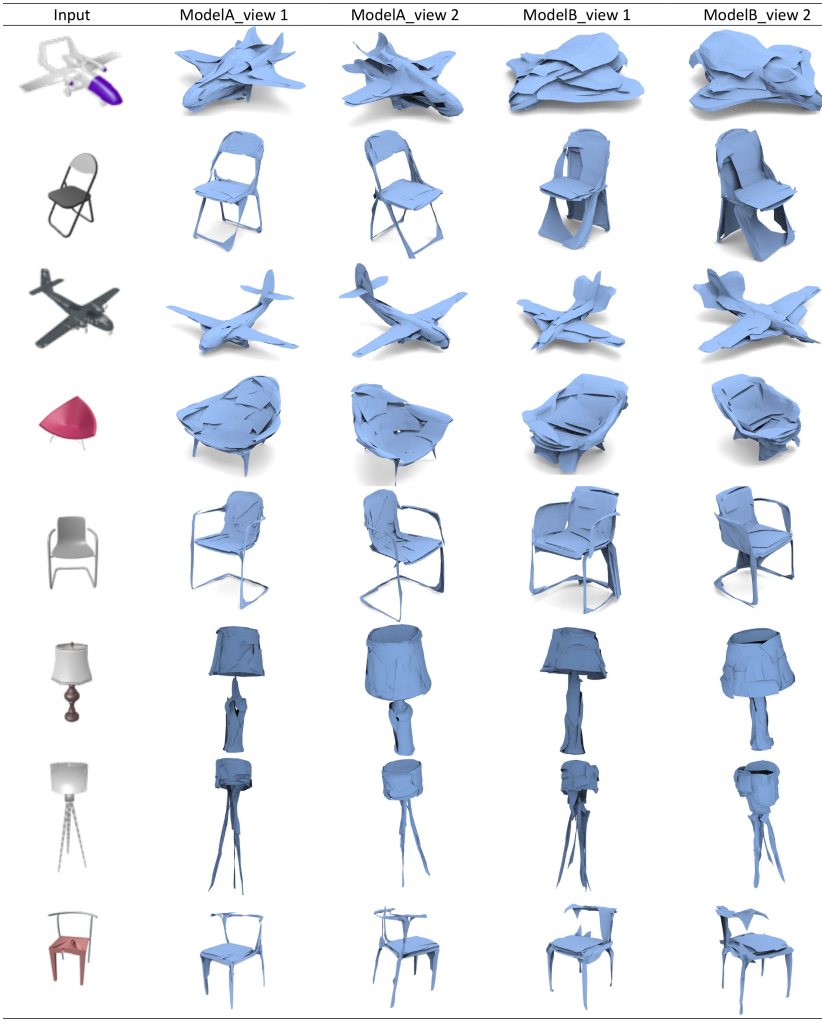

Fig. 11. Models used in PS-1 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

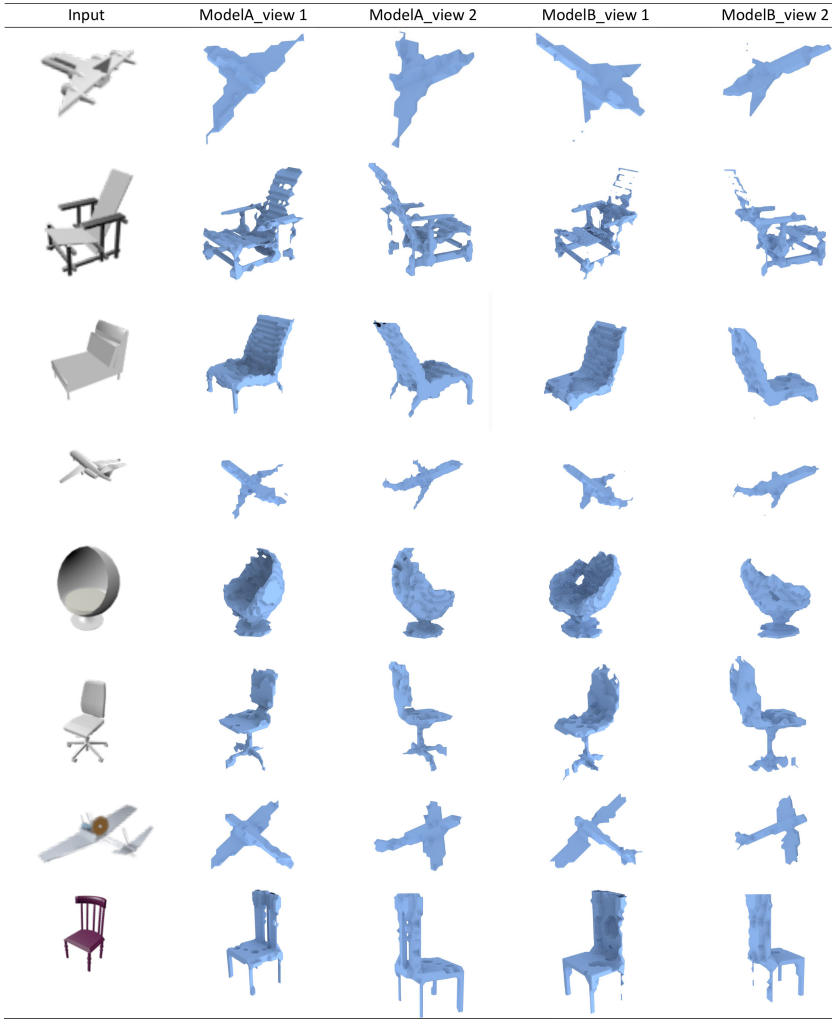

Fig. 12. Models used in PS-1 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

Fig. 13. Models used in PS-2 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

Fig. 14. Models used in PS-2 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

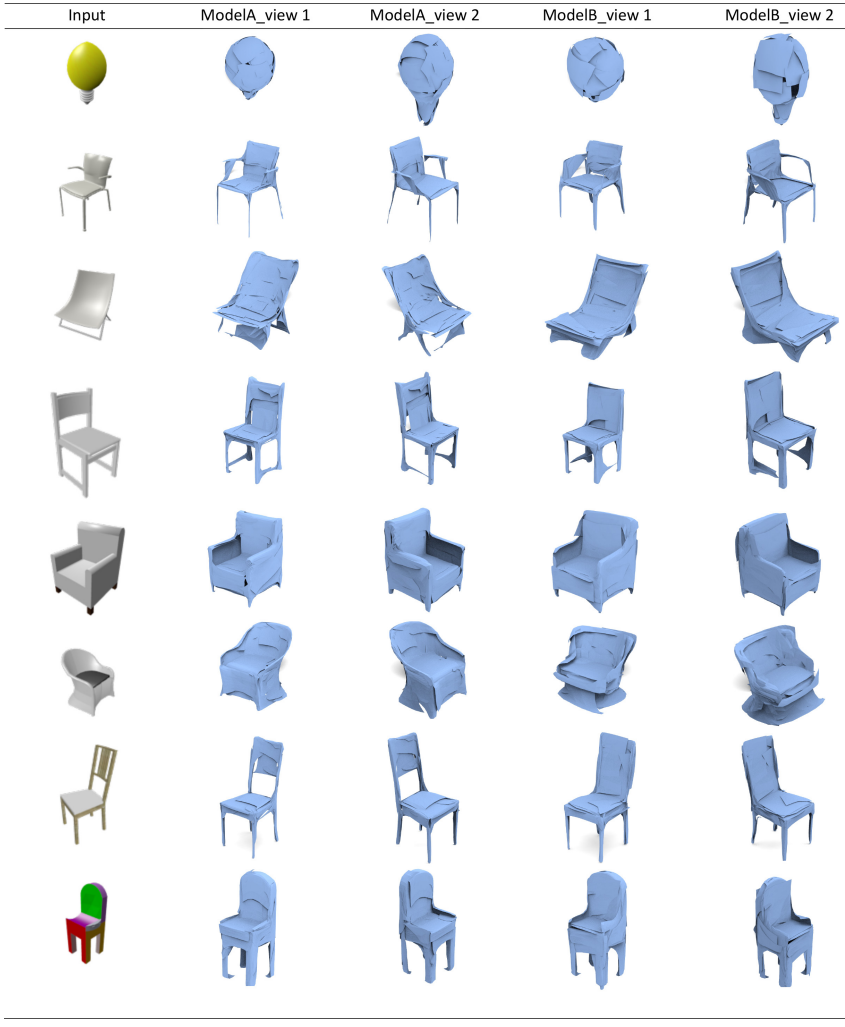

Fig. 15. Models used in PS-2 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

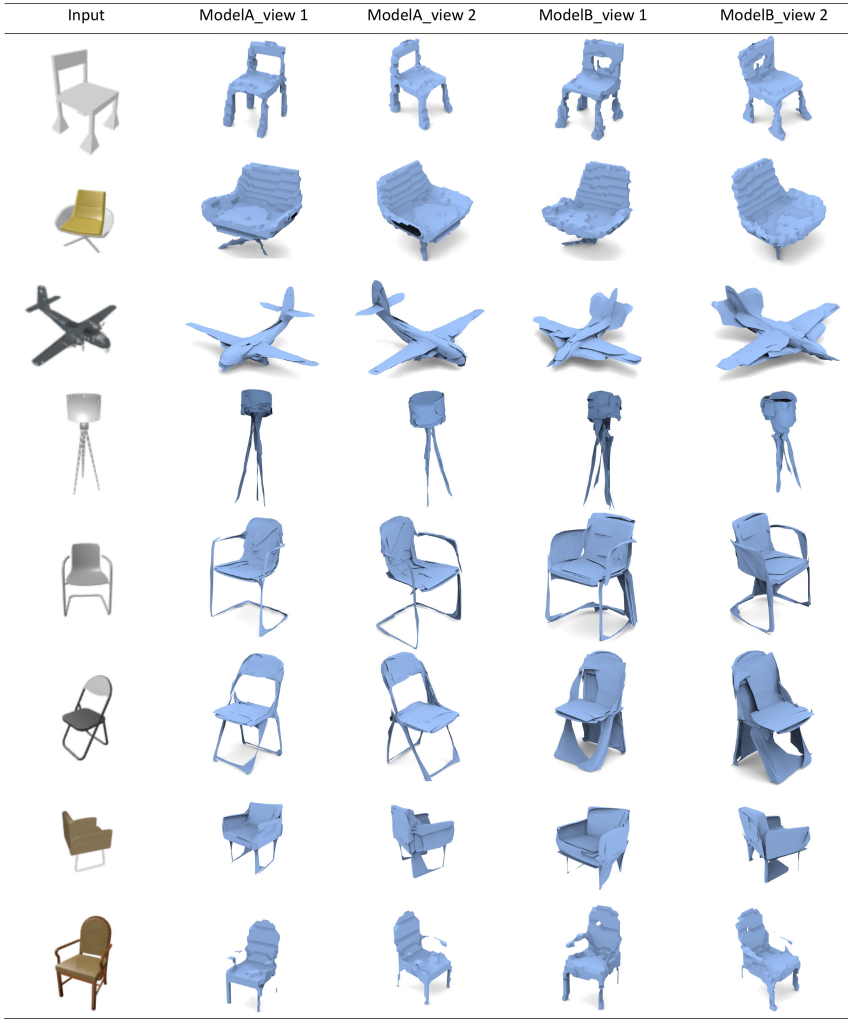

Fig. 16. Models used in PS-2 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

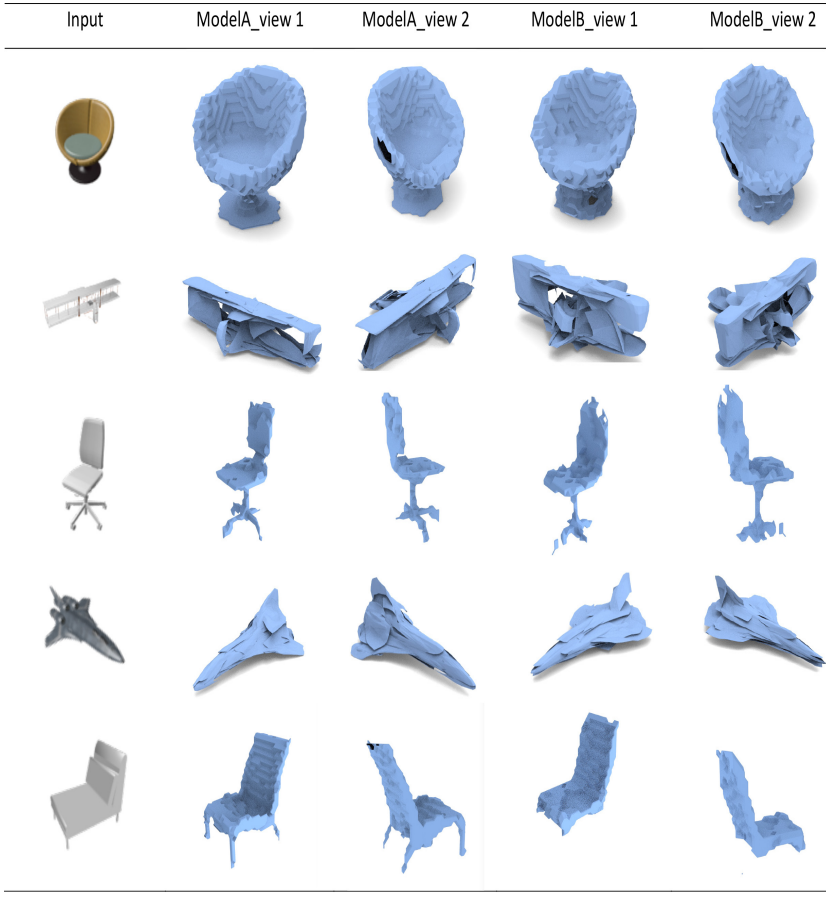

Fig. 17. Models used in PS-2 – Reconstructed model using DR-KFS loss is ModelA and that using the original loss is ModelB. Note that in the actual perceptual study, the order of the models is randomized, including their view-renderings.

References

1. Choy, C.B., Xu, D., Gwak, J., Chen, K., Savarese, S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In: European conference on computer vision. pp. 628–644. Springer (2016)
2. Groueix, T., Fisher, M., Kim, V.G., Russell, B., Aubry, M.: Atlasnet: A papier-mâché approach to learning 3d surface generation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
3. Liao, Y., Donne, S., Geiger, A.: Deep marching cubes: Learning explicit surface representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2916–2925 (2018)
4. Liu, S., Li, T., Chen, W., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In: ICCV (2019)
5. Mishchuk, A., Mishkin, D., Radenovic, F., Matas, J.: Working hard to know your neighbor's margins: Local descriptor learning loss. In: Advances in Neural Information Processing Systems. pp. 4826–4837 (2017)
6. Richter, S.R., Roth, S.: Matryoshka networks: Predicting 3d geometry via nested shape layers. In: CVPR (2018)
7. Tatarchenko, M., Dosovitskiy, A., Brox, T.: Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In: The IEEE International Conference on Computer Vision (ICCV) (Oct 2017)
8. Verdie, Y., Yi, K., Fua, P., Lepetit, V.: Tilde: a temporally invariant learned detector. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5279–5288 (2015)
9. Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., Jiang, Y.G.: Pixel2mesh: Generating 3d mesh models from single rgb images. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018)
10. Yi, K.M., Trulls, E., Lepetit, V., Fua, P.: Lift: Learned invariant feature transform. In: European Conference on Computer Vision. pp. 467–483. Springer (2016)