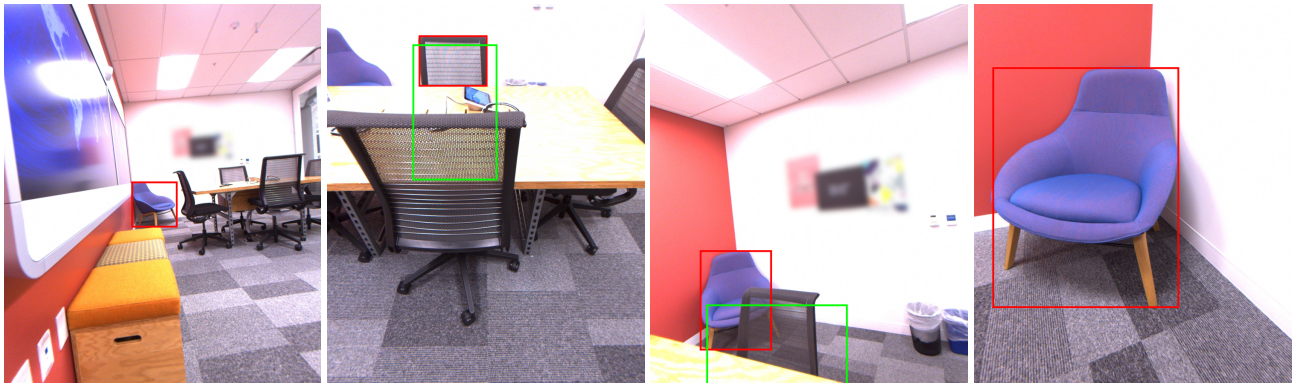
Supplemental Material for FroDO: From Detections to 3D Objects



a: Bad: small BB b: Bad: occlusion c: Bad: overlap d: Good

Figure 1: Several examples on bad 2D detection bounding box filtered out by the projection of 3D bounding box. The red and green box are 2D detected bounding box and the projection of the 3D bounding box respectively

1. Detection Pruning

After estimating the bounding box of an object, good views are selected by employing a pruning scheme. This scheme uses three criteria to reject frames based on (1) bounding box size, (2) occlusions and (3) overlap with other objects. (1) is implemented via a threshold on the width and height of the bounding box (BB), and (2,3) are implemented using a threshold on the intersection over union (IoU) of the bounding boxes. Figure 1 illustrates these strategies. The selection helps to identify better initial detections to extract good shape codes and later on leads to an optimization with clean energy terms. An ablation study on the detection pruning on Redwood-OS dataset is demonstrated in Table 1.

2. DP-means-based Line Segment Clustering

DP-mean clustering is similar to K-mean clustering as it also runs in an Expectation-Maximization manner. A key difference is that the total number of cluster is unknown at first. A new cluster is generated when the distance between a line segment and any existing clusters is larger than a cluster penalty threshold, which we set to 0.4. The distance we used is euclidean point (cluster) to line segment distance. Algorithm 1 details each steps.

Algorithm 1: DP-mean clustering for 3D line segments

Input: $\boldsymbol{r_1}, \boldsymbol{r_2}, ..., \boldsymbol{r_n}$: n rays, λ : cluster penalty threshold

Output: $c_1, c_2, ..., c_m$: m object clusters

1. init. $\boldsymbol{\mu_1} = mid(\boldsymbol{r_1})$, $mid()$ is to compute the middle point of a ray;
2. **while** *not converged* **do**
   - **for** *each* $\boldsymbol{r_i}$ **do**
     - $d_{ij} = \min_{\mu_1 ... k} dist(\boldsymbol{\mu_k}, \boldsymbol{r_i})$;
     - **if** $d_{ij} \geq \lambda$ **then**
       - set k = k + 1;
       - $\boldsymbol{\mu_k} = mid(\boldsymbol{r_i})$;
     - **else**
       - $z_i = j$;
     - **end**
   - **end**
   - k clusters are generated, where $c_k = \{\boldsymbol{r_i} | z_i = k\}$;
   - **for** *each* c_i **do**
     - $\boldsymbol{\mu_i} = lstsq(c_k)$;
   - **end**
3. **end**

Method	Vote	Average
CD [cm] w/o occlusion filter	12.17	11.73
CD [cm] w/ occlusion filter	11.97	10.57

Table 1: Ablation study on using the inferred 3D bounding box to filter occluded views.

3. Dense Shape Code Optimization

In order to implement the dense optimization efficiently, certain measures were taken to achieve an effective optimization procedure. The following two sections explain decisions for the formulation of analytical partial derivatives and for the implementation these.

3.1. Analytical Partial Derivatives

As the DeepSDF decoder yields signed distance values, in contrast to the pointcloud decoder which outputs object coordinates, neither E_g nor E_p are explicitly defined. Therefore, dense object coordinates are extracted by sampling the zero-crossing of the distance field and deriving relevant Jacobians analytically. While most of the derivatives of E_p and E_g wrt. code and pose follow the chain rule, the relevant term $\frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \frac{\partial \mathbf{x}}{\partial f_\theta}\frac{\partial f_\theta}{\partial \mathbf{z}}$ needs more attention.

Intuitively, the change of a surface coordinate $\mathbf{x}$ with respect to a change in code $\mathbf{z}$ is ambiguous due to potential changes in topology. However, as shown by [1] it is possible to derive a first-order approximation as follows. Given x_t as a function of code $\mathbf{z}$ on the surface at time t, it is given that corresponding distance value remains constant at the zero-crossing, implying:

$$\mathbf{x}_0 = x(\mathbf{z}_0) \tag{1}$$

$$0 = \left.\frac{\partial f_\theta(x(\mathbf{z}), \mathbf{z})}{\partial \mathbf{z}}\right|_{t=0} \tag{2}$$

Using the multivariate chain rule, and solving for $\frac{\partial \mathbf{x}}{\partial f_\theta}$, yields:

$$\left.\frac{\partial x}{\partial f_\theta}\right|_{t=0} = -\frac{\partial f_\theta}{\partial \mathbf{x}}^{-1}\frac{\partial f_\theta}{\partial \mathbf{z}} \approx -\vec{\mathbf{n}}\frac{\partial f_\theta}{\partial \mathbf{z}} \tag{3}$$

Where $\vec{\mathbf{n}}$ is the surface normal. Since the decoder is differentiable, $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ can be calculated easily using this first-order approximation. Figure 2 visualizes $\frac{\partial f_\theta}{\partial \mathbf{z}}$ for 9 different shape code components and demonstrates that many parts of an object are affected, when a single component changes.

3.2. Implementation Details

Multiple techniques are employed in order to speed up the dense optimization stage. Since executing the network is the most expensive step in the optimization, the number of forward and backward passes are kept to a minimum. At the beginning of each iteration the DeepSDF volume is sampled with an adaptive resolution. First, on a dense grid with a distance of δ between samples along a every dimension and afterwards at a higher resolution for voxels close the object boundary, i.e. when $-\delta < f_\theta < \delta$, where δ is the DeepSDF truncation factor as described by Park *et al.* [7]. A mesh corresponding to the current code estimate is then obtained by running marching cubes [2] on the samples and used as a proxy for the current state. Each vertex contains $\frac{\partial f_\theta}{\partial \mathbf{x}}$ and $\frac{\partial f_\theta}{\partial \mathbf{z}}$ and as a result data required to minimize the energy in Eq. 3 is generated by rendering the mesh to observed detections and is independent of sampling. Rasterized mesh data is directly copied from OpenGL to CUDA for the optimization.

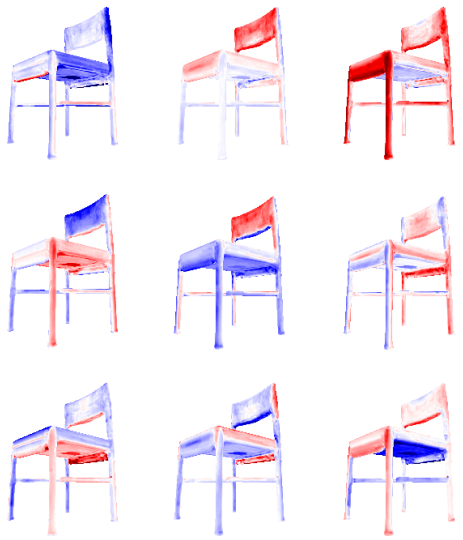


Figure 2: Visualization of $\frac{\partial f_\theta}{\partial \mathbf{z}}$ for 9 different code components, demonstrating shape changes when single components change. Red (positive values) indicates an intrusion, while blue (negative values) indicates an extrusion.

Further, a pyramid scheme is implemented which aids convergence when starting from a bad initialization, improves run-time and reduces the memory footprint of the optimization. For every detection with width w and height h, only pyramid levels l with a bounded shape ($r^2_{min} < \frac{w \cdot h}{4^l} < r^2_{max}$) are considered. In our experiments, r_{min} and r_{max} are always 40 and 400 respectively.

4. Network Architecture and Training

4.1. Decoder for Joint embedding

The shared joint embedding is with 64 dimensions. The network is split into two independent branches for point

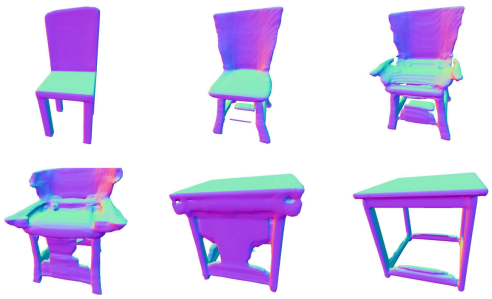


Figure 3: Interpolation of shape codes from different classes.

cloud and DeepSDF respectively. The point cloud branch is composed of four fully connected layers each with 512, 1024, 2048, 2048×3 as output dimensions. We use ReLU as a nonlinear activation function followed by each fully connected layers except for the last one. Readers can refer to [7] for detailed network architecture for the DeepSDF branch. The joint autodecoder is trained for 2000 epochs with a batch size of 64. There are 16384 SDF samples for each shape. The learning rate for network parameters and latent vectors are $5 \cdot 10^{-3}$, and 10^{-3} respectively, decayed by 0.5 for every 500 epochs.

4.2. Encoder Network

After we train the decoder network as described in Sec. 4.1, we obtain a set of embeddings $\mathbf{z} \in \mathcal{R}^{64}$ for the corresponding CAD models. In the second stage, we train an encoder network that maps an image to the latent code. We tailor ResNet50 to output a vector of dimension 64 and initialize the network with pretrained models. We train the network for 50 epochs to minimize a Huber loss with a polynomial decaying learning rate of 10^{-3}. The network for the embedding is trained in a way similar to Li *et al.* [4]. However, our deep learning based shape embedding is very different from the non-parametric embedding used in [4].

5. Latent Space Interpolation

We randomly sample three latent codes in the table and chair classes respectively. Figure 3 illustrates such an inter-class interpolation and a demonstrating video is also attached. It can be seen that the shape transition is smooth, which is suitable for our gradient based optimization. A second point worth noting is that when decoding a identical latent code the surfaces represented by the point cloud and mesh (generated by DeepSDF) are consistent.

6. CAD retrieval on Pix3D dataset

In Fig. 4 we show more examples of CAD model retrieval on Pix3D dataset [8]. We find the nearest CAD model based on Euclidean distance of our shape codes.

Method	sec. /iteration	# of iteration
PMO [5]	12.59	100
Optim. Dense	4.96	100
Optim. Sparse	0.07	200

Table 2: Speed comparison between PMO [5] and our method on the same sequence of 60 frames.

7. Dual-Representation Efficiency Gain

One motivation for using a shared code space for a point-based and SDF-based representation is that the efficiency of a sparse optimization can be leveraged, while exploiting richer information when subsequently applying fewer dense iterations. A comparison of optimization run-time shows that the pointcloud-based representation is approximately two orders of magnitude faster than the mesh optimization used by PMO [5] and DeepSDF [7] as shown in Table 2.

8. Redwood Dataset Augmentation

While Pix3d [8] is a real-world dataset available for single-view object shape reconstruction, multi-view datasets with ground truth 3D data are scarce. Previous multi-view learning based methods primarily evaluate their methods on synthetic data [5]. We post-process the Redwood-OS dataset [3] and will release it to facilitate benchmarking on real-world data. It contains RGBD sequences and dense full scene reconstructions. To isolate object shapes for evaluation, we manually select sequences that have full 3D object shapes and manually segment them from the full scene mesh. We run RGBD ORB-SLAM2 [6] to estimate camera poses.

9. Qualitative results on Redwood-OS dataset

Additional qualitative results on the Redwood-OS dataset are shown in Figure 5. As in the main paper, groundtruth, sparse COLMAP, dense COLMAP, our sparse reconstructions and our dense reconstructions are compared.

Figure 4: From left to right: input image and the nearest CAD models in latent space sorted by distances.

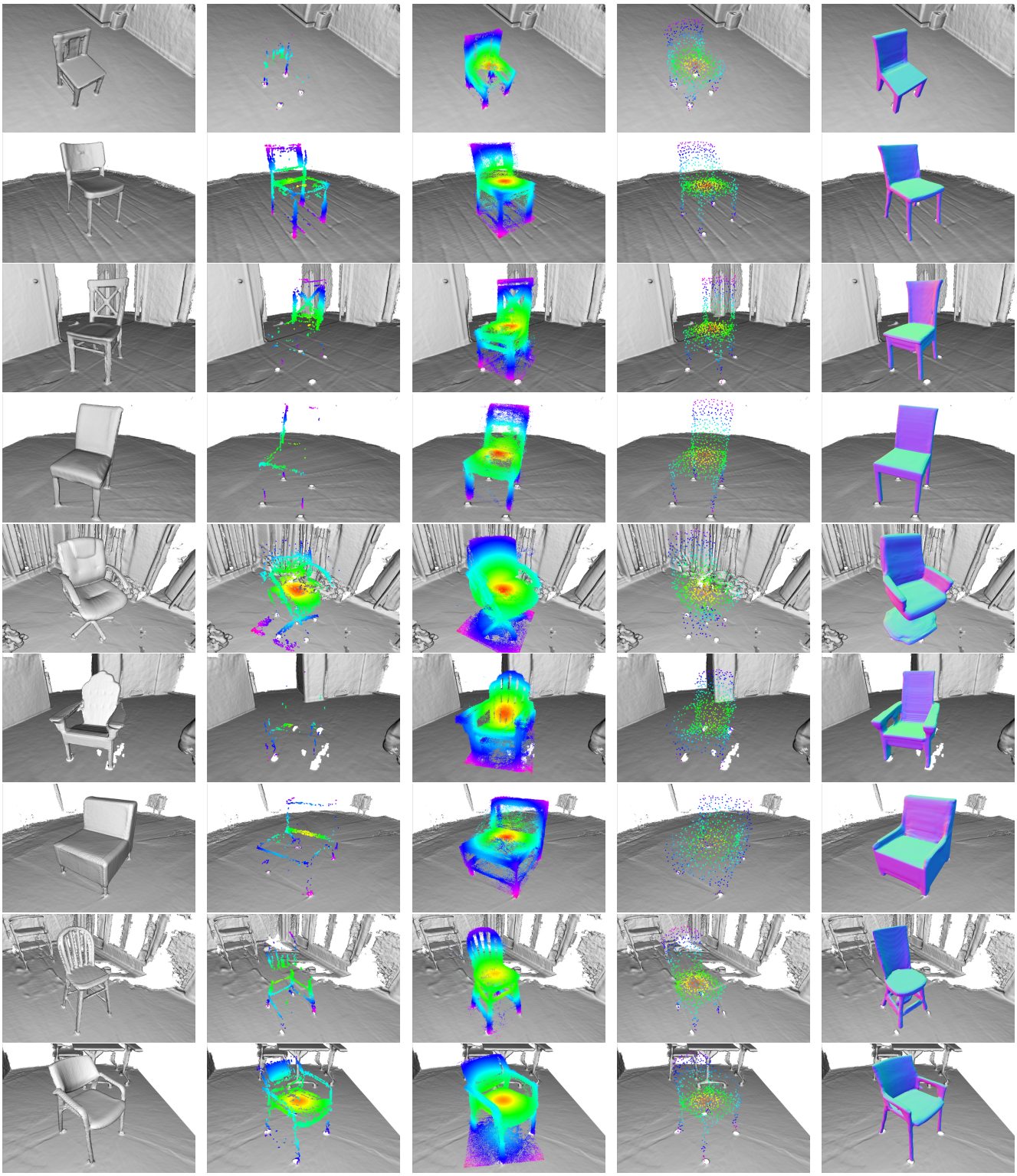


Figure 5: Comparison of different approaches to object shape reconstruction on some examples from Redwood-OS dataset.

References

[1] M. Atzmon, N. Haim, L. Yariv, O. Israelov, H. Maron, and Y. Lipman. Controlling neural level sets. In *Advances in Neural Information Processing Systems*, pages 2032–2041, 2019. 2

[2] E. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, 1995. 2

[3] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016. 3

[4] Y. Li, H. Su, C. R. Qi, N. Fish, D. Cohen-Or, and L. J. Guibas. Joint embeddings of shapes and images via cnn image purification. *ACM transactions on graphics (TOG)*, 34(6):234, 2015. 3

[5] C.-H. Lin, O. Wang, B. C. Russell, E. Shechtman, V. G. Kim, M. Fisher, and S. Lucey. Photometric mesh optimization for video-aligned 3d object reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 969–978, 2019. 3

[6] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. 3

[7] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 3

[8] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3