RL-ACD: Reinforcement Learning-based Approximate Convex Decomposition

YUZHE LUO, State Key Lab of CAD&CG, Zhejiang University, China and LIGHTSPEED, China
ZHERONG PAN, LIGHTSPEED, USA
KUI WU, LIGHTSPEED, USA
XINGYI DU, LIGHTSPEED, USA
YUN ZENG, LIGHTSPEED, USA
XIANGJUN TANG, State Key Lab of CAD&CG, Zhejiang University, China
YIQIAN WU, State Key Lab of CAD&CG, Zhejiang University, China
XIAOGANG JIN*, State Key Lab of CAD&CG, Zhejiang University, China
XIFENG GAO*, LIGHTSPEED, USA

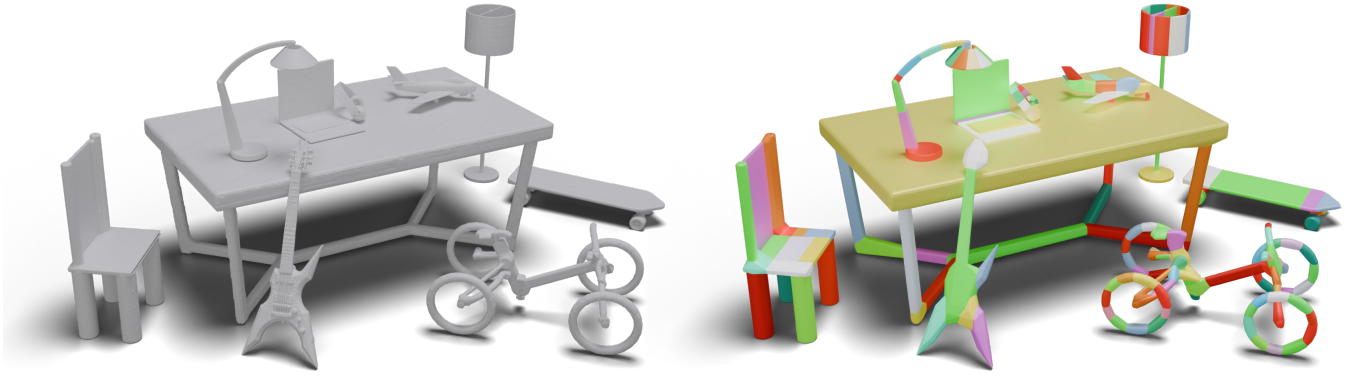

Fig. 1. In complex scenarios (10 dense meshes, 189k faces) (left), RL-ACD decomposes models via a lightweight neural policy, predicting near-optimal cutting planes to generate 216 compact convex hulls (right) at 2.6s/model while preventing sub-optimal cuts from prior solutions.

Approximate Convex Decomposition (ACD) aims to approximate complex 3D shapes with convex components, which is widely applied to create compact collision representations for real-time applications, including VR/AR, interactive games, and robotic simulations. Efficiency and optimality are critical for ACD algorithms in approximating large-scale, complex 3D shapes, enabling high-quality decompositions with minimal components. Unfortunately, existing methods either employ sub-optimal greedy strategies or rely on computationally intensive multi-step searches. In this work, we propose RL-ACD, a data-driven, reinforcement learning-based approach for efficient and near-optimal convex shape decomposition. We formulate ACD as a Markov Decision Process (MDP), where cutting planes are iteratively applied based on the current stage's mesh fragments rather than the entire fine-grained mesh, leading to a novel, efficient geometric encoding. To train near-optimal policies for ACD, we propose a novel dual-state Bellman loss and analyze its convergence using a Q-learning algorithm. Comprehensive evaluations across diverse datasets validate the efficiency and accuracy of RL-ACD for convex decomposition tasks. Our method outperforms the multi-step tree search by 15× in terms of computational speed, while reducing the number of resulting components by 16% compared to the current state-of-the-art greedy algorithms, significantly narrowing the sub-optimality gap and enhancing downstream task performance.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: Mesh Decomposition, Reinforcement Learning

ACM Reference Format:
Yuzhe Luo, Zherong Pan, Kui Wu, Xingyi Du, Yun Zeng, Xiangjun Tang, Yiqian Wu, Xiaogang Jin, and Xifeng Gao. 2025. RL-ACD: Reinforcement Learning-based Approximate Convex Decomposition. *ACM Trans. Graph.* 44, 6, Article 271 (December 2025), 12 pages. https://doi.org/10.1145/3763270

*Corresponding authors.

Authors' Contact Information: Yuzhe Luo, yzluo@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China and LIGHTSPEED, China; Zherong Pan, zherong.pan.usa@gmail.com, LIGHTSPEED, USA; Kui Wu, walker.kui.wu@gmail.com, LIGHTSPEED, USA; Xingyi Du, du.xingyi@wustl.edu, LIGHTSPEED, USA; Yun Zeng, iamzengxiang@gmail.com, LIGHTSPEED, USA; Xiangjun Tang, Xiangjun.Tang@outlook.com, State Key Lab of CAD&CG, Zhejiang University, China; Yiqian Wu, onethousand1250@gmail.com, State Key Lab of CAD&CG, Zhejiang University, China; Xiaogang Jin, jin@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Xifeng Gao, gxf.xisha@gmail.com, LIGHTSPEED, USA.

ACM 1557-7368/2025/12-ART271
https://doi.org/10.1145/3763270

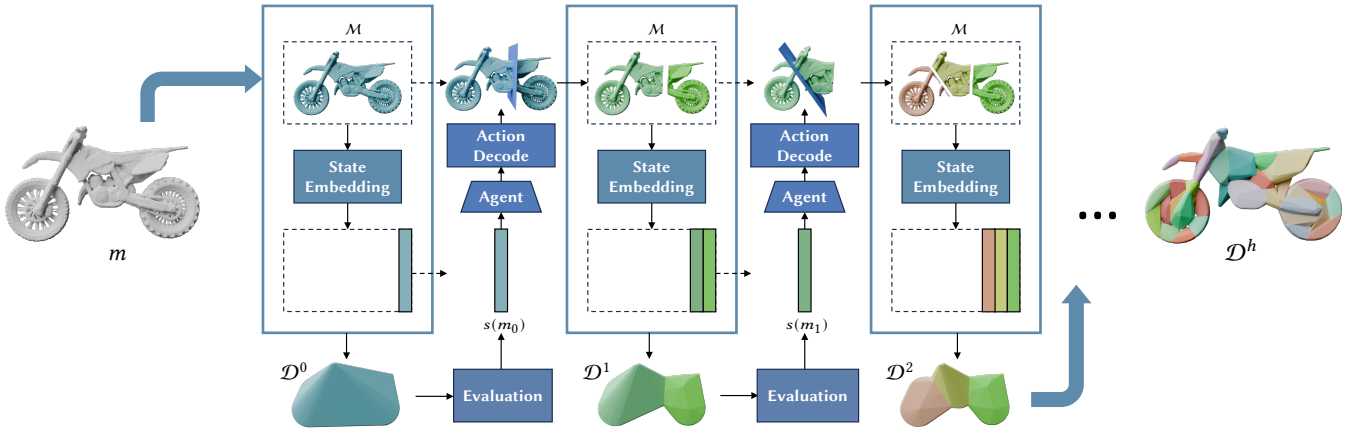

Fig. 2. The pipeline of RL-ACD: The input mesh m is processed iteratively, with each part m_i embedded into a state vector $s(m_i)$ and evaluated by the agent to determine the optimal cutting action. This process continues until all mesh components meet the predefined concavity threshold, indicating that the mesh is fully decomposed into smaller convex components, forming $\mathcal{D}^h$ with h being the number of decomposition steps.

1 Introduction

3D mesh-based geometric processing tasks like collision detection [Kockara et al. 2007] face rising complexity with mesh intricacy. In real-time applications like video games and VR/AR environments, detailed meshes are often substituted with low-cost representations to enable faster collision detection. Similarly, in robotics research, policy learning demands vast datasets generated by simulators, further emphasizing the need for efficient mesh simplifications. Approximate Convex Decomposition (ACD) algorithms are widely employed for these purposes, as they approximate detailed meshes using limited convex components. However, achieving both efficiency and near-optimality in ACD algorithms remains a significant challenge. For time-critical applications, practitioners seek near-optimal decompositions with the fewest possible components. Moreover, these algorithms must efficiently process tens of thousands of objects in large virtual environments, requiring rapid computations.

Existing ACD algorithms typically follow a common recipe consisting of two crucial components: 1) a concavity metric to measure how closely the decomposed components match their convex hulls; and 2) a decomposition strategy that recursively cuts the mesh with planes to reduce the concavity metric while using a minimal number of components. The choice of plane-selection strategy in this process significantly affects the quality of the decomposition. Unfortunately, most existing methods, such as V-HACD [Mamou et al. 2016], Thul et al. [2018], and NavACD [Andrews 2024] rely on sub-optimal greedy strategies, leading to shortsighted results with considerable redundant convex components. CoACD [Wei et al. 2022] introduces the Monte Carlo Tree Search (MCTS) to determine the cutting planes. Using a multistep search strategy, MCTS can achieve better decomposition results with fewer convex components. However, MCTS requires simulating plane cuts over multiple steps using a trial-and-error approach, leading to significant computational overhead.

To achieve both accuracy and efficiency, we introduce Reinforcement Learning-based ACD (RL-ACD). We train a lightweight neural policy network to predict the optimal cutting planes for mesh components, achieving both low policy inference cost and nearly optimal decomposition results. The primary challenge lies in the vast state space arising from the intricate mesh geometry. Indeed, identifying near-optimal cutting planes necessitates a policy attentive to fine-grained mesh details, traditionally requiring a high-resolution 3D state encoder. Training such detailed encoders can be both computation- and data-intensive. To address this challenge, we formulate the problem as a Markov Decision Process (MDP) with a novel policy representation. Our policy proposes cutting planes solely based on the mesh component being cut, rather than the entire mesh to start with. We show that our partial observation can be effectively encoded using standard point cloud features [Zhang et al. 2023]. To train our neural policies effectively, we propose a novel dual-state Bellman loss that approximates the value function of a plane-cut using the value sum of two after-cut components. Our analysis reveals that our modified Bellman loss leads to convergent Reinforcement Learning (RL) algorithm and our evaluations highlight that our decomposition policy exhibits superior performance compared to state-of-the-art baselines, achieving fewer convex components than NavACD [Andrews 2024] and CoACD [Wei et al. 2022], as well as significantly lower inference cost than CoACD. In summary, our work makes the following major contributions:

- Novel formulation of ACD as MDP with partial observation.
- Dual-state Bellman loss for efficient ACD policy learning.

2 Related Work

We discuss the optimality and efficacy of prior works on exact and approximate ACD, while also exploring related machine learning algorithms that are related to ACD.

Exact Convex Decomposition. These algorithms aim to decompose a 3D model into the fewest exact convex components, a task

proven to be NP-Hard [O'Rourke and Supowit 1983]. Heuristic techniques [Bajaj and Dey 1992; Bajaj and Pascucci 1996; Hershberger and Snoeyink 1998; Joe 1994] have been proposed to reduce complexity, but even advanced solvers frequently produce too many components, rendering them impractical for real-world applications.

Approximate Convex Decomposition. Pioneered by Lien and Amato [2004], ACD permits partial concavities in decomposed results to reduce component count. Subsequent ACD heuristics primarily differ in concavity metrics and plane-selection strategies. Concavity metrics measure decomposition quality by comparing a mesh component with its convex hull. The algorithm terminates when the concavity metric is lower than a predefined threshold. These metrics fall into three categories: (a) **Surface-based** metrics calculate geometric differences between the mesh surface and its convex hull [Lien and Amato 2004, 2007; Mamou and Ghorbel 2009]. (b) **Volume-based** metrics measure volume differences [Andrews 2024; Mamou et al. 2016; Thul et al. 2018; Wei et al. 2022]. CoACD [Wei et al. 2022] uses a hybrid collision-aware metric, while NavACD [Andrews 2024] incorporates navigation space constraints. (c) **Visibility-based** metrics utilize the ratio of mutually visible points within the shape [Ren et al. 2011]. The plane-selection strategy significantly impacts ACD effectiveness. Most techniques employ a greedy strategy that iteratively selects cutting planes to minimize concavity. CoACD applies time-consuming MCTS to approximate globally optimal plane search. V-HACD [Mamou et al. 2016] and CoACD sample equidistant cutting planes from each axis-aligned direction, while NavACD samples from concave edges and uses three axis-aligned bisection planes. However, these approaches may struggle with complex models due to limited or shortsighted search spaces.

Learning-based ACD.. Methods like BSP-Net [Chen et al. 2020] and CvxNet [Deng et al. 2020] rely on global feature encoding for 3D object representation using convex components. However, such global encoding strategies fundamentally constrain their applicability to industrial-scale complex models due to limited geometric adaptivity. In contrast, our method formulates the problem as a Markov Decision Process and trains a neural decomposition policy to learn near-optimal cutting planes for complex shapes. As our key innovation, our policy operates in a reduced observation space, focusing solely on the mesh component being cut, instead of the entire initial mesh. In this way, our policy can better attend to the key feature of the mesh that affects the quality of the cutting plane. Furthermore, we introduce a novel Bellman loss to facilitate policy training using Q-learning [Sutton and Barto 2018].

Reinforcement Learning. RL aims at solving long-horizon decision-making problems. The success of deep RL algorithms [Wang et al. 2022] has found abundant applications in computer graphics for character animation [Peng et al. 2018], animation control [Ma et al. 2018], and object packing [Zhao et al. 2023], scene optimization [Sun et al. 2024], to name just a few. The success of deep reinforcement learning (RL) is largely based on the ability of deep neural networks to represent optimal policies and value functions. However, RL has seen limited application in geometry processing, with only a few notable exceptions [Freymuth et al. 2023; Yang et al. 2023]. This is primarily due to the ultra-high and variable-dimensional state spaces inherent to geometric data, which often involve arbitrarily complex shapes, posing significant challenges to the expressive capacity of neural representations. In this work, we successfully adapt RL to address the problem of convex decomposition, demonstrating its potential in the geometric domain.

3 Methodology

This section details our approach, beginning with the problem formulation and its subsequent MDP reformulation. We then introduce our novel dual-state Bellman loss and the corresponding Q-learning algorithm used for policy optimization.

As shown in Figure 2, the input of our ACD algorithm is a solid mesh m, and the output is a set of mesh components $\mathcal{M} = \{m_1, m_2, \cdots\}$ such that $\cup_{m_i \in \mathcal{M}} = m$, and their corresponding convex hull set $\mathcal{D} = \{\mathrm{CH}(m_1), \mathrm{CH}(m_2), \cdots\}$, where $\mathrm{CH}(m_i)$ denotes the convex hull of m_i. The ACD algorithm is equipped with a concavity metric, denoted $\mathrm{Concavity}(m_i)$, which measures the difference between m_i and $\mathrm{CH}(m_i)$. The goals of ACD are two-fold: 1) minimize the number of components $|\mathcal{M}|$, to improve the computational efficiency of downstream applications, and 2) ensure that $\mathcal{D}$ approximates the original mesh m as tightly as possible by minimizing the total concavity metric $\sum_{m_i \in \mathcal{M}} \mathrm{Concavity}(m_i)$.

3.1 MDP Reformulation of ACD

We observe that the typical process of ACD, as described in prior works [Andrews 2024; Mamou et al. 2016; Wei et al. 2022], can be naturally formulated as a MDP [Puterman 1990]. These methods iteratively select a component $m_i \in \mathcal{M}$ and choose a cutting plane p to divide m_i into two sub-components, m_i^l and m_i^r. The selection of the cutting plane $p \in \mathbb{R}^4$ depends on the current state of the decomposition, i.e., $\mathcal{M}$. Thus, we define our ACD-MDP as the tuple $\langle \mathcal{S}, \mathcal{A}, r, T, \gamma \rangle$, consisting of state space $\mathcal{S}$, action space $\mathcal{A}$, reward function r, state transition function T, and discount factor γ. Our state space consists of the current partial decomposition. In other words, any $\mathcal{M}$ is considered as a state so that $\mathcal{M} \in \mathcal{S}$.

Our action space comprises two parts. First, we select a candidate component $m_i \in \mathcal{M}$ for dissection and then choose a cutting plane p. In other words, an action $\langle i, p \rangle \in \mathcal{A}$ is a tuple of the candidate component index and the cutting plane. Our state transition function employs a mesh Boolean operation to dissect m_i into m_i^l and m_i^r and update the state through the state transition function:

$$T(\mathcal{M}, \langle i, p \rangle) = \mathcal{M}' \triangleq \mathcal{M} \cup \{m_i^l, m_i^r\} - \{m_i\}. \tag{1}$$

Finally, we design our reward function, which is composed of two terms. The first term r_1 measures the reduction ratio of the convex hull volume between before and after the plane-cut compared to the origin solid mesh:

$$r_1(\mathcal{M}, \langle i, p \rangle) = \frac{\left[|\mathrm{CH}(m_i)| - |\mathrm{CH}(m_i^l)| - |\mathrm{CH}(m_i^r)|\right]}{|\mathrm{CH}(m)|}, \tag{2}$$

where $|\mathrm{CH}(\bullet)|$ is the volume of a convex hull. The second term avoids excessive cuts in parts with minimal gain. We assign a completion reward after the decomposition of $\mathcal{M}$ to encourage fewer cuts:

$$r_2(\mathcal{M}, \langle i, p\rangle) = \frac{\mathbb{I}[\mathcal{M}' \text{ is terminal}]}{1 + \log(|\mathcal{M}'| - 1)}, \tag{3}$$

where $\mathbb{I}[\mathcal{M}' \text{ is terminal}]$ is an indicator function that equals one and only if the partial decomposition state $\mathcal{M}'$ meets the required concavity threshold ϵ, i.e., $\text{Concavity}(m_i) < \epsilon$ for each $m_i \in \mathcal{M}$. If $\mathcal{M}'$ is the terminal state, r_2 encourages the policy to use fewer cuts by being inversely proportional to $|\mathcal{M}'| - 1$, which is the number of cuts to reach the state $\mathcal{M}'$. Our ultimate reward is the sum of the two terms above, i.e. $r = \lambda_1 r_1 + \lambda_2 r_2$, with λ_1, λ_2 being the weights to be fine-tuned. The goal of our work is to solve the MDP over a finite horizon of h steps, searching for a near-optimal policy $\pi(\langle i, p\rangle | \mathcal{M})$ that maximizes the cumulative reward.

3.2 Partially Observable Assumption

While deep neural policy learning techniques exhibit superior performance in high-dimensional state and action spaces, applying them to our ACD-MDP policy $\pi(\langle i, p\rangle | \mathcal{M})$ presents two key challenges. First, the state space representing 3D mesh components has a dynamic dimensionality stemming from the varying number of mesh components during the decomposition process. Second, after repeated plane cutting, the components m_i can become very small, making it difficult to recognize geometric structural features.

To address these challenges, we adopt two techniques that significantly reduce the amount of information a neural policy needs to encode. First, inspired by empirical studies such as [Zhao et al. 2023], we discretize the action space. This design choice significantly reduces policy complexity because the network only needs to evaluate a finite set of actions $\langle i, p\rangle$ through a lightweight Q-function, rather than modeling a continuous action distribution. With a discrete action space, we can then represent the optimal policy using a state-action value function $Q(\langle i, p\rangle, \mathcal{M})$, which estimates the expected cumulative reward obtained by taking action $\langle i, p\rangle$ in state $\mathcal{M}$ under our policy $\pi(\langle i, p\rangle | \mathcal{M})$. The optimal policy can be solved via:

$$\langle i, p\rangle \leftarrow \text{argmax}_{i=1,\cdots,|\mathcal{M}|, p\in\mathcal{P}} Q(\langle i, p\rangle, \mathcal{M}),$$

where $\mathcal{P}$ represents a pre-computed set of candidate cutting planes.

Second, to more efficiently represent the detailed mesh features, we adopt a novel observation space. Observation function is typically used by Partially Observable MDP (POMDP) [Kaelbling et al. 1998] to model the partial environmental information that an agent perceives. Although our problem setting allows the entire state to be observed, the complete state is challenging for a neural network to digest. Therefore, we borrow ideas from POMDP and introduce an index-dependent observation function $O(\mathcal{M}, i)$ that produces a partial observation of the state $\mathcal{M}$, under the assumption that we already know we want to cut the component m_i. In our problem, we introduce a key assumption below:

> To determine the optimal p for $m_i \in \mathcal{M}$, the policy only needs to observe the selected component m_i, readily ignoring other components $\mathcal{M} - \{m_i\}$.

Our assumption above is based on the observation that the reduction ratio of different convex hulls are used as reward signal r_1 and are thus additive. We then define the observation as $O(\mathcal{M}, i) = m_i$ and simplify the neural policy to represent the reduced state-action value function $Q(p, m_i)$, again under the assumption that we know m_i is to be cut, from which the optimal policy is solved via:

$$\langle i, p\rangle = \text{argmax}_{i=1,\cdots,|\mathcal{M}|, p\in\mathcal{P}(m_i)} Q(p, m_i), \tag{4}$$

where $\mathcal{P}(m_i)$ is a set of candidate cutting planes computed exclusively for m_i. This simplifying assumption effectively tackles the challenges posed by dynamic state space dimensions and the decreasing component size as decomposition progresses recursively. By parameterizing Q as a neural network, its input becomes a single mesh component instead of the entire set $\mathcal{M}$ of a changing size, allowing us to normalize m_i for the 3D shape encoder, denoted as $s(\bullet)$. For instance, using a 3D voxel-based encoder [Wu et al. 2016], we can re-scale m_i to occupy the entire voxel grid. Given such a policy, our ACD procedure is summarized in Figure 2, where we iteratively evaluate Equation 4 to choose the optimal candidate cutting plane and perform the plane cutting until the concavity of all the mesh components falls below a user-specified threshold ϵ.

3.3 Policy Parameterization

For our problem, the key to designing an efficient neural policy lies in the choice of an effective 3D shape encoder. We opt for point cloud-based encoding due to its robustness to variations in topology and geometric complexity. While point cloud sampling may result in some loss of geometric information, our procedure mitigates this issue by sampling and normalizing each partial mesh component during the iterative cutting process. This normalization, coupled with a powerful point cloud encoder, allows for effective feature extraction from the normalized components.

Specifically, as shown in Figure 3, for the current mesh component m_i undergoing cutting, we uniformly sample a fixed number of points on its normalized mesh surface. We then leverage pre-trained I2P-MAE [Zhang et al. 2023] point cloud encoder for feature extraction, denoted as $F(m_i)$. In addition to extracting features from the mesh itself, we also compute the point cloud sampled features of the mesh's convex hull as part of the state, denoted as $F(\text{CH}(m_i))$. As a result, we can define our state encoder as:

$$s(m_i) = (F(m_i), F(\text{CH}(m_i))). \tag{5}$$

Since for any m_i we use the same number of candidate cutting planes $|\mathcal{P}(m_i)|$, we can have the MLP output the state-action value function of all the cutting planes in a single inference, denoted as:

$$Q(p_j, m_i) \triangleq \text{MLP}_j \circ s(m_i), \tag{6}$$

where MLP_j is the jth output of the MLP. This combined with Equation 4 completes our parameterization of the policy.

3.4 Action Space Discretization

As previously discussed, the possible cutting planes for m_i could be any plane in space that intersects m_i. However, training a decision model by sampling from the entire continuous 3D plane space using a Gaussian distribution has proven ineffective [Zhao et al. 2023]. Therefore, we choose to discretize the action space by extending the previous methods' search spaces [Andrews 2024; Thul et al. 2018; Wei et al. 2022]. For each mesh component, as shown in Figure 4, our discrete candidate planes consist of the following components:

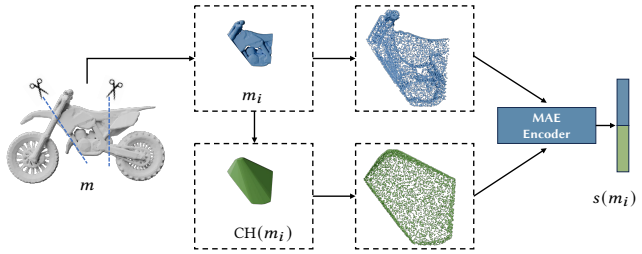

Fig. 3. Mesh state embedding. The mesh part m_i and its convex hull CH(m_i) form point cloud representations. The point clouds are normalized and processed by I2P-MAE to yield state embeddings.

(a) equidistantly sampled cutting planes along the spatial X-, Y-, and Z-directions; (b) equidistantly sampled cutting planes along the primary axes computed via PCA of each mesh part; and (c) candidate planes sampled from the concave edges of the mesh. Notably, during model training we exclusively employ strategies (a) and (b), reserving strategy (c) for supplementary candidate planes during algorithm deployment (see Section 4.2). We emphasize that all prior methods such as [Andrews 2024; Wei et al. 2022] can be modified to use our extended search space, with a much larger breadth of search. However, these methods need to perform exact plane-cutting for each candidate plane during runtime. Therefore, using our extended search space can significantly increase their computational overhead. In contrast, our method only requires performing multiple exact plane cuts during RL training. At runtime, we primarily rely on lightweight network inference for each candidate plane, supplemented by very limited cutting computations, leading to a significant runtime cost saving.

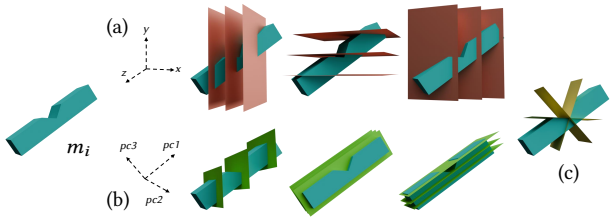

Fig. 4. Our potential planes for an input mesh component m_i include: (a) spatial axis-align planes (red); (b) PCA axis-align planes (green); and (c) planes sampled from concave edges (yellow). For each concave edge, three cutting planes are sampled: one bisecting the dihedral angle and two aligned with the edge's adjacent faces, as referenced NavACD [Andrews 2024].

3.5 Q-Learning using Dual-state Bellman Loss

We train our policy using deep Q-learning [Mnih et al. 2015]. This method is based on the Bellman loss that unrolls the Bellman optimality condition over one decision step, which involves the state-action values of a mesh component before and after plane-cutting. Under our policy parameterization, however, the value after plane-cutting involves two mesh components m_i^l and m_i^r, whose values are predicted separately via two Q-functions. We still need a method to define the combined value function of $\{m_i^l, m_i^r\}$. To this end, we propose our second assumption:

> The state value function is additively composable over mesh components, i.e. $V(\mathcal{M}) = \sum_{m_i \in \mathcal{M}} \mathcal{V}(m_i)$ and $Q(\langle i, p\rangle, \mathcal{M}) = Q(p, m_i) + \sum_{m_i \neq m_j \in \mathcal{M}} \mathcal{V}(m_j)$.

Here we denote $\mathcal{V}(m_i)$ with a single parameter as the state value function that can be readily computed by maximizing over the discrete action space:

$$\mathcal{V}(m_i) \triangleq \max_{p \in \mathcal{P}(m_i)} Q(p, m_i). \tag{7}$$

Under the composable assumption on the Q-function, we derive the following dual-state Bellman loss:

$$\mathbb{E}_{\langle \mathcal{M}, \langle i,p\rangle, \mathcal{M}', r\rangle \in \mathbb{D}} \| r + \gamma^l \mathcal{V}(m_i^l) + \gamma^r \mathcal{V}(m_i^r) - Q(p, m_i) \|^2, \tag{8}$$

where the expectation is taken over a replay buffer $\mathbb{D}$ of transition tuples. Note that our Bellman loss is only related to the chosen component m_i instead of the entire state $\mathcal{M}$, making it computationally efficient to evaluate. We use the standard deep soft Q-learning procedure [Haarnoja et al. 2017], with Equation 8 replacing the original Bellman loss. Note that we introduce two different discount factors γ^l and γ^r for the two Q-functions after plane cutting. In the standard setting, we can simply set $\gamma^l = \gamma^r = \gamma$. However, our experiments show that using different γ for the two mesh components leads to better training convergence speed. In practice, we set γ^l and γ^r to be the relative convex hull volume, i.e.:

$$\gamma^l = |\mathrm{CH}(m_i^l)|/|\mathrm{CH}(m_i)|, \gamma^r = |\mathrm{CH}(m_i^r)|/|\mathrm{CH}(m_i)|. \tag{9}$$

Note that such design ensures the inequality $\gamma^l + \gamma^r \leq 1$ holds because the two components m_i^l and m_i^r are non-overlapping after plane cutting. The above inequality ensures Q-learning convergence in the tabular setting, as discussed in our supplementary materials, and serves as a strong indicator of convergence under general neural policy representations.

4 Experiments

In this section, we detail our training and deploy steps, experimental setup, ablation study, and evaluations.

4.1 Network Training

All experiments were conducted on a workstation equipped with an AMD Ryzen 9 7950X3D 16-Core CPU, and an NVIDIA RTX 3090 GPU. For the Soft Q-learning implementation, we leveraged PyTorch with a lightweight multi-layer perceptron(MLP) policy network. The network architecture comprised four fully connected layers with dimensions {1024, 512, 256, 128}, interleaved with ReLU activation functions, mapping state representations to action-value predictions through nonlinear transforms. Training employed a 1×10^5-capacity experience replay buffer over 1×10^6 iterations. Exploration combined initial 1×10^4 random steps for network initialization, followed by ϵ-greedy search (initial $\epsilon = 0.2$, decay rate 0.99 per 1×10^3 steps). Each mesh component was sampled with 2,048 points, using reward weights $\lambda_1 = \lambda_2 = 1 \times 10^3$ for objective

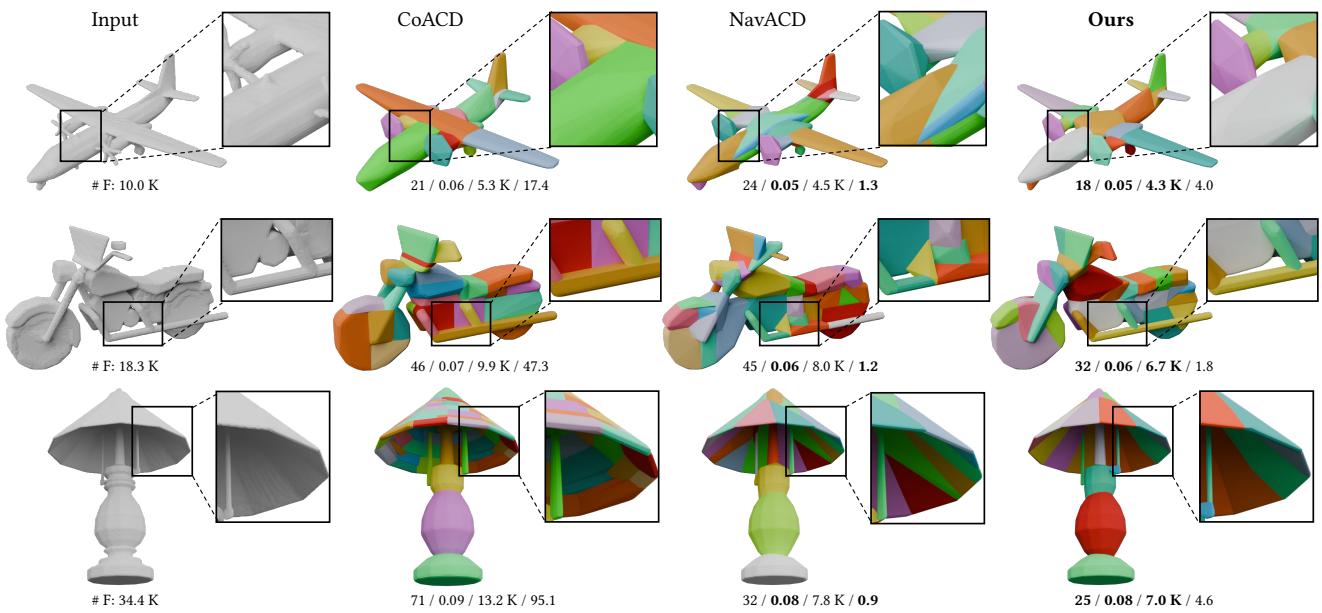

Fig. 5. Comparison with CoACD [Wei et al. 2022] and NavACD [Andrews 2024]. Numbers below each result indicate the number of convex parts, Hausdorff distance, face number, and decomposition time (s). Close-up views highlight decomposition quality.

balancing. The total training time was approximately 9 hours on the aforementioned workstation.

During training, CoACD's concavity metric enforced geometric constraints ($t_c = 4\%$) for surface-volume preservation. The action space $\mathcal{P}(m_i)$ contained 150 planes: 25 equidistant along global $x/y/z$-axes, 25 along component PCA axes, all origin-centered with 1/25 unit spacing.

4.2 Algorithm Deployment

We integrated our learning module into Unreal Engine for deployment. Following NavACD [Andrews 2024]'s framework, we adopt their navigation space paradigm as the decomposition termination criterion, which effectively ignores unreachable internal surfaces while accelerating convergence. To enhance cutting plane selection efficacy, we extend the candidate plane pool by integrating convex edge-derived planes through NavACD's extraction protocol, though we deliberately exclude concave edge sampling during training due to their representation of high-frequency geometric details that challenge our point cloud encoder's capture capacity.

Deployment followed three stages: (1) Policy network inference evaluates axis-aligned planes; (2) Top 10 network-predicted planes combine with 12 NavACD-derived convex-edge planes; (3) Physical cutting selects optimal plane by minimal convex hull volume. Post-processing merged redundant parts using NavACD's protocol (implementation details in NavACD Sections 4.1-5).

4.3 Datasets

We randomly sampled 1, 500 models from the ShapeNetCore [Chang et al. 2015] dataset, with 80% of the data designated for training. To enhance the robustness of our mesh cutting algorithm, we preprocess the data with the Watertight Manifold algorithm [Huang et al. 2018], which ensures the correct 2-manifold topology. Data augmentation is incorporated during training through random rotations applied to the input meshes. We randomly select one of the principal axes (x, y, or z) and a rotation angle from the set $(0, \pi/4, \pi/2, \pi)$. A rotation matrix is then calculated based on these selections, relative to the mesh's centroid, and applied to the mesh.

4.4 Baselines

We compare our method to V-HACD [Mamou et al. 2016], CoACD [Wei et al. 2022], and NavACD [Andrews 2024] on the test dataset. Specifically, for CoACD and NavACD, we fine-tune their concavity measure parameter thresholds to obtain decomposition results with the same average Hausdorff distance, i.e. $D_h = 0.15$ and $D_h = 0.10$ of Table 1. Our method follows NavACD's navigation space parameters since we also use navigation space as our stopping criterion. For V-HACD, since it controls decomposition by manually setting the desired number of components for each model, we compare against V-HACD under the condition of having the same number of components. Complete test data and statistics are in supplementary material.

4.4.1 Comparison with CoACD and NavACD. CoACD is configured with two precision levels for decomposition: $t_c = 12\%$ and $t_c = 4\%$. NavACD and our method use two different precision levels for the navigable space: (1) $r = 2.5\%, t = 5\%$ and (2) $r = 5.0\%, t = 1\%$. We conducted experiments on the ShapeNet test dataset with 305 models, and the average results are shown in Table 1.

CoACD, due to its MCTS algorithm, requires extensive cutting computations for each decision, resulting in significant time consumption (see the **t** (s) column of Table 1). NavACD's single-step greedy search strategy, combined with its extremely limited search

space, leads to some unnecessary cuts (see the **# P** column of Table 1). Our method, incorporating lightweight network inference and limited cutting computations, effectively optimizes the cutting plane selection process. Consequently, we achieve the desired decomposition precision with the fewest convex components (the **# P** column in Table 1). Since the number of triangles in the convex hulls plays a critical role in some applications, we also compare this metric in our experiment, where our method achieves the lowest triangle count (see the **# F** column in Table 1). More visual comparisons are shown in Figure 5 and Figure 11.

Table 1. ShapeNet decomposition performance (format: $\text{mean}^{\text{max}}_{\pm\text{std}}$). Parameters: t_c=convexity threshold(%), r=navigation space's radius(%), t=navigation spaces's tolerance distance(%). Metrics: D_h=Hausdorff distance, #P=part count, #F=face count (k=$\times 10^3$), T=computation time.

Method (Params)	$D_h \downarrow$	**#P** $\downarrow$	**#F (k)** $\downarrow$	**T (s)** $\downarrow$
CoACD (t_c12)	$0.15^{0.38}_{\pm 0.06}$	$10.7^{54}_{\pm 9.0}$	$5.5^{22.6}_{\pm 3.8}$	$21.9^{99.3}_{\pm 20.2}$
CoACD (t_c4)	$0.10^{0.31}_{\pm 0.05}$	$44.2^{278}_{\pm 45.7}$	$10.4^{47.7}_{\pm 7.2}$	$52.3^{233.5}_{\pm 43.8}$
NavACD (r2.5, t5)	$0.15^{0.45}_{\pm 0.06}$	$6.9^{34}_{\pm 5.1}$	$2.7^{9.0}_{\pm 1.6}$	$\underline{\mathbf{0.3}}^{1.6}_{\pm 0.3}$
NavACD (r5, t1)	$0.10^{0.29}_{\pm 0.05}$	$30.5^{184}_{\pm 22.3}$	$6.7^{22.8}_{\pm 3.5}$	$\underline{\mathbf{1.3}}^{3.7}_{\pm 0.6}$
Ours (r2.5, t5)	$0.15^{0.45}_{\pm 0.06}$	$\underline{\mathbf{6.0}}^{32}_{\pm 4.3}$	$\underline{\mathbf{2.6}}^{9.4}_{\pm 1.6}$	$1.3^{7.1}_{\pm 1.2}$
Ours (r5, t1)	$0.10^{0.31}_{\pm 0.05}$	$\underline{\mathbf{25.9}}^{163}_{\pm 20.8}$	$\underline{\mathbf{6.2}}^{21.7}_{\pm 3.5}$	$3.6^{12.6}_{\pm 2.3}$

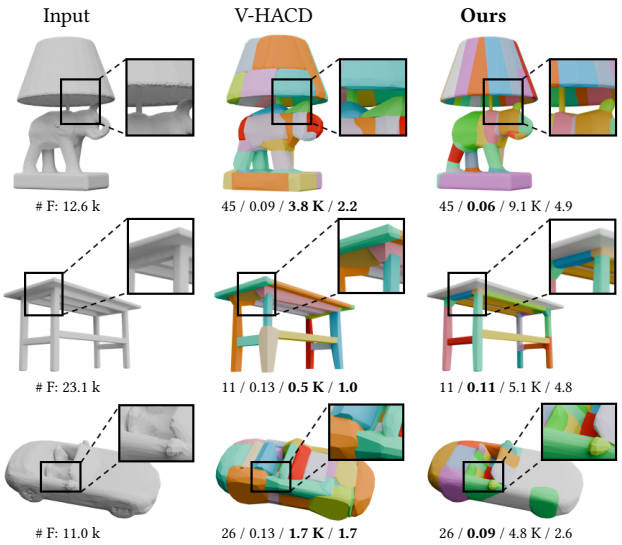

Fig. 6. Comparison with V-HACD [Mamou et al. 2016] using the same number of decomposed parts. Numbers below each result indicate the number of convex parts, Hausdorff distance, face number, and decomposition time (s). Close-up views highlight decomposition quality.

4.4.2 Comparison with V-HACD. We compared our method with V-HACD, ensuring an equal number of convex components for both. The average Hausdorff distances between V-HACD results and origin shapes at different precision levels are 0.19 and 0.14. Due to the voxel-based preprocessing in V-HACD, its fitting accuracy suffers significantly. Our method, in contrast, achieves superior geometric quality in the decomposition results, as demonstrated in Figure 6, which highlights the visual differences between the two methods.

4.4.3 Comparison with Single-Step Greedy Algorithm. To validate the benefits of our policy network arising from its long-term decision-making capabilities, we compare it against a single-step greedy algorithm. Within the high-precision navigable space setting, we replace our policy module with this greedy algorithm, which is specifically designed to select cutting planes. In the greedy approach, at each decomposition step, all candidate planes are evaluated by simulating the cut operation, and the plane that minimizes the combined convex hull volume is selected.

On the ShapeNet test set, the greedy algorithm yields an average of 28.3 convex hull components, requires 13.2 seconds for decomposition, and achieves an average Hausdorff distance of 0.10. By contrast, our policy network achieves a significantly reduced number of 25.9 components and a decomposition time of 3.6 seconds. While the Hausdorff distance accuracy of our method matches that of the greedy baseline, our approach outperforms in minimizing decomposed components and reducing computational time. This comparison highlights the benefits of our method's long-term decision-making perspective, which ultimately improves performance by reducing the reliance on computationally expensive actual cutting operations.

4.5 Ablation Study

In this section, we explore the effectiveness of different components in RL-ACD. We utilize our final solution with high precision as the baseline for ablation and report the number of decomposed parts for comparison.

4.5.1 Candidate Plane Configurations. We first evaluated the influence of three distinct candidate plane categories—spatial axis-align planes, PCA axis-align planes, and concave-edge planes—on convex decomposition efficacy. As quantified in Table 2, the synergistic integration of all three plane types yields optimal decomposition outcomes, minimizing component count and mesh face complexity while maintaining geometric fidelity.

Table 2. Quantitative results of the ablation study of candidate plane configurations. Different types of planes are beneficial to reducing the number of components.

Candidate Planes	$D_h \downarrow$	**# P** $\downarrow$	**# F (k)** $\downarrow$	**T (s)** $\downarrow$
All planes	0.10	<u>**25.9**</u>	<u>**6.2**</u>	3.6
W/O spatial axis-align planes	0.10	29.4	6.6	4.3
W/O PCA axis-align planes	0.10	27.9	6.3	4.5
W/O concave-edge planes	0.10	28.6	6.6	<u>**3.2**</u>

Number of Axis-align Planes. We systematically evaluated how selecting different quantities of high-value axis-aligned cutting planes

(from the RL model's 150 predicted candidates) impacts decomposition performance during deployment (Table 3, left). We observed that when the number of selected cutting planes is too few (e.g., 1 or 5), the policy results might be unstable due to randomness in network sampling. As the number of cutting computations increases, the computation time significantly increases. Meanwhile, the number of resulting parts initially decreases and then increases. This phenomenon arises because, with an increasing number of cuts, the algorithm progressively adopts a single-step greedy strategy, thereby reducing the effectiveness of the long-term reward-focused agent. Notably, when the number of calculated cuts reaches 150, the algorithm essentially performs as a single-step greedy method. We discovered that choosing ten planes with the highest action value for cutting yields excellent decomposition results with the least computational burden, achieving good model stability and global rewards.

Table 3. Quantitative ablation study: Impact of axis-aligned planes (#AP) and concave edges (#CE) counts. Our configuration (10 #AP + 4 #CE) balances component reduction (#P↓) and computation time (T(s)↓).

Axis-aligned Planes (#AP)			**Concave Edges (#CE)**		
#AP	**#P↓**	**T(s)↓**	**#CE**	**#P↓**	**T(s)↓**
1	29.4	2.8	0	28.6	3.2
5	27.8	3.1	2	26.4	3.5
10	25.9	3.6	4	25.9	3.6
20	25.8	4.9	8	27.1	4.4
50	26.3	7.7	16	26.5	4.9
100	26.8	12.4	32	26.3	6.0
150	28.3	13.2	64	26.1	8.4

Number of Concave Edges. Following NavACD [Andrews 2024], we further analyzed the cost-benefit trade-off of varying the number of sampled concave edges (Table 3, right). In our method, we observed that the average number of decomposed components does not consistently decrease as the number of sampled concave edges increases. This occurs because additional planes generated by sampling more concave edges weaken the influence of the axis-aligned values predicted by our policy, causing the decomposition results to increasingly align with those of a greedy algorithm.

4.5.2 State Encoding. For the design of the state space, we evaluated the number of components required after removing the encoded features of convex hull sampling. The number of components required was 26.6 (vs 25.9), indicating the importance of the convex hull feature to achieve efficient decomposition.

4.5.3 Reward Function. We evaluated the impact of various reward weights on the experimental outcome. When the reward term r_1 was omitted, the number of components required was 27.3, and without the reward term r_2, it was 27.1. Regarding the reward weights, the component counts were 27.0, 26.9, 25.9, 26.7, and 26.8 (vs 25.9) for $\lambda_1/\lambda_2 = 0.01$, 0.1, 1, 10, and 100, respectively. These results underscore the importance of each reward term and the need to appropriately weight them to optimize the decomposition process and minimize the number of components.

4.5.4 Discount factors. As analyzed in Section 3.5, discount factors γ^l, γ^r govern training convergence. Figure 7 shows training loss divergence when $\gamma^l + \gamma^r > 1$. Our handcrafted heuristic setup achieves faster convergence than baselines via balanced reward distribution, with detailed theoretical proofs in supplementary material.

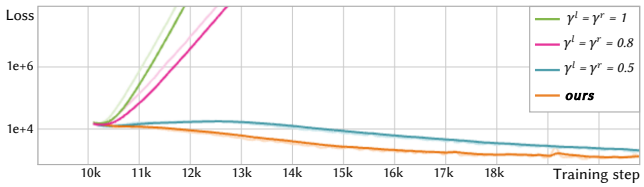

Fig. 7. Analyzes training convergence dynamics under varying discount factors (γ^l, γ^r). Results demonstrate that $\gamma^l + \gamma^r \leq 1$ (empirically set to relative convex hull volumes) ensures stable Q-learning convergence, outperforming static discounting schemes.

4.6 Generalization

Our model was initially trained on the ShapeNet, which primarily consists of man-made models (e.g., chairs, tables, and lamps). To rigorously evaluate its cross-domain generalization capability, we conducted zero-shot transfer tests on two unseen datasets: HumanBody [Maron et al. 2017] and COSEG [Wang et al. 2012]. For each target dataset, we randomly selected 20 models and applied our pre-trained model directly without fine-tuning or retraining. As shown in Figure 8 and Table 4, our model still achieved competitive performance relative to previous methods. We attribute this generalization capability to the partially observable assumption underlying our point cloud encoding strategy: by only requiring encoding of a small mesh component, the network exhibits low sensitivity to the global geometric structure of meshes, thereby enabling strong cross-dataset performance.

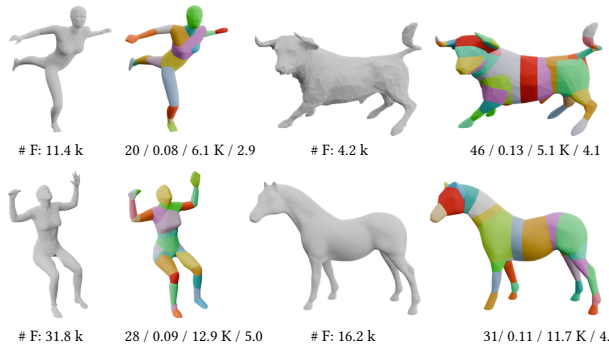

Fig. 8. RL-ACD decomposition results on the HumanBody (left) and COSEG (right) datasets. Our method demonstrates good generalization capability across different categories of previously unseen data.

Table 4. Quantitative results on the HumanBody and COSEG datasets, validating RL-ACD's generalization capability to unseen geometries.

Dataset	Method	$D_h \downarrow$	# P ↓	# F (k) ↓	T (s) ↓
HumanBody	CoACD ($t_c 4$)	0.22	23.4	8.2	22.5
	NavACD ($r5, t1$)	0.18	22.2	8.3	**1.0**
	Ours ($r5, t1$)	**0.17**	**20.2**	**8.1**	3.7
COSEG	CoACD ($t_c 4$)	0.15	33.1	7.7	23.8
	NavACD ($r5, t1$)	**0.13**	32.8	9.0	**1.1**
	Ours ($r5, t1$)	**0.13**	**29.6**	8.6	4.3

We conducted extended evaluations on high-genus models with complex topological structures to assess decomposition robustness. As demonstrated in Figure 9, while all comparative methods preserve original genus characteristics under optimized parameter configurations, our approach achieves superior component efficiency.

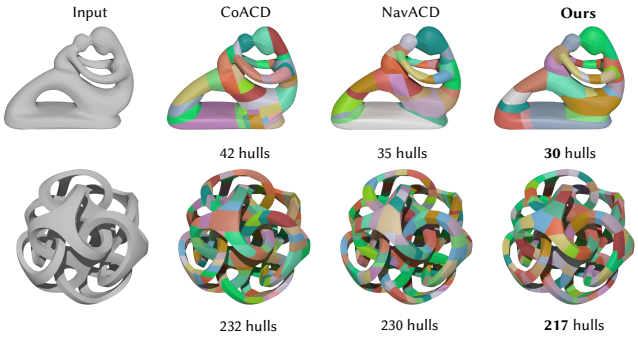

Fig. 9. Evaluates decomposition results on high-genus models with complex topologies. RL-ACD reduces component counts by 10% compared to baseline methods, demonstrating superior handling of intricate geometric features.

5 Conclusion and Discussion

We presented RL-ACD, a reinforcement-learning-based approach to approximate convex decomposition. This marks the first application of data-driven reinforcement learning techniques to tackle the well-established ACD problem. Under our innovative assumptions, we facilitate the use of a lightweight neural policy to approximate near-optimal multi-step decision rewards. This enables RL-ACD to surpass existing methodologies in terms of convex decompositions while maintaining interactive performance suitable for real-time applications. As a result, our approach yields more efficient ACD results for downstream applications, significantly enhancing the 3D asset creation workflow.

Notwithstanding its advantages, RL-ACD exhibits limitations. The reliance on predefined candidate planes and a static feature encoder may constrain its ability to identify optimal cuts for complex geometries, and suboptimal decomposition occurs when processing smooth residual meshes lacking discriminative geometric features, as shown in Figure 10. These failures stem primarily from the feature extractor's inability to encode homogeneous surfaces effectively.

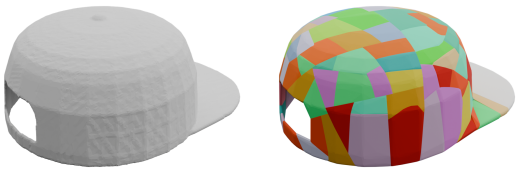

Fig. 10. Illustrates a failure mode in smooth, featureless geometries. Limited surface discriminability hinders the policy's ability to identify optimal cuts, highlighting opportunities for enhanced feature encoding in future work.

The identified limitations highlight promising avenues for future research, including exploring advanced state encoding techniques to capture complex geometries better, incorporating real-time feedback mechanisms to dynamically adapt the cutting strategy, and developing methods for dynamically generating candidate planes focused on areas of high concavity to expand the search space and potentially identify more optimal cuts. Finally, optimizing the reinforcement learning training process could improve efficiency and reduce computational requirements, ultimately enhancing the robustness and adaptability of RL-ACD for a wider range of complex shapes and scenarios.

Acknowledgments

Xiaogang Jin was supported by the National Natural Science Foundation of China (Grant Nos. 62036010, 62472373).

References

James Andrews. 2024. Navigation-driven approximate convex decomposition. In *ACM SIGGRAPH Conference Papers*.

Chanderjit L Bajaj and Tamal K Dey. 1992. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.* 21, 2 (1992), 339–364.

Chandrajit L Bajaj and Valerio Pascucci. 1996. Splitting a complex of convex polytopes in any dimension. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*. 88–97.

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

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. BSP-Net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 45–54.

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. 2020. CvxNet: Learnable convex decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 31–44.

Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, and Gerhard Neumann. 2023. Swarm reinforcement learning for adaptive mesh refinement. *Advances in Neural Information Processing Systems* 36 (2023), 73312–73347.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *Proceedings of the IEEE International Conference on Machine Learning (ICML)*. 1352–1361.

John E Hershberger and Jack S Snoeyink. 1998. Erased arrangements of lines and convex decompositions of polyhedra. *Computational Geometry* 9, 3 (1998), 129–143.

Jingwei Huang, Hao Su, and Leonidas Guibas. 2018. Robust watertight manifold surface generation method for ShapeNet models. *arXiv preprint arXiv:1802.01698* (2018).

Barry Joe. 1994. Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *Internat. J. Numer. Methods Engrg.* 37, 4 (1994), 693–713.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1 (1998), 99–134.

Sinan Kockara, Tansel Halic, Kamran Iqbal, Coskun Bayrak, and Richard Rowe. 2007. Collision detection: A survey. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 4046–4051.

Jyh-Ming Lien and Nancy M Amato. 2004. Approximate convex decomposition. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. 457–458.

Jyh-Ming Lien and Nancy M Amato. 2007. Approximate convex decomposition of polyhedra. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*. 121–131.

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. 2018. Fluid Directed Rigid Body Control using Deep Reinforcement Learning. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.

Khaled Mamou and Faouzi Ghorbel. 2009. A simple and efficient approach for 3D mesh approximate convex decomposition. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 3501–3504.

Khaled Mamou, E Lengyel, and A Peters. 2016. Volumetric hierarchical approximate convex decomposition. *Game Engine Gems* 3 (2016), 141–158.

Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 71.

Francisco S Melo. 2001. Convergence of Q-learning: A simple proof. *Institute of Systems and Robotics, Tech. Rep* (2001), 1–4.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

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

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)* 37, 4 (2018), 1–14.

Martin L Puterman. 1990. Markov decision processes. *Handbooks in Operations Research and Management Science* 2 (1990), 331–434.

Zhou Ren, Junsong Yuan, Chunyuan Li, and Wenyu Liu. 2011. Minimum near-convex decomposition for robust shape representation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 303–310.

Jia-Mu Sun, Jie Yang, Kaichun Mo, Yu-Kun Lai, Leonidas Guibas, and Lin Gao. 2024. Haisor: Human-aware Indoor Scene Optimization via Deep Reinforcement Learning. *ACM Trans. Graph.* 43, 2, Article 15 (Jan. 2024), 17 pages. doi:10.1145/3632947

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Daniel Thul, Lubor Ladicky, Sohyeon Jeong, and Marc Pollefeys. 2018. Approximate convex decomposition and transfer for animated meshes. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 226.

Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. 2022. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems* 35, 4 (2022), 5064–5078.

Yunhai Wang, Shmulik Asafi, Oliver Van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2012. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 165.

Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. 2022. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 42.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in Neural Information Processing Systems (NIPS)* 29 (2016), 82–90.

Zeshi Yang, Zherong Pan, Manyi Li, Kui Wu, and Xifeng Gao. 2023. Learning based 2D irregular shape packing. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–16.

Renrui Zhang, Liuhui Wang, Yu Qiao, Peng Gao, and Hongsheng Li. 2023. Learning 3d representations from 2d pre-trained models via image-to-point masked autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 21769–21780.

Hang Zhao, Zherong Pan, Yang Yu, and Kai Xu. 2023. Learning physically realizable skills for online packing of general 3D shapes. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 165.

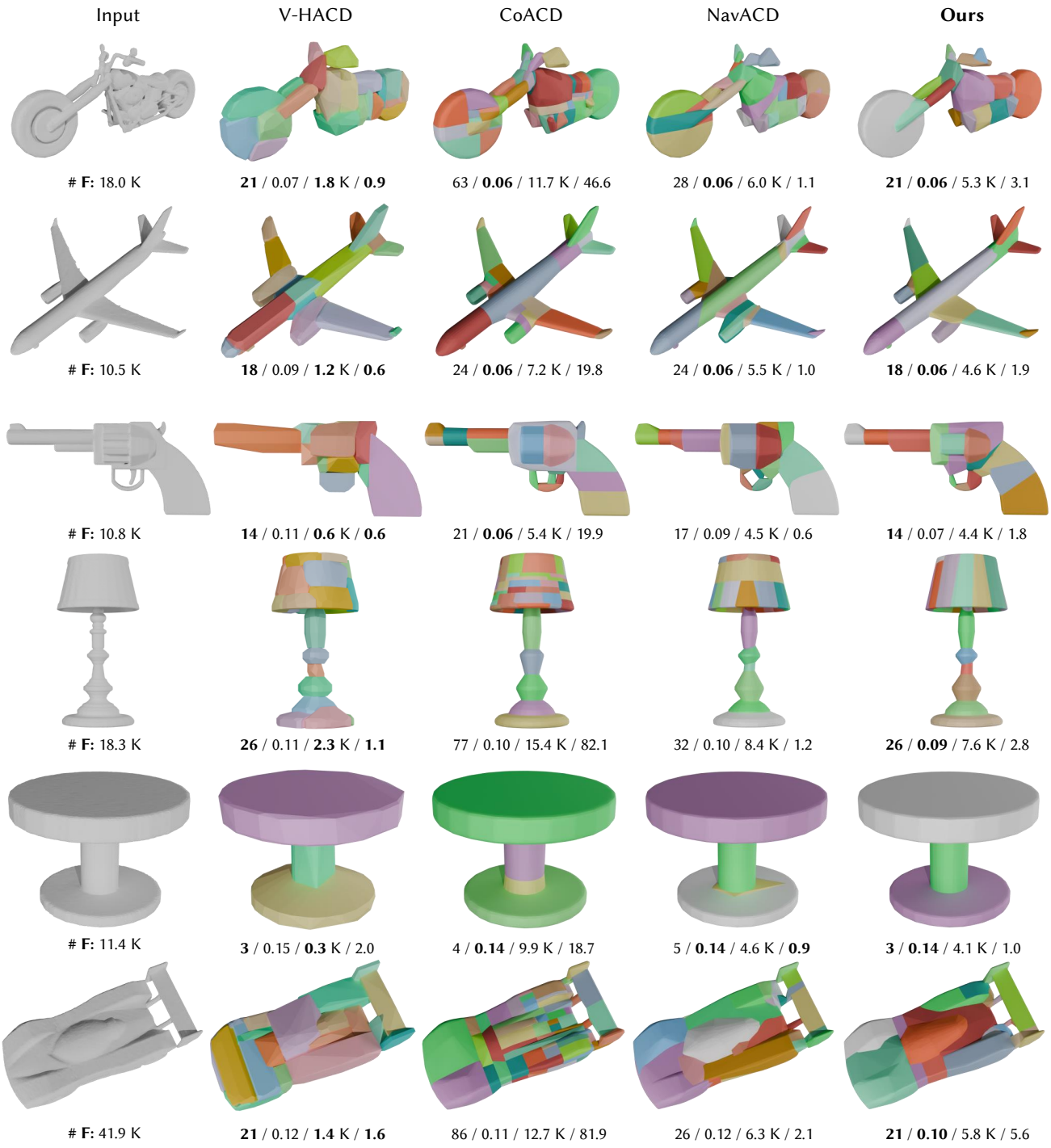

Fig. 11. Comparison with V-HACD [Mamou et al. 2016], CoACD [Wei et al. 2022] and NavACD [Andrews 2024]. Numbers below each result indicate the number of convex parts, Hausdorff distance, face number, and decomposition time (s). ©2025 ShapeNET.

Appendix

A Convergence Analysis

We analyze the convergence of our proposed dual-state Q-learning algorithm and show that our parameter choice for the discount factor is key to convergence. We emphasize that our analysis should not be considered as a convergence proof. Indeed, the convergence of deep Q-learning is very challenging due to the approximation error of neural network. Instead, we only assume a tabular setting, where we store and learn Q-values over all state-action pairs and a fixed horizon, but we believe that such analysis provides useful indications for choosing the two key parameters for the discount factor. Specifically, we take the following assumption:

Assumption 1. *Our training setup assumes:*

- *The dataset contains a finite number of N meshes $m^{1,\cdots,N}$.*
- *The feature of each mesh component $s(m_i^j)$ is precomputed.*
- *The horizon is at most H.*

Assumption 1 exactly matches our experimental setup. We have the following immediate result due to this setup:

Lemma A.1. *Under Assumption 1, the number of possible states $\mathcal{M}$ and encoded features $s(m_i)$ is finite.*

Proof. We can consider $s(\bullet)$ as a deterministic function due to our assumption that $s(\bullet)$ is precomputed, so we only need to show that the number of possible states $\mathcal{M}$ and components m_i^j is finite. For a possible state $s(m_i^j)$ experienced during Q-learning, the superscript j is finite due to the assumed finiteness of dataset. The subscript i is also finite by induction over H possible cut operations. Base Step: The starting state is $\mathcal{M} = \{m^j\}$, which is finite. Inductive Step: Assuming the number of possible states $\mathcal{M}$ is finite, then a state $\mathcal{M}' = T(\mathcal{M}, \langle i, p\rangle)$ can only be derived by cutting some $m_i \in \mathcal{M}$ using one of 3 types of cutting planes: 1) axis-aligned cutting planes; 2) primary axes aligned cutting planes; 3) planes aligned with convex edges. The number of each type of these cutting planes is finite, so the number of possible states $\mathcal{M}'$ is finite. □

Lemma A.1, allows us to consider a Q-learning algorithm in a tabular setting, where we store the finite number of action values $Q(p, m_i)$ in a table (superscript j omitted for brevity, because the learning problem for each mesh is separate during training in a tabular setting). We have the following result:

Proposition A.2. *Under Assumption 1 and using a tabular policy, the Q-learning algorithm minimizing the expected dual-state Bellman loss converges if $\gamma^l_{m_i,p} + \gamma^r_{m_i,p} \le 1$.*

Proof. We can adopt the tabular policy due to Lemma A.1, so the problem setup is well-defined. We adopt the argument in [Melo 2001], where the goal is to prove that Q-learning converges to a fixed point. In our case, given the following sampled data tuple with non-vanishing probability:

$$\langle \mathcal{M}, \langle i, p\rangle, \mathcal{M}', r\rangle,$$

at the tth update, our Q-learning update formula can be written as:

$$\begin{aligned} Q_{t+1}(p, m_i) = Q_t(p, m_i) + \alpha_t(p, m_i) \\ \left[r + \gamma^l \max_{p^l \in \mathcal{P}(m_i^l)} Q_t(p^l, m_i^l) + \right. \\ \left. \gamma^r \max_{p^r \in \mathcal{P}(m_i^r)} Q_t(p^r, m_i^r) - Q_t(p, m_i) \right]. \end{aligned}$$

We can define our contraction operator $\mathcal{H}$ as:

$$\begin{aligned} \mathcal{H}Q_t(p, m_i) = r + \\ \gamma^l \max_{p^l \in \mathcal{P}(m_i^l)} Q_t(p^l, m_i^l) + \gamma^r \max_{p^r \in \mathcal{P}(m_i^r)} Q_t(p^r, m_i^r). \end{aligned}$$

This operator is a contraction in the sup-norm, i.e.,

$$\|\mathcal{H}Q_t - \mathcal{H}Q'_t\|_\infty \le \|Q_t - Q'_t\|_\infty.$$

To prove this, we write:

$$\begin{aligned} &\|\mathcal{H}Q_t - \mathcal{H}Q'_t\|_\infty = \max_{m_i,p} \\ &\left| \gamma^l_{m_i,p} \max_{p^l} Q_t(p^l, m_i^l) + \gamma^r_{m_i,p} \max_{p^r} Q_t(p^r, m_i^r) - \right. \\ &\left. \gamma^l_{m_i,p} \max_{p^l} Q'_t(p^l, m_i^l) - \gamma^r_{m_i,p} \max_{p^r} Q'_t(p^r, m_i^r) \right| \\ \le &\max_{m_i,p} \left[\gamma^l_{m_i,p} \max_{p^l} \left| Q_t(p^l, m_i^l) - Q'_t(p^l, m_i^l) \right| + \right. \\ &\left. \gamma^r_{m_i,p} \max_{p^r} \left| Q_t(p^r, m_i^r) - Q'_t(p^r, m_i^r) \right| \right] \\ \le &\max_{m_i,p} \left[(\gamma^l_{m_i,p} + \gamma^r_{m_i,p}) \max_{m_i,p} \left| Q_t(p, m_i) - Q'_t(p, m_i) \right| \right] \\ \le &\max_{m_i,p} \left[(\gamma^l_{m_i,p} + \gamma^r_{m_i,p}) \right] \max_{m_i,p} \left| Q_t(p, m_i) - Q'_t(p, m_i) \right| \\ = &\max_{m_i,p} (\gamma^l_{m_i,p} + \gamma^r_{m_i,p}) \|Q_t - Q'_t\|_\infty \le \|Q_t - Q'_t\|_\infty. \end{aligned}$$

Here we introduce subscript for $\gamma^{l,r}_{m_i,p}$ to indicate that their choices are dependent on our state and action, but we always have $\gamma^l_{m_i,p} + \gamma^r_{m_i,p} \le 1$ by our construction. We also omit $p \in \mathcal{P}(m_i)$ and $p^{l,r} \in \mathcal{P}(m_i^{l,r})$ for brevity. By the Banach fixed point theorem, there exists a unique fixed point $Q^\star$ with $\mathcal{H}Q^\star = Q^\star$. The remaining argument towards convergence follows from Theorem 2 of [Melo 2001]. □

Note that this is not the optimal Q-table by our assumption, which suffices since we only aim to prove convergence.