Learning Convex Decomposition via Feature Fields

Yuezhi Yang[1,2*] Qixing Huang[2] Mikaela Angelina Uy[1†] Nicholas Sharp[1†]
[1]NVIDIA [2]The University of Texas at Austin

Abstract

This work proposes a new formulation to the long-standing problem of convex decomposition through learning feature fields, enabling the first feed-forward model for open-world convex decomposition. Our method produces high-quality decompositions of 3D shapes into a union of convex bodies, which are essential to accelerate collision detection in physical simulation, amongst many other applications. The key insight is to adopt a feature learning approach and learn a continuous feature field that can later be clustered to yield a good convex decomposition via our self-supervised, purely-geometric objective derived from the classical definition of convexity. Our formulation can be used for single shape optimization, but more importantly, feature prediction unlocks scalable, self-supervised learning on large datasets resulting in the first learned open-world model for convex decomposition. Experiments show that our decompositions are higher-quality than alternatives and generalize across open-world objects as well as across representations to meshes, CAD models, and even Gaussian splats. https://research.nvidia.com/labs/sil/projects/learning-convex-decomp/

1. Introduction

Convex decomposition approximates detailed, nonconvex 3D shapes with a simple set of convex bodies—these convex approximations are essential geometric accelerations for fast collision detection [22, 37, 53], signed-distance computation [32], motion animation [28, 39], and more, which all must efficiently compute distances and spatial bounds. Traditionally, such decompositions were manually authored by technical artists as part of the content creation process, but the prevalence of automated 3D pipelines and generative AI has created ever-increasing demand for algorithmic construction of convex decompositions. In particular, recent progress in physical robotics training environments demands robust, high-performance simulation of generated assets and world models—convex decompositions are essential to enable these simulations [3].

Figure 1. Our method takes an input shape (*top*), infers features from an open-world model learned with our new self-supervised geometric loss (*middle*), and clusters those features to fit the shape with a collection of tight convex bounding proxies (*bottom*).

However, convex decomposition has thus far remained a remarkably challenging algorithmic task; one must accurately approximate the starting shape in the geometric sense, as well as a combinatorial covering problem that is formally NP-hard in the worst case [10, 41]. Traditional approaches from computational geometry use branch-and-bound techniques to explore a search space of possible decompositions, but can be prohibitively computationally expensive [6, 9, 27]. The learned architectures have also explored representing shapes with convex primitives, but thus far have been limited to narrow families of objects rather than to general open-world content [12, 16]. Moreover, much past work has focused almost entirely on mesh decomposition, whereas geometry increasingly comes from

*Work done while interning at NVIDIA.
†Equal contribution.

imprecise representations such as Gaussian splats.

This work proposes a new formulation for convex decomposition, which allows us to train a feedforward open-world model directly producing high-quality convex decompositions of shapes. Our key insight is to adopt a *feature learning* approach. Rather than directly optimizing or learning a discrete set of primitives, we instead operate on continuous features defined along the shape, constructing a set of features such that clustering them yields a good convex decomposition. To make this possible, we introduce a new self-supervised contrastive loss on features, inspired by a classic geometric definition of convexity: lines connecting pairs of points should be contained within the shape.

We validate our approach against both classical and deep learning baselines, demonstrating superior performance across multiple datasets. Beyond quantitative gains, we also showcase the practical utility of our method, enabling collision detection, control over granularity (Fig. 2), and generalization to various input modalities such as Gaussian splats. The main contributions of this work are summarized as follows:

- We formulate convex decomposition as a contrastive learning problem and introduce our novel, self-supervised, geometric loss that enables scalable learning of convex decomposition on open-world data.
- We train a feedforward network that enables state-of-the-art performance on convex decomposition, fast inference, and generalization to different 3D modalities.
- We demonstrate the effectiveness of our approach in different downstream applications such as multi-granularity decompositions, collision detection and generalization to different input modalities.

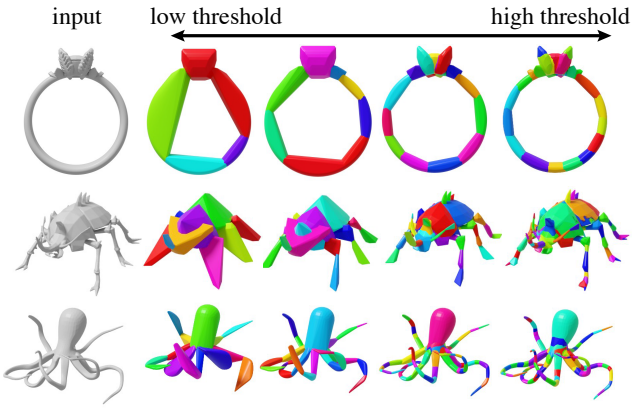


Figure 2. By adjusting the clustering threshold, our method can generate decompositions at varying granularity, all from the same feature field.

2. Related Work

Classic Convex Decomposition. Convex decomposition is a long-studied problem in computational geometry [5, 9, 37, 41] enabling applications such as fast and precise collision detection [22, 37, 53], shape deformation [59], distance computation [23, 49], skeleton extraction [28], animation [39], segmentation [18], simulation and gaming [3]. The task of *exact convex decomposition*—decomposing a shape into the minimum number of strictly convex components—is known to be NP-hard [10, 41]. Multiple works [5, 6, 9, 24, 25] introduce different heuristics to tackle this problem but often produce a large number of components that limit practical use. This led to the problem of *approximate convex decomposition* [27] that instead only requires components to be *nearly* convex, where they first define a concavity metric that measures the deviation of a component with its convex hull, then iteratively decomposes a shape until the concavity metric for all components is below a selected threshold. Concavity metrics have been defined based on the boundary of the component [21, 27, 29, 35], volume [4, 36, 51] or surface visibility [30, 46] with respect to its corresponding convex hull. Notably, V-HACD [36] has long been widely-used due to easily-available implementations and robust performance, while recently CoACD [58] improved it by introducing a collision-aware concavity metric based on both the shape's boundary and interior volume. RL-ACD [34] replaces searching with reinforcement learning to improve efficiency. Although these methods perform acceptably well on general shapes, the search space over all partitions to minimize concavity is enormously large, resulting in methods that are computationally slow even with assumptions such as axis-aligned cuts [58] and voxelization [36] to remain tractable.

Learning-based Methods. The prevalence of deep learning has spurred efforts to replace these classical search algorithms with data-driven optimization. Lacking ground-truth labels for optimal convex decompositions, learning-based methods such as BSP-Net [12] and Cvx-Net [16], adopt a self-supervised paradigm, where they define a fixed set of half-planes, and learn to optimize and assemble them to reconstruct the target shape. Although these approaches have shown promise in category-level datasets [8], the reconstruction objective fundamentally limits their scalability to large-scale topological and geometric variations in open-world data [14, 15]. Related to this line of work is learning-based shape abstraction, where the goal is to approximate the input shape into simple primitives. These works are commonly self-supervised with reconstruction [19, 43, 47, 52] or rendering [20, 38] to abstract the shape into simple primitives such as cuboids [48, 52, 60, 64], superquadrics [17, 20, 32, 43, 57], superellipses [63] or differentiable support functions [42]. Part-based decomposi-

tion [31, 44, 50, 62] has also been explored and shares some similarities with our setting, however, it does not result in a good solution for convex decomposition.

3. Method

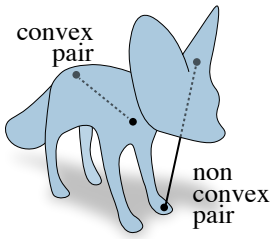
Convex decomposition takes a 3D shape as input and outputs a collection of convex bodies tightly approximating the shape. The core of our approach is to generate a set of features on the surface of the shape such that the distance in the feature space indicates points which should lie in the same convex body. These features are fitted using a new self-supervised geometric loss (Sec 3.3), and are ultimately be learned with a feedforward model (Sec 3.4). Any application-appropriate clustering-like strategy could be applied to these features; we propose a simple recursive algorithm well-suited to this setting (Sec 3.5) to partition the surface, and finally evaluate the convex hull of each partition as the resulting decomposition. Figure 3 shows an overview of our pipeline.

3.1. Formulation

We introduce our approach abstractly on a general shape $\mathcal{M}$, and will later demonstrate how to apply it to various shape representations like meshes, point clouds, and Gaussian splats. Specifically, let $\mathcal{M} \subset \mathbb{R}^3$ denote the shape surface that we assume to be the boundary of a solid, and $\mathrm{Vol}(\mathcal{M}) \subset \mathbb{R}^3$ as the solid volume.

Convex decomposition can be viewed as finding a partition of $\mathcal{M}$ into a set of disjoint components $\{S_i\}$, that is $\bigcup_i S_i = \mathcal{M}$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. A good partition is one where (i) the number of components is minimized, and (ii) each component is a tight convex approximation of the shape, *i.e.* the deviation of S_i from its convex hull $\mathrm{hull}(S_i)$ is small. We call the latter measure *concavity*, and in Sec. 4.2 we will discuss exactly how it is evaluated. We can also define an assignment function $G : \mathcal{M} \to \{S_i\}$ that takes a point on the shape and returns which segment it is a member of.

Convex Pairs. Our method is inspired by a classic geometric definition of convexity. A shape is convex if for any two points, the line segment connecting them is entirely contained inside the shape



$$\lambda x + (1-\lambda) y \in \mathrm{Vol}(\mathcal{M}) \quad \forall x, y \in \mathcal{M}, \quad \forall \lambda \in [0,1] \tag{1}$$

we refer to such points as a *convex pair*. On a potentially-nonconvex shape we can define set of all convex pairs of points $\mathcal{C}(\mathcal{M}) \subseteq \mathcal{M} \times \mathcal{M}$ as

$$\begin{aligned} \mathcal{C}(\mathcal{M}) = \{ & (x \in \mathcal{M}, y \in \mathcal{M}) : \\ & \lambda x + (1-\lambda) y \in \mathrm{Vol}(\mathcal{M}) \quad \forall \lambda \in [0,1] \} \end{aligned} \tag{2}$$

noting that it is sufficient to test only points which lie on the surface. Any pairs $(x, y) \notin \mathcal{C}$ are *nonconvex*, meaning the line segment connecting them passes outside the shape. In practice, a pair of points on a surface can be efficiently tested for convexity by casting a ray between its endpoints and checking for intersections with the surface.

This notion of convex pairs allows a formalization of the optimization problem for what it means to be a good, tightly-convex decomposition. Good decompositions maximize the number of convex pairs which belong to the same segment, or equivalently minimize the number of convex pairs split into different segments

$$\underbrace{\max_{\{S_i\}}}_{\substack{\text{optimize over}\\ \text{partitions}}} \underbrace{\iint_{x,y \in \mathcal{C}(\mathcal{M})}}_{\substack{\text{for all convex pairs}\\ \text{of points on the shape}}} \underbrace{\mathbf{1}_{G(x)=G(y)} \, dx \, dy}_{\substack{\text{counting pairs which are}\\ \text{in the same component}}} \tag{3}$$

where $\mathbf{1}_{G(x)=G(y)}$ is an indicator function that equal to 1 if x,y belongs to the same component. Directly optimizing this objective would amount to an intractable combinatorial partitioning problem, but we will show how to relax it to a continuous feature embedding problem.

3.2. Convex Decomposition as Feature Learning

We observe that the objective in Eq. 3 shares a similarity with unsupervised clustering, and draw inspiration from this insight to reformulate the original convex decomposition objective as a *feature learning* problem. Here, the goal is to learn continuous features that can later be clustered to yield the desired convex decomposition. We consider a field of k-dimensional features defined at each point on the shape $f : \mathcal{M} \to \mathbb{R}^k$, with some notion of feature distance $d(f_x, f_y)$, moving the argument of $f(\cdot)$ to a subscript for brevity. Under this perspective, the objective from Eq. 3 can be relaxed as

$$\underbrace{\min_{f}}_{\substack{\text{optimize over}\\ \text{features}}} \underbrace{\iint_{x,y \in \mathcal{C}(\mathcal{M})}}_{\substack{\text{for all convex pairs}\\ \text{of points on the shape}}} \underbrace{d\big(f_x, f_y\big) \, dx \, dy}_{\substack{\text{distance between}\\ \text{their features}}} \tag{4}$$

where f is the desired feature field on the shape. As-written, Eq. 4 tries to pull pairs of points together, and thus has a trivial degenerate solution with $f(x) = \textit{constant}$, so we balance the objective with a second term that tries to push nonconvex pairs apart

$$\min_{f} \iint_{x,y \in \mathcal{C}(\mathcal{M})} d\big(f_x, f_y\big) \; - \iint_{x,y \notin \mathcal{C}(\mathcal{M})} d\big(f_x, f_y\big) \tag{5}$$

which gives our feature embedding objective, restricting $||f|| = 1$ to prevent unbounded growth. From a machine learning viewpoint, this objective is *self-supervised*, it allows us to optimize for a good set of features, and hence a

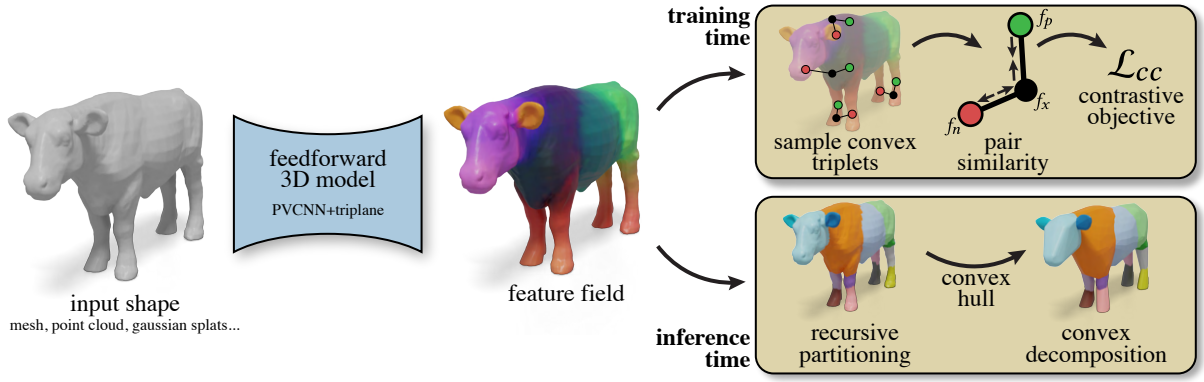


Figure 3. An overview of our convex decomposition pipeline. We train a feedforward model that takes a point-sampled 3D shape as input and predicts a feature field represented defined over the object. At training time, these features are fit with a self-supervised geometric objective derived from the definition of convexity. At inference time, the features are clustered to split the shape into components, and the convex hull of each component becomes the decomposition. Note that feature colors are visualized by running PCA on the feature field.

good decomposition, purely from the geometry of the input shape. In practice, optimization amounts to first sampling many pairs x, y of points on a shape, geometrically testing whether each pair $x, y \in \mathcal{C}(\mathcal{M})$ based on whether the line between the points is contained in the shape, and finally continuously optimizing for features to minimize the embedding objective.

3.3. Contrastive Feature Learning

Eq. 5 is already a well-posed embedding problem, but rather than optimizing it directly we follow previous works [11, 31] to recast it as relative *contrastive learning*, which is more amenable to high-dimensional stochastic optimization without imposing a metric structure. The contrastive objective is defined by gathering triplets of points $x, p, n \in \mathcal{M}$ forming a positive pair $x, p \in \mathcal{C}(\mathcal{M})$ and negative pair $x, n \notin \mathcal{C}(\mathcal{M})$, with the goal that the distance between the positive pair should be smaller than the distance between the negative pair. The loss is then a log-normalized balance

$$\mathcal{L}_{\text{cc}} = -\tfrac{1}{2}\left[\log \frac{\text{sim}(f_x, f_p)}{\text{sim}(f_x, f_p) + \text{sim}(f_x, f_n)} + \log \frac{\text{sim}(f_p, f_x)}{\text{sim}(f_p, f_x) + \text{sim}(f_p, f_n)}\right], \tag{6}$$

where we also exchange smaller-is-closer distance for larger-is-closer similarity, typically the exponential of cosine distance on sphere-normalized features $\text{sim}(x, y) = \exp\big(x \cdot y/\tau\big)$, with τ as a temperature hyperparameter akin to a softmax.

Triplet Sampling and Hard Negatives. The triplets (x, p, n) could be generated by uniform rejection sampling, but this is computationally expensive to begin with, and furthermore there is opportunity to seek out *hard negative* samples which are particularly informative to the optimization process. We first choose an anchoring sample $x \in \mathcal{M}$ uniformly from the object's surface. To get a sample $p \in \mathcal{M}$ forming a positive pair with x, we cast a random ray into the hemisphere opposite the surface-normal direction of x, into the interior of the shape, and takes the point where it exits the surface as p. To get a sample $n \in \mathcal{M}$ forming a negative pair with x, we rejection sample by generating candidates on the surface of $\mathcal{M}$ and testing whether the line segment connecting x and n exits the shape. Rather than uniformly gathering points on the surface, we prefer negatives spatially close to x, sampling inversely proportional to the Euclidean distance $\mathrm{P}(n) = \frac{1}{\|n-x\|^2}$. These negatives are likely to be challenging pairs, enabling more efficient optimization of features.

The geometric queries in this sampling procedure can be implemented efficiently and robustly, as it requires only sampling points and casting rays, the latter of which can be hardware-accelerated with libraries like Intel Embree and NVIDIA OptiX for fast on-the-fly triplet generation.

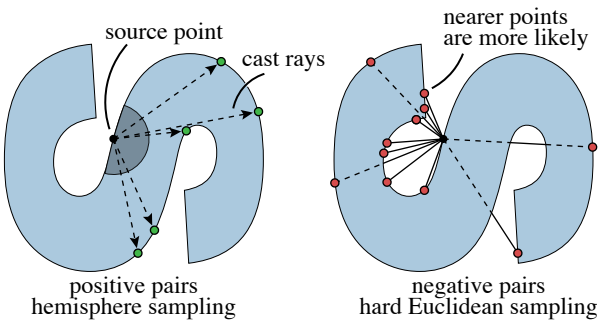


Figure 4. Contrastive training triplets are formed by a source point, a positive pair generated by casting a ray in a random inward direction (*left*), and a negative pair rejection-sampled from all points on the surface weighted to prefer nearby points (*right*).

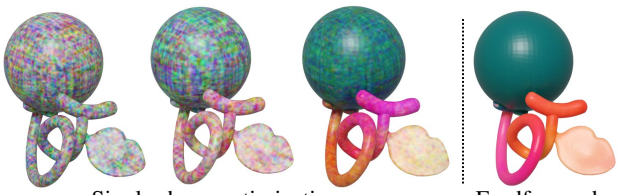


Figure 5. Left: Visualization of the learned feature field during our single-shape optimization, shown at different optimization stages. Right: Feature fields produced directly by our feed-forward model.

Single shape optimization. Given the sampling and optimization procedures described above, we can optimize our feature fields directly on a single shape to obtain its convex decomposition. Figure 5 shows an example of such optimization. Although already effective, we take advantage of feature-based setting to train a feedforward model to generate smoother features, rather than fitting them per-shape.

3.4. Feedforward Model

Feature-based formulations have been used throughout visual computing as a framework for large-scale learning of segmentation, embeddings, and more [7, 40, 45]. Reformulating convex decomposition as a feature learning problem makes it amenable to this approach, allowing us to train a feedforward model predicting the field f conditioned on an input shape $\mathcal{M}$, self-supervised across a large dataset of open-world 3D shapes [14, 15]. Our self-supervised geometric loss (Eq. 6) is essential, sidestepping the lack of high-quality ground truth supervision for this task to learn solely from the shapes' geometry. Compared with per-shape optimization, the feedforward model offers three main advantages: (a) fast inference, (b) smooth feature fields that are robust to input noise or incomplete geometry (c) generalization across 3D input modalities, i.e. our model can be applied to different 3D representations at inference time, without requiring a watertight mesh.

We adopt an architecture similar to prior work on open world-shape learning [31]. The model takes as input a point cloud sampled from the shape's surface $\mathcal{M}$, and outputs a triplane-encoded feature field that can be evaluated at any spatial location. This network consists of two main stages. First, a PVCNN encoder [33] encodes an input shape $\mathcal{M}$ represented as a point cloud and extracts per-point features from the input. These features are orthogonally projected onto three axis-aligned 2D feature planes via mean reduction to form an initial triplane representation. These initial triplanes are processed by a 2D CNN for downsampling, then reshaped and passed through a transformer module, and finally upsampled via a transposed 2D CNN to reconstruct final feature triplanes. Hence for any 3D query point we have its corresponding feature is retrieved by aggregating corresponding features from the final feature triplane.

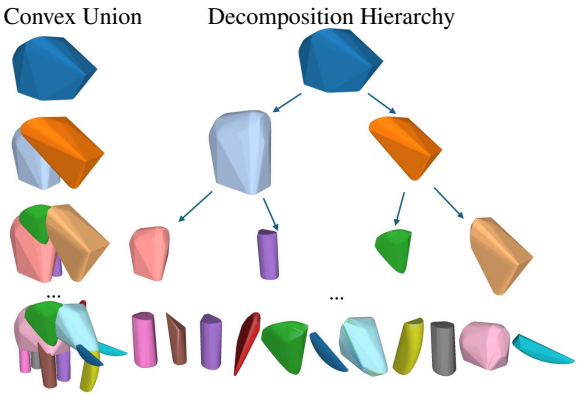


Figure 6. A hierarchical decomposition tree from our algorithm.

3.5. Recursive Decomposition on Features

At inference time, our trained model generates feature fields $f : \mathcal{M} \rightarrow \mathbb{R}^k$, such that similar features indicate surface regions that should appear together in the decomposition. Concretely, we densely sample features on the surface of the input shape S using the predicted field and apply off-the-shelf clustering algorithms to partition S into approximate convex components $\{S_i\}$. In principle any clustering algorithm could be used for this purpose, common choices include k-means for fast performance, or agglomerative clustering to take connectivity into account. For mesh-based inputs, features are sampled per face and clustered using agglomerative clustering with mesh connectivity. For other modalities such as point clouds, we blend feature-space distance and Euclidean distance and apply k-means clustering. Finally, we compute convex hulls $\{\text{hull}(S_i)\}$ for each cluster, whose union approximates the input geometry.

There is no obvious strategy to choose the number of clusters in the decomposition of a shape, so we adopt a recursive strategy of repeatedly binary clustering until a desired user-specified concavity threshold is reached. This threshold allows control over the granularity of the output; importantly this granularity need only be set during the clustering post-processing, and our learned features can be used for decomposition at any granularity. Algorithm 1 describes the divide-and-conquer strategy. For each cluster which is not yet a sufficiently tight convex proxy for its covered region, we apply binary clustering to split it, then recurse on both subcomponents. Components are processed in order of concavity until all reach the target threshold, or a user-specified maximum component count is reached. Figure 6 illustrates an example of this process.

4. Experiments

4.1. Implementation details

We train our model on the Objaverse dataset [14]. Low-quality data, such as scans with broken or incomplete geometry, are filtered out, leaving approximately 340K shapes for training. All shapes are normalized to the range $[-1, 1]^3$, and 100,000 points are uniformly sampled from each shape

Algorithm 1: Our Recursive Decomposition

Require: Mesh $\mathcal{S}$, threshold ε, max hulls K
Ensure: Parts $\mathcal{R}$

$P \leftarrow \textsc{Sample}(\mathcal{S})$; $F \leftarrow \textsc{Model}(P)$
$C_{\mathcal{S}} \leftarrow \textsc{Concavity}(\mathcal{S})$
$\mathcal{Q} \leftarrow \{(\mathcal{S}, C_{\mathcal{S}})\}$ {max-heap by concavity}
$\mathcal{R} \leftarrow \emptyset$; $N \leftarrow 0$
while $\mathcal{Q} \neq \emptyset$ **and** $N < K$ **do**
  $(P, C_P) \leftarrow \textsc{PopMax}(\mathcal{Q})$
  if $C_P < \varepsilon$ **then**
    $\mathcal{R}.\textsc{Add}(P)$; $N \leftarrow N + 1$
  else
    $(P_1, P_2) \leftarrow \textsc{Clustering}(P, 2)$
    $C_1 \leftarrow \textsc{Concavity}(P_1)$;
    $C_2 \leftarrow \textsc{Concavity}(P_2)$
    $\textsc{Push}(\mathcal{Q}, (P_1, C_1))$; $\textsc{Push}(\mathcal{Q}, (P_2, C_2))$
  end if
end while

return $\mathcal{R}$

as network input. For every shape, we generate 10,000 triplets; for each triplet, 64 positive and 512 negative samples are drawn to construct candidate pairs.

The feature field has a dimensionality of 448. The triplane representation has a spatial resolution of 512×512 with 128 channels, and the transformer backbone consists of six layers. We train the model for 600,000 steps on eight NVIDIA A100 GPUs over one week, using a batch size of two per GPU. It takes 5s to generate features and 13s for the recursive algorithm to perform decomposition.

4.2. Baseline Comparisons

Evaluation Datasets. We evaluate our method on three datasets: V-HACD [36], PartObjaverse-Tiny [61] and ShapeNet [8] datasets. The V-HACD dataset contains 61 3D models including mechanical objects, animals, and human body parts, and is commonly used for evaluating approximate convex decomposition algorithms. The PartObjaverse-Tiny dataset is a curated subset of the large-scale Objaverse collection, containing 200 3D objects spanning diverse semantic categories. ShapeNet [8] is a large dataset for man-made shapes commonly used for evaluating learning based 3D shape reconstruction. We use 13 most common categories in ShapeNet and randomly sample 10% of the shapes from the test set resulting in 900 models.

Evaluation Metric. Following previous work, we evaluate all methods using the concavity metric defined in [58]. We compute $\max_i(\text{Concavity}(\mathcal{M} \cap \text{hull}(S_i))$, where $\text{Concavity}(P)$ is the maximum of the surface and volumetric Chamfer distances between P and its convex hull. We also further report the overall reconstruction accuracy defined as the Chamfer distance between the input shape and the union of all convex hulls, i.e. $\text{Chamfer}(\mathcal{M}, \cup_i \text{hull}(S_i))$.

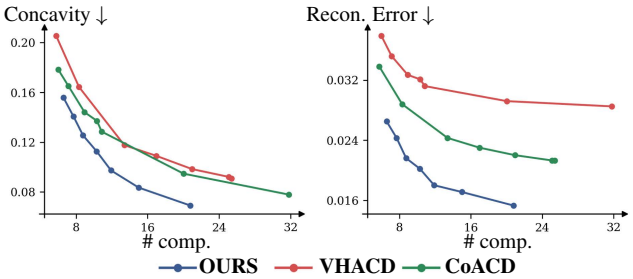


Figure 7. Quantitative comparison with CoACD and VHACD on the VHACD dataset under different granularity.

Baselines. We compare our method against both (i) classical convex decomposition approaches: CoACD [58] and V-HACD [36], as well as (ii) learning-based methods: Cvx-Net [16] and BSP-Net [12]. For fair comparison, we report both Cvx-Net and BSP-Net using their original setup trained on ShapeNet (Cvx-S, BSP-S) as well as both models trained on Objaverse, a larger dataset corpus (Cvx-O, BSP-O). We use their recommended parameter settings in all experiments. Quantitative evaluation for CoACD and V-HACD are computed by setting hyper-parameters that maintain the same level of granularity with our method and visualize results at different granularity level qualitatively.

Results. Quantitative results are reported in Table 1, where we see that across all datasets, our method consistently outperforms both classical and learning-based baselines in terms of concavity and reconstruction error at any given component count. Figure 7 further comprehensively compares our method against V-HACD and CoACD showing superior performance at multiple granularity levels. Figure 12 shows our qualitative results. While these classical approaches remain strong baselines, they often fail to capture inclined convex regions, leading to unnecessary splits (top row: hermit crab shell) due to their axis-aligned cut assumption. In contrast, our approach better preserves large convex structures (third row: submarine), separates nearby convex parts (fourth row: elephant), and performs well even when restricted to a small number of components (second row: airplane). On learning-based baselines, both Cvx-S and BSP-S struggle to generalize on shapes beyond those in ShapeNet as shown by their poor reconstruction results (see airplane in the second row vs other examples), even when using a substantially larger number of convexes. While both Cvx-O and BSP-O improve in reconstruction quality, their resulting convex decomposition are still far from ideal, suggesting that both do not scale well to open-world shapes.

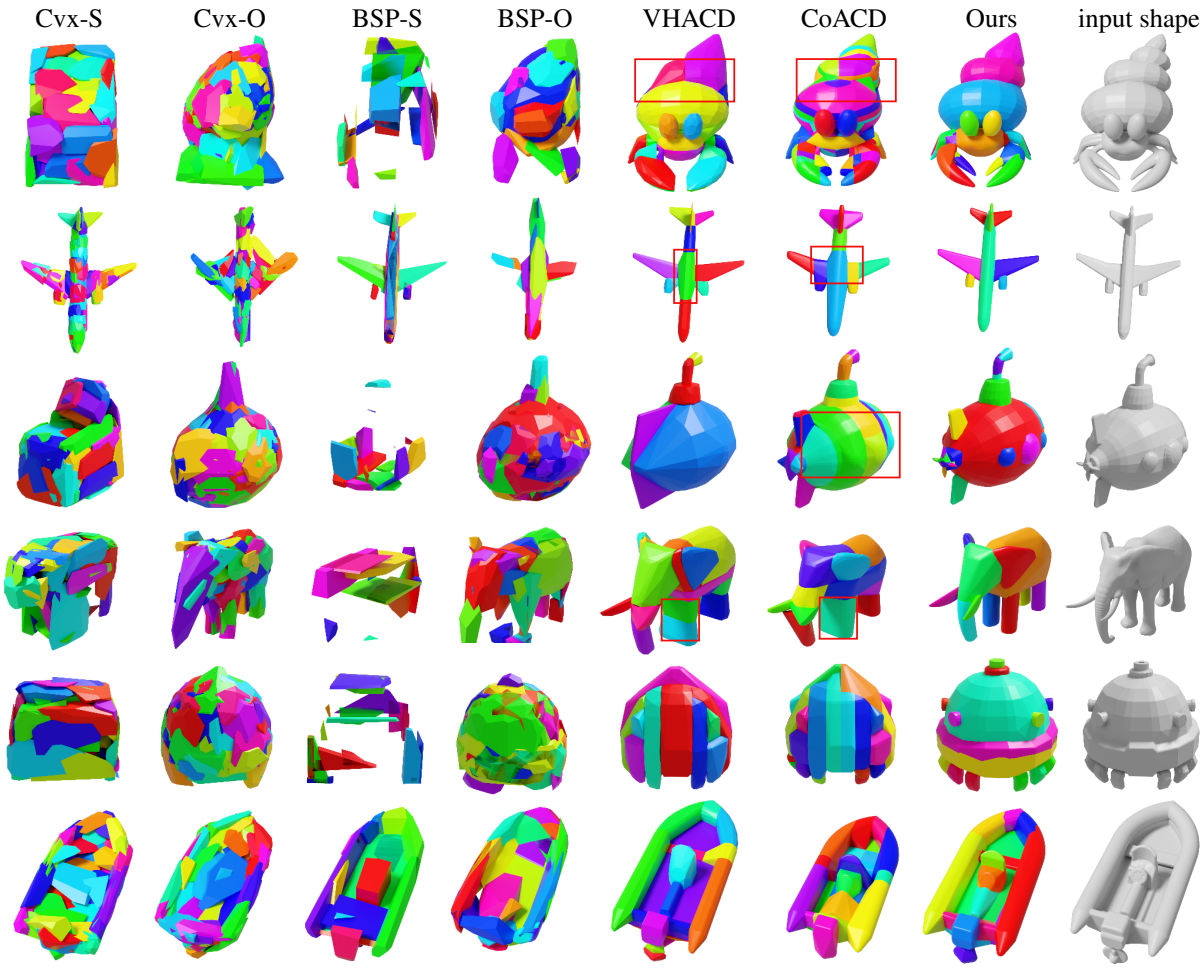


Figure 8. Qualitative results against the baselines.

	VHCD Dataset			PartObjaverse-Tiny			ShapeNet		
	# comp.	concavity	recon.	# comp.	concavity	recon.	# comp.	concavity	recon.
VHCD	13.39	0.1176	0.0213	21.94	0.1800	0.0318	14.73	0.1155	0.0201
COACD	14.47	0.1095	0.0321	23.63	0.1388	0.0321	13.02	0.0746	0.0172
BSP-Net (ShapeNet)	12.90	0.2249	0.1227	12.33	0.2593	0.1258	21.58	0.2107	0.0273
BSP-Net (Objaverse)	22.26	0.1857	0.0297	21.44	0.1964	0.0315	16.15	0.1592	0.0273
Cvx-Net (ShapeNet)	50.00	0.4673	0.0814	50.00	0.5079	0.0808	50.00	0.1795	0.0234
Cvx-Net (Objaverse)	50.00	0.2880	0.0373	50.00	0.2970	0.0359	50.00	0.2191	0.0258
Ours	**11.90**	**0.0973**	**0.0180**	**21.18**	**0.1257**	**0.0254**	**12.63**	**0.0656**	**0.0142**

Table 1. Quantitative comparison across three datasets. #comp. denotes the number of convex components, and *recon.* denotes reconstruction accuracy measured using Chamfer distance. For all metrics, lower ↓ means better.

4.3. Analysis and Ablation

Concavity thresholds ϵ. Figure 2 shows the convex decomposition produced under different user-specified concavity thresholds ϵ, which directly controls the decomposition granularity. Lower thresholds produce more components and preserve finer geometric details, while higher thresholds yield fewer components and result in coarser approximations of the input shape.

Ablation Study. We evaluate variants of our framework to understand the impact of each component. First, we remove hard-negative sampling when selecting negative pairs and hemisphere-based sampling when selecting positive

	VHCD Dataset			Objaverse-Tiny		
	#comp	conc.	recon.	#comp	conc.	recon.
- Hard-neg	12.88	0.0993	0.0186	21.60	0.1293	0.0268
- Hemis-Pos	13.08	0.0999	0.0219	23.27	0.1286	0.0263
- Recur. Clus.	12.25	0.1292	0.0191	25.44	0.1375	0.0288
Single opt.	12.40	0.1134	0.0198	22.35	0.1300	0.0266
Ours	**11.90**	**0.0973**	**0.0180**	**21.18**	**0.1257**	**0.0254**

Table 2. Quantitative ablation on components of our approach.

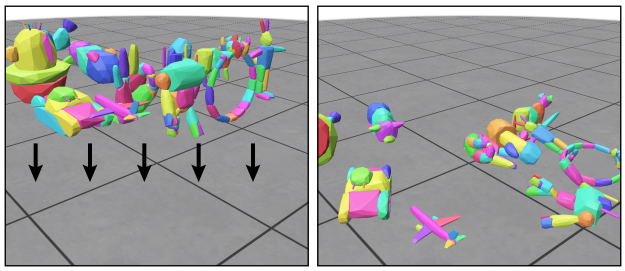


Figure 9. Convex decompositions are used to accelerate collision handling in physical simulations, shown here on rigid bodies, decomposed with our method and simulated under gravity in the Newton engine [2] .

pairs in the contrastive triplet construction (Section 3.3). We also compare against a version that does not use our recursive decomposition strategy (Section 3.5), instead increasing the number of clusters in a flat clustering procedure until the stopping criterion is met. Finally, we include an ablation where optimization is performed directly on a single shape rather than through the feedforward model. Quantitative results for all settings are shown in Table 2.

3D segmentation features. Although 3D segmentation and convex decomposition share conceptual similarities, segmentation features are not suitable for convex partitioning. To highlight this difference, we apply our recursive decomposition algorithm to feature fields produced by a recent 3D segmentation model, PartField [31]. PartField's features reflect semantic structure, which leads to poor convex partitions when clustering is applied. In contrast, our method learns features that are convex-aware, enabling accurate and geometrically meaningful decompositions.

4.4. Applications

Collision Detection. We showcase our convex decomposition results by applying the default collision detection implementation in Newton [2] as shown in Figure 9. Our convex approximation yields a 5x faster simulation step v.s. using the original meshes (8ms vs 40ms).

Input Modalities. Our feed-forward model operates directly on point clouds, allowing it to be applied to 3D shapes in a wide range of modalities. Figure 10 shows convex decompositions generated from CAD models of mechanical parts from the ABC dataset [26], real-world 3D scans [1], and AI-generated 3D Gaussian splats. Although trained primarily on point clouds sampled from human-authored meshes, the model generalizes well across these diverse input types, demonstrating broad applicability.

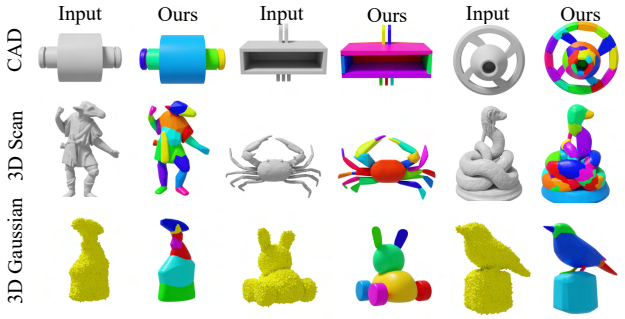


Figure 10. Our model can take different modalities as input, including CAD models, 3D scans, and 3D Gaussian splats, and perform convex decomposition.

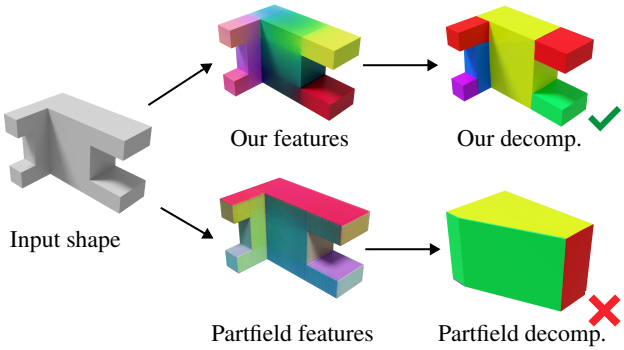
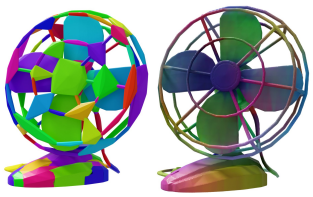


Figure 11. Our geometric objective produces convex-aware features for accurate decomposition (top). PartField instead yields nearly uniform features on flat regions. For example, the L-shaped surface appears almost entirely red—resulting in inferior decomposition (bottom). Feature colors are visualized by principle component analysis on the feature field.

5. Limitations and Future Work



Our model was trained on clean object-level data, and does not necessarily generalize to scene-scale or highly-incomplete geometry; one route to enhance this capability is training on scenes and noise-injected data. Our model also struggles with complex thin structures such as the frame of the fan (inset). More broadly, we are optimistic about further extending the generation of convex decompositions with feature learning, such as learning semantically-aware proxies adapted to likely motions of an object.

Acknowledgments. We would like to thank Guying Lin, Anka Chen and Jun Gao for helpful discussions. Yuezhi Yang and Qixing Huang were supported by NSF-2047677, 2413161, 2504906, and 2515626; GIFTs from Adobe and Google; and computing support on the Vista GPU Cluster through the Center for Generative AI (CGAI) and the Texas Advanced Computing Center (TACC) at UT Austin.

References

[1] Threedscans: Free 3d scan archive. https://threedscans.com/, 2025. 8

[2] Newton: An open-source gpu-accelerated physics simulation engine. https://github.com/newton-physics/newton, 2025. GitHub repository, Apache-2.0 license. 8

[3] James Andrews. Navigation-driven approximate convex decomposition. In *Proceedings of the ACM SIGGRAPH 2024 Conference Papers*, pages 1–9, 2024. 1, 2

[4] Marco Attene, Michela Mortara, Michela Spagnuolo, and Bianca Falcidieno. Hierarchical convex approximation of 3d shapes for fast region selection. *Comput. Graph. Forum*, 27: 1323–1332, 2008. 2

[5] Chanderjit L. Bajaj and Tamal K. Dey. Convex decomposition of polyhedra and robustness. *SIAM Journal on Computing*, 21(2):339–364, 1992. 2

[6] Chandrajit L. Bajaj and Valerio Pascucci. Splitting a complex of convex polytopes in any dimension. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, page 88–97, New York, NY, USA, 1996. Association for Computing Machinery. 1, 2

[7] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 5

[8] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2, 6

[9] Bernard Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984. 1, 2

[10] Bernard Chazelle, David P. Dobkin, Nadia Shouraboura, and Ayellet Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry*, 7(5–6): 327–342, 1997. 1, 2

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, 2020. 4

[12] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bspnet: Generating compact meshes via binary space partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 2, 6

[13] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation in robotics, games and machine learning. https://pybullet.org, 2016. 1

[14] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. *arXiv preprint arXiv:2212.08051*, 2022. 2, 5

[15] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-xl: A universe of 10m+ 3d objects. *arXiv preprint arXiv:2307.05663*, 2023. 2, 5

[16] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. 2020. 1, 2, 6

[17] Elisabetta Fedele, Boyang Sun, Leonidas Guibas, Marc Pollefeys, and Francis Engelmann. SuperDec: 3D Scene Decomposition with Superquadric Primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 2

[18] Matheus Gadelha, Rui Wang, and Subhransu Maji. Label-efficient learning on point clouds using approximate convex decompositions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 2

[19] Aditya Ganeshan, Matheus Gadelha, Thibault Groueix, Zhiqin Chen, Siddhartha Chaudhuri, Vladimir Kim, Wang Yifan, and Daniel Ritchie. Residual primitive fitting of 3d shapes with superfrusta. 2025. 2

[20] Zhirui Gao, Renjiao Yi, Yuhang Huang, Wei Chen, Chenyang Zhu, and Kai Xu. Self-supervised learning of hybrid part-aware 3d representation of 2d gaussians and superquadrics. 2025. 2

[21] Mukulika Ghosh, Nancy Amato, Yanyan Lu, and Jyh-Ming Lien. Fast approximate convex decomposition using relative concavity. *Computer-Aided Design*, 45:494–504, 2013. 2

[22] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. 1, 2

[23] Elmer G Gilbert, Daniel W Johnson, and Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. 2

[24] J.E. Hershberger and J.S. Snoeyink. Erased arrangements of lines and convex decompositions of polyhedra. *Computational Geometry*, 9(3):129–143, 1998. 2

[25] Barry Joe. Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *International Journal for Numerical Methods in Engineering*, 37(4):693–713, 1994. 2

[26] Sebastian Koch, Aleksey Matveev, Kai Jiang, Shi Wu, Binbin Pan, Vladimir Kim, Wojciech Matusik, Barbara Solenthaler, Markus Gross, and Robert W. Sumner. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9601–9611, 2019. 8

[27] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007*

ACM Symposium on Solid and Physical Modeling, pages 121–131, 2007. 1, 2

[28] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, page 219–228, New York, NY, USA, 2006. Association for Computing Machinery. 1, 2

[29] Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. Nearly convex segmentation of polyhedra through convex ridge separation. *Computer-Aided Design*, 78, 2016. 2

[30] Hairong Liu, Wenyu Liu, and Longin Jan Latecki. Convex shape decomposition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 97–104, 2010. 2

[31] Minghua Liu, Mikaela Angelina Uy, Donglai Xiang, Hao Su, Sanja Fidler, Nicholas Sharp, and Jun Gao. Partfield: Learning 3d feature fields for part segmentation and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 3, 4, 5, 8

[32] Wei Liu, Yun Wu, Seo Ruan, and Gregory S. Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14796–14806, 2023. 1, 2

[33] Zhijian Liu, Haotian Tang, Yueqi Lin, Song Han, and William Han. Point-voxel cnn for efficient 3d deep learning. In *NeurIPS*, 2019. 5

[34] Yuzhe Luo, Zherong Pan, Kui Wu, Xingyi Du, Yun Zeng, Xiangjun Tang, Yiqian Wu, Xiaogang Jin, and Xifeng Gao. Rl-acd: Reinforcement learning-based approximate convex decomposition. *ACM Trans. Graph.*, 44(6), 2025. 2

[35] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3501–3504, 2009. 2

[36] Khaled Mamou, E Lengyel, and AK Peters. Volumetric hierarchical approximate convex decomposition. In *Game Engines Gems 3*, pages 141–158, 2016. 2, 6, 1

[37] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, 1998. 1, 2

[38] Tom Monnier, Jake Austin, Angjoo Kanazawa, Alexei A. Efros, and Mathieu Aubry. Differentiable Blocks World: Qualitative 3D Decomposition by Rendering Primitives. In *NeurIPS*, 2023. 2

[39] Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Trans. Graph.*, 32(4), 2013. 1, 2

[40] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 5

[41] Joseph O'Rourke and Kenneth Supowit. Some np-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29(2):181–190, 1983. 1, 2

[42] Sunkyung Park, Jeongmin Lee, and Dongjun Lee. Shape abstraction via marching differentiable support functions. In *CVPR*, pages 16902–16911, 2025. 2

[43] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[44] Despoina Paschalidou, Luc van Gool, and Andreas Geiger. Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[45] Alec Radford, Jong Wook Kim, Chris Hallacy, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 5

[46] Zhou Ren, Junsong Yuan, Chunyuan Li, and Wenyu Liu. Minimum near-convex decomposition for robust shape representation. In *2011 International Conference on Computer Vision*, pages 303–310, 2011. 2

[47] Gopal Sharma, Rishabh Tiwari, and Subhransu Maji. Prifit: Learning to fit primitives improves few-shot point cloud segmentation. *Computer Graphics Forum*, 41(5):–, 2022. 2

[48] Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Trans. Graph.*, 38(6), 2019. 2

[49] Min Tang, Yu-Kun Wang, Kai Tang, and Dinesh Manocha. Fast and exact continuous collision detection for topologically complex objects. In *ACM SIGGRAPH Asia 2014 Technical Briefs*, pages 1–4. ACM, 2014. 2

[50] Konstantinos Tertikas, Despoina Paschalidou, Boxiao Pan, Jeong Joon Park, Mikaela Angelina Uy, Ioannis Emiris, Yannis Avrithis, and Leonidas Guibas. Generating part-aware editable 3d shapes without 3d supervision. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3

[51] Daniel Thul, L'ubor Ladický, Sohyeon Jeong, and Marc Pollefeys. Approximate convex decomposition and transfer for animated meshes. 37(6), 2018. 2

[52] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 2

[53] Gino van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *J. Graphics, GPU, & Game Tools*, 4:7–25, 1999. 1, 2

[54] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt,

and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 1

[55] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. Embree: a kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics (TOG)*, 33(4):1–8, 2014. 1

[56] Peng-Shuai Wang. mesh2sdf: Converts an input mesh to a signed distance field (sdf). https://github.com/wang-ps/mesh2sdf, 2022. GitHub repository. 1

[57] Yuhan Wang, Weikai Chen, Zeyu Hu, Runze Zhang, Yingda Yin, Ruoyu Wu, Keyang Luo, Shengju Qian, Yiyan Ma, Hongyi Li, Yuan Gao, Yuhuan Zhou, Hao Luo, Wan Wang, Xiaobin Shen, Zhaowei Li, Kuixin Zhu, Chuanlang Hong, Yueyue Wang, Lijie Feng, Xin Wang, and Chen Change Loy. Light-sq: Structure-aware shape abstraction with superquadrics for generated meshes. 2025. SIGGRAPH Asia 2025. 2

[58] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 2, 6, 1

[59] Martin Wicke, Mario Botsch, and Markus Gross. A Finite Element Method on Convex Polyhedra. *Computer Graphics Forum*, 2007. 2

[60] Kaizhi Yang and Xuejin Chen. Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds. *ACM Trans. Graph.*, 40(4), 2021. 2

[61] Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Lam Edmund Y., Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv preprint arXiv:2411.07184*, 2024. 6

[62] Yuezhi Yang, Haitao Yang, Kiyohiro Nakayama, Xiangru Huang, Leonidas Guibas, and Qixing Huang. Genanalysis: Joint shape analysis by learning man-made shape generators with deformation regularizations. *ACM Transactions on Graphics (TOG)*, 44(4):1–19, 2025. 3

[63] Mingrui Zhao, Yizhi Wang, Fenggen Yu, Changqing Zou, and Ali Mahdavi-Amiri. Sweepnet: Unsupervised learning shape abstraction via neural sweepers. In *European Conference on Computer Vision*. Springer, 2024. 2

[64] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *ICCV*, 2017. 2

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:

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

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:

$$\mathrm{Concavity}(S) = \max\big(H(\partial S,\, \partial\,\mathrm{hull}(S)),\; H(\mathrm{Vol}(S),\, \mathrm{Vol}(\mathrm{hull}(S)))\big) \tag{8}$$

where $H(\cdot)$ denotes the Hausdorff distance, ∂S is the surface of S, and $\mathrm{Vol}(S)$ 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.

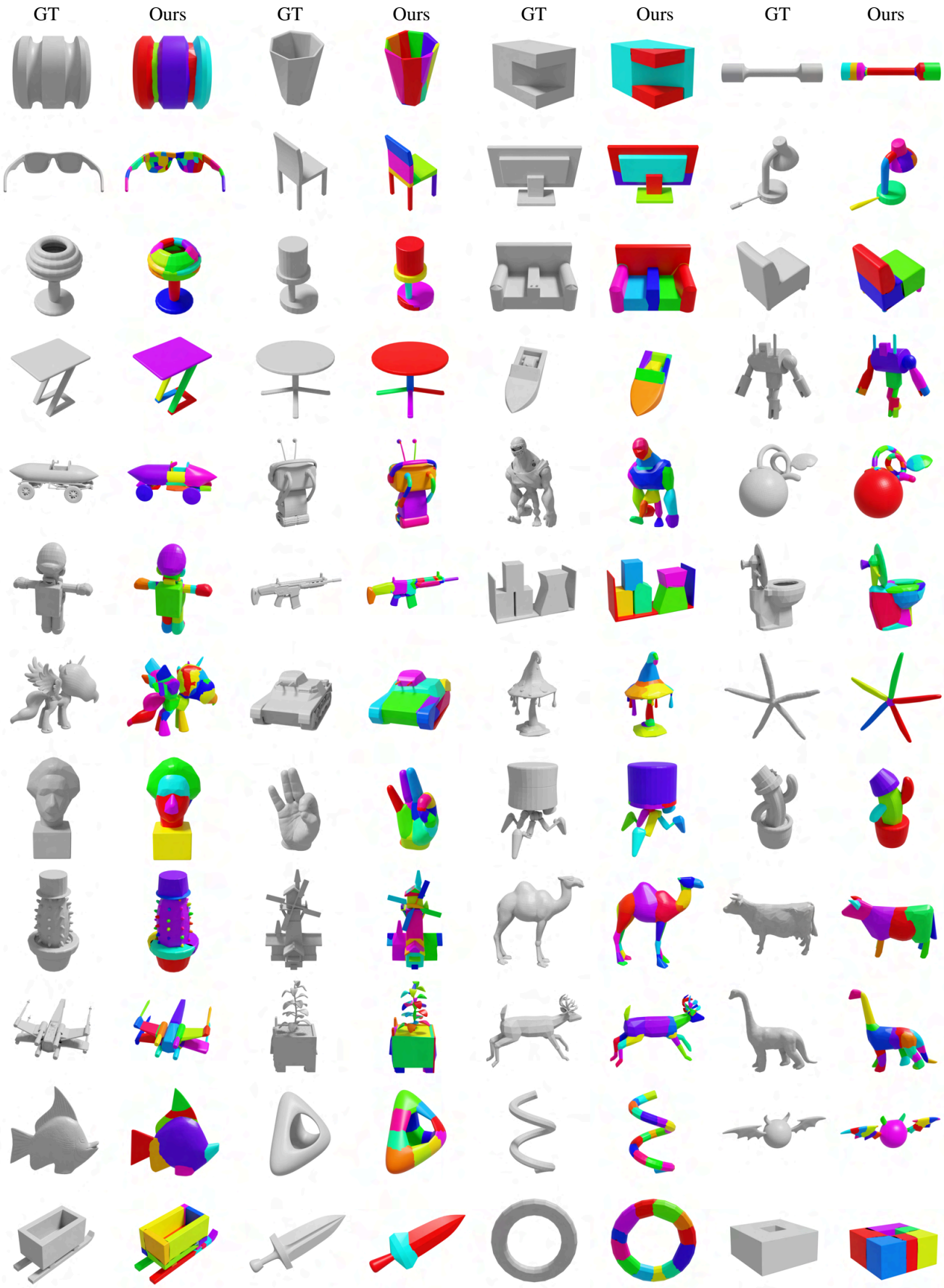


Figure 12. Additional result.