

Fully Understanding Generic Objects: Modeling, Segmentation, and Reconstruction — Supplementary Material

Feng Liu Luan Tran Xiaoming Liu
Michigan State University, East Lansing MI 48824
{liufeng6, tranluan, liuxm}@msu.edu

In this supplementary material, we provide:

i) Implementation details, including

- ◊ Training details;
- ◊ Network structures;
- ◊ Linear search and linear-binary search algorithms.

ii) Additional experimental results, including

- ◊ Expressiveness;
- ◊ Over-fitting problem study;
- ◊ Additional qualitative performances on 3D segmentation, reconstruction and image decomposition.

1. Implementation Details

1.1. Training Details

Data Preparation.

Following the work [1], we first obtain color voxelization in different resolutions ($16^3 \times 3$, $32^3 \times 3$, $64^3 \times 3$) for ShapeNet 3D models. Fig. 1 shows two examples of $64^3 \times 3$ colored voxel. Then, similar to the sampling strategy of [3], we obtain the triple data $\{\mathbf{x}_j, o_j, c_j\}_{j=1}^K$ offline for each colored voxel. \mathbf{x}, o, c are the spatial point and the corresponding occupancy label and albedo.

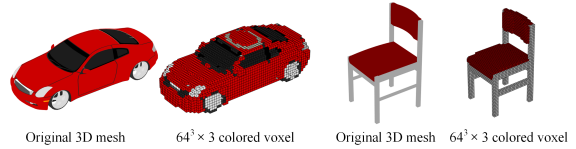


Figure 1. Color voxelization of ShapeNet models. Original 3D mesh (left) and $64^3 \times 3$ colored voxel (right).

Table 1. Training process.

	Network	Training data	Loss
Stage 1	$\mathcal{E}', \mathcal{D}_S, \mathcal{D}_A$	$64^3 \times 3$ colored voxels and sampled point-values	\mathcal{L}_1
Stage 2	$\mathcal{E}, \mathcal{D}_A$	synthetic images	\mathcal{L}_2
Stage 3	$\mathcal{E}, \mathcal{D}_A$	real images	\mathcal{L}_3

Training Process.

We summarize the training process in Tab. 1. In stage 1, we adopt a progressive training technique [3], to train our model on gradually increasing resolution data ($16^3 \rightarrow 32^3 \rightarrow 64^3$), which stabilizes and significantly speeds up the training process.

Hyperparameters.

In our experiments, we set $l_C = 3$, $l_S = 256$ and $l_A = 256$. We set $\lambda_1 = 1$, $\lambda_2 = 0.1$, $\lambda_3 = 10$, $\lambda_C = 1$, $\lambda_S = 5$, $\lambda_A = 5$, $\lambda_P = 10$ to balance the losses. We set $q = 50$ in the local feature extraction. Adam optimizer is used with a learning rate of 0.0001 in all stages.

1.2. Network Structures

Colored Voxel Encoder \mathcal{E}' .

To auto-encode 3D shape and albedo simultaneously, we adopt a 3D CNN [2, 3] as encoder \mathcal{E}' to learn the embedding category, shape and albedo features $\mathbf{f}_C, \mathbf{f}_S, \mathbf{f}_A$ from $64^3 \times 3$ colored voxel. The architecture of \mathcal{E}' is depicted in Tab. 2.

Table 2. Colored voxel encoder network structure.

Layer	Kernel size	Stride	Activation function	Output size (d1,d2,d3,C)
input	-	-	-	(64, 64, 64, 3)
conv3d	(4, 4, 4)	(2, 2, 2)	LReLU	(32, 32, 32, 32)
conv3d	(4, 4, 4)	(2, 2, 2)	LReLU	(16, 16, 16, 64)
conv3d	(4, 4, 4)	(2, 2, 2)	LReLU	(8, 8, 8, 128)
conv3d	(4, 4, 4)	(2, 2, 2)	LReLU	(4, 4, 4, 256)
conv3d	(4, 4, 4)	(1, 1, 1)	-	(1, 1, 1, 515)
\mathbf{f}_C	-	-	-	3
\mathbf{f}_S	-	-	-	256
\mathbf{f}_A	-	-	-	256

Image Encoder \mathcal{E} .

As shown in Tab. 3, we use a modified ResNet-18 architecture, which was pre-trained on ImageNet, as our image encoder. However, we adjust the last fully-connected layer to project the features to four embeddings $\mathbf{L}, \mathbf{P}, \mathbf{f}_S$ and \mathbf{f}_A .

Table 3. Image encoder network structure (slightly modified from ResNet-18).

Layer	Kernel size	Stride	Activation function	Input size	Output size
input	-	-	-	-	(128, 128, 3)
conv1	(7, 7)	(2, 2)	BN, LReLU	(128, 128, 3)	(32, 32, 64)
conv2 (ResNet block)	(3, 3)	-	-	(32, 32, 64)	(32, 32, 64)
conv3 (ResNet block)	(3, 3)	-	-	(32, 32, 64)	(16, 16, 128)
conv4 (ResNet block)	(3, 3)	-	-	(16, 16, 128)	(8, 8, 256)
conv5 (ResNet block)	(3, 3)	-	-	(8, 8, 256)	(4, 4, 512)
average pool	(4, 4)	-	-	(4, 4, 512)	(1, 1, 512)
\mathbf{FC}_L	-	-	-	512	27
\mathbf{FC}_P	-	-	-	512	12
\mathbf{FC}_{f_C}	-	-	-	512	3
\mathbf{FC}_{f_S}	-	-	-	512	256
\mathbf{FC}_{f_A}	-	-	-	512	256

Shape and Albedo Decoders $\mathcal{D}_S, \mathcal{D}_A$.

The shape decoder architecture is followed the work of [2] (unsupervised case). The network takes shape latent representation \mathbf{f}_S and a spatial point (x, y, z) as inputs. It is composed of 3 fully connected layers each of which is applied with Leaky ReLU, except the final output is applied *Sigmoid* activation (Fig. 2). The albedo decoder architecture is similar, with only two differences. The inputs to the network have an additional vector, albedo latent representation \mathbf{f}_A . The output is applied *Tanh* activation. Fig. 3 depicts the albedo decoder architecture.

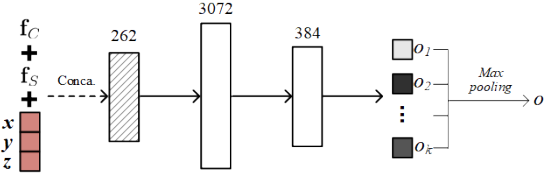


Figure 2. The shape decoder network is composed of 3 fully connected layers, denotes as “FC”. The \mathbf{f}_C (3-dim), \mathbf{f}_S (256-dim) are concatenated, denoted “+”, with the xyz query, making a 262-dim vector, and is provided as input to the first layer. The Leaky ReLU activation is applied to the first 2 FC layers while the final value is obtained with *Sigmoid* activation.

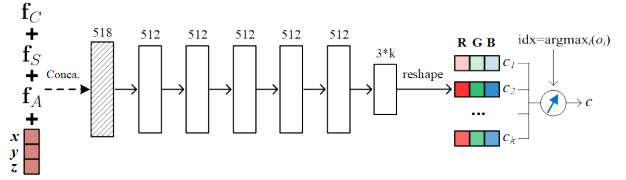
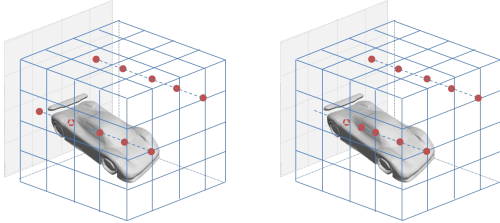


Figure 3. The albedo decoder network is composed of 6 fully connected layers. Specifically, it takes the point coordinate (x, y, z) , along with $\mathbf{f}_C, \mathbf{f}_S, \mathbf{f}_A$, and outputs the RGB color values. The Leaky ReLU activation is applied to the first 5 FC layers while the final value is obtained with *Tanh* activation.

1.3. Linear Search and Linear-Binary Search Algorithms

For efficient network training, instead of finding exact surface points, we approximate them using Linear search or Linear-Binary search (Fig. 4). The detailed algorithms for Linear Search and Linear-Binary Search are represented in Algorithm 1 and 2 respectively.

Empirically, with the same number of evaluating steps, the proposed Linear-Binary search better approximates the object surface. This is demonstrated on render images of surface normals (Fig. 5). While the ad-hoc Linear search approach leads to artifacts, Linear-Binary search results in smooth continuous surfaces.



(a) Linear Search (b) Linear-Binary Search

Figure 4. Ray tracing for surface points detection. In Linear search, candidates (red points) are uniformly distributed in the grid. In Linear-Binary search, after the first point inside the object found, Binary search will be used between the last outside point and current inside point for all remaining iterations.

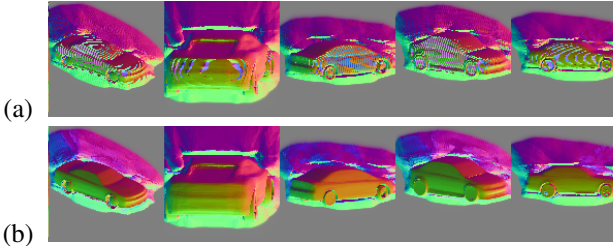


Figure 5. Compare surface normal rendering quality of (a) Linear vs. (b) Linear-Binary search. With the same computation budget, Linear-Binary search better approximates surface points, which leads to smooth normal computation meanwhile Linear search causes artifacts on rendering, which spoils model training.

Algorithm 1: Linear Search Ray Tracing

```

input : Shape decoder  $\mathcal{D}_S$ , projection  $\mathbf{P}$ , error margin  $\epsilon$ , image size  $W, H$ 
output:  $W \times H$  surface points  $\mathbf{X}_{\text{final}}$ 

// Step 1: Generate candidate grid
1  $\mathbf{X}_{\text{boundary}} \leftarrow [[-1, -1, -1], [-1, -1, 1], [-1, 1, -1], [-1, 1, 1], \dots$ 
    $[1, -1, -1], [1, -1, 1], [1, 1, -1], [1, 1, 1]]$ ;
2  $\mathbf{U}_{\text{boundary}} \leftarrow \text{apply\_projection}(\mathbf{P}, \mathbf{X}_{\text{boundary}})$ ;
3  $d_{\min} \leftarrow \min(\mathbf{U}_{\text{boundary}}[:, 2])$ ;
4  $d_{\max} \leftarrow \max(\mathbf{U}_{\text{boundary}}[:, 2])$ ;
5  $\mathbf{d} \leftarrow (d_{\min} : d_{\max} : \epsilon)$ ;
6  $\mathbf{U}_{\text{candidate}} \leftarrow \text{meshgrid}((1 : W) * \mathbf{d}, (1 : H) * \mathbf{d}, \mathbf{d})$ ;
7  $\mathbf{X}_{\text{candidate}} \leftarrow \text{apply\_projection}(\mathbf{P}^{-1}, \mathbf{U}_{\text{candidate}})$ ;
// Step 2: Surface point selection
// Step 2.1: Select first surface point ( $o > 0.5$ )
8 for  $k \leftarrow \lceil \frac{1}{\epsilon} \rceil$  to 1 do
   // Evaluate the occupancy field
   9  $\mathbf{X}_{\text{curr.candidate}} \leftarrow \mathbf{X}_{\text{candidate}}[:, :, k]$ ;
   10  $\mathbf{O}_{\text{curr}} \leftarrow \mathcal{D}_S(\mathbf{f}_S, \mathbf{X}_{\text{curr.candidate}})$ ;
   // Selection (As backward tracing, the last
   // selected point is the closest to the camera
   11  $\mathbf{X}_{\text{final}}[\mathbf{O}_{\text{curr}} > 0.5] \leftarrow \mathbf{X}_{\text{curr.candidate}}[\mathbf{O}_{\text{curr}} > 0.5]$ ;
   12  $\mathbf{O}[:, :, k] \leftarrow \mathbf{O}_{\text{curr}}$ ;

// Step 2.2: Select largest closest point to the
// surface for the background
13  $\mathbf{X}_{\text{closest}} \leftarrow \mathbf{X}_{\text{candidate}}[\text{argmax}(\mathbf{O}, \text{axis} = 2)]$ ;
14  $\mathbf{X}_{\text{final}}[\mathbf{X}_{\text{final}} = 0] \leftarrow \mathbf{X}_{\text{closest}}[\mathbf{X}_{\text{final}} = 0]$ ;

```

Algorithm 2: Linear-Binary Search Ray Tracing

```

input : Shape decoder  $\mathcal{D}_S$ , projection  $\mathbf{P}$ , error margin  $\epsilon$ , image size  $W, H$ 
output:  $W \times H$  surface points  $\mathbf{X}_{\text{final}}$ 

// Step 1: Calculate initial position and ray
// direction
1  $\mathbf{X}_{\text{boundary}} \leftarrow [[-1, -1, -1], [-1, -1, 1], [-1, 1, -1], [-1, 1, 1], \dots$ 
    $[1, -1, -1], [1, -1, 1], [1, 1, -1], [1, 1, 1]]$ ;
2  $\mathbf{U}_{\text{boundary}} \leftarrow \text{apply\_projection}(\mathbf{P}, \mathbf{X}_{\text{boundary}})$ ;
3  $d_{\min} \leftarrow \min(\mathbf{U}_{\text{boundary}}[:, 2])$ ;
4  $d_{\max} \leftarrow \max(\mathbf{U}_{\text{boundary}}[:, 2])$ ;
5  $\mathbf{d} \leftarrow (d_{\min} : d_{\max} : \epsilon)$ ;
6  $\mathbf{U}_{\text{init}} \leftarrow \text{meshgrid}((1 : W) * \mathbf{d}, (1 : H) * \mathbf{d}, \mathbf{d})$ ;
7  $\mathbf{X}_{\text{init}} \leftarrow \text{apply\_projection}(\mathbf{P}^{-1}, \mathbf{U}_{\text{init}})$ ;
8  $\mathbf{X}_{\text{init.v}} \leftarrow \text{cal\_direction}(\mathbf{P})$ ;
// Step 2: Surface point selection
9  $\mathbf{O}_{\text{max}} \leftarrow 0$ ;
10  $\mathbf{O}_{\text{argmax}} \leftarrow 0$ ;
11  $\mathbf{X}_{\text{final}} \leftarrow \mathbf{X}_{\text{init}}$ ;
12 for  $k \leftarrow 1$  to  $\lceil \frac{1}{\epsilon} \rceil$  do
   // Evaluate the occupancy field
   13  $\mathbf{X}_{\text{curr.candidate}} \leftarrow \mathbf{X}_{\text{final}} + \mathbf{X}_{\text{init.v}}$ ;
   14  $\mathbf{O}_{\text{curr}} \leftarrow \mathcal{D}_S(\mathbf{f}_S, \mathbf{X}_{\text{curr.candidate}})$ ;
   // Move outside points forward
   15  $\mathbf{X}_{\text{final}}[\mathbf{O}_{\text{curr}} < 0.5] \leftarrow \mathbf{X}_{\text{final}}[\mathbf{O}_{\text{curr}} < 0.5] + \mathbf{X}_{\text{init.v}}[\mathbf{O}_{\text{curr}} < 0.5]$ ;
   // Reduce velocity of inside points for binary
   // search
   16  $\mathbf{X}_{\text{init.v}}[\mathbf{O}_{\text{curr}} \geq 0.5] \leftarrow \mathbf{X}_{\text{init.v}}[\mathbf{O}_{\text{curr}} \geq 0.5] / 2$ ;
   // Update max occupancy value
   17  $\mathbf{O}_{\text{max}}[\mathbf{O}_{\text{curr}} > \mathbf{O}_{\text{max}}] \leftarrow \mathbf{O}_{\text{curr}}[\mathbf{O}_{\text{curr}} > \mathbf{O}_{\text{max}}]$ ;
   18  $\mathbf{O}_{\text{argmax}}[\mathbf{O}_{\text{curr}} > \mathbf{O}_{\text{max}}] \leftarrow \mathbf{X}_{\text{final}}[\mathbf{O}_{\text{curr}} > \mathbf{O}_{\text{max}}]$ ;

// Update background spatial points
19  $\mathbf{X}_{\text{final}}[\mathbf{O}_{\text{max}} < 0.5] \leftarrow \mathbf{O}_{\text{argmax}}[\mathbf{O}_{\text{max}} < 0.5]$ ;

```

2. Additional Experimental Results

2.1. Expressiveness

Given our disentangled intrinsic 3D representations, we are able to manipulate any individual component describing the image.

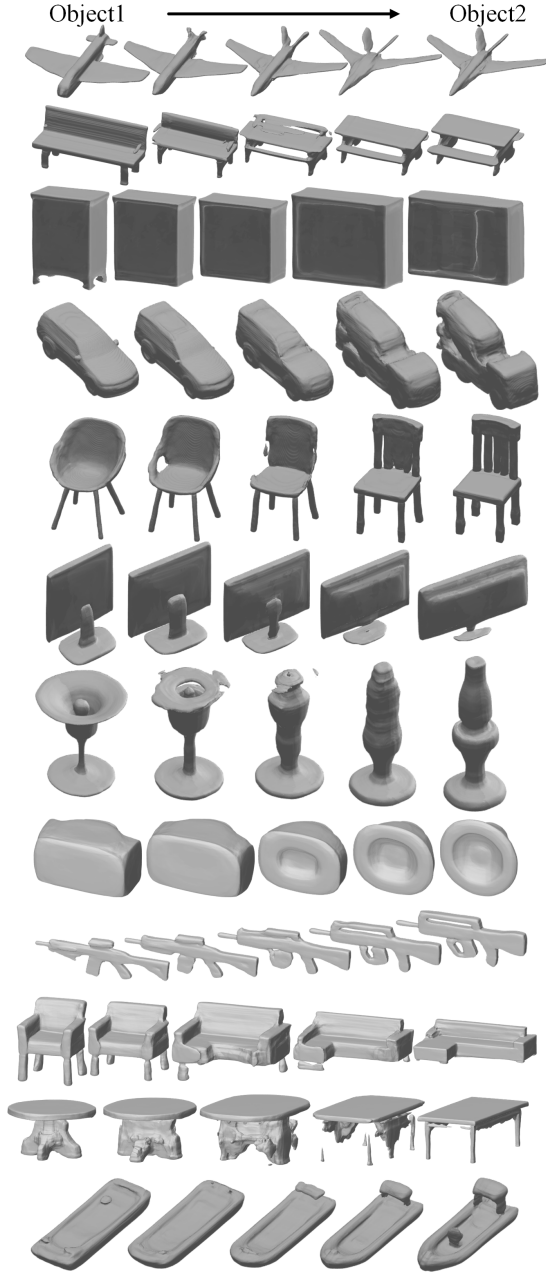


Figure 6. Linear interpolation of two objects within the same category in shape latent space.

For example, changing \mathbf{P} allows us to virtually rotate the object or alternative \mathbf{L} can switch the image lighting. Here we demonstrate our ability to interpolate between two objects. We can interpolate objects in shape/albedo latent space $\alpha \mathbf{f}_{S/A}^{(1)} + (1 - \alpha) \mathbf{f}_{S/A}^{(2)}$ ($\alpha \in [0,1]$) within the same category (Fig. 6 and 7).

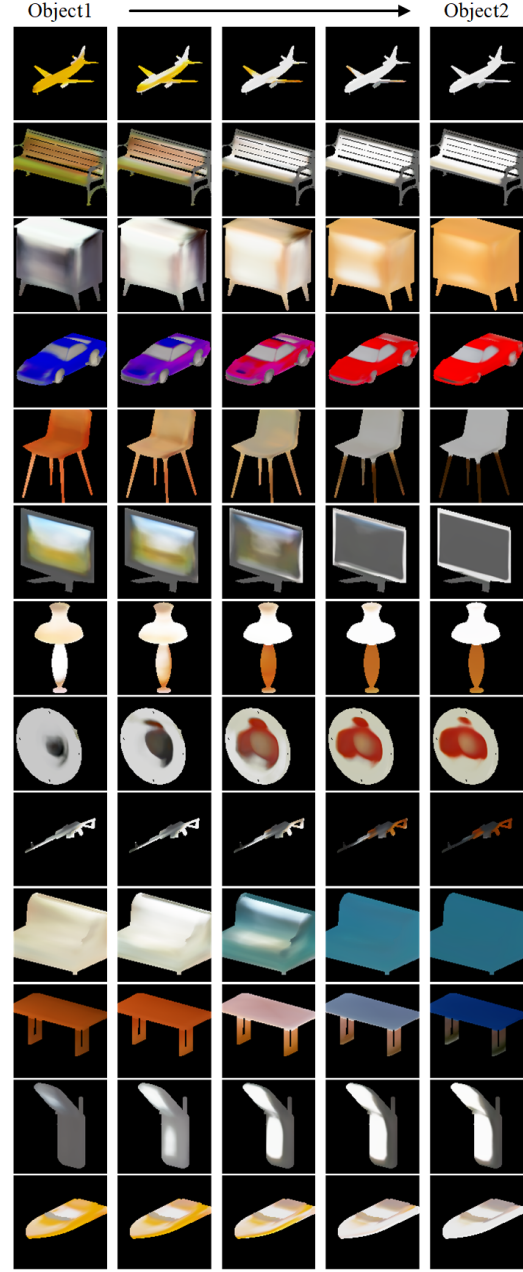


Figure 7. Linear interpolation of two objects within the same category in albedo space.

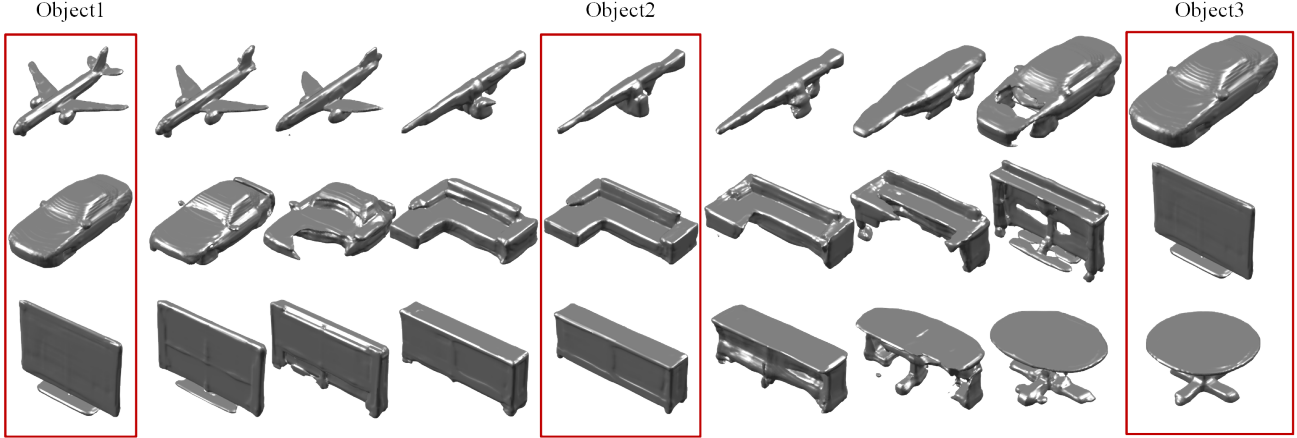


Figure 8. Linear interpolation of objects' shape in category latent space.

Further, the trained single universal model also allows us to show the interpolation performance across different categories. Fig. 8 shows object interpolation in category latent space $\alpha \mathbf{f}_C^{(1)} + (1-\alpha) \mathbf{f}_C^{(2)}$ ($\alpha \in [0,1]$), while having the *same* shape latent code. It can be observed that our model allows us to synthesize 3D object with new shape, albedo and even category by sampling the latent spaces.

2.2. Over-Fitting Problem Study

Reconstruction implies reasoning about the 3D structure of the input image using cues such as texture, shading, and perspective effects. Recently, the work of [5] argues that most of reconstruction methods do not actually perform reconstruction but retrieve due to they only simply map the input image to 3D space through a latent representation. Thanks to the combination of 3D decomposition and modeling, our framework does learn both high-level semantic understanding and low-level image cues from images.

To further prove that, we provide evidences that our model does not actually overfit to training shapes, nor treat 3D reconstruction as a retrieval problem. We statistically evaluate the distance between the latent codes (combination of \mathbf{f}_C and \mathbf{f}_S) of train and test samples together with their corresponding closet pre-learned ground-truth latent codes. As shown in Fig. 9, we visualize the distribution of distance scores for the testing and training samples. It is obvious that the distributions of training and testing features are similar, which indicates our model does not perform retrieval. If we performed retrieval, the test sample distribution would move to the origin of the x-axis and be very close to zero.

2.3. Additional Qualitative Performances

3D Image Decomposition.

We provide several 3D image decomposition results on real-world car images (see Fig. 10). Our framework pro-

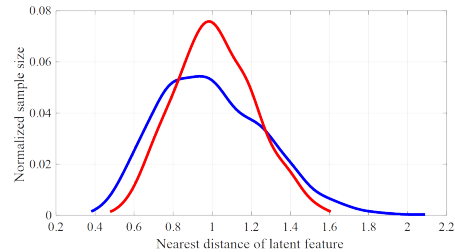


Figure 9. Distribution of nearest distance of latent features (combination of \mathbf{f}_C and \mathbf{f}_S) for 13 categories. **Blue**: training samples; **Red**: testing samples.

duces good visual decompositions for real images.

Branched Albedo Visualization. We assign a color for the output of each branch of our shape decoder and reasonable parts are obtained. Since our segmentation is unsupervised and the model for each category is trained separately, our results are not guaranteed to produce the same part counts for all categories. Fig. 11 shows the estimations of albedo colors of valid branches. The albedo branches do represent the dominant albedo colors of the objects.

Reconstruction Comparisons on Synthetic and Real Images. Fig. 12 shows more qualitative comparisons with Front2Back method [8] (F2B, CVPR 20') on 13 categories of ShapeNet. Obviously, our model is able to estimate 3D shapes that closely resemble the ground truth shapes. To provide more comprehensive comparisons on the 3D reconstruction quality. We provide more reconstruction results on Pascal3D+ [7] (Fig. 13) and Pix3D [4] datasets (Fig. 14). Comparison is made with ShapeHD [6] using trained model provided by the authors.

References

- [1] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape:

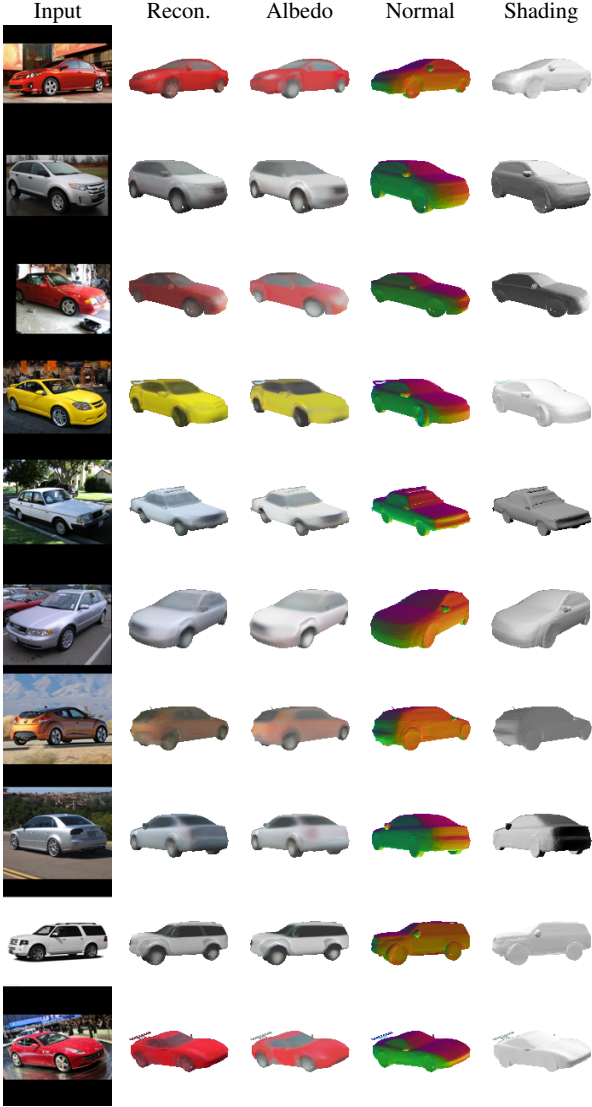


Figure 10. 3D image decomposition on real-world car images. Our work decomposes a 2D image of generic objects into albedo, completed 3D shape and illumination.

Generating shapes from natural language by learning joint embeddings. In *ACCV*, 2018.

- [2] Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang. BAE-NET: Branched autoencoder for shape co-segmentation. In *ICCV*, 2019.
- [3] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019.
- [4] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3D: Dataset and methods for single-image 3D shape modeling. In *CVPR*, 2018.
- [5] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view

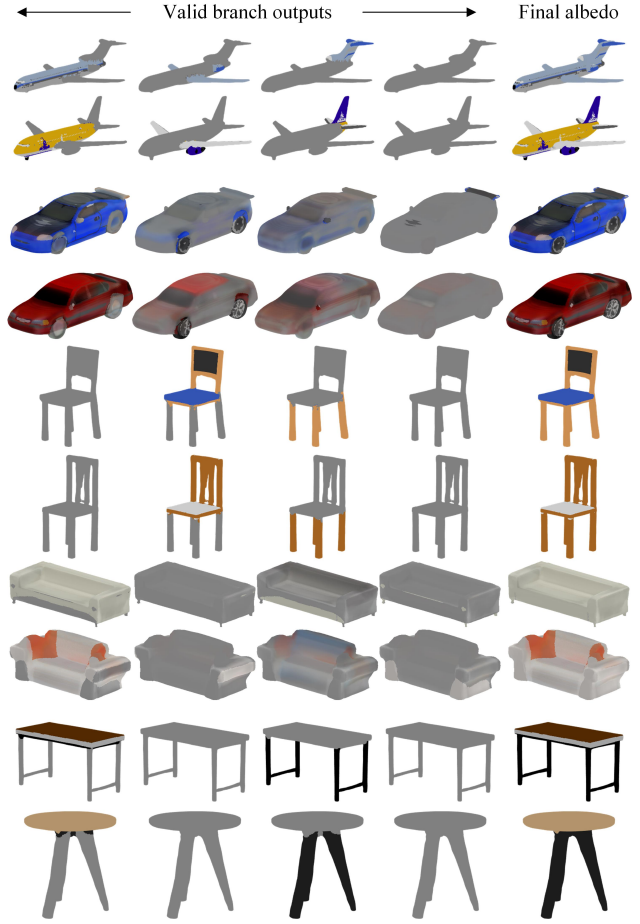


Figure 11. Visualization of albedo branch outputs. We render the albedo with reconstructed mesh.

3D reconstruction networks learn? In *CVPR*, 2019.

- [6] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. Learning shape priors for single-view 3D completion and reconstruction. In *ECCV*, 2018.
- [7] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3D object detection in the wild. In *WACV*, 2014.
- [8] Yuan Yao, Nico Schertler, Enrique Rosales, Helge Rhodin, Leonid Sigal, and Alla Sheffer. Front2Back: Single view 3D shape reconstruction via front to back prediction. In *CVPR*, 2020.



Figure 12. Reconstruction comparison with Front2Back method [8] (F2B) of 13 categories on ShapeNet. Our reconstructions more closely match the ground-truth shapes.

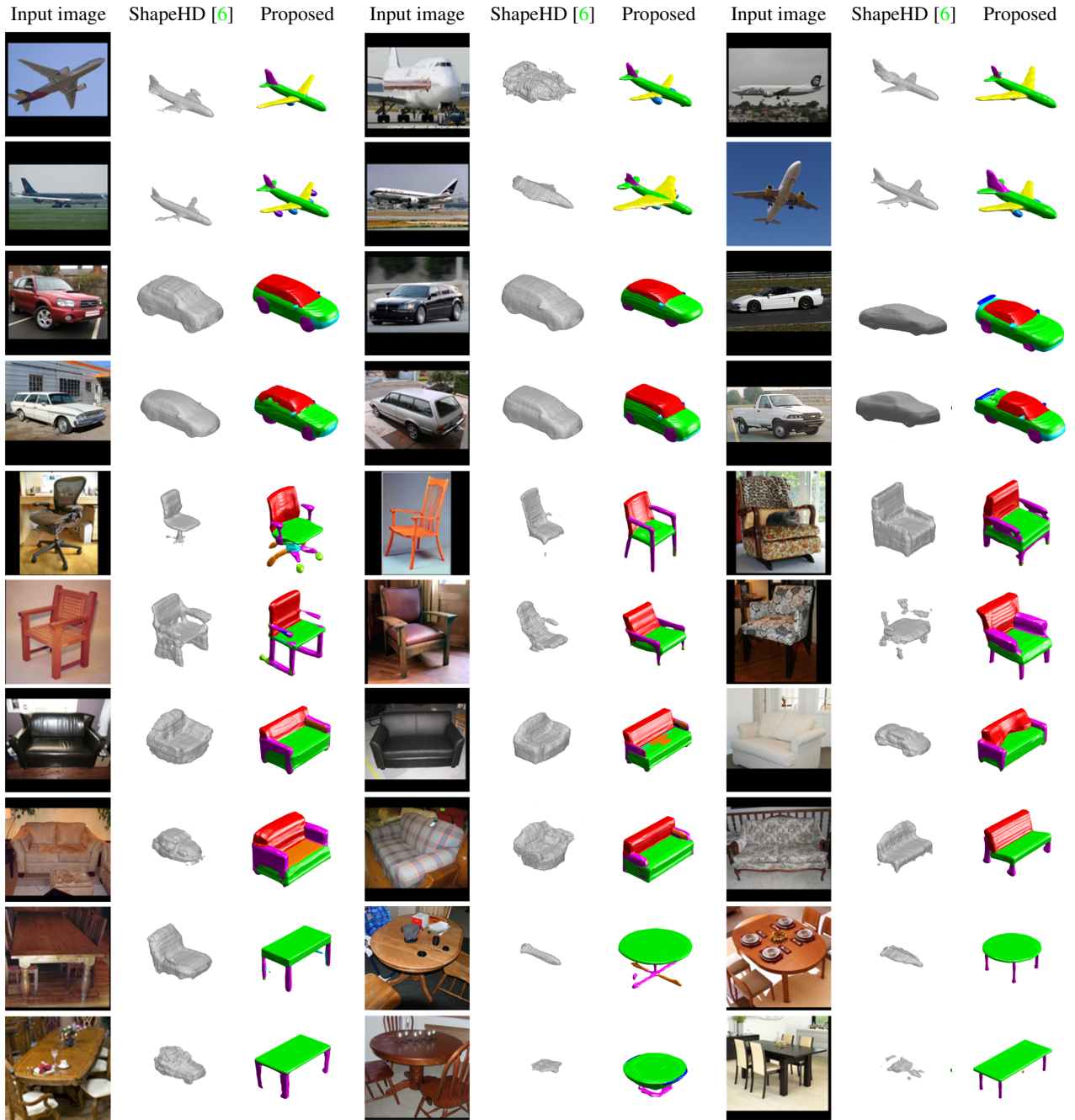


Figure 13. Additional 3D reconstruction results on Pascal3D+ [7] dataset. Compared to ShapeHD [6], our method reconstructs 3D shape with more details.



Figure 14. Additional 3D reconstruction results on Pix3D [4]. For each input image, we show reconstructions by ShapeHD [6], and ground truth. Our reconstructions resemble the ground truth.