

# Split, Merge, and Refine: Fitting Tight Bounding Boxes via Over-Segmentation and Iterative Search

Chanyeok Park Minhyuk Sung

KAIST

{chpark1111, mhsung}@kaist.ac.kr

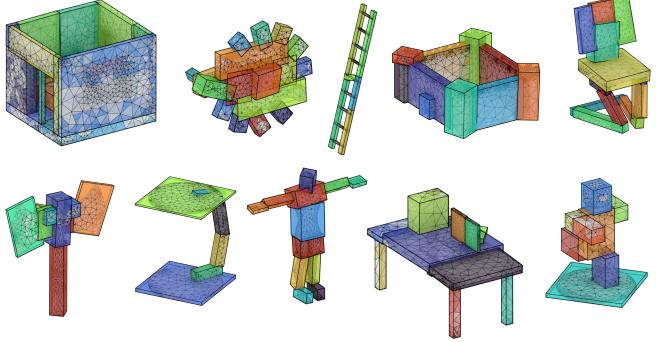
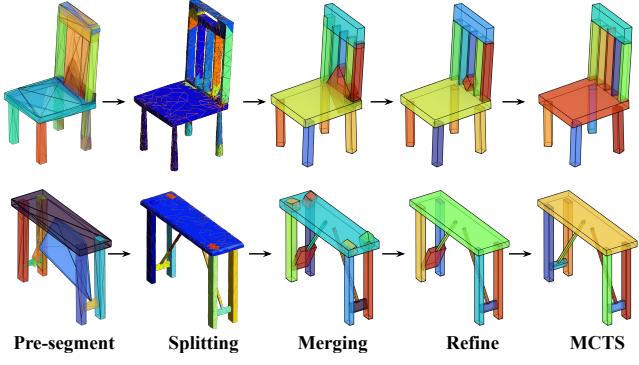


Figure 1. Our framework dubbed **SMART** finds a set of tight bounding boxes of a 3D shape. (a) Given a 3D shape represented as a volumetric mesh, it first takes **over-segmentation** by post-processing (second column) any pre-segmentation (first column) and then performs hierarchical **merging** with tightness-aware criteria (third column). Then, it **refines** the bounding box parameters with a sequence of actions in a discrete space (fourth column). The results are further improved with long-sighted exploration by our **MCTS** accelerated with our technique (last column). (b) Results of SMART on Objaverse [12] overlaid with the input mesh represented as grey.

## Abstract

Achieving tight bounding boxes of a shape while guaranteeing complete boundness is an essential task for efficient geometric operations and unsupervised semantic part detection. But previous methods fail to achieve both full coverage and tightness. Neural-network-based methods are not suitable for these goals due to the non-differentiability of the objective, while classic iterative search methods suffer from their sensitivity to the initialization. We propose a novel framework for finding a set of tight bounding boxes of a 3D shape via over-segmentation and iterative merging and refinement. Our result shows that utilizing effective search methods with appropriate objectives is the key to producing bounding boxes with both properties. We employ an existing pre-segmentation to split the shape and obtain over-segmentation. Then, we apply hierarchical merging with our novel tightness-aware merging and stopping criteria. To overcome the sensitivity to the initialization, we also define actions to refine the bounding box parameters in an Markov Decision Process (MDP) setup with a soft reward function promoting a wider exploration. Lastly, we further

improve the refinement step with Monte Carlo Tree Search (**MCTS**) based multi-action space exploration. By thoughtful evaluation on diverse 3D shapes, we demonstrate full coverage, tightness, and an adequate number of bounding boxes of our method without requiring any training data or supervision. It thus can be applied to various downstream tasks in computer vision and graphics.

## 1. Introduction

Approximating complex 3D shapes using primitives offers several capabilities, including shape structure analysis, shape abstraction, and efficient geometric computations. To achieve these, many recent self-supervised learning approaches [10, 28–30, 37, 38, 41] have successfully addressed the problem while exploring different types of primitives.

Despite the recent advances, in this work, we pay attention to some desired yet underinvestigated properties of the bounding primitives of a shape and propose a novel approach aiming to achieve them. The properties are 1) **full coverage** — guaranteeing the boundness of the entire shape

by the primitives, 2) **tightness**, and 3) **adequate number of primitives**. Attaining these three properties is particularly crucial for the downstream applications requiring *efficient geometric computations*, such as intersection tests [19, 20], robust transmissions [5], ray tracing [6], or proximity computations [14]. The primitives that just approximate but do not fully cover the shape can result in imprecise computation in such tasks. Also, loosely bounding primitives and too many primitives increase the computation time. Moreover, the bounding primitives satisfying these properties typically provide a better abstraction of the shape aligned with the *human perception* of the shape decomposition.

Although neural-network-based methods have demonstrated their powerful generalizability and expressivity, they typically fail to achieve these three properties mainly due to the *non-differentiable* nature of the objectives. The full coverage and tightness can be computed with the volumetric intersection or difference operations, which are not differentiable. Finding the proper number of parts is also a discrete problem that cannot be easily solved via backpropagation.

As tight bounding primitives have been essential in various applications, there also has been a line of work [5, 22, 34] addressing the problem before the deep learning era. One notable example is the work by Lu et al. [26] that proposes to find the tight bounding boxes by iteratively exchanging point-to-primitive assignments, starting from an initialization. The major drawback of such a method is its dependency and sensitivity to the *initialization*, resulting in a suboptimal output when it starts from a poor initialization.

To address the problem of achieving the three properties that introduce challenges of *non-differentiability* for neural networks and *initialization sensitivity* for iterative search methods, we propose a framework that performs **Splitting, Merging, And Refinement Techniques**, and is thus dubbed **SMART**. We first find that a simple post-processing applied to a pre-segmentation, can provide appropriate **over-segmentation** of a 3D shape. Hence, we perform **hierarchical merging** to find the adequate number of parts. We introduce *tightness-aware* merging and stopping criteria that enable selection of the optimal number of parts the best number of parts given the trade-off between tightness and parsimony in the decomposition.

While merging over the over-segments already produces promising results, the results are yet dependent on the quality of the over-segmentation (third column of Fig. 1-a). To overcome the dependency to the initialization, we present the next **refinement** step that adjusts the bounding box parameters following a sequence of predefined actions. We design a Markov Decision Process (MDP) setup with a *soft* reward function that allows the bounding boxes to break the full coverage in the *middle* of the process. This is the key to having more flexibility in traversing various cases and obtaining better results at the end (fourth column of Fig. 1-

a). Lastly, we extend the refinement step to see not only a single step of actions but multiple steps in a small sequence. We utilize **MCTS** [7] and introduce acceleration techniques (in the supplementary) to speed up the additional refinement (last column of Fig. 1-a).

In our experiments with ShapeNet [8], we demonstrate that our method guaranteeing full coverage provides better tightness and reconstruction compared with the baseline methods while approximating the shapes into similar numbers of cuboids. We additionally show that the decomposition of a shape based on our bounding boxes is better aligned with the semantic parts than other methods. Furthermore, we provide the result of SMART on Objaverse [12] and OmniObject3D [40] to show its applicability to real data and various categories of 3D shapes.

To summarize,

- We present a novel framework for finding a set of tight bounding boxes of a 3D shape by optimizing volumetric objectives with iterative search methods.
- We first propose a hierarchical merging method exploiting over-segmentation from a pre-segmentation and tightness-aware merging and stopping criteria.
- We also introduce a bounding box refinement process with an effective soft reward function that allows wider exploration in the action space.
- Finally, we present an MCTS-based efficient exploration of multi-action sequences with acceleration techniques.

## 2. Related Work

**Learning-Based Shape Abstraction.** As neural network-based learning approaches have shown powerful performance, various unsupervised learning-based approaches have attempted to abstract/reconstruct the shape with simple primitives. For primitives, cuboids [37, 38, 41], superquadrics [29], convexes [10, 13] and implicit fields [9, 28, 30] have been explored. Tulsiani et al. [38] use deep convolutional neural networks, directly predicting volumetric primitives (VP) and their translation parameters. Sun et al. [37] try to discover the adaptive hierarchical cuboid abstraction (HA) to exploit the structural coherence of 3D shapes. Also, Yang and Chen [41] propose an unsupervised approach that jointly predicts the cuboid parameters and segmentation (CA) of input point clouds giving feedback to each other. Paschalidou et al. [29] use superquadrics (SQ) as primitives for abstraction which can represent cylinders, spheres, cuboids, ellipsoids, etc. Furthermore, convexes and implicit fields which are more general representations have been used as primitives to enhance the performance of reconstruction. One such approach by Chen et al. [10] design a network to learn the associations of binary space partitioning (BSP-Net) as an implicit representation.

**Volume-Based 3D Shape Decomposition.** Before learning-based approaches became popular, classical

approaches tried to segment 3D shapes by exploiting volumetric information. Lu *et al.* [26] propose a variational formulation using volumetric information to compute a tight bounding volume and segmentation. Also, Attene *et al.* [2] define part concavity with volumetric information to obtain weakly convex decompositions. Asafi *et al.* [1]’s work is based on visibility (line of sight), which is one of the volumetric properties. Shapira *et al.* [32] define a shape diameter function that expresses the diameter of the object’s volume in the nearby point on the surface for consistent mesh partitioning. Liu *et al.* [25] design a part-aware surface metric that considers the volumetric context when encoding part information. Kaick *et al.* [21] merge weakly convex components taking account of the volumetric profile of the parts to obtain the final segmentation. In the sense of exploiting volumetric information to capture part-level segmentation, SMART is similar to these approaches. However, such classical approaches usually start without any initialization while SMART can effectively utilize shape-dependent initialization learned by neural networks (BSP-Net) [10]. Moreover, we propose an additional step to refine the errors and failures of these approaches.

**Efficient Search for 3D Applications.** Efficient search is crucial for tackling high-dimensional search spaces since we cannot exhaustively search all the possible solutions. Continuity and extra dimensionality in 3D data make the search space extremely large, requiring dedicated algorithms for efficient searches. For this purpose beam search [15, 33], monte carlo tree search (MCTS) [16, 35, 36, 39] and learning [24, 33] approaches have been explored to solve various 3D computer vision tasks such as shape reconstruction, scene detection and convex decomposition. Sharma *et al.* [33] propose a neural shape parser that learns to generate CSG grammar to challenge the huge search spaces. After obtaining the results from the shape parser, they apply beam search to find the best result. Similarly, Lin *et al.* [24] propose a learning-based approach to reconstruct the given shape that effectively searches the large discrete action spaces. Hampali *et al.* [16] use MCTS to detect 3D objects in the scenes to guide the solution faster. Wei *et al.* [39] also utilize MCTS to overcome the limitations of one-step greedy search. In SMART, we change the optimization problem of finding tight bounding boxes to a search problem and apply MCTS to effectively tackle the huge search space to overcome the limitations of our one-step greedy heuristic search.

### 3. Problem Definition and Overview

Our goal is to determine a set of bounding boxes, denoted as  $\{B_i\}_{i=1}^M$ , for a given 3D shape  $S$  represented as a tetrahedral mesh. It is worth noting that any watertight mesh can be converted into a tetrahedral mesh using an off-the-shelf technique [17]. To define the desired set of bounding boxes,

we first specify three criteria: (1) **coverage**, (2) **tightness**, and (3) the **number of boxes**.

The measure of the **coverage** ( $\text{Cov}$ ) is defined as follows:

$$\text{Cov}(S, \{B_i\}) = 1 - \frac{\text{vol}\left(S \setminus (\bigcup_{i=1}^M B_i)\right)}{\text{vol}(S)}, \quad (1)$$

where  $\text{vol}(\cdot)$  is a function measuring the volume of either a mesh or a bounding box.  $\text{Cov}(S, \{B_i\})$  measures the proportion of the volume of the input shape  $S$  that is covered by the bounding boxes. Our goal is to find a set of bounding boxes that *fully* cover the given shape, and hence  $\text{Cov}(S, \{B_i\})$  needs to be equal to one.

The **tightness** ( $\text{Tgt}$ ) is measured as follows, inspired by the variational formulation by Lu et al. [26]:

$$\text{Tgt}(S, \{B_i\}) = \sum_{i=1}^M \frac{\text{vol}(B_i)}{\text{vol}(S)}. \quad (2)$$

When the coverage is one, the ideal set of bounding boxes that tightly fits the input shape has the minimum value of  $\text{Tgt}(S, \{B_i\})$ , which is close to one. Hence, our objective function for finding the tightest set of bounding boxes while fully covering the shape is formulated as follows:

$$\arg \min_{\{B_i\}_{i=1}^M} \text{Tgt}(S, \{B_i\}) \text{ s.t. } \text{Cov}(S, \{B_i\}) = 1. \quad (3)$$

There can be multiple configurations of bounding boxes that achieve similar tightness with full coverage. Among these configurations, we aim to identify the set of bounding boxes with the **minimum number of boxes**.

The objective function for coverage and tightness mentioned above involves non-differentiable operations, such as volume union and difference. Consequently, using gradient-descent-based methods to solve the problem is not feasible. Moreover, finding the minimal set of bounding boxes involves minimizing discrete variables, making the problem highly non-trivial and unsuitable for solving using neural network-based methods. As a result, recent learning-based methods fitting bounding boxes [37, 38, 41] fail to achieve at least one of the desired criteria above: full coverage, tightness, or the minimum number of bounding boxes.

To this end, we propose a novel *non-learning* framework that finds a set of bounding boxes satisfying the above criteria through three main steps: **split**, **merge**, and **refinement**. We first **split** the given 3D shape using existing techniques [10, 39] and obtain an over-segmentation. It allows us to achieve high performance regardless of the choice of over-segmentation techniques [10, 39].

In the subsequent **merging** phase, we revisit the classical hierarchical merging methods [2, 21, 26] with our volumetric criteria (BAVF) to merge the partitions (Sec. 4.2). BAVF allows us to determine the appropriate grouping of

the initial segments that provides tightness and an adequate number of parts while guaranteeing full coverage.

We aim to improve the bounding boxes through the **refinement** step, which transforms our optimization problem (Eq. 3) into a search problem by iteratively applying discrete unit actions (Sec. 4.3.1). Our key observation in this phase is that guaranteeing full coverage may lead to sub-optimal results in the refinement. Hence, we treat coverage as a soft constraint, penalizing low coverage rather than guaranteeing full coverage. Then, we recover full coverage at the end of the process using a simple heuristic. This approach significantly helps to avoid getting trapped in local minima during the optimization process.

Finally, we additionally address the limitations of the one-step greedy search by proposing a general solution with MCTS, which allows us to perform a multi-step search (Sec. 4.3.2). We also propose techniques to accelerate the MCTS in the supplementary.

## 4. SMART Framework

### 4.1. Initialization via Over-Segmentation

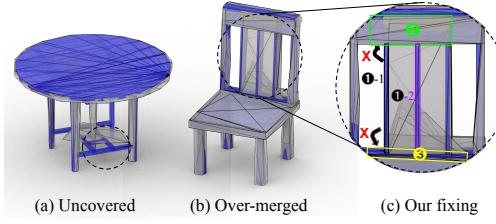


Figure 2. Typical failure cases of pre-segment and our fixing (Blue: Target shape, Grey: BSP-Net). (a) pre-segment that does not fully cover the entire shape, (b) pre-segment that over-merges the closely located parts, and (c) how our initialization can fix the over-merged cases using the flood fill algorithm.

Pre-segments are easier to obtain compared to part-level segmentation or partitions in an unsupervised manner. For example, BSP-Net [10] or convex decomposition [39] (CoACD) can be used as pre-segments in our initialization. However, using them directly as initial over-segmentation is not straightforward due to their common problems: (1) uncovered parts (Fig. 2-a), (2) overlapping segments, and (3) merging of closely located parts (Fig. 2-b). Ignoring these problems will cause ambiguity in parsing volumetric information thereby induce inferior result.

To handle these problems and obtain an initial over-segment (partition) for our purpose, we first split the overlaps and regroup the partitions by using our simple flooding algorithm (Fig. 2-c).

Concretely, to resolve the second limitation, we split each overlap generated by pre-segments into different partitions. For the third limitation, we observe that separated parts in the shape merged by pre-segments typically resemble the case shown at Fig. 2-b. In such cases, if the upper

and lower segments are correctly partitioned, we can separate the parts merged by pre-segments using a flood fill algorithm (Fig. 2-c). Lastly, for the first limitation, we merge the nearby uncovered parts to form each partition. The details are described in the supplementary.

### 4.2. Hierarchical Merging

**Bounding-Box-Aware Volume Function.** We introduce the Bounding-box-Aware Volume Function (BAVF) as our merging criteria in the hierarchical clustering [31] process to obtain the part-level segmentation. This volume-based criterion is the key to grouping the over-segments into part-level without requiring any exhaustive search. BAVF calculates the decrease in the bounding volumes that results from merging the two partitions as follows:

$$\begin{aligned} \text{BAVF}(S_i^t, S_j^t) &= \text{Tgt}(\{B_i\}^t) - \text{Tgt}(\{B_i\}^{t+1}) \\ &= \frac{\text{vol}(B_i^t) + \text{vol}(B_j^t) - \text{vol}(B_{ij}^{t+1})}{\text{vol}(S)}. \end{aligned} \quad (4)$$

$S_i^t$  is the partition of the tetrahedral mesh  $S$  at timestep  $t$ ,  $B_i^t$  and  $B_j^t$  are the oriented bounding boxes calculated by finding the minimum volume of the vertices of  $S_i^t$  and  $S_j^t$ , respectively, and  $B_{ij}^{t+1}$  is the bounding box of  $S_i^t \cup S_j^t$ .

By utilizing BAVF in hierarchical clustering, we can achieve two objectives simultaneously: minimizing  $\text{Tgt}(S, \{B_i\})$  and reducing the number of partitions, while also being able to distinguish between the part-level merging that should occur and not (as shown in Fig. 3). Furthermore, in the early stages of clustering, BAVF tends to prioritize merging nearby partitions rather than distant ones, which aids in achieving the goal of obtaining a part-level segmentation.

**Hierarchical Clustering.** Hierarchical clustering starts by identifying the partition pairs with the largest BAVF value, and merging them if the BAVF value is positive. The threshold  $\epsilon_{merge}$  is used as a stopping criterion to determine when to stop merging. Because of the low resolution of tetrahedral meshes and errors in initialization, a small negative value is used to force the merging of small errors (Fig. 4-a). If the BAVF value is negative, the trade-off between decreasing the number of cuboids and reducing the  $\text{Tgt}(S, \{B_i\})$  is considered. When the BAVF value is larger than the threshold, the partitions are still merged. However, if the BAVF value is smaller than the threshold, partitions are no longer merged, and the hierarchical clustering is terminated. The details are described in the supplementary.

After obtaining the part-level segmentation with merging, the oriented bounding box that covers each part-level segmentation with minimum volume is calculated.

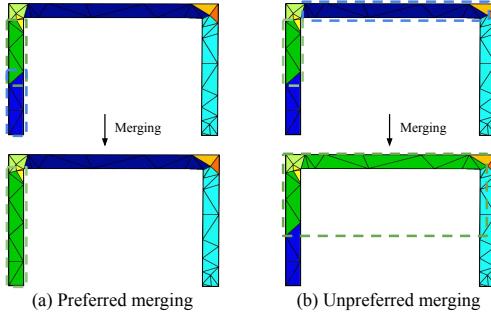


Figure 3. When merging the green and blue partitions, (a) shows a preferred merging case that decreases the bounding volume. (b) shows an unpreferred merging case that significantly increases bounding volume.

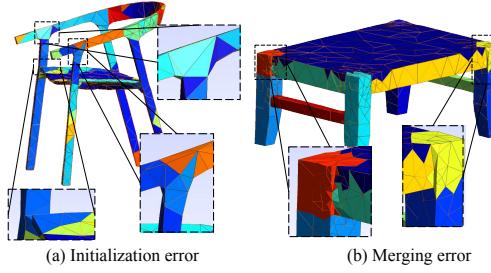


Figure 4. (a) shows the initialization errors where a partition that should not be in the same partition is initialized together. (b) shows the merging errors where partitions that should be merged are not.

### 4.3. Bounding Box Refinement

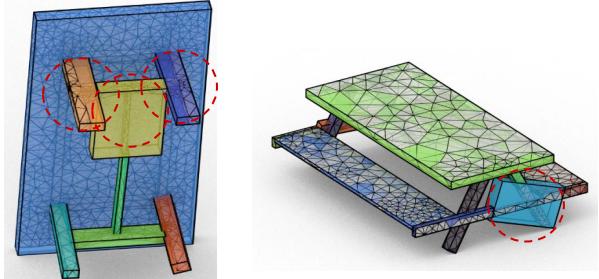
#### 4.3.1 Greedy Refinement

Although the bounding boxes obtained from merging (Sec. 4.2) hold potential, they are not as tightly fitted as desired. Also, errors can occur during the initialization step (as shown in Fig. 4-a), causing the bounding boxes to become loose. Additionally, the merging step (Fig. 4-b) can result in some unmerged bounding boxes, leading to have unnecessary boxes.

Our refinement step targets to fix these errors by formulating the optimization problem into Markov Decision Process (MDP) [4]. We can define the state and action space like below and give reward as the improvement of  $\text{Tgt}(S, \{B_i\})$  while  $\text{Cov}(S, \{B_i\})$  is 1 following our hard objective (Eq. 3). We iteratively apply greedy actions that minimize  $\text{Tgt}(S, \{B_i\})$  but do not decrease  $\text{Cov}(S, \{B_i\})$ .

**State.** Each bounding box is parameterized with the two diagonal points of the cuboid and a rotation matrix  $R$  at time-step  $t$ . If one of the coordinates of the left vertex becomes bigger than the right vertex, we treat the bounding box as deleted. Note that  $(lx_i, ly_i, lz_i)$  represents the left diagonal point of the bounding box.

$$\{B_i^t\}_{i=1}^M = \{(lx_i, ly_i, lz_i, rx_i, ry_i, rz_i, R_i)\}_{i=1}^M \quad (5)$$



(a) The case needs to break hard Cov (b) Failure case of soft refinement

Figure 5. (a) shows the case that needs to perform actions violating Cov constraint. (b) shows the failure case of soft refinement that can not handle the rotation of the bounding box decided by the wrong segmentation.

**Action.** There are two types of actions available for each bounding box: (1) adding or subtracting a predefined unit scale to one of the parameters in  $(lx_i, ly_i, lz_i, rx_i, ry_i, rz_i)$  and (2) changing the rotation matrix  $R_i$  to orient the bounding box correctly based on the points it covers. The second action is necessary to correct the wrong rotations derived from poor segmentation. In total, there are  $6 \times 2 + 1 = 13$  actions for modifying each bounding box.

**Soft Refinement.** With the hard constraint, obtaining a satisfactory refinement of cases similar to Fig. 5-a is impossible. To address this, we make adjustments to the initial objective function (Eq. 3) by transforming the hard constraint into a flexible soft constraint. However, to ensure  $\text{Cov}(S, \{B_i\})$  to be close to 1, we introduce a coefficient denoted as  $\alpha$ . We iteratively apply one-step greedy actions that minimize the objective function

$$\arg \min_{\{B_i\}_{i=1}^M} \text{Tgt}(S, \{B_i\}) - \alpha \text{Cov}(S, \{B_i\}). \quad (6)$$

This updated objective function enables a trade-off between coverage and minimizing the bounding volume. Consequently, refinement attempts can be made to break  $\text{Cov}(S, \{B_i\}) = 1$ , while still finding other actions that guarantee a  $\text{Cov}(S, \{B_i\})$  of almost 1, leading to a decrease in  $\text{Tgt}(S, \{B_i\})$ . With an appropriate  $\alpha$ , our soft refinement approach can refine such cases (Fig. 5-a), while maintaining  $\text{Cov}(S, \{B_i\})$  to be almost 1.

**Post-Processing.** The soft refinement may neglect small-volume parts of the shape even with such high  $\alpha$ . So, to guarantee full coverage, we post-process the outputs by fitting boxes for each nearby uncovered part or using a ternary search to fit the boundaries of the bounding boxes which are smaller than the predefined unit scale.

#### 4.3.2 Monte Carlo Tree Search (MCTS)

Fig. 5-b demonstrate the insufficiency of one-step greedy search methods (soft refinement) in resolving our problem.

	Class	#Box	Tgt↓	Cov↑	MOV↓	TOV↓	CD↓	VIoU↑	mAP↑
Table	HA [37]	4.59	-	0.69	-	-	1.41	0.41	0.31
	HA [37]+Ref.	4.39	3.69	0.99	6.78	2.64	2.05	0.39	-
	CA [41]	7.49	-	0.61	-	-	1.19	0.45	0.51
	CA [41]+Ref.	5.31	3.23	<b>1.00</b>	6.05	2.21	1.75	0.40	-
	BSP-Net [10]	12.9	-	-	-	-	-	-	0.51
	Ours (Merge)	8.89	2.21	<b>1.00</b>	3.95	1.04	0.76	0.60	0.58
	Ours (H Ref.)	8.43	2.00	<b>1.00</b>	4.72	0.95	0.74	0.63	0.59
	Ours (Ref.)	7.93	1.72	<b>1.00</b>	2.82	0.69	0.64	0.67	<b>0.60</b>
	Ours (MCTS)	7.87	<b>1.69</b>	<b>1.00</b>	<b>2.58</b>	<b>0.67</b>	<b>0.63</b>	<b>0.68</b>	<b>0.60</b>
Chair	HA [37]	6.34	-	0.76	-	-	1.42	0.49	0.38
	HA [37]+Ref.	5.70	3.23	<b>1.00</b>	5.82	2.19	2.76	0.41	-
	CA [41]	9.10	-	0.72	-	-	<b>0.81</b>	0.56	0.57
	CA [41]+Ref.	7.78	2.54	<b>1.00</b>	5.14	1.50	1.55	0.49	-
	BSP-Net [10]	20.5	-	-	-	-	-	-	0.48
	Ours (Merge)	11.58	2.09	<b>1.00</b>	4.29	0.83	0.96	0.63	0.58
	Ours (H Ref.)	11.28	1.86	<b>1.00</b>	6.02	0.76	0.98	0.66	<b>0.59</b>
	Ours (Ref.)	10.84	1.64	<b>1.00</b>	2.96	0.58	0.84	<b>0.69</b>	<b>0.59</b>
	Ours (MCTS)	10.81	<b>1.62</b>	<b>1.00</b>	<b>2.73</b>	<b>0.56</b>	0.83	<b>0.69</b>	<b>0.59</b>
Airplane	HA [37]	7.39	-	0.68	-	-	0.52	0.50	-
	HA [37]+Ref.	7.00	2.91	<b>1.00</b>	4.82	1.82	0.89	0.37	-
	CA [41]	9.48	-	0.59	-	-	0.45	0.46	-
	CA [41]+Ref.	8.16	2.83	<b>1.00</b>	5.50	1.72	0.69	0.38	-
	Ours (Merge)	13.2	2.62	<b>1.00</b>	3.82	1.21	0.42	0.47	-
	Ours (H Ref.)	13.0	2.34	<b>1.00</b>	4.56	1.14	0.45	0.48	-
	Ours (Ref.)	12.6	2.06	<b>1.00</b>	3.81	0.93	0.39	<b>0.53</b>	-
	Ours (MCTS)	12.6	<b>2.05</b>	<b>1.00</b>	<b>3.77</b>	<b>0.92</b>	<b>0.38</b>	<b>0.53</b>	-

Table 1. Comparison of SMART and baselines. Ref. stands for soft refinement (Eq. 6) in our framework and H Ref. stands for refinement using the hard objective (Eq. 3). # Box denotes average number of primitives. Note that all CD is scaled by 1000.

Due to the huge search space of size  $(M \times 13)^T$  with multiple local minima, it fails to find actions that can fix the wrong rotation. It occurs because rotating typically does not result in an immediate improvement in  $\text{Tgt}(S, \{B_i\})$ , making it unfixable through a greedy approach. To mitigate this issue, we propose utilizing MCTS, which allows us to efficiently search the huge search space by simulating multiple steps while saving the explored results and utilizing them for further searches.

**Tree Structure and Iteration.** In our tree search, each node represents the state of our MDP setup. The state is characterized by the parameters of the bounding boxes  $\{B_i^t\}_{i=1}^M$ , and each edge represents an action. The root node represents the initial bounding boxes, and we aim to tightly refine the initial bounding boxes using a sequence of actions within a limited time-step  $T$ . Each node has a child node that corresponds to the state of the bounding boxes after applying possible actions to its node’s state. After running pre-defined iterations, we take the best bounding boxes as the result. Refer to the supplementary for the details of MCTS and its acceleration.

## 5. Results

Our bounding boxes are tight bounding volumes that can preserve structural details, resulting in precise reconstruction results without requiring any training data. Also, they provide instance-level segmentation since each box covers each part without overlaps. We comprehensively evaluate

our approach’s tightness, reconstruction, and instance-level segmentation performance compared to other cuboid abstraction baselines. And we conduct an ablation study to validate our refinement step and MCTS. Furthermore, we test SMART on diverse categories to show its applicability.

### 5.1. Implementation Details

We evaluate SMART on the ShapeNet [8] dataset using 9 categories including 3496 tables, 2500 chairs, and 531 airplanes. Additionally, we randomly selected shapes from Objaverse [12] and OmniObject3D [40] to evaluate SMART. BSP-Net [10] was used as the pre-segment in the initialization step for the ShapeNet meshes, while CoACD [39] was used as the pre-segment for the Objaverse and OmniObject3D meshes. Furthermore, for the volume calculation of meshes, we converted them into watertight tetrahedral mesh by using the algorithm of Huang *et al.* [18] and fTetwild [17]. Note that finding an oriented bounding box is implemented with trimesh [11]. In SMART, we used  $\epsilon_{merge}$  as -0.02 and  $\alpha$  as 100 and  $c$  with 0.001 in all of our experiments.

### 5.2. Evaluation Metrics

For evaluating the tightness of our bounding boxes, we use TOV and MOV proposed by Lu *et al.* [26]. Tgt and Cov are also used since they are the main objectives of our problem. Also, to evaluate the reconstruction performance, we adopt the commonly-used chamfer distance [3] (CD) and volumetric IoU (VIoU). Part-level segmentation is evaluated by calculating the mean Average Precision (mAP) using instance labels from PartNet [27].

- **Total Outside Volume:**

$$\text{TOV}(B) = \frac{\text{vol}(\bigcup_i B_i \setminus S)}{\text{vol}(S)}$$

- **Maximum local Outside Volume:**

$$\text{MOV}(B, \{S_i\}) = \max_i \frac{\text{vol}(B_i \setminus S)}{\text{vol}(S_i)}$$

- **Volumetric Intersection over Union:**

$$\text{VIoU}(B) = \frac{\text{vol}(S \cap \bigcup_i B_i)}{\text{vol}(S \cup \bigcup_i B_i)}$$

### 5.3. Tightness of Bounding Boxes

We first evaluate the tightness of the bounding boxes with Tgt, Cov, MOV [26], and TOV [26]. Since there is no baseline work that directly solves the problem of finding tight bounding boxes, we apply our refinement algorithm to the recent state-of-the-art cuboid abstraction works HA [37] and CA [41] to achieve the  $\text{Cov}(S, \{B_i\})$  to be 1. This is necessarily for a fair comparison since TOV, MOV can be properly evaluated only if  $\text{Cov}(S, \{B_i\})$  is 1. We report the result of the refined baselines (row 2, 4 in each class) and SMART (last four row in each class) in Tab. 1. Even compared without refinement on our merging step, SMART

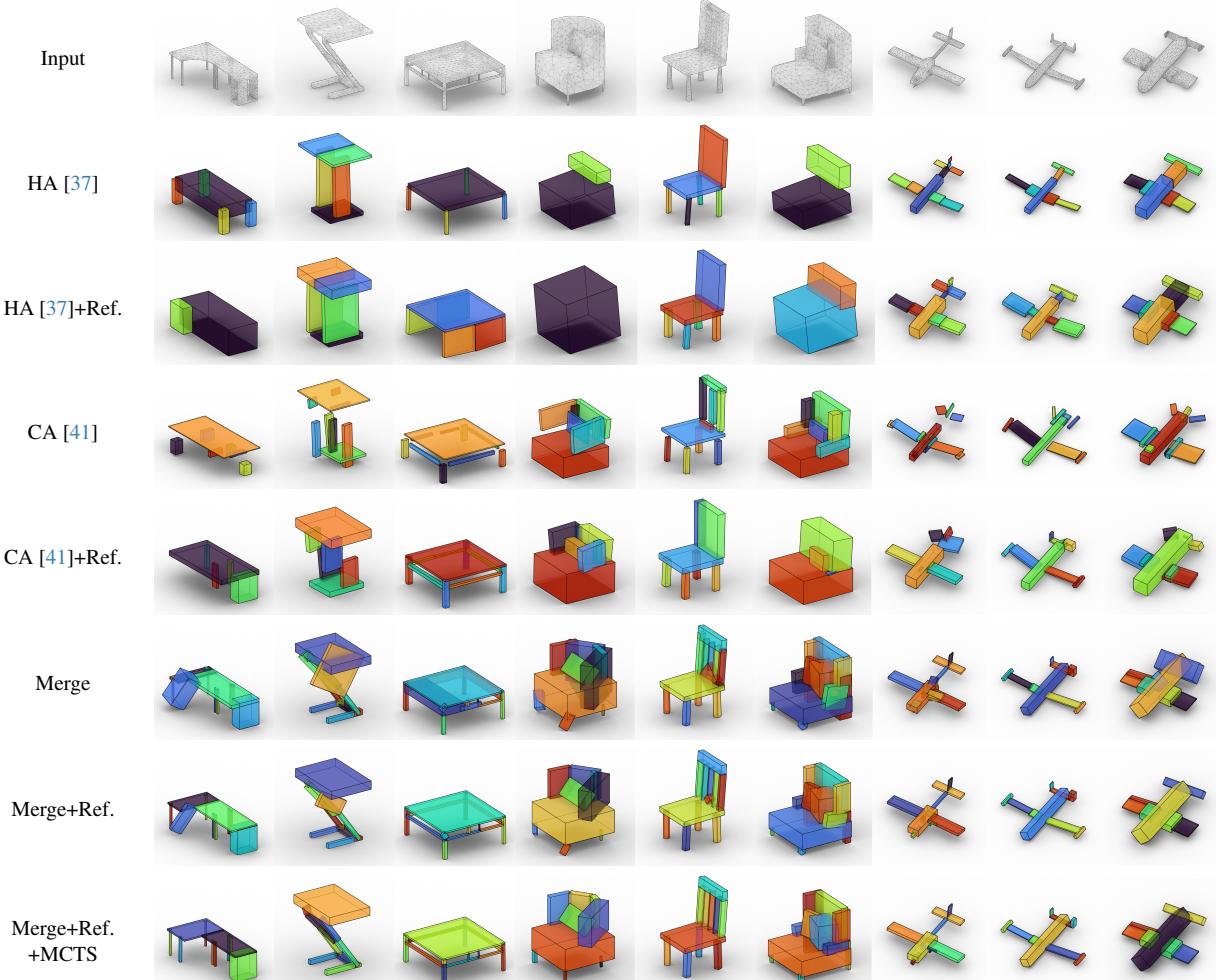


Figure 6. Qualitative comparison between SMART and baselines.

outperforms the baselines in all metrics showing big margins. We can also see that our refinement algorithm successfully tightens the output of our merging step. We give a qualitative comparison in Fig. 6.

#### 5.4. Shape Reconstruction

We compare the reconstruction performance of SMART to CA [41] and HA [37] to demonstrate its superiority. We use CD and VIoU to evaluate the reconstruction performance, where CD was calculated by uniformly sampling 4096 points from the obtained cuboids and the ShapeNet [8] mesh. VIoU was calculated by measuring the IoU between the input mesh and the merged bounding boxes. As shown in Tab. 1 (sixth and seventh columns of metric), SMART outperforms all baselines in VIoU while achieving better or comparable performance in CD. It is important to note that, unlike other cuboid abstraction baselines, we did not directly optimize for CD. For qualitative comparison, refer to Fig. 6, where it can be observed that other base-

lines fail to properly abstract the shape or capture details, while SMART accurately captures them even with the restriction of covering the entire shape ( $\text{Cov}(S, \{B_i\}) = 1$ ).

#### 5.5. Shape Instance Segmentation

Since our bounding boxes are based on part-level segmentation, we also measure its performance by measuring the mean average precision (mAP) of instance segmentation with the labels obtained from PartNet [27]. In addition to HA [37] and CA [41], we also compare with BSP-Net [10] which was used to get our initial segment. We follow the evaluation setting of PartNet [27] to report the instance segmentation results. The segmentation was obtained by identifying the nearest point for each primitive and assigning them to obtain the segmentation mask. Subsequently, we computed the IoU between each segmentation mask and the ground-truth mask and considered a segmentation mask as true positive only when IoU was greater than 0.5. Note that instance segmentation for the airplane category is not reported because PartNet [27] does not have ground-truth la-

belts. The results are reported in the last column of Tab. 1. Compared to other baselines we achieve the best result showing that SMART captures the individual parts in the shape better than other works having more alignment with the human perception of the shape decomposition.

## 5.6. Ablation Study

To examine the benefits introduced by the refinement step and MCTS, we provide a quantitative and qualitative comparison. The last four rows of each class in Tab. 1 shows the results for these cases. MCTS successfully improves the failure cases of one-step greedy refinement by performing multi-step searches for all the categories. Fig. 6 shows the qualitative comparison between our methods. Our soft-refinement fails to handle the rotations, while our MCTS successfully addresses the issue.

## 5.7. More Diverse Categories

We also provide quantitative and qualitative results for other shape categories in ShapeNet [8] in Fig. 7 and Tab. 2, including Bench, Cabinet, Couch, Display, Lamp, and Rifle. Note that other baselines such as HA [37] and CA [41] do not provide any pre-trained models or experiments for other shape categories. Additionally, Fig. 1-b shows the qualitative result of Objaverse [12].

Category	# Box	Tgt $\downarrow$	Cov	MOV $\downarrow$	TOV $\downarrow$	CD $\downarrow$	VIoU $\uparrow$
Bench	10.59	1.92	1.00	3.28	0.87	0.49	0.61
Cabinet	5.5	1.45	1.00	3.85	0.43	1.11	0.81
Couch	6.59	1.30	1.00	3.59	0.25	1.04	0.82
Display	4.45	1.37	1.00	1.24	0.34	0.82	0.76
Lamp	8.93	2.30	1.00	5.75	1.20	1.49	0.54
Rifle	9.06	1.70	1.00	2.79	0.62	0.23	0.63

Table 2. Quantitative result on diverse categories in ShapeNet. Note that all CD is scaled by 1000 and # Box denotes average number of primitives.



Figure 7. Qualitative result on diverse categories in ShapeNet [8].

## 5.8. Application to Real Data

For testing applicability and robustness to real data, we show the result of applying SMART to OmniObject3D [40]

dataset at Fig 8. Note that OmniObject3D meshes are reconstructed from real 3D objects using multi-view images.

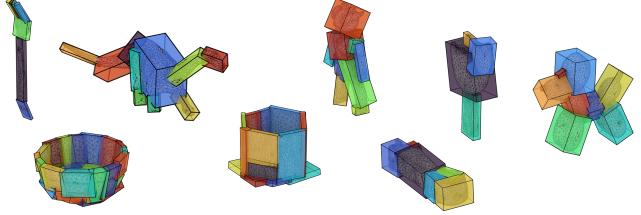


Figure 8. Qualitative result on OmniObject3D [40].

## 5.9. Timing and Scalability Analysis

We give a timing and scalability analysis of our framework. Using our *unoptimized* coded implemented in *Python*, all the 3496 tables in ShapeNet [8] were processed within a day using 48 CPU cores. For further analysis, we randomly sampled 100 shapes from the ShapeNet table category and ran the entire pipeline. For 90% of the shapes, it had fewer than 15 bounding boxes after processing. With a single CPU core, it took 8.48 minutes to run the entire pipeline for each shape on average. Within this time, the *optional* MCTS step took 6.22 minutes per shape on average.

## 6. Conclusion

We presented SMART, a novel framework for finding tight bounding boxes of 3D shapes with pre-segment-based over-segmentation and multiple iterative searches. The framework consists of four steps: 1) **over-segmentation** using pre-segment and post-processing, 2) **merging** in a hierarchical way with tightness-aware criteria, 3) **refinement** with a discrete action space with a soft reward function, and 4) **additional refinement** via multi-action space exploration using MCTS and its acceleration. The experimental results showed the tightness and parsimony of our bounding boxes that also guarantee the full coverage of the shape.

**Limitation.** We have successfully demonstrated the robustness and versatility of our framework across various categories and datasets, all without necessitating any training data. However, we acknowledge certain limitations. As our framework relies on volumetric information to fit cuboids, it requires watertight tetrahedral mesh as input to accurately define the volume of the parts. Additionally, due to the nature of MCTS optimization, it solely can not guarantee  $\text{Cov}(S, \{B_i\})$  to be 1 requiring additional post-processing.

**Acknowledgement.** This work was partly supported by NRF grant (RS-2023-00209723) and IITP grant (2022-0-00594, RS-2023-00227592) funded by the Korean government (MSIT), Seoul R&BD Program (CY230112), and grants from ETRI, KT, NCSOFT, and Samsung Electronics.

## References

- [1] Shmuel Asafi, Avi Goren, and Daniel Cohen-Or. Weak convex decomposition by lines-of-sight. *Comput. Graph. Forum*, 2013. 3
- [2] Marco Attene, Michela Mortara, Michela Spagnuolo, and Bianca Falcidieno. Hierarchical convex approximation of 3d shapes for fast region selection. *Comput. Graph. Forum*, 2008. 3
- [3] Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen Cf Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, 1977. 6
- [4] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957. 5
- [5] S. Bischoff and L. Kobbelt. Ellipsoid decomposition of 3d-models. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*, 2002. 2
- [6] Christian Bouville. Bounding ellipsoids for ray-fractal intersection. In *SIGGRAPH*, 1985. 2
- [7] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012. 2, 12
- [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*, abs/1512.03012, 2015. 2, 6, 7, 8, 11
- [9] Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang. Bae-net: Branched autoencoder for shape co-segmentation. In *ICCV*, 2019. 2
- [10] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *CVPR*, 2020. 1, 2, 3, 4, 6, 7, 11
- [11] Dawson-Haggerty et al. trimesh. 6
- [12] 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. 1, 2, 6, 8
- [13] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvnet: Learnable convex decomposition. In *CVPR*, 2020. 2
- [14] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2
- [15] Abhinav Gupta, Martial Hebert, Takeo Kanade, and David Blei. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. *NeurIPS*, 2010. 3
- [16] Shreyas Hampali, Sinisa Stekovic, Sayan Deb Sarkar, Chetan S Kumar, Friedrich Fraundorfer, and Vincent Lepetit. Monte carlo scene search for 3d scene understanding. In *CVPR*, 2021. 3
- [17] Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM TOG*, 2020. 3, 6
- [18] Jingwei Huang, Yichao Zhou, and Leonidas Guibas. Manifoldplus: A robust and scalable watertight manifold surface generation method for triangle soups. *arXiv preprint arXiv:2005.11621*, abs/2005.11621, 2020. 6
- [19] P. Jiménez, F. Thomas, and C. Torras. 3d collision detection: a survey. *Computers & Graphics*, 2001. 2
- [20] J Jylanki. An exact algorithm for finding minimum oriented bounding boxes. *Semantic Scholar*. Available online: <https://pdfs.semanticscholar.org/a76f/7da5f8bae7b1fb4e85a65bd3812920c6-d142.pdf>, 2015. 2
- [21] Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-OR. Shape segmentation by approximate convexity analysis. *ACM TOG*, 2015. 3
- [22] Aravind Kalaiah and Amitabh Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM TOG*, 2005. 2
- [23] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, 2006. 13
- [24] Cheng Lin, Tingxiang Fan, Wenping Wang, and Matthias Nießner. Modeling 3d shapes by reinforcement learning. In *ECCV*, 2020. 3
- [25] Rong Liu, Hao Zhang, Ariel Shamir, and Daniel Cohen-Or. A part-aware surface metric for shape analysis. *Comput. Graph. Forum*, 2009. 3
- [26] Lin Lu, Yi-King Choi, Wenping Wang, and Myung-Soo Kim. Variational 3d shape segmentation for bounding volume computation. *Comput. Graph. Forum*, 2007. 2, 3, 6
- [27] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *CVPR*, 2019. 6, 7
- [28] Chengjie Niu, Manyi Li, Kai Xu, and Hao Zhang. Rimnet: Recursive implicit fields for unsupervised learning of hierarchical shape structures. In *CVPR*, 2022. 1, 2
- [29] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *CVPR*, 2019. 2
- [30] Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In *CVPR*, 2021. 1, 2
- [31] Sakshi Patel, Shivani Sihmar, and Aman Jatain. A study of hierarchical clustering algorithms. In *2015 2nd international conference on computing for sustainable global development (INDIACom)*, pages 537–541. IEEE, 2015. 4
- [32] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Vis. Comput.*, 2008. 3
- [33] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *CVPR*, 2018. 3

- [34] Patricio D Simari and Karan Singh. Extraction and remeshing of ellipsoidal representations from mesh data. In *Graphics Interface*, 2005. 2
- [35] Sinisa Stekovic, Mahdi Rad, Friedrich Fraundorfer, and Vincent Lepetit. Montefloor: Extending mcts for reconstructing accurate large-scale floor plans. In *ICCV*, 2021. 3
- [36] Sinisa Stekovic, Mahdi Rad, Alireza Moradi, Friedrich Fraundorfer, and Vincent Lepetit. Mcts with refinement for proposals selection games in scene understanding. *IEEE TPAMI*, 2022. 3
- [37] Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM TOG*, 2019. 1, 2, 3, 6, 7, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
- [38] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 1, 2, 3
- [39] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM TOG*, 2022. 3, 4, 6, 11
- [40] Tong Wu, Jiarui Zhang, Xiao Fu, Yuxin Wang, Jiawei Ren, Liang Pan, Wayne Wu, Lei Yang, Jiaqi Wang, Chen Qian, et al. Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. In *CVPR*, pages 803–814, 2023. 2, 6, 8
- [41] Kaizhi Yang and Xuejin Chen. Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds. *ACM TOG*, 2021. 1, 2, 3, 6, 7, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

## Appendix

### A.1. Comparison of Different Initialization

We show two possible pre-segment initialization method introduced in Sec. 4.1 and its quantitative result. One is the output of BSP-Net [10] and the other is convex decomposition method [39]. Tab. A3 shows the quantitative results with Tables, Chairs, and Airplanes in ShapeNet [8] averaged all together. It demonstrates that our method is *not* sensitive to the choice of the pre-segment in the initialization and other pre-segments can also be used with our algorithm.

	Table, Chair, and Airplane				
Initialization	Tgt↓	Cov↑	MOV↓	TOV↓	VIoU↑
BSP-Net [10]	1.79	1.00	3.03	0.72	0.63
CoACD [39]	1.82	1.00	2.75	0.76	0.62

Table A3. Comparison of SMART by using different initializations. Both initializations are processed by Merge+Ref.+MCTS.

### A.2. Effect of $\epsilon_{merge}$ in Hierarchical Merging (Sec. 4.2 in the Main Paper)

To investigate the trade-off between reducing the number of boxes and minimizing Tgt in the merging step (Sec. 4.2), we varied  $\epsilon_{merge}$  to  $(0, -0.004, -0.02, -0.1)$  on the Table category of ShapeNet [8]. The quantitative results are presented in Tab. A4. It is observed that reducing  $\epsilon_{merge}$  decreases the number of bounding boxes while slightly worsening other volumetric metrics showing the trade-off. We chose our  $\epsilon_{merge}$  as  $-0.02$  because it has the best mAP value meaning that it has the best alignment with human perception giving us more plausible part-level segmentation with adequate number of boxes. However, we can notice that other metrics remain similar by changes of  $\epsilon_{merge}$  indicating that intensive hyper-parameter tuning is not necessary for our merging step.

$\epsilon_{merge}$	# boxes↓	mAP↑	Tgt↓	Cov↑	MOV↓	TOV↓	VIoU↑
0	14.25	0.57	2.20	1.00	3.89	1.01	0.61
-0.004	11.41	0.56	2.19	1.00	3.95	1.02	0.61
-0.02	8.89	<b>0.58</b>	2.21	1.00	3.95	1.04	0.60
-0.1	6.59	0.57	2.30	1.00	4.16	1.15	0.57

Table A4. Effect of  $\epsilon_{merge}$  in merging step (Sec. 4.2) at ShapeNet [8] Table category. The best mAP is marked in **bold** showing the best alignment with human perception.

### A.3. Effect of $\alpha$ in Bounding Box Refinement (Sec. 4.3.1 in the Main Paper)

We examine the benefits of converting the hard constraint objective (Eq. 3) to the soft constraint objective (Eq. 6) during the refinement step. We begin by formally introducing

hard refinement, which uses a one-step greedy approach with the hard constraint. We then present the outcomes of both hard refinement and soft refinement using different values of  $\alpha$  to illustrate their impact.

**Hard Refinement.** A simple one-step greedy approach, hard refinement can be employed by finding the optimal actions that reduce  $\text{Tgt}(S, \{B_i\})$  where  $\text{Cov}(S, \{B_i\})$  is 1. Although this approach can enhance the overlaps of bounding boxes, it cannot address scenarios like Fig. 5-a, where breaking the  $\text{Cov}(S, \{B_i\})$  constraint is necessary to refine the boxes along the action sequence.

**Effect of  $\alpha$ .** To investigate the trade-off between coverage and tightness, we demonstrate the result of changing the  $\alpha$  to  $\{1, 10, 100, 1000, \infty\}$  on ShapeNet [8] table category. Note that  $\infty$  represents the result for the hard refinement. Tab. A5 shows the quantitative results.

$\alpha$	# boxes↓	Tgt↓	Cov↑	MOV↓	TOV↓	VIoU↑
1	8.37	1.00	0.7628	—	—	0.66
10	7.77	1.43	0.9887	1.34	0.42	0.74
100	7.90	1.72	0.9996	2.85	0.69	0.67
1000	8.09	1.85	0.9999	4.30	0.81	0.66
$\infty$ (Hard)	8.41	2.02	1.0000	5.18	0.97	0.62

Table A5. Effect of  $\alpha$  in the refinement step (Sec. 4.3.1) at ShapeNet [8] Table category.

By using various  $\alpha$ , we can analyze the trade-off between Cov and other volumetric metrics, such as Tgt, TOV, MOV, and VIoU. Employing hard refinement with  $\alpha = \infty$  ensures a  $\text{Cov}(S, \{B_i\})$  of 1 for all the shapes. Soft refinement ( $\alpha \neq \infty$ ), on the other hand, may result in rare failures where  $\text{Cov}(S, \{B_i\})$  may not reach exactly 1 for some of the shapes while most of the shapes still satisfy the full coverage. However, it generally leads to an overall increase in quality, as illustrated in Tab. A5. Typically, these rare failures occur in situations where bounding boxes fail to cover small, detailed parts with a small volume in the shape. Still, for those rare failure cases, we can simply use the post-processing introduced in the main paper to guarantee that all the shapes have full coverage. This soft refinement objective (Eq. 6) provides a chance to fix the wrong bounding boxes resulting in more tightness.

### A.4. Algorithm Details

In this section, we provide a detailed explanation of our initialization (Sec. 4.1) and hierarchical merging (Sec. 4.2) and MCTS (Sec. 4.3.2) algorithm in the following sections.

#### A.4.1 Initialization via Over-Segmentation (Sec. 4.1 in the Main Paper)

We start by recapping the three limitations of pre-segments. Then, we explain how we address these limitations and give a detailed explanation of our algorithm.

The limitations of the pre-segments are: (1) uncovered parts, (2) overlapping segments, and (3) the merging of closely located parts. To address these issues, our algorithm treats the centroids of tetrahedral mesh covered by the pre-segments (*main-part*) and the uncovered centroids (*sub-part*) differently to deal with the first limitation. We first partition the *main-part* and then handle the *sub-part* partition, as the *sub-part* represents the failure cases of pre-segments. To resolve the second limitation, we separate each overlap generated by pre-segments into different partitions in the *main-part*. For the third limitation, we observe that separated parts in the shape merged by pre-segments typically resemble the case shown at Fig. 2-b. In such cases, if the upper and lower segments are correctly predicted, we can separate the parts merged by pre-segments using a depth-first search (Fig. 2-c). After partitioning all the *main-part*, we merge the nearby left *sub-part* to form each partition.

These ideas are directly implemented in Algorithm 1. We separate the centroids into *main-part* and *sub-part* by calculating the SDF of centroids  $\{c_i\}_{i=1}^M$  of P with each pre-segments  $\{Q_j\}_{j=1}^N$ . To determine which pre-segments  $Q_j$  cover the centroid  $c_i$ , we obtain a length N boolean mask. We mark the boolean mask to 1 if SDF value of  $c_i$  and  $Q_j$  is positive (Line 16 in Algorithm 1). These masks separate every overlap generated by pre-segments into a different partition. Then we regroup the *main-part* using MainDFS that have the same *mask* (Line 20 in Algorithm 1). Finally, we merge the left nearby *sub-part* using SubDFS to obtain the initial over-segmentation (Line 25 in Algorithm 1).

#### A.4.2 Hierarchical Merging (Sec. 4.2 in the Main Paper)

As shown in Algorithm 2, we first start by calculating Bounding-box-Aware Volume Function (BAVF, Eq. 4) for all the partition pairs. Then, we select the pair that has the maximum value of BAVF. If such value is larger than  $\epsilon_{merge}$ , we merge the partitions and if not, we terminate our hierarchical merging algorithm. Since merging of  $S_i$  and  $S_j$  does not change the BAVF value for other pairs, we effectively cache those values to reduce the time complexity of  $\mathcal{O}(N^3)$  to  $\mathcal{O}(N^2)$ .

---

#### Algorithm 1: Computing Over-Segmentation using Pre-Segments

---

```

Input: Pre-segments  $\{Q_i\}_{i=1}^N$ , tetrahedral mesh P
        having  $\{c_j\}_{j=1}^M$  tetrahedrals
Output: Initial over-segmentation  $\{S_k\}_{k=1}^K$  of P
1 Function MainDFS ( $j, k, mask$ ) :
  2    $visited[j] \leftarrow$  True
  3    $S_k \leftarrow S_k + \{c_j\}$ 
  4   for  $id \in P.\text{nearby}(c_j)$  do
  5     if  $mask[id] == mask[j]$  and not  $visited[id]$ 
  6       then
  7         MainDFS ( $id, k$ )
  8
7 Function SubDFS ( $j, k$ ) :
  9    $visited[j] \leftarrow$  True
 10   $S_k \leftarrow S_k + \{c_j\}$ 
 11  for  $id \in P.\text{nearby}(c_j)$  do
 12    if not  $visited[id]$  then
 13      SubDFS ( $id, k$ )
 14
 15  $k = 1$ 
 16 for  $j \in \{1\dots M\}$  do
 17    $visited[j] \leftarrow$  False
 18    $mask[j] \leftarrow \text{GetSDFMask}(c_j, Q)$ 
 19   for  $j \in \{1\dots M\}$  do
 20     if not  $visited[j]$  and  $mask[j] \neq 0$  then
 21        $S_k \leftarrow \emptyset$ 
 22       MainDFS ( $j, k, mask$ )
 23        $k \leftarrow k + 1$ 
 24
 25 for  $j \in \{1\dots M\}$  do
 26   if not  $visited[j]$  then
 27      $S_k \leftarrow \emptyset$ 
 28     SubDFS ( $j, k$ )
 29      $k \leftarrow k + 1$ 

```

---

#### A.4.3 Monte Carlo Tree Search (Sec. 4.3.2 in the Main Paper)

As Algorithm 3 shows, our MCTS consists of iteratively running  $iter$  iterations of the four steps: (1) selection, (2) expansion, (3) evaluation, and (4) backpropagation similar to the original MCTS [7] (Line 1 in Algorithm 3). We start from the root node  $n_0$  corresponding to the initial bounding boxes  $\{B_i\}_0^0$  we want to tighten, using less than  $T$  actions. In each node, we save the information of  $Q(\cdot)$  which is the value function, where  $N(\cdot)$  indicates the number of visits and its expanded child nodes and untried actions.

In the selection and expansion step, we select the node to expand and expand it. To select the node, we start from the

---

**Algorithm 2:** Hierarchical Clustering with BAVF

---

**Input:** Initial over-segmentation  $S = \{S_i\}_{i=1}^K$   
**Output:** Merged part-level partition  $S = \{S_i\}_{i=1}^N$

```

1 while True do
2    $n \leftarrow \text{len}(S)$ 
3   for  $i, j \leq n$  do
4     └ Calculate BAVF( $S_i, S_j$ )
5     if  $\max_{i \neq j} \text{BAVF}(S_i, S_j) < \epsilon_{\text{merge}}$  then
6       └ break
7      $i^*, j^* \leftarrow \arg \max_{i \neq j} \text{BAVF}(S_i, S_j)$ 
8     Remove  $S_{i^*}, S_{j^*}$  from  $S$ .
9     Add  $S_{i^*} \cup S_{j^*}$  to  $S$ .

```

---

root node and choose the child node with the best UCB [23]:

$$\text{UCB}(n) = Q(n) + c \sqrt{\frac{2 \ln N(n_p)}{N(n)}} \quad (7)$$

if all the possible actions had been tried (in other words, all of its child nodes are expended) (Line 14 in Algorithm 3). Note that  $n$  is the node we want to calculate and  $n_p$  is the parent node of  $n$ . If there is an untried action, we randomly select an untried action and expand that node as its child node (Line 18 in Algorithm 3). When expanding the node, we set the untried actions as all the possible actions except that to go back to the parent state to remove unnecessary expansion.

During the evaluation step, evaluating the expanded node is a non-trivial task when applying MCTS to our problem. To address this challenge, we employ a one-step greedy approach where we try out all possible actions on the expanded node and select the best action iteratively by computing its corresponding  $\text{Score}(\{B_i\}^0, \{B_i\}^t)$  (Line 27 in Algorithm 3, Eq. 8), which reflects the improvement from the initial bounding boxes. Note that  $\text{Score}(\{B_i\}^0, \{B_i\}^t)$  is calculated as:

$$\begin{aligned} \text{Score}(\{B_i\}^0, \{B_i\}^t) = \\ \text{Tgt}(S, \{B_i\}^0) - \text{Tgt}(S, \{B_i\}^t) \\ - \alpha \text{Cov}(S, \{B_i\}^0) + \alpha \text{Cov}(S, \{B_i\}^t). \end{aligned} \quad (8)$$

If the action does not improve the  $\text{Score}$ , we terminate the evaluation process (Line 29 in Algorithm 3).

The last backpropagation step is simple and intuitive. We simply update the number of visits  $N(\cdot)$  and value function  $Q(\cdot)$  along the path (Line 34 in Algorithm 3).

## A.5. MCTS Acceleration Techniques and Ablation Study

In this section, we discuss our dedicated acceleration techniques used in MCTS and demonstrate its performance

through an ablation study.

**Greedy Bounding Box Pruning.** The primary time constraint in the MCTS is selecting the greedy action during node evaluation (Line 27 in Algorithm 3), which involves testing all potential one-step actions. To optimize this process and improve evaluation efficiency, we calculate the greedy action by the unit of each bounding box. If all the actions corresponding to the  $i$ -th bounding box fail to yield any improvement, we do not attempt any action on that bounding box until the end of the evaluation.

**Evaluation Expansion (EE).** To reduce the number of greedy actions, we not only expand the node in the selection step but also expand the node at the evaluation step (Line 31 in Algorithm 3). However, this expansion differs from normal expansion as it only permits the previously taken greedy action at that node. This technique has the same outcome as saving the greedy action at each node, thereby reducing the time of calculating repetitive greedy actions as the time step increases.

**Prioritized Node Selection (PNS).** Since we are using discrete unit actions to refine the bounding boxes, there is a possibility that applying an action that results in a *score* increase may do so again if applied repeatedly. Therefore, prioritizing such actions during node expansion in the selection step can facilitate faster optimization than random node expansion, by deepening the tree instead of widening it.

To implement this technique, we calculate the *score* increase for each expansion action ( $a_u$  at Line 20 in Algorithm 3) and increase the probability of random selection for expansion (Line 19 in Algorithm 3) proportional to the past score increase history while allowing to skip the untried actions of the node (execute Line 14 in Algorithm 3 though the condition is false) with some probability if it is on the path to finding the best reward. We use 0.9 for the skipping probability in our PNS.

**Does the Acceleration in MCTS Really Help?** We now give an ablation study of our acceleration techniques used in MCTS. Due to the time-consuming nature without the greedy bounding box pruning, we only conducted ablations on evaluation expansion (EE) and prioritized node selection (PNS). (Greedy bounding box pruning is the default.) Fig. A9 illustrates the result of the best score and elapsed time as the iteration of the tree search increases. Our EE technique allows us to reduce the required time, and PNS enables us to identify the best score in earlier iterations than without it. These techniques all bring acceleration and additional improvements.

---

**Algorithm 3:** MCTS for Tight Bounding Boxes

---

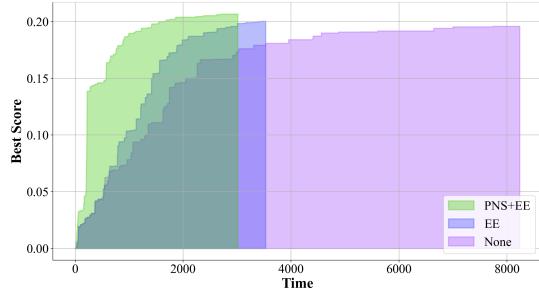


Figure A9. Ablation study on the techniques used in MCTS. X-axis represents the elapsed time in seconds and Y-axis shows the found best score.

```

1 Function MCTS ( $\{B_i\}$ , iter, T) :
2   Create root node  $n_0$  with initial bounding boxes
    $B^0 = \{B_i\}^0$ 
3   while iter iterations do
4      $\{B^1, \dots, B^s\}, n_s \leftarrow \text{Select}(n_0, T)$ 
5      $\{B^{s+1}, \dots, B^t\}, n_t \leftarrow \text{Eval}(n_s, T)$ 
6      $r \leftarrow \text{Score}(B^0, B^t)$  where Score is
       Eq. 8.
7     Backup ( $n_t, r$ )
8      $n^* \leftarrow \arg \max_{n_c \in \text{children of } n_0} Q(n_c)$ 
9   return Action sequence from  $n_0$  to  $n^*$ 

10 Function Select ( $n, T$ ) :
11    $P \leftarrow \emptyset$ 
12   while  $\text{depth}(n) < T$  do
13      $t = \text{depth}(n)$ 
14     if all node of  $n$  are expanded then
15        $n_{bst}, a_{bst} \leftarrow \text{Select the best child and}$ 
         action of  $n$  by the UCB( $n$ ) (Eq. 7).
16       Apply action  $a_{bst}$  to the bounding boxes
          $B^t$  to get  $B^{t+1}$ .
17        $P, n \leftarrow P + \{B^{t+1}\}, n_{bst}$ 
18     else
19       Randomly select a untried action  $a_u$ .
20       Apply action  $a_u$  to the bounding boxes
          $B^t$  to get  $B^{t+1}$ .
21       Create a new child  $n_{chd}$  with  $B^{t+1}$  to  $n$ .
22     return  $P + \{B^{t+1}\}, n_{chd}$ 

23 return  $P, n$ 

24 Function Eval ( $n, T$ ) :
25    $P \leftarrow \emptyset$ 
26   while  $\text{depth}(n) < T$  do
27      $a_{grd} \leftarrow \text{Select one-step greedy action on } n.$ 
28     Apply action  $a_{grd}$  to the bounding boxes  $B^t$ 
       to get  $B^{t+1}$ .
29     if  $\text{Score}(B^0, B^{t+1}) \leq \text{Score}(B^0, B^t)$ 
       then
30       break
31     Create a new child  $n_{grd}$  with  $B^{t+1}$  to  $n$ 
       while removing all untried actions of  $n$ .
32      $P, n \leftarrow P + \{B^{t+1}\}, n_{grd}$ 
33   return  $P, n$ 

34 Function Backup ( $n, r$ ) :
35   while  $n$  is not null do
36      $N(n) \leftarrow N(n) + 1$ 
37      $Q(n) \leftarrow \max(Q(n), r)$ 
38      $n \leftarrow \text{Parent of } n$ 

```

---

## A.6. More Qualitative Results

We provide more results of the comparison with the other methods below, as shown in Fig. 6. We show two lines for each shape for a better view of tightness and coverage: the upper line overlaying the input mesh and the lower line without it.

