Learning Convex Decomposition via Feature Fields

Supplementary Material

This document supplements the main paper *Learning Convex Decomposition via Feature Fields*. In particular, we provide additional implementation details (Sec. A), baseline comparison details (Sec. B), runtime comparison with baselines (Sec. C), an alternative training objective ablation (Sec. D) and more qualitative results on our proposed method (Sec. E).

A. Implementation Details

A.1. Triplet Sampling

Positives and negatives for our contrastive loss are sampled on the fly during training. For each shape, we first run marching cubes on a precomputed 256^3 signed distance function grid to obtain a watertight mesh. We then uniformly sample 1024 surface points as anchor points. For each anchor point, we cast 64 random rays into the hemisphere opposite the surface normal to obtain 64 positive candidate pairs. Negative candidates are generated via rejection sampling over the surface points of the entire shape until we obtain 1024 valid negative samples for each anchor. Triplets are then constructed by pairing positives and negatives that share the same anchor point. Specifically, for each anchor point we sample one positive pair (x,p) uniformly from the 64 positive candidates. Each positive pair is then matched with 512 negatives $(x, \{n\})$: 256 sampled uniformly and 256 selected via hard-negative mining from the pool of 1024 negative candidates. This produces triplets $(x, p, \{n\})$, and the contrastive loss is defined as:

$$\mathcal{L} = -\frac{1}{2}\left[\log\left(\frac{\operatorname{sim}\big(f(x), f(p)\big)}{\operatorname{sim}\big(f(x), f(p)\big) + \sum_n \operatorname{sim}\big(f(x), f(n)\big)}\right) + \log\left(\frac{\operatorname{sim}\big(f(p), f(x)\big)}{\operatorname{sim}\big(f(p), f(x)\big) + \sum_n \operatorname{sim}\big(f(p), f(n)\big)}\right)\right] \tag{7}$$

We utilize Embree [55] via pyembree to accelerate ray–mesh intersection.

A.2. Input Preprocessing and Clustering

For all experiments except those using 3D Gaussian splats as input, we preprocess each shape into a manifold, watertight mesh following the protocol of [56]. Clustering is then performed using agglomerative clustering from SciPy [54]. For experiments that use 3D Gaussian splats as input, we instead operate directly on point features using K-means clustering.

A.3. Metric Definitions

For easy comparison, we adopt the definition of the concavity metric from [58] as:

$$\text{Concavity}(S) = \max\left(H(\partial S,\, \partial\,\text{hull}(S)),\; H(\text{Vol}(S),\, \text{Vol}(\text{hull}(S)))\right) \tag{8}$$

where $H(\cdot)$ denotes the Hausdorff distance, ∂S is the surface of S, and $\text{Vol}(S)$ denotes points sampled in its interior. We uniformly sample 20,000 points per component to compute this metric. For reconstruction metric, we measure mean surface Chamfer distance between original shape and the union of convex hull, we also sample 20,000 points to calculate this metrics.

B. Baseline Comparisons Details

BSP-Net [12]. We use the official PyTorch codebase and the released checkpoint trained on ShapeNet. In addition, we train the network on the Objaverse subset used by our model. Following the official training protocol, we train the auto-encoding network (voxel input) for 8M steps at voxel resolutions 16 and 32, and for 16M steps at resolution 64 during Phase 0. We then train for another 16M steps at resolution 64 for Phase 1. We use the default network architecture and optimizer settings.

Cvx-Net [16]. We use the official TensorFlow implementation. Since no pretrained checkpoint is available, we train the network separately on ShapeNet and on our Objaverse subset. The model takes 20 depth images as input. Following their original protocol, for each shape, we sample random camera positions uniformly on a sphere of radius 1.5 centered at the origin and render depth maps at a resolution of 224×224. We train the network for 1M steps.

VHACD [36]. We run VHACD using the implementation in the PyBullet[13] package. All shapes are processed with a 1M-voxel approximation. Like our approach, the algorithm allows the user to select a threshold to control the granularity of the decomposition; finer decomposition granularity generally improves quality metrics at the expense of generating more components. For fair comparison, we follow a similar approach to [58] in selecting the threshold value and manually search for a suitable value that roughly results in the same granularity, i.e. the same number of resulting hulls. For Table 1 and Table 2, we use a fixed threshold of 0.01.

CoACD [58]. We use their official implementation in our comparisons. Similar to VHACD, the algorithm requires specifying a threshold to control the granularity of the decomposition; finer decomposition granularity generally improves

	VHACD	CoACD	CVX	BSP	Ours
time (s)	16.11	25.86	1.46	1.02	13.32
memory (GB)	0.5	5.5	1.78	1.38	2.61
GPU memory (GB)	-	-	2.08	1.65	3.61
model size (Mparams)	-	-	44	39	106

Table 3. Per-shape computation cost on VHACD dataset

	VHCD Dataset			Objaverse-Tiny		
	#comp	conc.	recon.	#comp	conc.	recon.
Alter. objective	15.24	0.1243	0.0260	27.36	0.1653	0.0303
Ours	**11.90**	**0.0973**	**0.0180**	**21.18**	**0.1257**	**0.0254**

Table 4. Quantitative ablation on alternative training objectives.

quality metrics at the expense of generating more components. For fair comparison, we follow their evaluation protocol in selecting the threshold value and manually search for a suitable value that roughly results in the same granularity, i.e. the same number of resulting hulls. For Table 1 and Table 2, we use a fixed threshold of 0.10.

C. Run time comparison with baseline methods

Table 3 compares run time statistics with baseline methods. Our approach is somewhat faster than optimization-based methods and somewhat slower than feedforward learning model, but offers quality improvements over both.

D. Additional Ablation with Alternative Training Objectives

We provide an additional ablation to validate the effectiveness of our loss function. Specifically, we ablate on our triplet-based contrastive learning objective. We compare it against a network trained on our feature embedding objective defined in Equation 5, but *without* using triplets. Concretely, we train the model with the following loss:

$$L = \frac{1}{2}\Big[-f(x)^\top f(p) \; + \; f(x)^\top f(n)\Big] \tag{9}$$

As shown in Table 4, the triplet-based contrastive loss used in our main method outperforms this alternative objective, demonstrating its effectiveness.

E. More Results

We present additional qualitative results across various granularities for further evaluation.

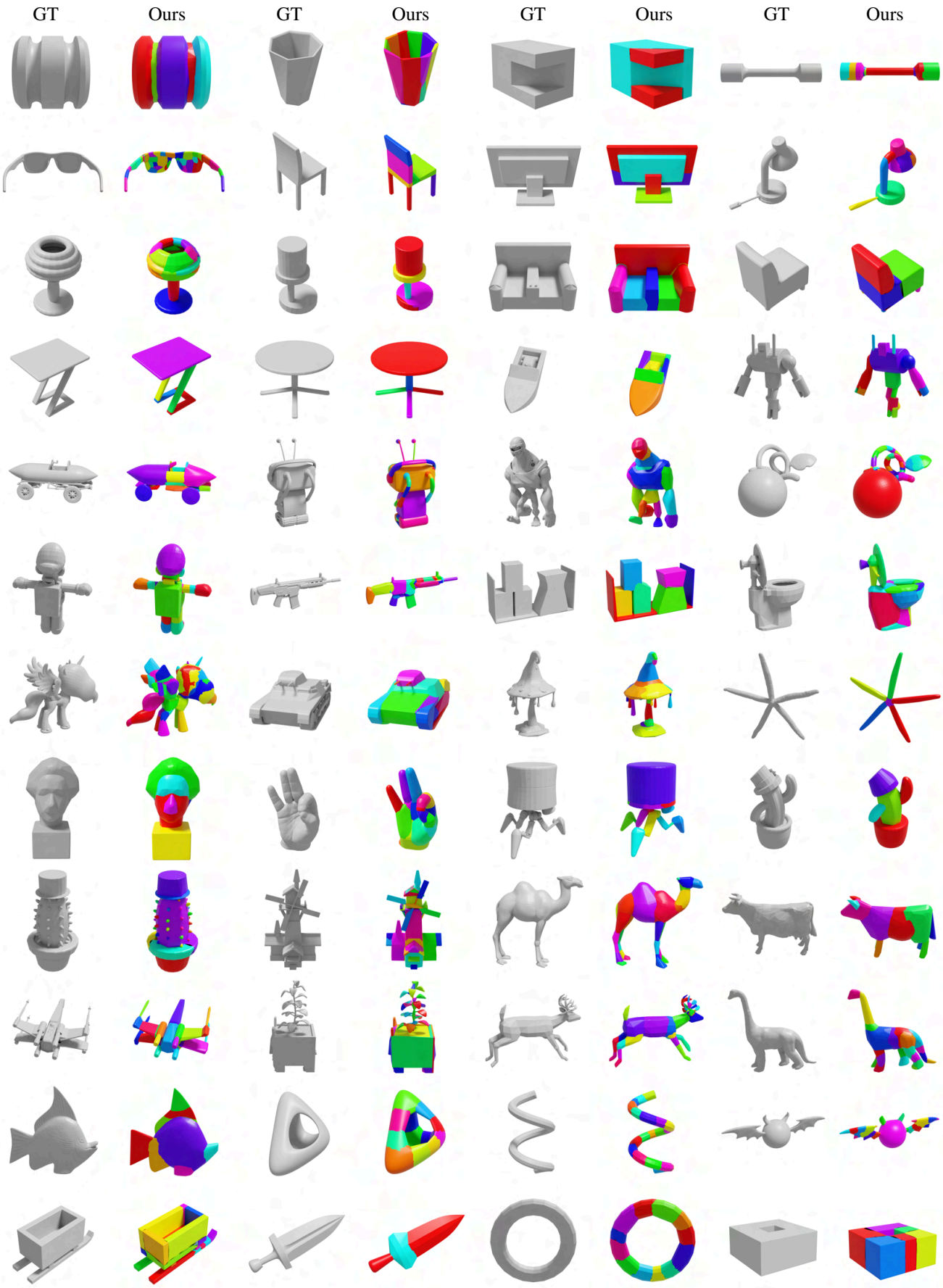

Figure 12. Additional result.