Supplement Materials for Unsupervised 3D Shape Reconstruction by Part Retrieval and Assembly

Xianghao Xu[1], Paul Guerrero[2], Matthew Fisher[2], Siddhartha Chaudhuri[2], Daniel Ritchie[1]
[1]Brown University (USA), [2]Adobe Research (UK, USA, India)
{xianghao_xu, daniel_ritchie}@brown.edu, {guerrero, matfishe, sidch}@adobe.com

1. Additional Implementation Details

In this section we provide additional details for our method and experiments. The code of this paper will be available at https://github.com/xxh43/PartRetrievalAndAssembly.

Method Details Please see Algorithms 1, 2 and 7 for pseudo-code of Phase I (Part Optimization), Phase II (Part Shift) and Phase III (Part Borrowing), respectively. For Phase II, we provide additional pseudo-code for helper functions $Swap()$ (Algo. 3), $NNSegment()$ (Algo. 4), $Filter()$ (Algo. 5) and $FathestConnectedComponent()$ (Algo. 6). Intuitively, these helper functions perform the following operations: Firstly, target to optimized parts distance matrix $\mathbf{Q}$ is computed, with rows(axis 0) represent points in target, and columns(axis 1) represent parts. Then in $Swap()$, the minimum distance matrix $\mathbf{Q}_{min}$ is computed by taking the minimum value of each row of $\mathbf{Q}$. The region in the target $\mathbf{T}$ that is least covered by any of the parts is extracted as $\mathbf{T}^{-}$. Then for each part $\mathbf{D}_i \in \mathcal{D}$, we compute the distance matrix $\mathbf{Q}^s$ without the effect of $\mathbf{D}_i$, and set the values of region $\mathbf{T}^{-}$ in $\mathbf{Q}^s$ to zero. The new minimum distance matrix after swap is computed as $\mathbf{Q}^s_{min}$. The part $\mathbf{D}_s$ that leads to the minimum value of $avg(\mathbf{Q}^s_{min})$ will be replaced by $\mathbf{T}^{-}$. Then in $NNSegment()$, the target point cloud $\mathbf{T}$ is segmented into a set of segments where $\mathbf{S}_i$ consists of all points from $\mathbf{T}$ which are closer to $\mathbf{D}_i$ than to any other decoded part. Then in $Filter()$, segment S_i is updated with discarding a small amount of points that are already well covered. Finally in $FarthestConnectedComponent()$, the self connectivity matrix $\mathbf{W}_{adj}$ is computed by using point to point connection when their distance is below a threshold. This connectivity matrix $\mathbf{W}_{adj}$ is used to construct the graph g, followed by the connected components $\mathcal{C}$ are extracted from the graph g and the connected component $\mathbf{C}_i$ which is farthest from any of the other segments is selected.

Algorithm 1 Phase I Part Optimization

```
 1: Input
 2:     Target shape $\mathbf{T}$
 3:     Latent codes $\mathbf{e}_i$, $i \in k$
 4:     Translation vectors $\mathbf{t}_i$, $i \in k$
 5:     Rotation angles $r_i$, $i \in k$
 6:     Pre-trained part decoder $Decode$
 7: Output
 8:     Updated latent codes $\mathbf{e}_i$, $i \in k$
 9:     Updated translation vectors $\mathbf{t}_i$, $i \in k$
10:     Updated rotation angles $r_i$, $i \in k$
11: procedure Optimize
12:     $\mathbf{D}_i \leftarrow Decode(\mathbf{e}_i)$
13:     $\mathbf{D}_i \leftarrow Pose(\mathbf{D}_i, \mathbf{t}_i, r_i)$
14:     $\mathcal{L} \leftarrow \mathcal{L}_{recon}(\mathcal{D}, \mathbf{T}) + \mathcal{L}_{overlap}(\mathcal{D})$
15:     $\mathbf{e}_i \leftarrow \mathbf{e}_i + \nabla\mathcal{L}(\mathbf{e}_i)$
16:     $\mathbf{t}_i \leftarrow \mathbf{t}_i + \nabla\mathcal{L}(\mathbf{t}_i)$
17:     $r_i \leftarrow r_i + \nabla\mathcal{L}(r_i)$
18:     return $\mathbf{e}_i, \mathbf{t}_i, r_i$
19: end procedure
```

Dataset Preprocessing Our method takes volumetric point clouds as input. In our experiments, we use the meshes provided in PartNet dataset to generate the corresponding point clouds. In the dataset preprocessing, all target shapes are centered to the origin. All parts in the part library are centered to the origin and rotated so the axes of their minimum volume bounding boxes align with the world axes. We then applied a hole filling method [1] to make all target shape meshes and part meshes watertight and applied rejection sampling to generate the volumetric point clouds for both target shapes and parts. For detecting symmetry planes for each shape, we iterate over several candidate planes that are perpendicular to xz plane that are centered at the object center. Then the points on one size of the plane are reflected to the other size, if the reflected points are overlap with the points on the other size, the symmetry plane is found. The overlap is determined by a threshold

Algorithm 2 Phase II Part Shift

```
Input
    Target shape $\mathbf{T}$
    Latent codes $\mathbf{e}_i$, $i \in k$
    Translation vectors $\mathbf{t}_i$, $i \in k$
    Rotation angles $r_i$, $i \in k$
    Pre-trained part decoder $Decode$
    Pre-trained part encoder $Encode$
Output
    Updated latent codes $\mathbf{e}_i$, $i \in k$
    Updated translation vectors $\mathbf{t}_i$, $i \in k$
    Updated rotation angles $r_i$, $i \in k$

procedure Shift
    $\mathbf{D}_i \leftarrow Decode(\mathbf{e}_i)$
    $\mathbf{D}_i \leftarrow Pose(\mathbf{D}_i, \mathbf{t}_i, r_i)$
    $\mathbf{Q} \leftarrow cdist(\mathbf{T}, \mathcal{D})$
    $\mathbf{D}_i \leftarrow Swap(\mathbf{D}_i, \mathcal{D}, \mathbf{T}, \mathbf{Q})$
    $\mathbf{S}_i \leftarrow NNSegment(\mathbf{D}_i, \mathbf{T}, \mathbf{Q})$
    $\mathbf{S}_i \leftarrow Filter(\mathbf{S}_i, \mathbf{T}, \mathbf{Q})$
    $\mathbf{C}_i \leftarrow FarthestConnectedComponent(\mathbf{S}_i, \mathcal{S}_{\neq i})$
    $\mathbf{t}_i \leftarrow Center(\mathbf{C}_i)$
    $r_i \leftarrow Rotation(\mathbf{C}_i)$
    $\mathbf{C}_i \leftarrow Pose(\mathbf{C}_i, -\mathbf{t}_i, -r_i)$
    $\mathbf{e}_i \leftarrow Encode(\mathbf{C}_i)$
    return $\mathbf{e}_i, \mathbf{t}_i, r_i$
end procedure

```

Algorithm 3 Phase II helper Swap()

```

procedure Swap($\mathbf{D}_i, \mathcal{D}, \mathbf{T}, \mathbf{Q}$)
    $\mathbf{Q}_{min} \leftarrow min(\mathbf{Q}, axis = 1).val$
    $\mathbf{T}^- \leftarrow \mathbf{T}[ind(topK(\mathbf{Q}_{min}))]$
    $d_{min} \leftarrow avg(\mathbf{Q}_{min})$
    $\mathbf{D}_s \leftarrow None$
    for $\mathbf{D}_i \in \mathcal{D}$ do
        $\mathbf{Q}^s \leftarrow (\mathbf{Q}[:, 0 : i-1] \cup \mathbf{Q}[:, i+1 :])$
        $\mathbf{Q}^s_{min} \leftarrow min(\mathbf{Q}^s, axis = 1).val$
        $\mathbf{Q}^s_{min}[ind(\mathbf{T}^-)] \leftarrow 0$
        $d \leftarrow avg(\mathbf{Q}^s_{min})$
        if $d < d_{min}$ then
            $d_{min} \leftarrow d$
            $\mathbf{D}_s \leftarrow \mathbf{D}_i$
        end if
    end for
    $\mathbf{D}_s \leftarrow \mathbf{T}^-$
    return $\mathbf{D}_i$
end procedure
```

Algorithm 4 Phase II helper NNSegment()

```

procedure NNSegment($\mathbf{D}_i$, $\mathbf{T}$, $\mathbf{Q}$)
    $\mathbf{S}_i \leftarrow \mathbf{T}[min(\mathbf{Q}, axis = 1).ind == i]$
    return $\mathbf{S}_i$
end procedure

```

Algorithm 5 Phase II helper Filter()

```

procedure Filter($\mathbf{S}_i$, $\mathbf{Q}$)
    $\mathbf{S}_i \leftarrow \mathbf{S}_i[ind(topK(\mathbf{Q}[:, i]))]$
    return $\mathbf{S}_i$
end procedure

```

Algorithm 6 Phase II helper FarthestConnectedComponent()

```

procedure FarthestConnectedComponent($\mathbf{S}_i$, $\mathcal{S}_{\neq i}$)
    $\mathbf{W} \leftarrow cdist(\mathbf{S}_i, \mathbf{S}_i)$
    $\mathbf{W}_{adj} = \mathbf{W}[\mathbf{W} < \tau]$
    $g \leftarrow Graph(\mathbf{W}_{adj})$
    $\mathcal{C} \leftarrow g.connected\_components$
    $d_{max} \leftarrow 0$
    for $\mathbf{C} \in \mathcal{C}$ do
        $d \leftarrow avg(cdist(Center(\mathbf{C}), \mathcal{S}_{\neq i}))$
        if $d > d_{max}$ then
            $d_{max} \leftarrow d$
            $\mathbf{C}_i \leftarrow \mathbf{C}$
        end if
    end for
    return $\mathbf{C}_i$
end procedure

```

one for category with relative sparse point clouds such as Table and one for categories with relative dense point cloud such as Faucet and Lamp.

Experiments For Phase I optimization, we use the Adam optimizer [2] with learning rate 0.008. For NP, the model that performs best on evaluation set during training is used to evaluate both train and test performance.

Part VAE Architectures Please see Table 1 and Table 2 for architecture of the Part Encoder and Part Decoder.

2. Additional Ablation Experiments

We conducted several additional ablation studies on the Faucet category to investigate more about our method.

Algorithm 7 Phase III Part Borrowing

```
1:  Input
2:      Target reconstruction errors: $h_i, i \in N$
3:      Target to Target distance matrix $\mathbf{M}$
4:  Output
5:      Updated latent codes $\mathbf{e}_i, i \in k$
6:      Updated translation vectors $\mathbf{t}_i, i \in k$
7:      Updated rotation angles $r_i, i \in k$
8:  procedure Borrow
9:      $nb \leftarrow -1$
10:     for $j \in ind(botK(\mathbf{M}[i,:]))$ do
11:         if $h_i[j] \leq \epsilon$ then
12:             $nb \leftarrow j$
13:             $break$
14:         end if
15:     end for
16:     if $nb \geq 0$ then
17:         $\mathbf{e}_i \leftarrow \mathbf{e}_{nb}$
18:         $\mathbf{t}_i \leftarrow \mathbf{t}_{nb}$
19:         $r_i \leftarrow r_{nb}$
20:     else
21:         $\mathbf{e}_i \leftarrow rand()$
22:         $\mathbf{t}_i \leftarrow rand()$
23:         $r_i \leftarrow rand()$
24:     end if
25:     return $\mathbf{e}_i, \mathbf{t}_i, r_i$
26: end procedure
```

Part Encoder
Conv1d $(3, 32, 1)$
Batchnorm1d
LeakyRelu
Conv1d $(32, 64, 1)$
Batchnorm1d
LeakyRelu
Conv1d $(64, 64, 1)$
Batchnorm1d
LeakyRelu
Conv1d $(64, 64, 1)$
Batchnorm1d
LeakyRelu
MaxPool
FC(64×64)

Table 1. Detailed architecture of the Part Encoder

Part library size We investigate the influence of varying the size of the input part library. We cluster all parts according to the shape similarity, then we randomly choose certain number of clusters combined as the part library for our experiments. Please see Table 3 and Figure 1 for the results. As you can see, with fewer parts available in the part library, the performance of both our method and Neural Parts (NP) drops, but our method drops more gracefully (slower) than Neural Parts (NP) does. We think this shows our method can effectively take advantage of its awareness of the available parts.

Part Decoder
FC(64×512)
Batchnorm1d
LeakyRelu
FC(512×512)
Batchnorm1d
LeakyRelu
FC(512×1024)
Batchnorm1d
LeakyRelu
FC(1024×1024)
Batchnorm1d
LeakyRelu
FC$(1024 \times (512 * 3))$
Reshape

Table 2. Detailed architecture of the Part Decoder

Method	**Train (SCD)** $\downarrow$	**Train (VCD)** $\downarrow$	**Test (SCD)** $\downarrow$	**Test (VCD)** $\downarrow$
NP (107 parts)	0.668	0.394	0.623	0.353
Ours (107 parts)	**0.382**	**0.201**	**0.437**	**0.225**
NP (432 parts)	0.488	0.274	0.488	0.256
Ours (432 parts)	**0.302**	**0.156**	**0.340**	**0.163**
NP (789 parts)	0.326	0.171	0.370	0.174
Ours (789 parts)	**0.256**	**0.135**	**0.288**	**0.134**

Table 3. Part library size ablation (Faucet). Note: numbers are multiplied by 100.

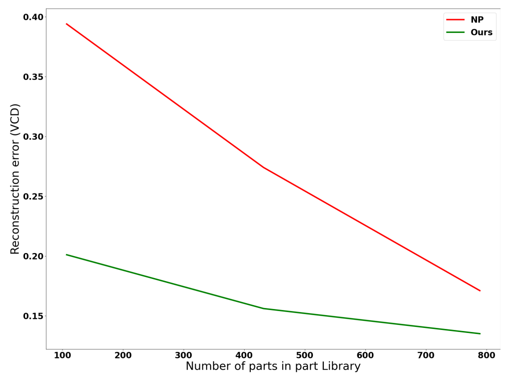

Figure 1. Part library size ablation (training target statistics).

Retrieval candidate part number We investigate the influence of varying size of the set of the parts considered as candidates for retrieval. In our main experiments, for all methods, we iterate over the entire part library to find the

Method	Train (SCD) ↓	Train (VCD) ↓	Test (SCD) ↓	Test (VCD) ↓
NP (100% of all parts)	0.326	0.171	0.370	0.174
Ours (5% of all parts)	0.271	0.142	0.337	0.162
Ours (25% of all parts)	0.261	0.137	0.297	0.143
Ours (100% of all parts)	0.256	0.135	0.288	0.134

Table 4. Retrieval part candidate number ablation (Faucet). Note: numbers are multiplied by 100.

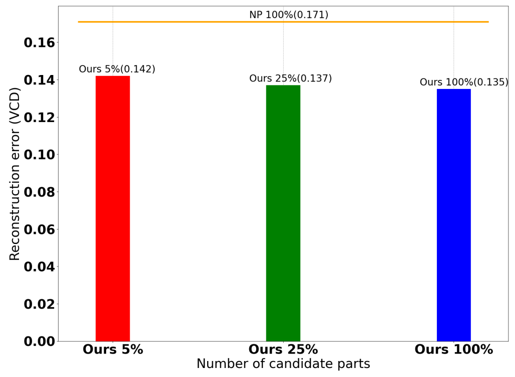

Figure 2. Retrieval part candidate number ablation (training target statistics).

part that fits best to each segment in the target as the final part retrieval. This is because for Neural Parts (NP) and Brute Force (BF), there is no information to tell us which subsets of all parts to focus as the retrieval candidate parts. However, in our method, we have a part encoding latent space and for each optimized part we have an optimized part latent code. Our method can take advantage of this to just iterate and retrieve a subset of parts that are in the vicinity of the optimized part latent code in the latent space. We conducted the experiments with iterating only the nearest 5%, 25% of the entire part library for the final retrieval. Please see Table 4 and Figure 2 for the results. As you can see, even with much smaller amount of part retrieval candidates, our method still outperforms NP. This proves (1) the effectiveness of our latent space and (2) the potential speed-up advantage of our method when dealing with large input part libraries.

Training target number We investigate the influence of varying size of the set of input training target shapes on our amortized inference procedure. Please see Table 5 and Figure 3 for the results. As you can see, the performance of amortized inference does not drop much with fewer training target shapes.

3. Failure Cases

Figure 4 shows some failure patterns we identified for our method: (1) Insufficient points sampled on thin regions of the target shape; (2) Sub-optimal hyper-parameters, e.g.

Method	Test (SCD) ↓	Test (VCD) ↓
NP (250 train shapes)	0.370	0.174
Ours (10 train shapes)	0.297	0.150
Ours (100 train shapes)	0.292	0.140
Ours (250 train shapes)	0.288	0.134

Table 5. Training target number ablation (Faucet). Note: numbers are multiplied by 100.

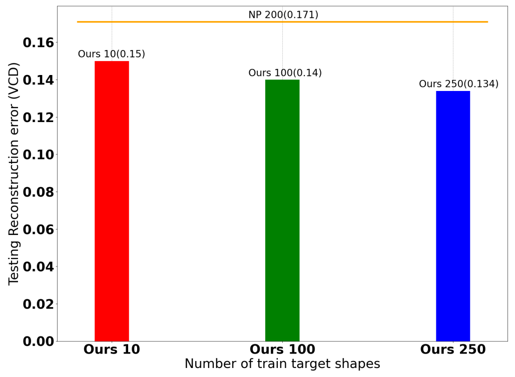

Figure 3. Training target number ablation.

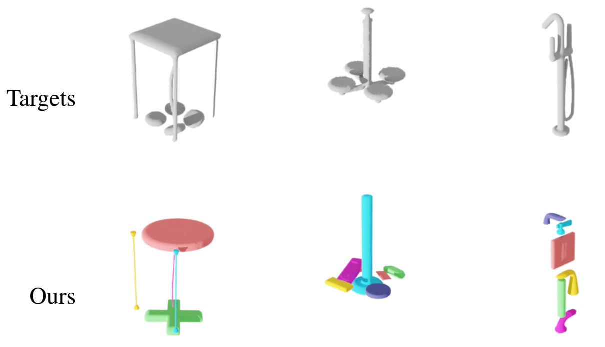

Figure 4. Failure Cases

the shape is not judged as a symmetrical shape so that the symmetry constraint is not applied, and the connected component selection threshold in Phase II is too large so that disconnected geometry is the target is not successfully represented by different parts; (3) complex cluttered geometry details are missed due to the limitation of the point cloud representation (the small scale cluttered geometry details are represented by several points during the optimization).

4. Model-based Inference

We also tried to use neural networks to perform fast inference, but we found that it did not perform as well as the amortized inference used in our method. We designed a Re-

Method	**Test (SCD) ↓**	**Test (VCD) ↓**
NP	0.370	0.174
Model Infer	0.370	0.185
Ours	0.288	0.134

Table 6. Comparison results NP vs. Model Infer vs. Ours (Faucet). Note: numbers are multiplied by 100.

trievalNet which takes in a target shape point cloud and outputs k part latent codes $\mathbf{e}_i$ in the latent space, together with an AssembleNet which takes in a target shape point cloud and a part latent code $\mathbf{e}_i$ and outputs a translation vector $\mathbf{t}_i$ and a rotation angle r_i for part i. These networks are trained to directly regress the latent code, translation vector and rotation angle that we acquired from our direct optimization on the training target shapes. Then inference is performed on testing target shapes.

See Figure 5 for an overview of this inference mode; see Table 6 for the reconstruction results on testing shapes from the Faucet category. The model based inference results are not as good as our amortized inference. We think the reason might be: each part output slot of the RetrievalNet plays a fixed role (for example, part i is always made to reconstruct similar region on every different target shape.), so that the direct regression of high dimensional vectors which are independently optimized (can play different roles) can be difficult. Thus, it cannot properly handle large structural difference across target shapes.

5. Additional experiment results

Please see Figure 6 and Figures 7 for additional JRD vs. Ours qualitative comparison results. Please see Figure 8–22 for additional NP vs. Ours qualitative comparison results. Please see Figures 23 and 24 for additional qualitative cross-category reconstruction results.

References

[1] Jingwei Huang, Hao Su, and Leonidas Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018. 1

[2] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR 2015*, 2015. 2

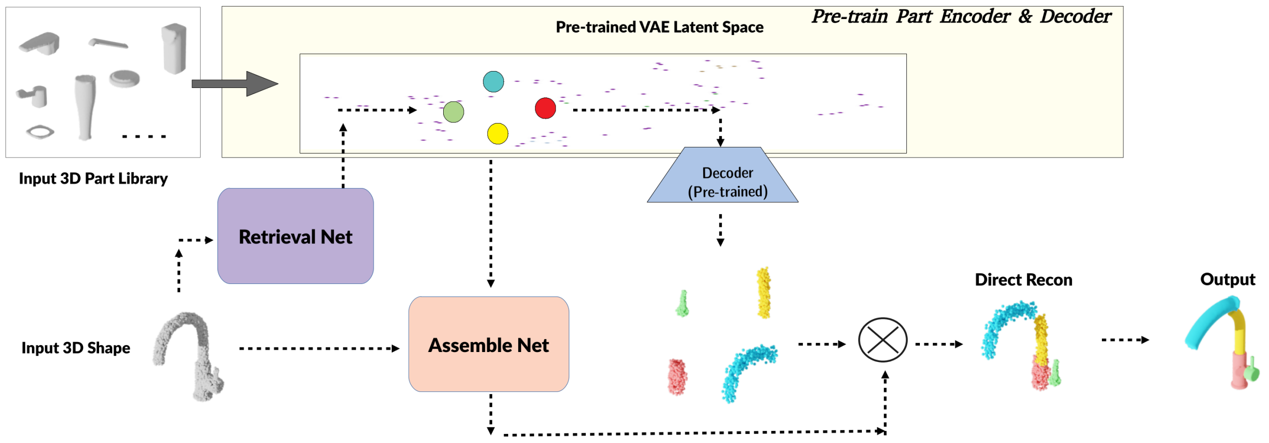

Figure 5. Model based Inference Overview: RetrievalNet takes in a target point cloud $\mathbf{T}$ and output k part latent codes $\mathbf{e}_i, i \in k$ in the latent space. AssembleNet takes in a target shape point cloud $\mathbf{T}$ and a part latent code $\mathbf{e}_i$ to output a translation vector $\mathbf{t}_i$ and a rotation angle r_i for part i. The decoded parts are translated and rotated according to $\mathbf{t}_i$ and r_i. The same segmentation and retrieval operations as our method are performed to generate the final output.

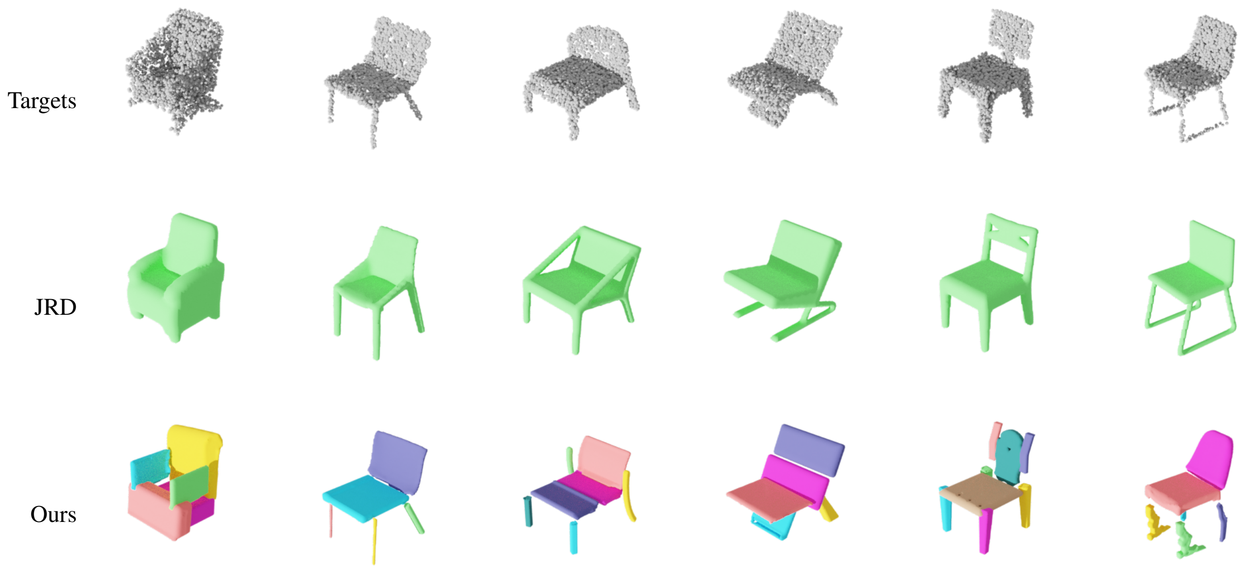

Figure 6. JRD vs. Ours on testing Chair shapes (1)

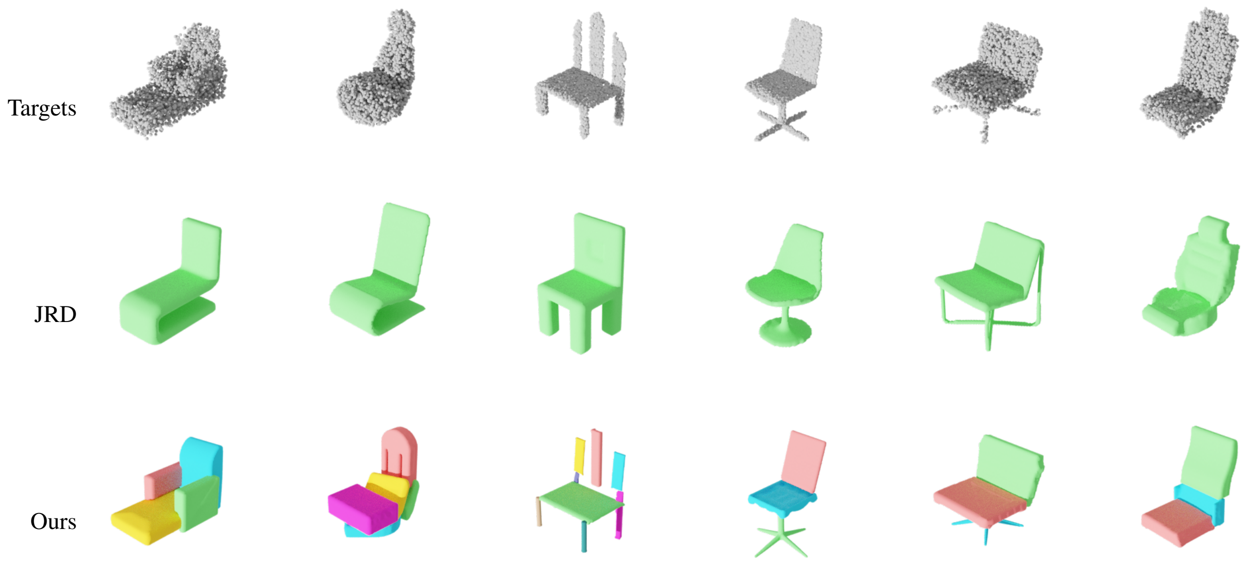

Figure 7. JRD vs. Ours on testing Chair shapes (2)

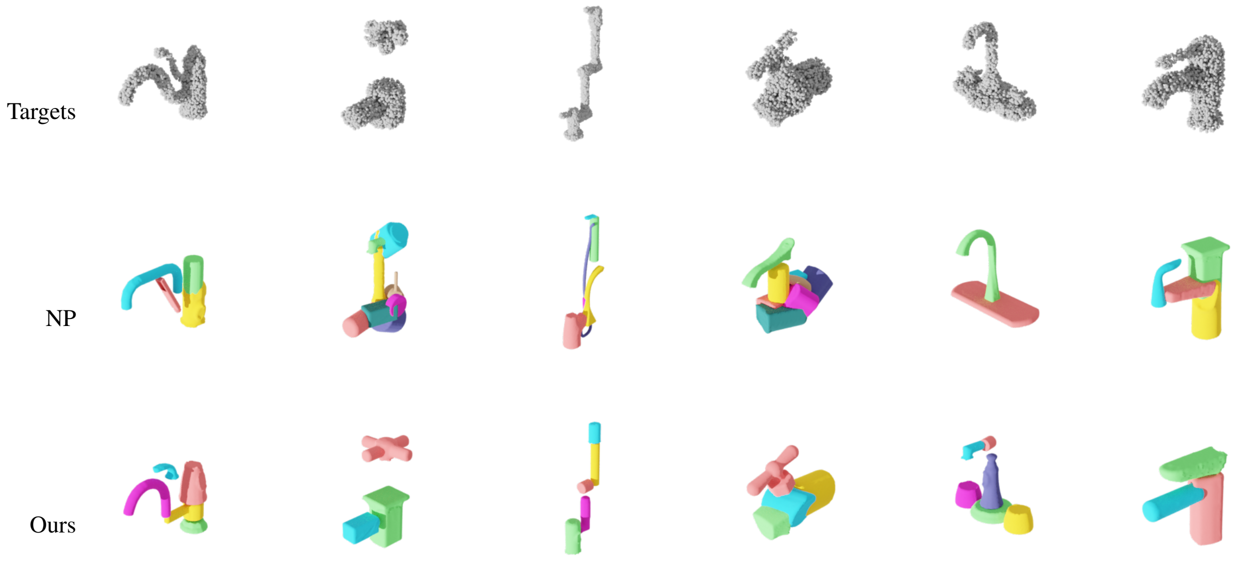

Figure 8. NP vs. Ours on Faucet category (1)

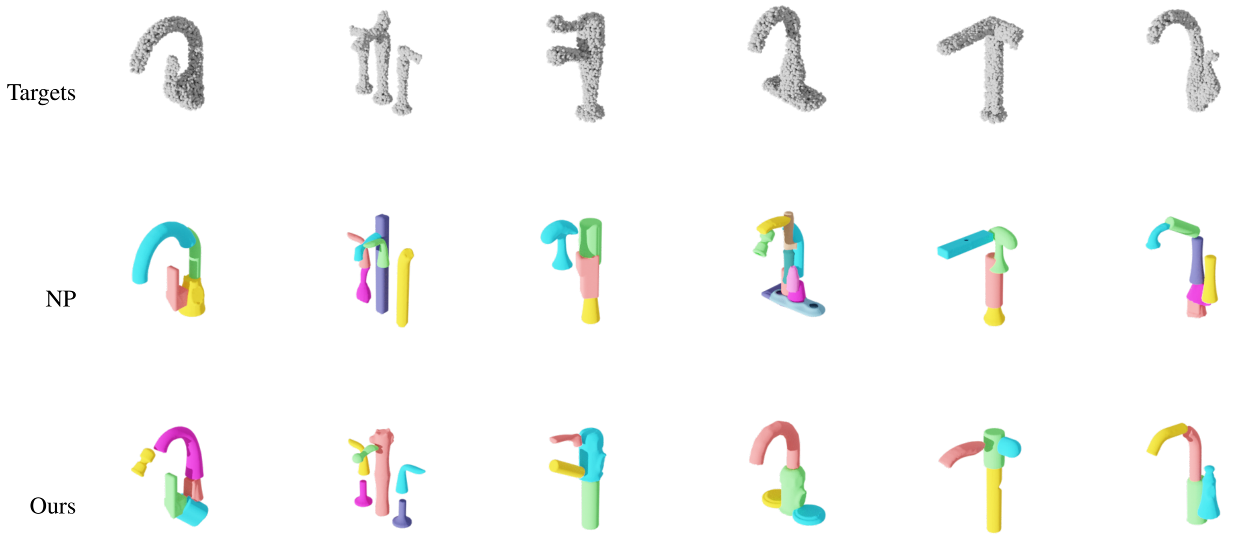

Figure 9. NP vs. Ours on Faucet category (2)

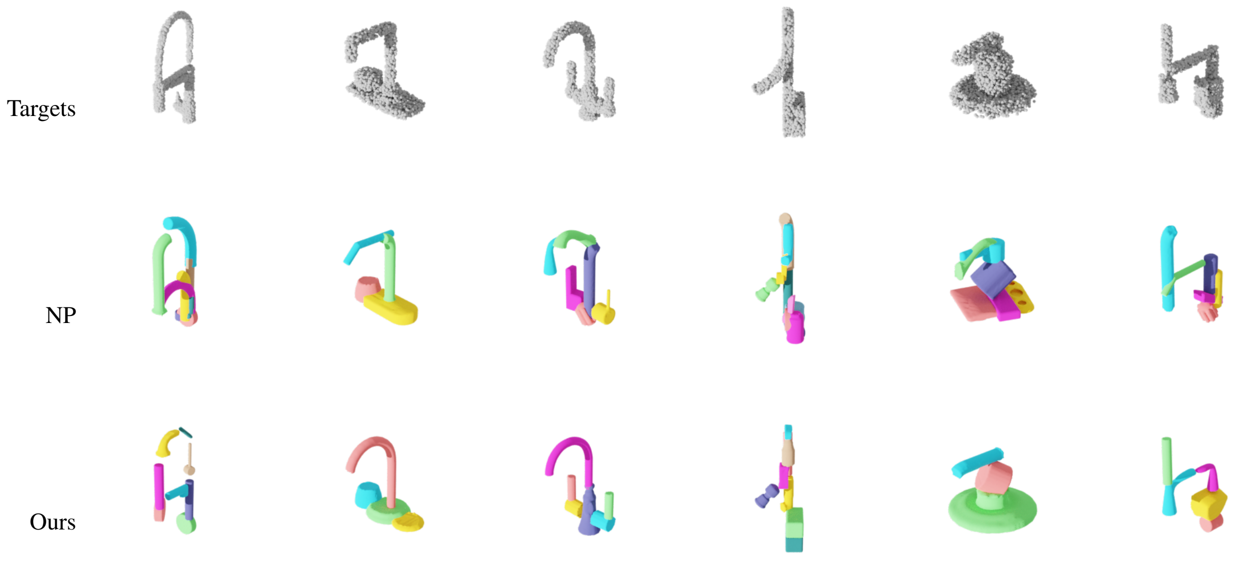

Figure 10. NP vs. Ours on Faucet category (3)

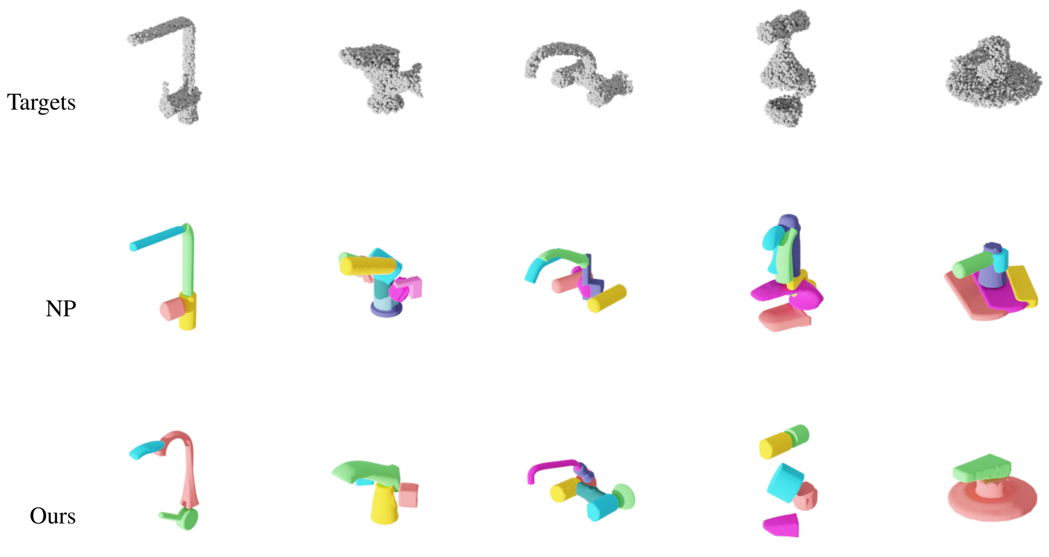

Figure 11. NP vs. Ours on Faucet category (4)

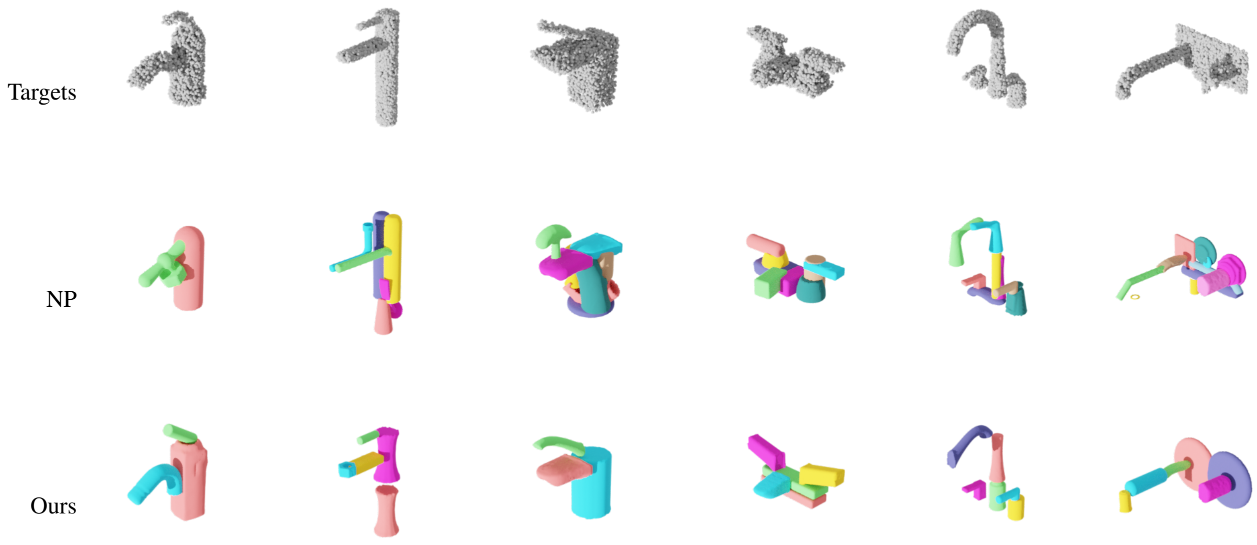

Figure 12. NP vs. Ours on Faucet category (5)

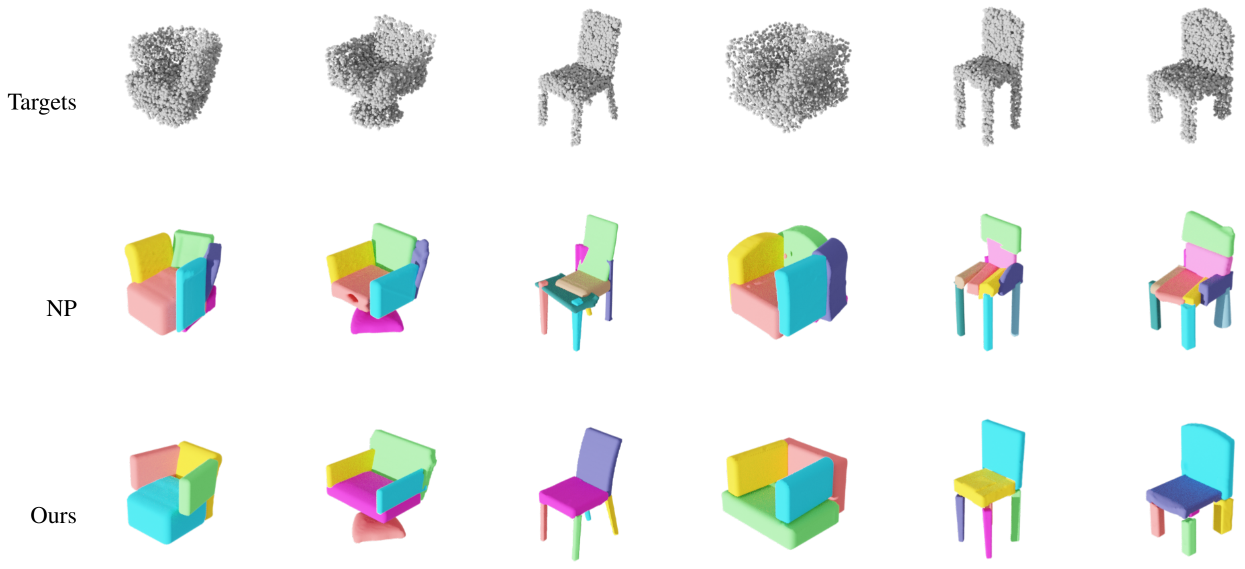

Figure 13. NP vs. Ours on Chair category (1)

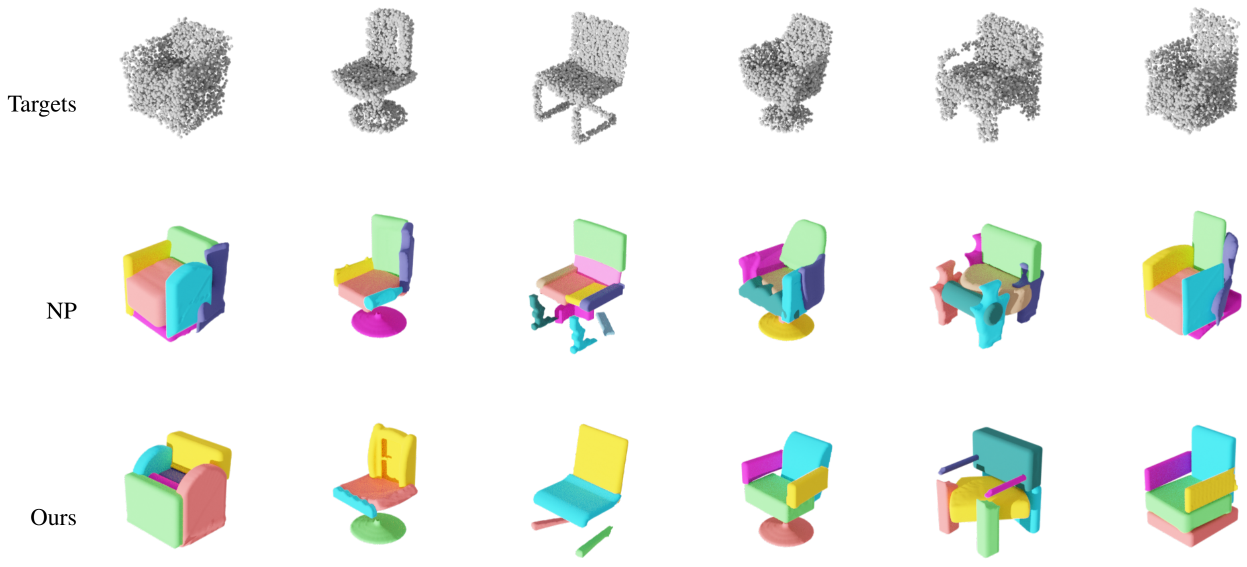

Figure 14. NP vs. Ours on Chair category (2)

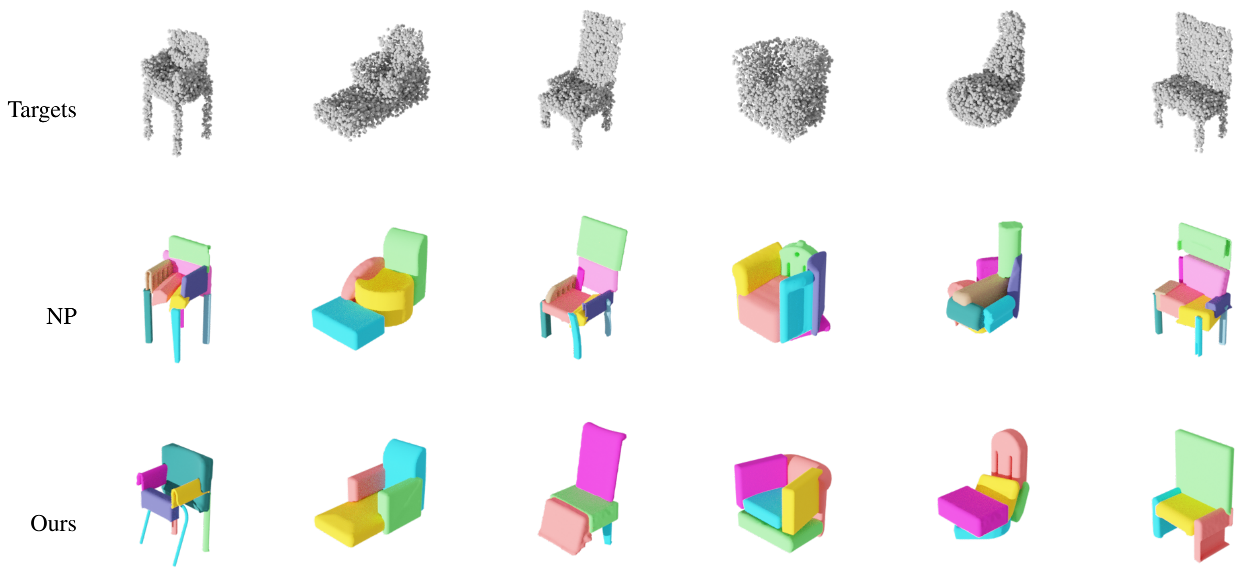

Figure 15. NP vs. Ours on Chair category (3)

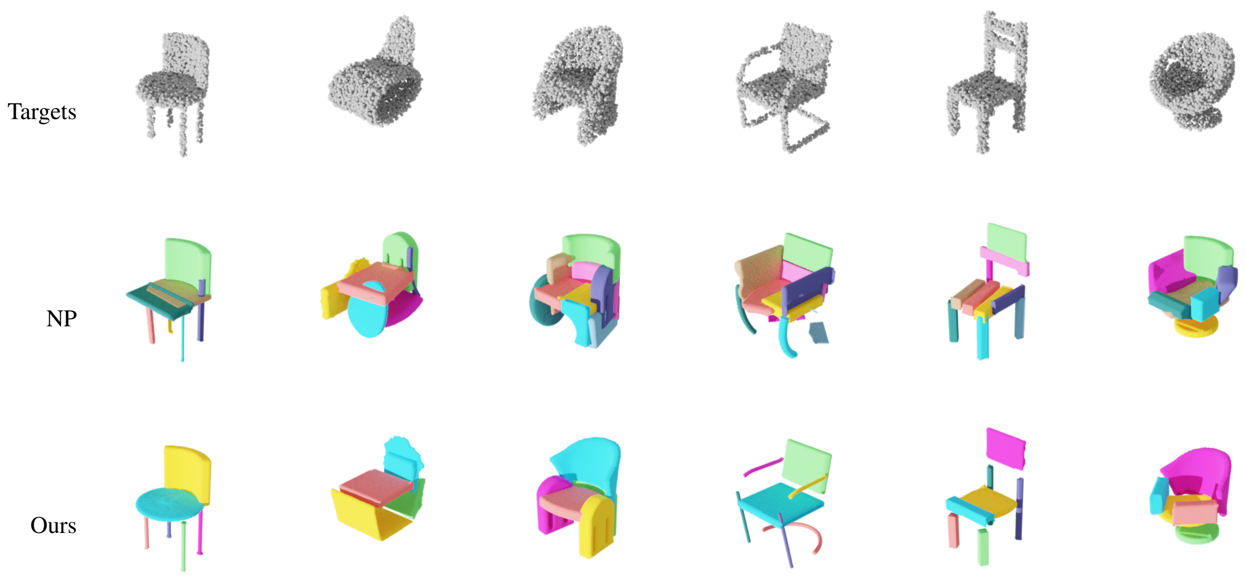

Figure 16. NP vs. Ours on Chair category (4)

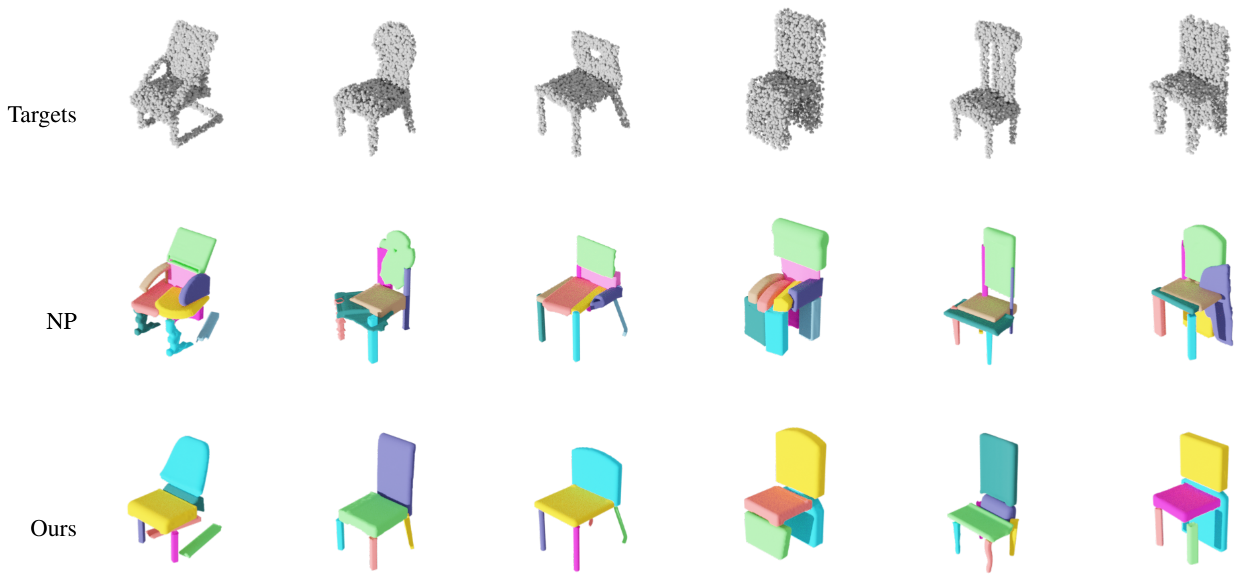

Figure 17. NP vs. Ours on Chair category (5)

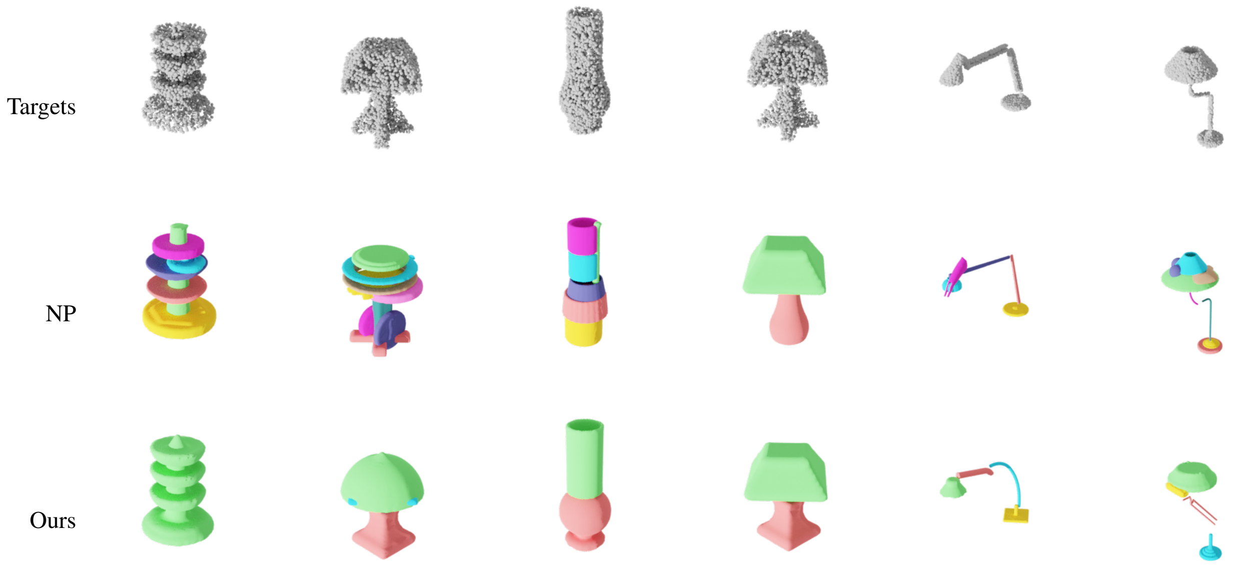

Figure 18. NP vs. Ours on Lamp category (1)

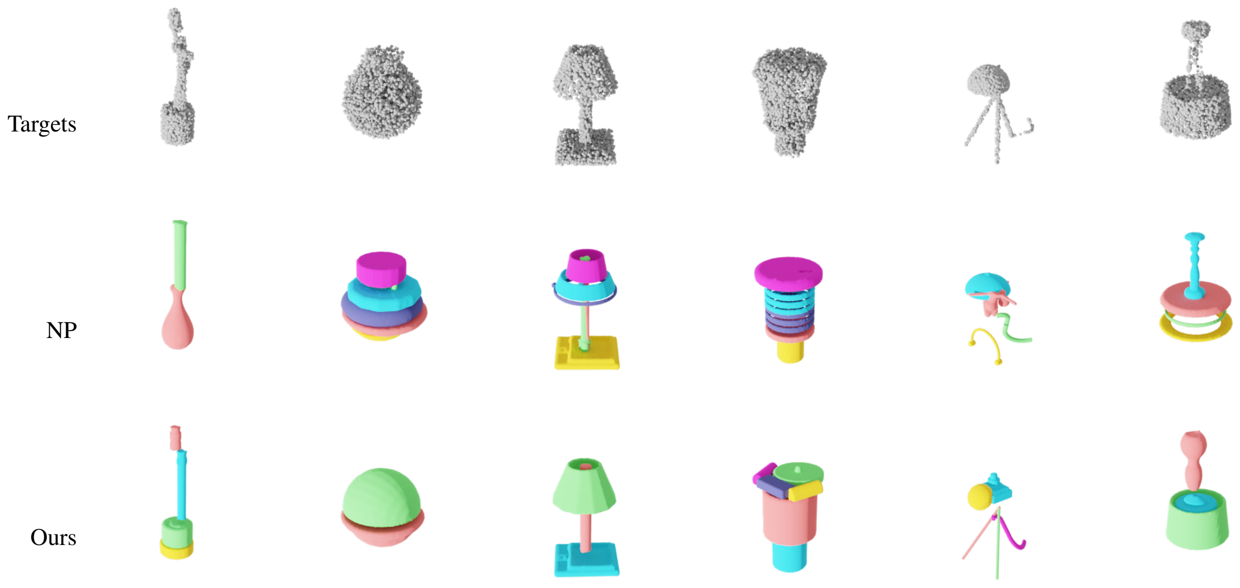

Figure 19. NP vs. Ours on Lamp category (2)

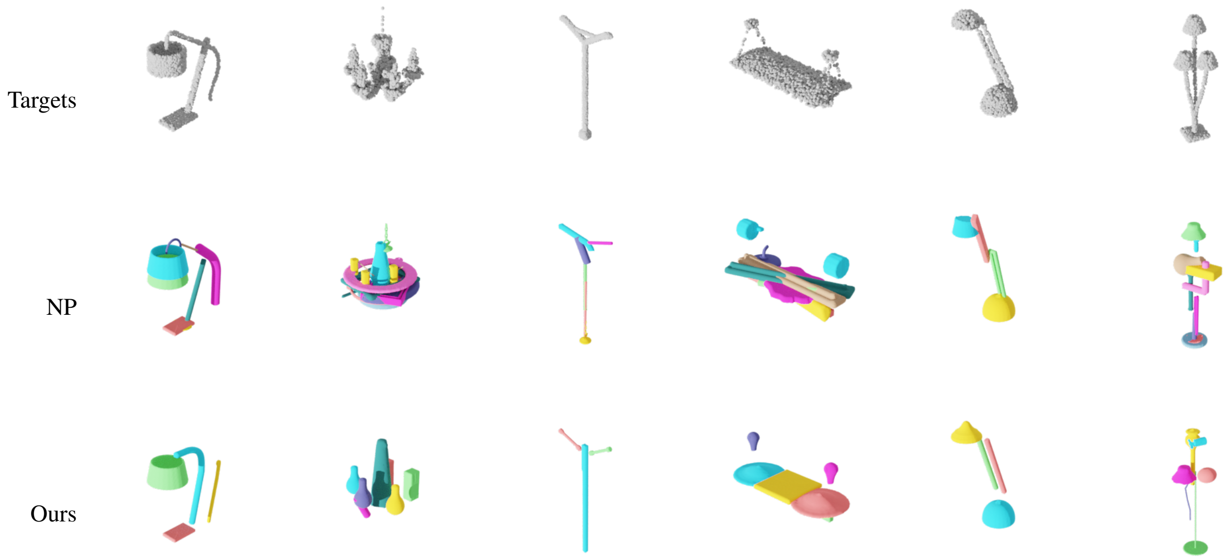

Figure 20. NP vs. Ours on Lamp category (3)

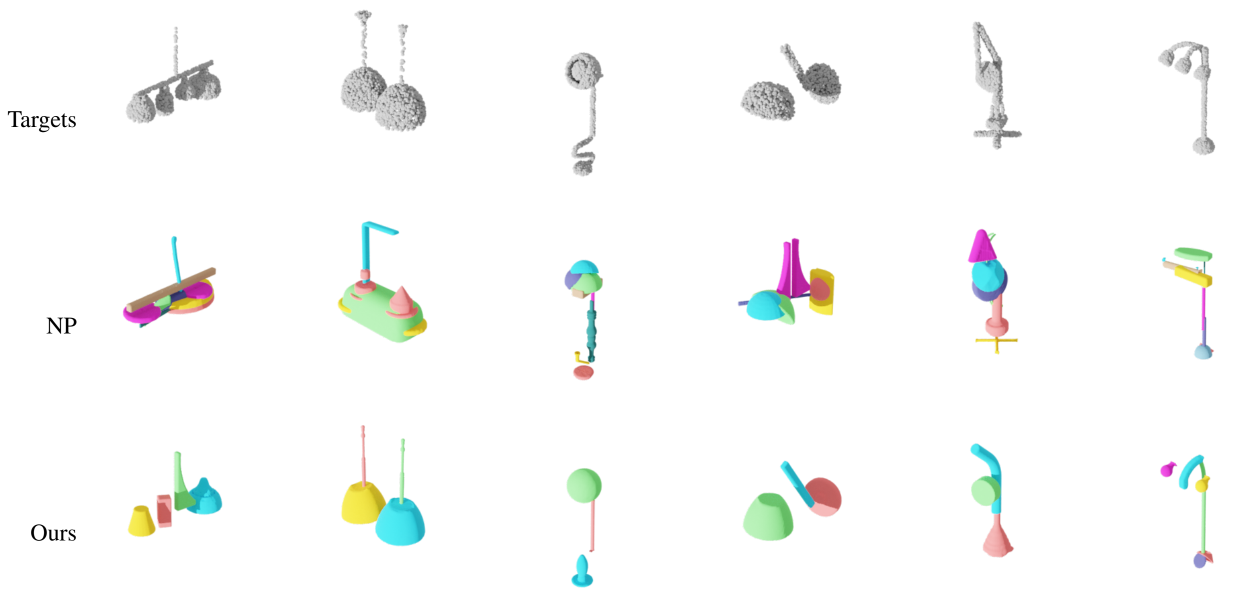

Figure 21. NP vs. Ours on Lamp category (4)

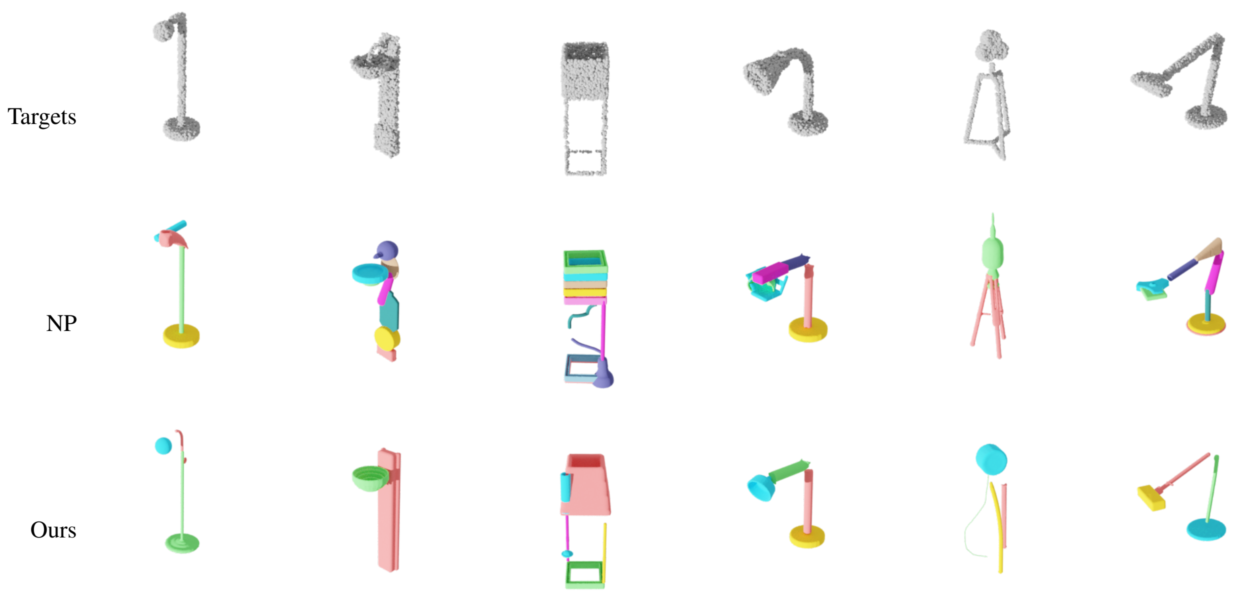

Figure 22. NP vs. Ours on Lamp category (5)

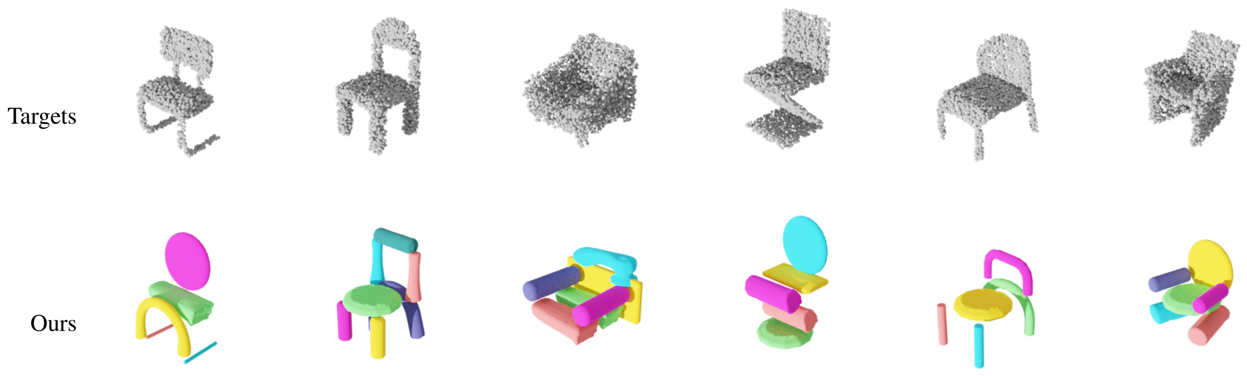

Figure 23. Faucet Parts to Chairs

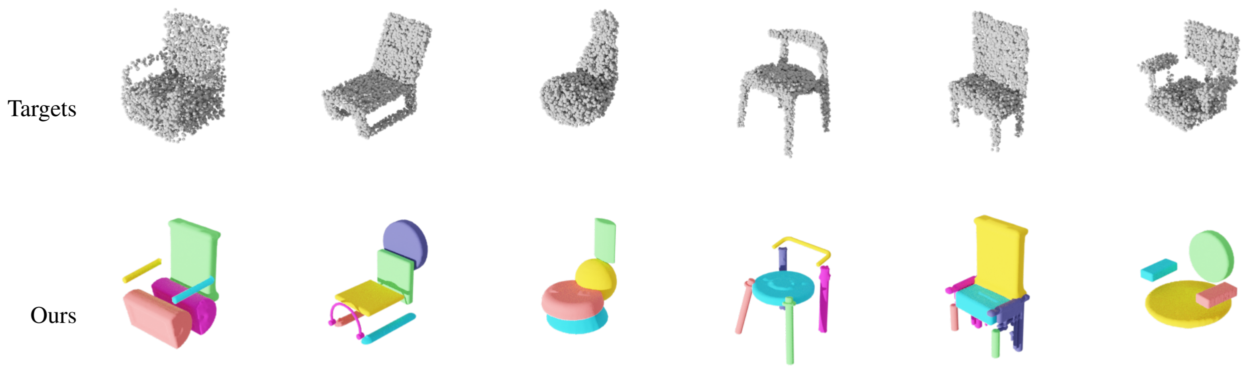

Figure 24. Lamp Parts to Chairs