arXiv:2205.02961v1 [cs.GR] 5 May 2022

# Approximate Convex Decomposition for 3D Meshes with Collision-Aware Concavity and Tree Search

XINYUE WEI* and MINGHUA LIU*, University of California San Diego, USA
ZHAN LING, University of California San Diego, USA
HAO SU, University of California San Diego, USA

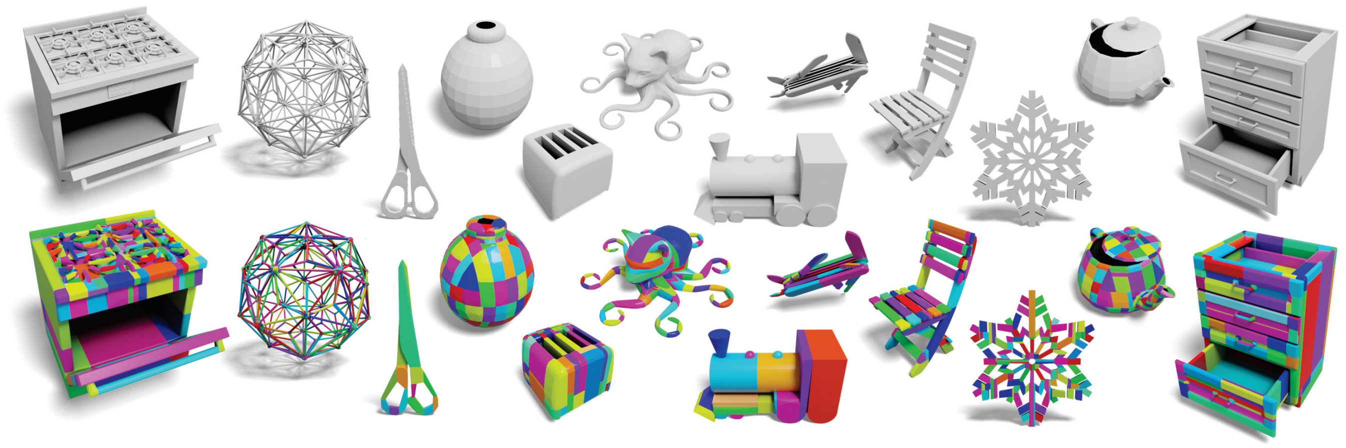


Fig. 1. We decompose a solid mesh into a set of components and utilize the convex hulls of the components (shown in different colors) to represent the original shape. Compared to prior works, we can better capture the fine-grained structures of the input shape with fewer components. See handles of the oven and the cabinet, slots of the toaster, and the spout of the kettle (zoom in for details). The high-quality decomposition enables delicate object interaction in downstream applications (e.g., a robot opens the drawer by grabbing the handles).

Approximate convex decomposition aims to decompose a 3D shape into a set of almost convex components, whose convex hulls can then be used to represent the input shape. It thus enables efficient geometry processing algorithms specifically designed for convex shapes and has been widely used in game engines, physics simulations, and animation. While prior works can capture the global structure of input shapes, they may fail to preserve fine-grained details (e.g., filling a toaster's slots), which are critical for retaining the functionality of objects in interactive environments. In this paper, we propose a novel method that addresses the limitations of existing approaches from three perspectives: (a) We introduce a novel collision-aware concavity metric that examines the distance between a shape and its convex hull from both the boundary and the interior. The proposed concavity preserves collision conditions and is more robust to detect various approximation errors. (b) We decompose shapes by directly cutting meshes with 3D planes. It ensures generated convex hulls are intersection-free and avoids voxelization errors. (c) Instead of using a one-step greedy strategy, we propose employing a multi-step tree search to determine the cutting planes, which leads to a globally better solution and avoids unnecessary cuttings. Through extensive evaluation on a large-scale articulated object dataset, we show that our method generates decompositions closer to the original shape with fewer components. It thus supports delicate and efficient object interaction in downstream applications.

*Both authors contributed equally to this research.

Authors' addresses: Xinyue Wei, xiwei@ucsd.edu; Minghua Liu, minghua@ucsd.edu, University of California San Diego, USA; Zhan Ling, University of California San Diego, USA, z6ling@eng.ucsd.edu; Hao Su, University of California San Diego, USA, haosu@eng.ucsd.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2022 Copyright held by the owner/author(s).
0730-0301/2022/7-ART42
https://doi.org/10.1145/3528223.3530103

CCS Concepts: • **Computing methodologies** → **Mesh geometry models**; **Mesh models**; *Shape analysis.*

Additional Key Words and Phrases: convex decomposition, shape decomposition, concavity, shape similarity, tree search, geometry processing

**ACM Reference Format:**
Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. 2022. Approximate Convex Decomposition for 3D Meshes with Collision-Aware Concavity and Tree Search. *ACM Trans. Graph.* 41, 4, Article 42 (July 2022), 18 pages. https://doi.org/10.1145/3528223.3530103

## 1 INTRODUCTION

With the development of 3D depth sensors, VR/AR, and physics simulation, large-scale detailed 3D models have become more common. In addition to employing data structures such as octrees and bounding volume hierarchies (BVH) to accelerate specific geometry processing algorithms, another common strategy for handling complex 3D models is to decompose them into simpler components. In particular, convex decomposition has aroused great interest. Many fundamental geometry problems in rendering and physics simulation are non-trivial and computationally expensive to solve for general shapes. However, if input shapes are convex polyhedra, many of them can be formulated as convex optimization problems, and efficient algorithms can be specifically designed. Examples include determining whether a 3D point lies inside or outside of a mesh [Snoeyink 2017], checking whether two meshes intersect [Bergen 1999; Liu et al. 2008; Mirtich 1998], and calculating the minimum distance between two meshes [Gilbert et al. 1988].

Decomposing a 3D solid shape into a minimum number of exact convex components is the *exact convex decomposition* (ECD) problem, which has proven to be NP-hard [Chazelle et al. 1997; O'Rourke and Supowit 1983]. Although many suboptimal heuristics [Chazelle 1981] have been proposed, they usually output a large number of small components, preventing them from practical applications. Instead, the *approximate convex decomposition* (ACD) problem [Lien and Amato 2007] proposes to lift the strict convexity constraint and only requires the decomposed components to be approximately convex. Since ACD approaches [Lien and Amato 2007; Mamou and Ghorbel 2009; Mamou et al. 2016; Thul et al. 2018] typically generate a much smaller number of components, whose convex hulls can then be used to approximate the original shape and speed up downstream applications, ACD works have recently received more attention. For example, V-HACD [Mamou et al. 2016] is currently one of the most popular open-source ACD methods and has been adopted by a wide range of game engines and physics simulation SDKs.

Existing ACD methods share a similar overall pipeline. In order to quantify the decomposition quality, they first define a concavity metric to measure the similarity between a decomposed component and its convex hull. They then design a heuristic cost function to decompose the 3D meshes greedily. There are three major shortcomings of existing ACD methods: **(a) Concavity metric:** Prior works mainly utilize two types of metrics: boundary-distance-based concavity [Ghosh et al. 2013; Lien and Amato 2004, 2007, 2008; Liu et al. 2016; Mamou and Ghorbel 2009], which measures the distance between the boundary surfaces of the shape and its convex hull; and volume-based concavity [Attene et al. 2008; Mamou et al. 2016; Thul et al. 2018], which calculates the volume difference between the solid shape and its convex hull. However, both metrics may fail to preserve the collision conditions in some cases, which means some positions in the space are unlikely to collide shape, but collide with the decomposition results. The insensitivity of existing concavity metrics to changes in collision conditions can be fatal for preserving object functionality. For example, they might cause an algorithm to stuff the slots of a toaster. **(b) Component representation:** There are two common strategies for representing components and decomposing shapes. The first one is to decompose shapes by grouping the triangle faces [Lien and Amato 2007; Liu et al. 2016; Mamou and Ghorbel 2009], which results in zig-zag boundaries of the components and intersecting convex hulls. In contrast, V-HACD [Mamou et al. 2016] first voxelizes the input mesh and then decomposes the voxels. However, the voxelization introduces discretization artifacts, which even makes the algorithm unable to recognize already convex shapes. **(c) One-step greedy search:** Most previous works [Mamou et al. 2016; Thul et al. 2018] decompose the shapes by recursively performing locally optimal actions. They often take short-sighted actions and end up generating more components. Furthermore, considering only one step may lead to various corner cases, which requires different heuristic terms [Mamou et al. 2016] as workarounds.

In this paper, we introduce a novel approximate convex decomposition method for 3D meshes, which effectively addresses the limitations of existing approaches from three corresponding perspectives: (a) We propose a novel collision-aware concavity metric that examines the component from both the boundary surface and shape interior by sampling points and calculating Hausdorff distance. The proposed concavity encourages preserving the collision conditions by penalizing the inclusion of regions that are far away from the original shape. We also propose an efficient way to calculate the concavity to speed up the decomposition. (b) We decompose shapes by directly cutting 3D solid meshes with 3D planes, which results in flat boundaries between components. It ensures intersection-free convex hulls and avoids the defects caused by voxelization. We also provide a lightweight mesh cutting implementation, which is 100x faster than off-the-shelf libraries. (c) We propose utilizing the Monte Carlo tree search to determine cutting planes, which simulates and searches multiple future actions before each cutting. Compared to the one-step greedy search, we are more likely to find cutting planes that lead to a better global solution and avoid unnecessary cuttings. In addition, by considering multiple steps, we no longer need various heuristic terms to prevent corner cases.

We evaluate our method on the V-HACD benchmark [Mamou et al. 2016] and PartNet Mobility [Xiang et al. 2020], a large-scale articulated object dataset. We show that our method better preserves the collision conditions and accurately approximates the fine-grained structures (e.g., drawer handles, kettle spouts, inner rings of scissors, and toaster slots) with fewer convex components. Our decomposition results thus enable delicate and fast object interaction in downstream applications. See Figure 1 for some examples. Our code is available at https://github.com/SarahWeiii/CoACD.

## 2 RELATED WORK

### 2.1 Application of Convex Shapes

Many efficient geometric algorithms require convex shapes as input. For example, collision detection between shapes is the cornerstone in physics simulation, virtual reality, game engines, and animations. Fast and precise collision detection algorithms have been specially designed for convex shapes [Bergen 1999; Gilbert et al. 1988; Liu et al. 2008; Mirtich 1998; Weller 2013]. Point location, which checks if a point is within a shape, is an important task in rendering and simulation. It can also be accelerated if input shapes are convex [Snoeyink 2017]. Moreover, by abstracting a 3D shape with a set of convex components, many downstream applications are developed, such as skeleton extraction [Lien et al. 2006], tetrahedral mesh generation [Joe 1994], mesh deformation [Jacobson et al. 2011; Liu et al. 2021; Wang et al. 2015; Wicke et al. 2007; Xian et al. 2012], and real-time animation [Müller et al. 2013].

### 2.2 Convex Decomposition

The problem of *exact convex decomposition* (ECD) was proved to be NP-hard [Chazelle et al. 1997; O'Rourke and Supowit 1983], and many suboptimal heuristic algorithms have been proposed [Bajaj and Dey 1992; Bajaj and Pascucci 1996; Chazelle 1984, 1981;

Hershberger and Snoeyink 1998; Joe 1994]. However, ECD works typically outputs a substantial amount of components and slows down practical applications. People thus turned to *approximate convex decomposition* (ACD) [Lien and Amato 2004, 2007], which lifts the strict convexity constraint and only requires the decomposed components to be almost convex.

ACD works first define a concavity metric, which measures the difference between a shape and its convex hull. They then iteratively decompose a 3D shape through top-down partition or bottom-up clustering, until the concavity of each decomposed component is within a pre-defined threshold. There are mainly three families of concavity metrics:

**Boundary-distance-based:** Lien and Amato [2004, 2007, 2008] propose to measure the distance between the shape boundary and its convex hull by mimicking the process of inflating a balloon. FACD [Ghosh et al. 2013] further extends this idea by introducing a relative concavity to enhance the details of local structures. CoRise[Liu et al. 2016] utilizes the shortest geodesic paths on the mesh surface and calculates the distance between the points on the path and edges of the convex hull. HACD [Mamou and Ghorbel 2009] projects the mesh vertices to the convex hull surface along normal directions and then measures the distance between the vertices and their projection. Most boundary-distance-based concavities involve intricate geometric processing and are inefficient to calculate for 3D shapes. Moreover, only requiring a small distance between the shape boundary and its convex hull is not enough to guarantee plausible decomposition.

**Volume-based:** There are also many works utilizing the volume difference between the shape and its convex hull as the concavity metric. Attene et al. [2008] tetrahedralizes the input mesh and hierarchically cluster the tets using the volume-based concavity. V-HACD [Mamou et al. 2016] first voxelizes the input mesh and then greedily decomposes the voxels with axis-aligned cutting planes and volume-based concavity. Due to its open-source code and good performance in general cases, V-HACD is currently one of the most widely used convex decomposition algorithms. Thul et al. [2018] also utilizes the volume-based metric and extends the task from a single static mesh to temporal coherent animated meshes. While computationally efficient, many unreasonable decompositions may not be penalized using volume differences alone. For example, it's easy for V-HACD to stuff the slots of a toaster since the relative volume difference may be small.

**Visibility-based:** Liu et al. [2010]; Ren et al. [2011] count the pairs of surface points that are mutually visible within the inner volume of the shape, and utilize the ratio of visible pairs as the concavity metric. However, it may be biased by the positions of the concave parts and inefficient to calculate for complex shapes.

Recently, many learning-based methods [Chen et al. 2020; Deng et al. 2020] also attempted to represent 3D meshes with assemblies of convex polyhedra. However, they generate convex components based on a global embedding feature of the shape and may thus fail to preserve many input structures. Their poor generalizability on novel shapes also prevents them from extensive practical use.

**Algorithm 1:** Approximate Convex Decomposition

```
Input: A 2-manifold solid mesh $\mathcal{S}$, a concavity threshold $\epsilon$
Output: Approximate convex decomposition $\mathcal{D}$
1  $Q \leftarrow \{\mathcal{S}\}$                                  // processing queue
2  $\mathcal{D} \leftarrow \emptyset$                            // decomposition results
3  while $Q$ is not empty do
4      $C \leftarrow \underbrace{Q.dequeue()}$
5      if Concavity($C$) $< \epsilon$ then                       // Section 4
6          $\mathcal{D}.enqueue(C)$
7      else
8          $\mathcal{P} \leftarrow$ MCTS($C$)      // search for cutting plane, Section 6
9          $\mathcal{P} \leftarrow$ Refine($C, \mathcal{P}$)                   // Section 6.5
10         $C_L, C_R \leftarrow$ Cut($C, \mathcal{P}$)                        // Section 5
11         $Q.enqueue(C_L, C_R)$
12 $\mathcal{D} \leftarrow$ Merge($\mathcal{D}, \epsilon$)                    // Section 6.5
```

## 2.3 Shape Abstraction

Another related direction is shape abstraction. Unlike convex decomposition, which accurately approximates original shapes, shape abstraction extracts the global structure and pays less attention to low-level details. Existing approaches utilize conventional optimization or deep learning to abstract a 3D shape into a set of primitives, such as cuboids [Calderon and Boubekeur 2017; Gadelha et al. 2020; Mo et al. 2019; Smirnov et al. 2020; Sun et al. 2019; Tulsiani et al. 2017; Zou et al. 2017], superquadrics [Paschalidou et al. 2020, 2019], sphere-trees [Bradshaw and O'Sullivan 2004], sphere-meshes [Gadelha et al. 2020; Thiery et al. 2013], generalized cylinders [Zhou et al. 2015], and their combination [Li et al. 2019]. Although most geometric primitives are convex, shape abstraction is hard to capture fine-grained structures due to the low-dimensional expressibility of the primitives.

## 3 PROBLEM DEFINITION AND METHOD OVERVIEW

We aim to decompose a solid shape $\mathcal{S}$ (represented by a 2-manifold mesh) into a set of almost convex polyhedra $\{\mathcal{S}_i\}$, such that the solid shape determined by the union $\bigcup\{\mathcal{S}_i\}$ is equivalent to the original shape $\mathcal{S}$, and there is no intersection between the components $\mathcal{S}_i$ except for the boundary. We utilize a *concavity* metric to measure the difference between each component and its convex hull. Our objective is to minimize the number of components while ensuring the concavity of each component is within a pre-defined threshold $\epsilon$. After decomposition, the convex hulls of the generated components can be used to approximate the original shape $\mathcal{S}$. By adjusting the threshold $\epsilon$, users can balance the number of components and the level of detail of the decomposition. We assume that the input is a 2-manifold solid mesh and we can convert imperfect input (e.g., non-watertight or non-manifold meshes) by pre-processing with an off-the-shelf manifold conversion algorithm [Huang et al. 2018].

As shown in Algorithm 1, we utilize a divide-and-conquer strategy to recursively decompose the solid shape. For each component whose concavity is greater than the threshold $\epsilon$, we search for the best cutting plane and use it to split the component into two (lines 8

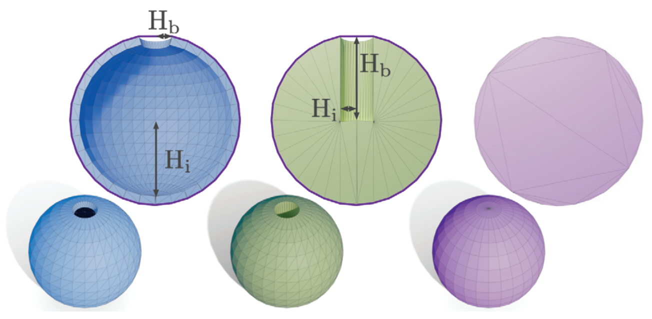


Fig. 2. From left to right: a spherical shell with a small opening, a solid sphere with a deep hole, and a solid sphere indicating the convex hull of the left two shapes. Both $H_i(\mathcal{S})$ and $H_b(\mathcal{S})$ are necessary to measure the difference between a shape and its convex hull. In the blue example, $H_i(\mathcal{S}) \gg H_b(\mathcal{S})$, while in the green example, $H_b(\mathcal{S}) \gg H_i(\mathcal{S})$. The purple polygons surrounding the cross-sections indicate the boundary surface of the convex hull.

to 10). The cutting process is recursively applied until all the components satisfy the concavity constraint. At last, a post-processing step is applied in order to merge the generated components and further reduce the number of components (line 12).

Our high-level pipeline is similar to V-HACD, and the differences mainly come from three aspects. First, we introduce a novel collision-aware concavity metric that is more sensitive to detect various approximation errors and leads to a more reasonable and robust decomposition. Also, we propose an efficient way to calculate the concavity (Section 4). Second, without voxelizing the input mesh and splitting the voxels, we directly decompose the shapes by cutting meshes with 3D planes, which avoids the discretization errors and supports more precise and efficient decomposition (Section 5). Third, instead of greedily searching for the locally optimal decompose actions with one-step results, we propose leveraging multi-step tree search to achieve globally better decomposition (Section 6).

## 4 COLLISION-AWARE CONCAVITY METRIC

### 4.1 Concavity Definition

ACD works utilize *concavity* to measure the difference between a solid shape $\mathcal{S}$ and its convex hull $\mathrm{CH}(\mathcal{S})$, which can be used to quantify the quality of the decomposed components. The ideal *concavity* should be able to recognize all unreasonable approximations and penalize them with a high cost. It should also be efficient to calculate, as numerous concavity calculations are needed during the decomposition process.

There is no consensus on the definition of the concavity among existing ACD works. Some of them [Ghosh et al. 2013; Lien and Amato 2004, 2007, 2008; Liu et al. 2016; Mamou and Ghorbel 2009] focus on the distance between the boundary surfaces of $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$, while other works [Attene et al. 2008; Mamou et al. 2016; Thul et al. 2018] utilize the volume difference between $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$ as the concavity. Please refer to Section 2.2 for more details.

A reasonable decomposition should preserve the collision conditions of the input shape, which means any position in the space that is unlikely to collide with the input shape (i.e., far away from

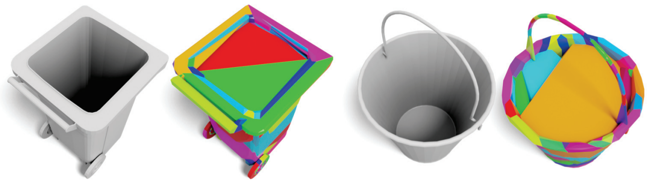


Fig. 3. Failure cases of the boundary-distance-based methods (from HACD [Mamou and Ghorbel 2009]). Focusing only on the boundary distance between the shape and its convex hull, HACD may fail to handle the hollow structures and fill the interior space.

the input shape), should not collide with the convex decomposition results as well. Otherwise, both the structure and functionality of the input shape can be greatly affected. A good concavity metric should thus be sensitive to detect the approximation errors that significantly change the collision conditions. However, we argue that both the boundary-distance-based and the volume-based metrics may fail to do so and lead to undesirable decomposition results:

- *Boundary-distance-based metrics alone are insufficient for preserving collision conditions.* As shown in Figure 2, if $\mathcal{S}$ is a hollow spherical shell (blue example), the boundary distance between $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$ may be quite small, and the algorithm may thus fill the interior space and approximate $\mathcal{S}$ with a solid convex hull. However, it is inappropriate to do so if we want to exploit the free space inside $\mathcal{S}$. In fact, such shell-like structures are quite common in applications. For example, in physical simulators, a teapot needs to hold water, and it's inappropriate to approximate the body of the teapot with a solid convex hull. Otherwise, the water particles collide with the interior of the teapot. Figure 3 shows some failure cases of the boundary-distance-based concavity.
- *Volume-based metrics alone are insufficient for preserving collision conditions.* In some cases, the volume difference between $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$ may be very small, but significant differences may exist at the boundary of $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$. For example, in Figure 2, if $\mathcal{S}$ is a solid sphere with a deep hole (green one), the relative volume of the deep hole is small. However, it is inappropriate to approximate $\mathcal{S}$ with its convex hull, which fills the hole. Figure 4 shows common failure cases of the volume-based concavity, where V-HACD tends to utilize thin planar components to close the holes.

The failure cases of existing concavity metrics are fatal to the functionality of the objects in downstream applications. For example, suppose we use the decomposition results shown in Figure 4 in a simulator. In that case, robots can no longer grasp the scissors by the circles, can no longer use the kettle to pour water, and can no longer interact with the water dispenser on the refrigerator.

However, it's non-trivial to directly combine the existing boundary-distance-based metrics and volume-based metrics since they have different geometric meanings and scales. On the other hand, many existing boundary-distance-based metrics involve intricate geometry processing and are cumbersome to calculate.

To overcome the limitations of existing approaches, we propose a novel *collision-aware concavity metric* that examines the decomposed component from both the boundary and the interior with a unified metric. To introduce the metric, we first review the definition of the

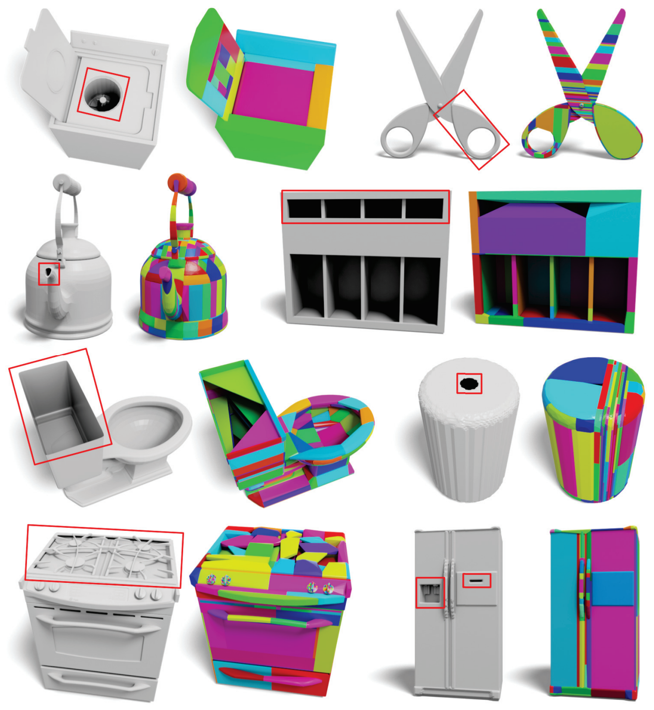


Fig. 4. Failure cases of the volume-based methods (from V-HACD [Mamou et al. 2016]). Focusing on the volume difference, V-HACD may fill holes when the relative volume of the errors is not too large (e.g., thin planar structures). The red rectangles highlight the error-prone regions.

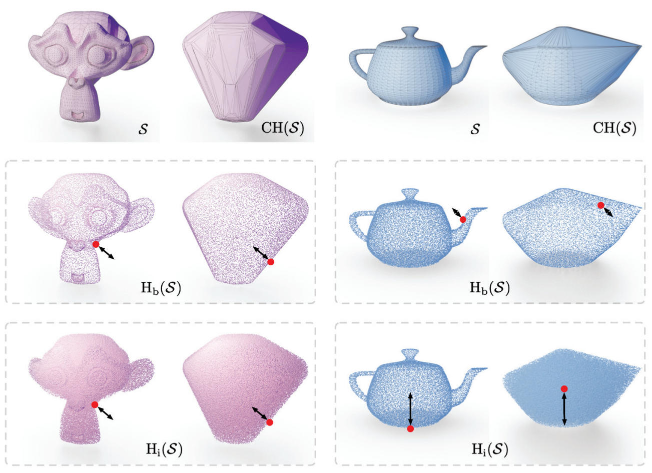

$\mathcal{S}$
$\mathrm{CH}(\mathcal{S})$
$\mathcal{S}$
$\mathrm{CH}(\mathcal{S})$
$\mathrm{H_b}(\mathcal{S})$
$\mathrm{H_b}(\mathcal{S})$
$\mathrm{H_i}(\mathcal{S})$
$\mathrm{H_i}(\mathcal{S})$

Fig. 5. The figure illustrates how to calculate the concavity for "Blender's Suzanne" and "Utah teapot". The first and third columns show the input shapes and their sampled point clouds, while the second and fourth columns show the convex hulls and their sampled point clouds. From top to bottom: manifold meshes, point clouds sampled from the boundary surfaces (for calculating $\mathrm{H_b}(\mathcal{S})$), and point clouds sampled from the interior of the shapes (for calculating $\mathrm{H_i}(\mathcal{S})$). The teapot is hollow since its inside is connected to the outside world through the spout. In each dotted square, the red points indicate the pair of points that achieves $\mathrm{H_i}(\mathcal{S})$ or $\mathrm{H_b}(\mathcal{S})$.

Hausdorff distance for two point sets:

$$\mathrm{H}(A,B) = \max\{\sup_{a\in A} d(a,B), \sup_{b\in B} d(b,A)\} \tag{1}$$

where $A$ and $B$ are two point sets, $d(x,Y) = \inf_{y\in Y} d(x,y)$ and $d(x,y)$ indicates the Euclidean distance between the two points.

As shown in Figure 5, we sample two pairs of point sets to measure the distance between a solid shape $\mathcal{S}$ and its convex hull $\mathrm{CH}(\mathcal{S})$. Specifically, by sampling the points from the two boundary surfaces, we define $\mathrm{H_b}(\mathcal{S})$ as:

$$\mathrm{H_b}(\mathcal{S}) = \mathrm{H}(\mathrm{Sample}(\partial\mathcal{S}), \mathrm{Sample}(\partial\,\mathrm{CH}(\mathcal{S}))) \tag{2}$$

where Sample() indicates the point set sampling operation, and $\partial$ denotes the boundary surface of a solid shape. Similarly, by sampling points from the interior of the shapes, we define $\mathrm{H_i}(\mathcal{S})$ as:

$$\mathrm{H_i}(\mathcal{S}) = \mathrm{H}(\mathrm{Sample}(\mathrm{Int}\,\mathcal{S}), \mathrm{Sample}(\mathrm{Int}\,\mathrm{CH}(\mathcal{S}))) \tag{3}$$

where Int denotes the interior of a solid shape.

The *concavity* of a solid shape $\mathcal{S}$ is then proposed to be:

$$\mathrm{Concavity}(\mathcal{S}) = \max(\mathrm{H_b}(\mathcal{S}), \mathrm{H_i}(\mathcal{S})) \tag{4}$$

By measuring the distance from both the boundary surface and the interior, the proposed concavity can better capture shape differences and well address failure cases of the prior concavity metrics.

We argue that both terms in Equation 4 are necessary and complementary for measuring the shape difference. Only using $\mathrm{H_b}(\mathcal{S})$ may fail to handle the shell-like structures. Taking the spherical shell (the blue one shown in Figure 2) as an example, $\mathrm{H_b}(\mathcal{S})$ only measures the radius of the small opening on the boundary surface, which is quite small. In contrast, $\mathrm{H_i}(\mathcal{S})$ recognizes the large difference from the interior and penalizes it with the inner radius of the shell. On the other hand, only using $\mathrm{H_i}(\mathcal{S})$ may fail to capture the difference between boundary surfaces. For the solid sphere with a deep hole (the green one shown in Figure 2), $\mathrm{H_i}(\mathcal{S})$ only measures the radius of the hole, no matter how deep the hole is. Instead, $\mathrm{H_b}(\mathcal{S})$ is able to measure the depth of the hole, which better captures the difference.

A notable property of the proposed concavity metric is its *collision awareness*. Our concavity encourages decompositions to preserve the collision conditions by penalizing the distances between points in $\mathrm{CH}(\mathcal{S}) - \mathcal{S}$ (i.e., the extra volume introduced by the convex hull) and the original shape $\mathcal{S}$. Therefore, our metric is sensitive to detecting approximation errors that significantly alter the collision conditions, no matter they are fine-grained structures with small volume or thin planar structures. Instead of calculating an overall average difference, the proposed concavity focuses on the worst case (i.e., the farthest point pair). This is because that in many applications (e.g., robot simulation), we need a guarantee about the worst case to avoid fatal approximations to certain parts of the shape, even though the approximation may look good overall.

Moreover, by using our proposed metric, it's more intuitive for users to set and adjust the concavity threshold $\epsilon$, since one can interpret the threshold as the degree to which the original shape becomes thicker. In contrast, the volume-based concavity may not have such an intuitive interpretation, and the change caused by adjusting the volume difference threshold may be less predictable.

### 4.2 Efficient Concavity Calculation

Besides recognizing all implausible approximations, a good concavity metric should also be efficient to calculate. Our proposed metric samples points from the shape and its convex hull, and then calculates Hausdorff distance between the point sets, which avoids intricate geometry processing required by existing boundary-distance-based metrics. The nearest neighbor calculation in Hausdorff distance can be accelerated by approximation approaches and parallel computation [Arya et al. 1998; Blanco and Rai 2014]. Moreover, when calculating $\mathrm{H_b}(\mathcal{S})$, we exploit point-to-triangle-face distances to improve the precision further and reduce the number of samples.

That being said, calculating accurate $\mathrm{H_i}(\mathcal{S})$ can still be time-consuming. The calculation of $\mathrm{H_b}(\mathcal{S})$ only needs sampled points from the boundary surfaces, while the calculation of $\mathrm{H_i}(\mathcal{S})$ needs much more sampled points from the interiors of the shapes to achieve high precision. See Figure 5 for differences between the sampled points. To further accelerate the concavity calculation, we propose a surrogate term $\mathrm{R_v}(\mathcal{S})$ for $\mathrm{H_i}(\mathcal{S})$:

$$\mathrm{R_v}(\mathcal{S}) = \sqrt[3]{\frac{3(\mathrm{Vol}(\mathrm{CH}(\mathcal{S})) - \mathrm{Vol}(\mathcal{S}))}{4\pi}} \tag{5}$$

where $\mathrm{Vol}(\mathrm{CH}(\mathcal{S}))$ - $\mathrm{Vol}(\mathcal{S})$ indicates the volume difference between the convex hull and the input shape. The geometric interpretation of $\mathrm{R_v}(\mathcal{S})$ is the radius of a sphere with volume $\mathrm{Vol}(\mathrm{CH}(\mathcal{S}))$ - $\mathrm{Vol}(\mathcal{S})$, which is potentially the largest inscribed sphere within the difference of $\mathrm{CH}(\mathcal{S})$ and $\mathcal{S}$. $\mathrm{R_v}(\mathcal{S})$ serves a similar role as $\mathrm{H_i}(\mathcal{S})$ to recognize the differences within the interior of solid shapes. Moreover, we can actually prove a theoretical guarantee for $\mathrm{R_v}(\mathcal{S})$:

THEOREM 1. *For every solid shape $\mathcal{S}$, we have:*

$$\sqrt{2}\max(\mathrm{H_b}(\mathcal{S}), \mathrm{R_v}(\mathcal{S})) \geq \max(\mathrm{H_b}(\mathcal{S}), \mathrm{H_i}(\mathcal{S}))$$

The theorem indicates that we can use $\mathrm{H_b}(\mathcal{S})$ and $\mathrm{R_v}(\mathcal{S})$ to bound the proposed $\mathrm{Concavity}(\mathcal{S})$ and still be able to recognize any unreasonable approximations. Please refer to the supplementary materials for detailed proof. In practice, we find that $\mathrm{R_v}(\mathcal{S})$ often overestimates $\mathrm{H_i}(\mathcal{S})$. We thus utilize:

$$\widetilde{\mathrm{Concavity}}(\mathcal{S}) = \max(\mathrm{H_b}(\mathcal{S}), k\,\mathrm{R_v}(\mathcal{S})) \tag{6}$$

where $k$ is a coefficient less than 1, as the concavity metric to achieve a better approximation. It's much faster to calculate $\widetilde{\mathrm{Concavity}}(\mathcal{S})$, since it avoids sampling points from the interior of the shapes and the corresponding nearest neighbor calculation.

## 5 SHAPE DECOMPOSITION BY CUTTING MESHES

As introduced in Section 3, our method recursively decomposes a shape $\mathcal{S}$ into smaller components. There are various ways to represent a component and different geometry processing strategies to decompose the shape. We discuss the geometric design in this section and leave the search strategy to the next section.

In general, prior works mainly take two types of decomposing strategies. For triangle-grouping-based methods [Liu et al. 2016; Mamou and Ghorbel 2009], they preserve the triangle faces of the input mesh, and group the faces by top-down division or bottom-up clustering. For volume-based methods, like V-HACD, they first voxelize the input mesh, use voxels to represent the shape, and then divide the voxels. However, both strategies have some apparent limitations. Specifically, triangle-grouping-based methods often output components with zigzag boundaries (see Figure 6). As a result, the convex hulls of the decomposed components usually intersect with each other, which is undesirable in many applications. On the other hand, although volume-based methods avoid crooked boundaries of the components, their voxelization pre-processing may introduce discretization artifacts. More importantly, volume-based methods may fail to recognize already convex components. As shown in Figure 7, although the input shapes are already convex, V-HACD still regards them as non-convex due to the discretization error, and may try to further divide the voxels. Not only that, but to achieve high precision, V-HACD would require a large number of voxels which would slow down the decomposition algorithm.

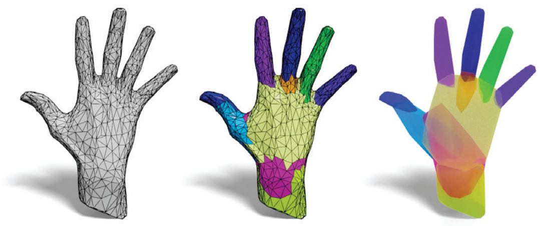


Fig. 6. An example of triangle-grouping-based methods (from HACD). From left to right: (a) Input triangle mesh. (b) Grouping results of the triangle faces, where each color indicates a component. There are zig-zag boundaries between different components. (c) Corresponding convex hulls of each component, and they intersect with each other.

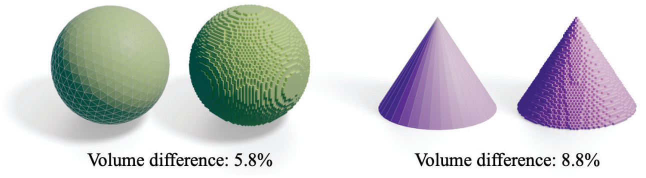
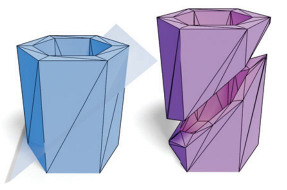


Fig. 7. Each pair shows an input mesh and its voxelization (64×64×64). Even the input shapes are already convex, there are noticeable volume differences between the voxels and their bounding convex hulls, making V-HACD fail to recognize already convex parts.


Instead, we follow [Thul et al. 2018] to utilize triangle meshes to represent solid components during decomposition and directly cut the manifold meshes with 3D planes. As shown in the left inset, given a manifold mesh and a 3D cutting plane, we split the mesh into two parts along the plane surface. Each resulting part is still a manifold mesh with flat boundaries and can be recursively decomposed. In this way, we ensure convex hulls of the decomposed components are intersection-free. Moreover, without voxelization as pre-preprocessing, we preserve fine-grained details and avoid over-decomposing convex shapes.

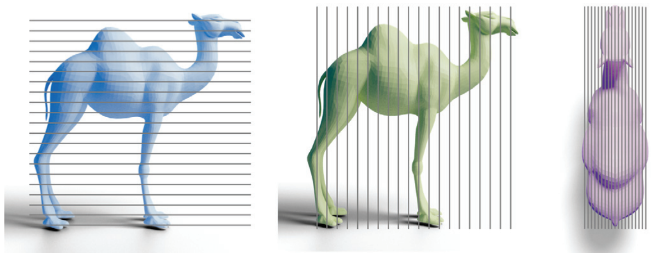


Fig. 8. We sample $m$ equally-spaced candidate cutting planes (illustrated by straight lines here) from each axis-aligned direction. After finding the best candidate with a tree search, we also refine the plane's position.

However, existing mesh cutting functions from off-the-shelf computational geometry libraries (e.g., CGAL [Fabri and Pion 2009]) are very heavy and time-consuming, which slows down our decomposition algorithm. Therefore, we implement a lightweight cutting function that is about 100x faster than CGAL's implementation. Specifically, the implementation mainly includes four steps: (a) Find triangles that are not intersecting with the cutting plane, and group them into two sets according to which side of the plane the triangle is in. The two triangle sets are later used to form the two parts. (b) Split each intersecting triangle into two with the cutting plane, and then add them into the two sets. (c) Add new surfaces (overlapping with the cutting plane) for the two parts to form solid meshes. To achieve this, we solve the constrained Delaunay triangulation [Shewchuk 1996], where the intersecting edges serve as boundary constraints. (d) Remove redundant triangles (if any) introduced in step (c), which correspond to holes on the newly added surfaces.

## 6 MONTE CARLO TREE SEARCH FOR CUTTING PLANE

### 6.1 Search Space

During the decomposition process, for each intermediate component $C$ whose $\widetilde{\text{Concavity}}(C)$ is greater than the threshold $\epsilon$, we find a cutting plane to split $C$ into two parts $C_L$ and $C_R$. Since there are infinite cutting planes in the 3D space, we follow V-HACD [Mamou et al. 2016] to restrict the candidate planes to be axis-aligned (parallel to $xy$, $xz$, or $yz$ planes), and sample $m$ equal-spaced candidates along each direction, as shown in Figure 8. Axis-aligned discretized candidate planes enable a feasible search space and avoid irregular cutting results. The found optimal discretized candidate will be refined in the continuous local neighborhood for a more accurate cutting. (section 6.5) Also, we optionally perform a PCA [Pearson 1901] for the input shape at the beginning of the decomposition algorithm to align the cutting plane directions with the principal axes of the input shape.

### 6.2 One-Step Greedy vs. Multi-Step Search

Intuitively, prior works [Mamou et al. 2016; Thul et al. 2018] greedily find a cutting plane from the candidates, which minimizes:

$$\max(\widetilde{\text{Concavity}}(C_L), \widetilde{\text{Concavity}}(C_R)) \tag{7}$$

However, the one-step greedy search may be short-sighted and fail to find cutting planes that result in better global decomposition,

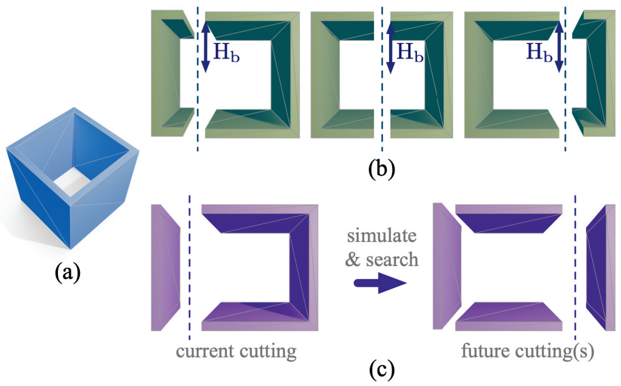


Fig. 9. Comparison between one-step greedy and multi-step search. (a) Input shape (a cube without top and bottom). (b) The one-step greedy algorithm fails to find the proper first cutting plane, since all candidate cutting planes lead to the same cost (Equation 7) as illustrated by the blue arrows ($H_b$). (c) The multi-step search algorithm can instead find the proper first cutting plane by simulating and searching future cuttings, which leads to the globally optimal solution (decomposed into exactly four pieces).

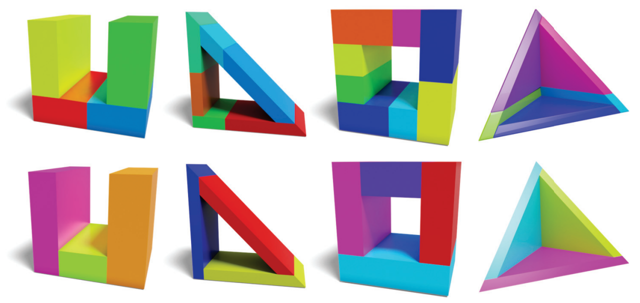


Fig. 10. Failure cases of one-step greedy search. First row: V-HACD employs a greedy search and generates redundant components. Second row: our method utilizes a multi-step tree search and solves the cases perfectly.

and end up with more decomposed components. Figure 9 shows such an example, where the optimal solution is to decompose the input shape into exactly four square parts. However, when only one step is taken into account, the greedy search tries to cut the shape from the middle, resulting in more components. Figure 10 compares decomposition results of some simple primitives. Since V-HACD utilizes a greedy search, it fails to decompose the cases perfectly.

Moreover, for the one-step greedy search, only using a concavity metric to find the cutting plane may often produce poor results and even block the decomposition algorithm. For example, when only considering one cutting, many candidate planes may lead to the same concavity deduction (Equation 7), and the cutting plane selection thus becomes arbitrary in these draw situations. To this end, prior works [Mamou et al. 2016] introduce various auxiliary heuristic terms (e.g., balance term and symmetry term) for Equation 7 as a workaround to make the algorithm more robust to various cases.

We instead propose to take multiple steps into account when searching for a cutting plane. Specifically, we solve a Monte Carlo

**Algorithm 2:** Search for Cutting Plane

```
Function MCTS($C, t, d$):
    Create root node $v_0$ with input mesh $C$
    while within $t$ iterations do
        $\{\mathcal{P}_1, \cdots, \mathcal{P}_l\}, v_l \leftarrow$ TreePolicy ($v_0$, $d$)
        $\{\mathcal{P}_{l+1}, \cdots, \mathcal{P}_d\} \leftarrow$ DefaultPolicy ($v_l$, $d$)
        $q \leftarrow$ Quality($\{\mathcal{P}_1, \cdots, \mathcal{P}_d\}$)    // Calculate a score
        Backup($v_l, q$)    // Update the score along the tree path
    $v^* \leftarrow \arg\max_{v' \in \text{children of } v_0} Q(v')$
    return corresponding plane of $v^*$
Function TreePolicy($v$, $d$):
    $\mathcal{S} \leftarrow \emptyset$    // Selected cutting planes
    while $depth(v) < d$ do    // From the root to the leaf
        $c^* = \arg\max_{c_i \in \text{components of } v} \text{Concavity}(c_i)$
        if all cutting plane candidates of $c^*$ are expanded then
            $v \leftarrow$ best child of $v$ according to the UCB formula
            $\mathcal{S} \leftarrow \mathcal{S} + \{$corresponding plane of $v\}$
        else    // Expand a new child for $v$
            Randomly select a untried cutting plane $\mathcal{P}$ of $c^*$
            Cut $c^*$ into $c_l^*$ and $c_r^*$ with $\mathcal{P}$
            Create a new child $v'$ to $v$ with $\mathcal{P}, c_l^*$ and $c_r^*$
            return $\mathcal{S} + \{\mathcal{P}\}, v'$
    return $\mathcal{S}, v$
Function DefaultPolicy($v$, $d$):
    $\mathcal{S} \leftarrow \emptyset$    // Selected cutting planes
    $\{c_i\}$ = Copy(components of $v$)    // Avoid affecting tree nodes
    for $i \in$ range($d - depth(v)$) do
        $c^* = \arg\max_{c_i} \text{Concavity}(c_i)$
        for direction in $\{xy, xz, yz\}$ do
            Try to cut $c^*$ into $c_l^*$ and $c_r^*$ from middle with a
             plane along the direction
            $q \leftarrow -\max(\text{Concavity}(c_l^*), \text{Concavity}(c_r^*))$
        $\mathcal{P} \leftarrow$ cutting plane that lead to the largest $q$
        Cut $c^*$ into $c_l^*$ and $c_r^*$ with $\mathcal{P}$
        $\mathcal{S} \leftarrow \mathcal{S} + \{\mathcal{P}\}$
    return $\mathcal{S}$
Function Backup($v, q$):
    while $v$ is not null do    // From the leaf to the root
        $N(v) \leftarrow N(v) + 1$    // Visit times
        $Q(v) \leftarrow \max(Q(v), q)$    // Value function
        $v \leftarrow$ parent of $v$
```

tree search (MCTS), which simulates multiple future cuttings, to find a cutting plane for each intermediate component $C$. This way, we pay more attention to long-term interests and are more likely to find cutting planes that lead to global optimal decompositions. Moreover, we find that by considering multiple steps during the tree search, we no longer need any other heuristic terms to prevent various corner cases.

## 6.3 Search Tree Structure

It's non-trivial to apply MCTS to our cutting plane search, and many dedicated designs are involved. Specifically, in our tree search, each node represents a set of decomposed components $\{c_i\}$ for $C$, and the root node contains a single component $\{C\}$. For each node, we aim to cut the component $c^*$ with the largest concavity among those components associated with the node, and each child node corresponds to a cutting action for $c^*$. Since a cutting plane splits $c^*$ into two parts, the number of a child node's components is equal to the number of its parent node's components plus one. Also, since we sample $m$ candidate cutting planes from each axis-aligned direction, each node contains at most $3m$ child nodes.

As shown in Algorithm 2, there is only a single root node in the search tree at the beginning, and $t$ iterations of tree search are performed. In each iteration, we first utilize the `TreePolicy()` to select a tree node for expansion by balancing the exploration and exploitation. We then evaluate the newly expanded tree node with the `DefaultPolicy()`. Specifically, for the `TreePolicy()`, we start from the root node and select successive child nodes until a node $l$ with unexpanded child nodes is reached. During the node selection, the UCB (Upper Confidence Bound) [Kocsis and Szepesvári 2006] value is used to balance the exploration (gather more information about less-visited nodes) and exploitation (choose the optimal node based on existing information):

$$Q(n) + c\sqrt{\frac{2\ln \mathrm{N}(n')}{\mathrm{N}(n)}} \tag{8}$$

where $n$ indicates the current node, $n'$ is its parent node, $Q()$ is the value function, $N()$ indicates the number of visit times, and $c$ is the exploration parameter. After reaching $l$, we then expand one child node for $l$ by randomly selecting an untried cutting plane $\mathcal{P}$ for the component $c^*$ and cutting $c^*$ with $\mathcal{P}$ (lines 18 to 20).

## 6.4 Tree Node Evaluation

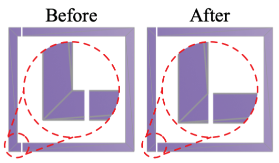
It's non-trivial to evaluate the expanded node, and one of the challenges is to compare decompositions with different number of cuttings (nodes at different depths). We thus employ a `DefaultPolicy()` to complete one playout, which leads to a fixed number of $d + 1$ components with one-step greedy cuttings. Specifically, the greedy strategy first tries to cut the component $c^*$ with the largest concavity from the middle along three axis-aligned directions (line 29). By comparing the one-step concavity reduction (Equation 7), the strategy then keeps the best cutting results from the three trials. We repeatedly apply this greedy cutting strategy until we get $d + 1$ decomposed components, and then calculate a score for the node. Please note that, in the tree search, we do not wait until all components are almost convex. Because then it could be much more time-consuming, and most cuttings may be achieved by the default policy, making the results less relevant to the searched tree nodes.

After applying the `TreePolicy()` and `DefaultPolicy()`, we get $d$ cutting planes and $d + 1$ resulting components, where the first $l$ planes $\{\mathcal{P}_1, \cdots, \mathcal{P}_l\}$ are associated with a tree path starting from the root node to the leaf node, and the remaining planes $\{\mathcal{P}_{l+1}, \cdots, \mathcal{P}_d\}$ come from `DefaultPolicy()`. To evaluate the decomposition (line 6), we not only assess the $d+1$ decomposed components after $d$ cuttings, but also examine the intermediate results. In this way, we can differentiate paths leading to similar final results and pick the path that achieves good results at a earlier stage. Specifically, the score for the set of cutting planes is calculated as:

$$\text{Quality}(\{\mathcal{P}_1, \cdots, \mathcal{P}_d\}) = \frac{1}{d}\sum_{i=1}^{d} -\max_{j=1}^{i+1} \text{Concavity}(c_{ij}) \quad (9)$$

where $c_{ij}$ represents one of the $i+1$ components after the $i$th cutting. At the end of each iteration, we update the scores for all nodes along the path (line 7).

## 6.5 Plane Refinement and Component Merging



After completing an MCTS search ($t$ iterations), we take the optimal cutting plane of the root node (from the child with the highest score). Since we sample discretized equally-spaced planes as candidates, it's likely that the searched plane may not be the optimal one from a continuous space. As shown in the left inset, we thus locally refine the searched plane. Specifically, we find the $d$ cutting planes $\{\mathcal{P}_1, \cdots, \mathcal{P}_d\}$ that correspond to the optimal path in the search tree. We finetune the first plane $\mathcal{P}_1$ within a small range using a greedy ternary search, while other $d-1$ cutting planes and the score function (Equation 9) remain the same. In this way, we can find high-resolution cutting planes without increasing the complexity of tree search. The refined plane is then used to cut $C$ into two parts. Please note that, in order to accelerate the tree search, we use $\mathrm{R_v}$ as the concavity within the MCTS. However, outside MCTS, we still use $\max(\mathrm{H_b}(\mathcal{S}), k\,\mathrm{R_v}(\mathcal{S}))$ to determine whether a component satisfies the concavity constraint and whether further cuttings are needed.

Since we recursively split a component into two, it's possible that among the set of decomposed components, there exist some components that could be merged to form a larger component that is still almost convex. We thus perform a post-processing to merge the generated components and further reduce the number of components. Specifically, we traverse all pairs of adjacent components and check the $\widetilde{\text{Concavity}}$ of the merged component. If it's within the threshold $\epsilon$, we will replace the two components with the merged one. We repeat the process until no more components to merge.

## 7 EVALUATIONS

### 7.1 Comparing with Existing Methods

We evaluate the methods on the V-HACD dataset [Mamou et al. 2016] and PartNet-Mobility dataset [Xiang et al. 2020]. V-HACD dataset contains 61 shapes and most shapes are animals or humans. PartNet-Mobility dataset contains 2,346 shapes covering a wide range of indoor articulated objects (e.g., cabinets and scissors), which can be used for robotics simulation. Compared to the V-HACD dataset, shapes in the PartNet-Mobility dataset contain more complex inner structures and delicate details. It also requires higher quality decomposition to enable fine-grained object interaction. Each shape in the PartNet-Mobility dataset may contain multiple parts and we decompose each part individually.

Table 1. Quantitative comparison on the V-HACD dataset and PartNet-Mobility dataset. For both HACD and V-HACD, we aim to compare the number of decomposed components. For Animation, we aim to match the number of decomposed components and compare the concavity scores. The runtime is in seconds.
[1] Animation is run with a different system configuration.

| dataset | method | **# component** ↓ | concavity ↓ | runtime ↓ |
|---|---|---|---|---|
| V-HACD | HACD | 57.6 | 0.118 | 67.2 |
| | Ours | **29.6** | 0.084 | 201.0 |
| PartNetM | HACD | 33.5 | 0.414 | 268.9 |
| | Ours | **7.3** | 0.204 | 194.4 |

| dataset | method | **# component** ↓ | concavity ↓ | runtime ↓ |
|---|---|---|---|---|
| V-HACD | V-HACD | 60.2 | 0.067 | 192.1 |
| | Ours | **29.8** | 0.044 | 201.9 |
| PartNetM | V-HACD | 44.6 | 0.055 | 206.0 |
| | Ours | **20.1** | 0.052 | 253.4 |

| dataset | method | # component ↓ | **concavity** ↓ | runtime ↓ |
|---|---|---|---|---|
| V-HACD | Animation | 34.4 | 0.069 | 28.5[1] |
| | Ours | 34.5 | **0.049** | 229.8 |

We compare our proposed method with existing approximate convex decomposition methods, HACD [Mamou and Ghorbel 2009], V-HACD [Mamou et al. 2016], and Animation [Thul et al. 2018]. It's non-trivial to compare different decomposition methods, since some methods use the concavity threshold as the termination rule, while other methods take the expected number of components as input. Moreover, different methods may have different concavity definitions. As a result, we compare our proposed method with each of the baseline method separately. Specifically, for comparison with HACD and V-HACD, we first run their methods by setting hyper-parameters that encourage as fine-grained decomposition as possible. After that, we calculate a concavity score for each of their generated decomposition solution:

$$\text{Score}(\mathcal{S}, \{\mathcal{CH}_1, \cdots, \mathcal{CH}_n\}) = \max_i \widetilde{\text{Concavity}}(\mathcal{S} \cap \mathcal{CH}_i) \quad (10)$$

where $\{\mathcal{CH}_1, \cdots, \mathcal{CH}_n\}$ indicates the set of generated convex hulls for the input shape $\mathcal{S}$, and $\mathcal{S} \cap \mathcal{CH}_i$ calculates the intersection between the input solid shape and the $i$th convex hull. We then utilize that score as the concavity threshold for running our method. In this way, our method will generate decomposition with finer details, and we aim to compare the number of decomposed components.

For comparison with Animation, we first run our method with a fixed concavity threshold of 0.05. We then run Animation for each shape to generate the same number of components as our results. After that, we can fairly compare the concavity scores (Equation 10) of the two methods. Since Animation didn't release their code due to commercial reasons, we ask the authors to help us run the method

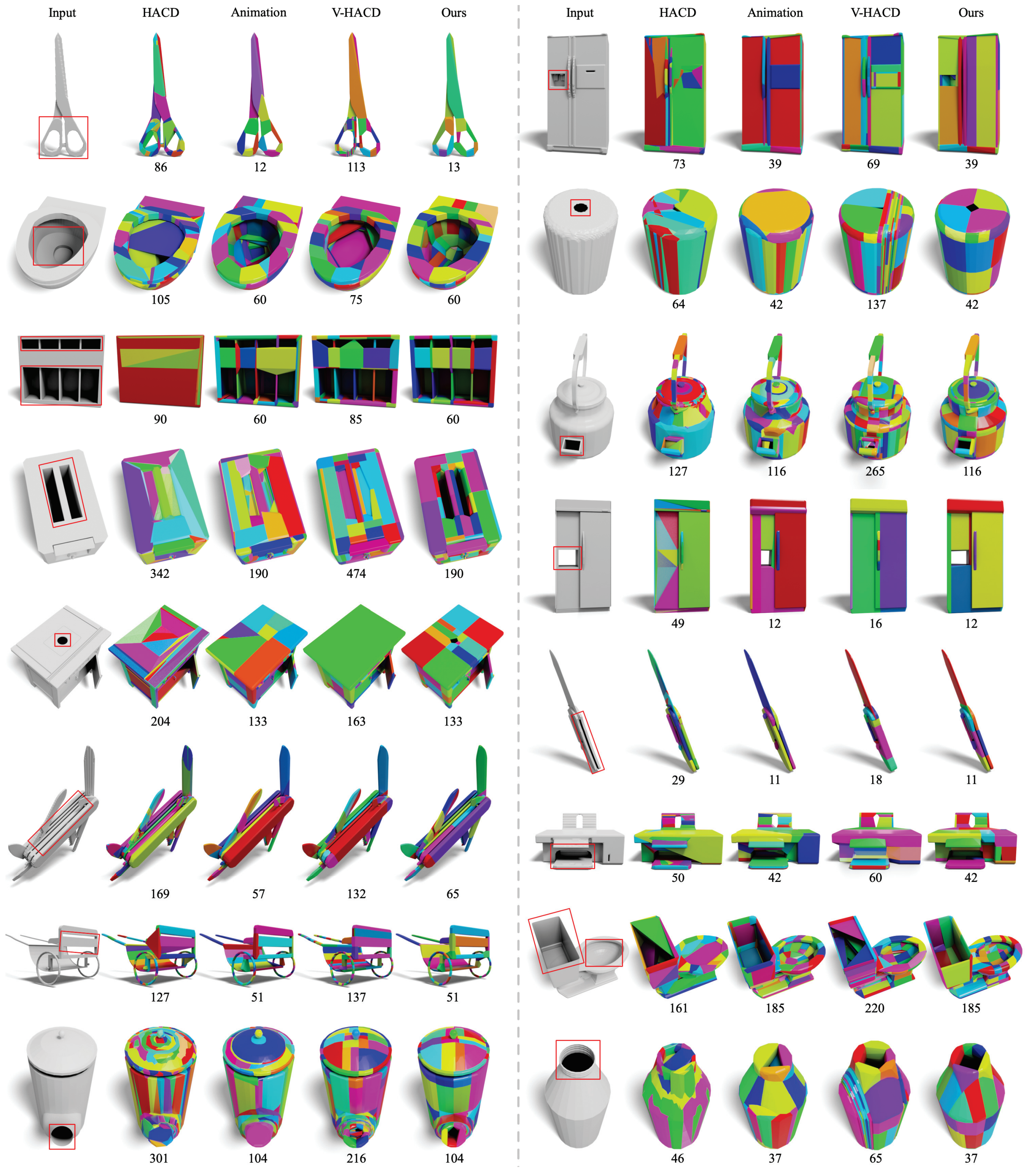


Fig. 11. We compare our method with HACD, Animation, and V-HACD. Please zoom in for the details. The input shapes come from the PartNet-Mobility dataset. Each shape may consist of multiple parts (e.g., two blades of a scissor), and each part is decomposed individually. The red rectangles on the input shapes highlight the error-prone regions. The numbers below the results indicate the number of decomposed components. For both HACD and V-HACD, we set parameters to encourage fine-grained decomposition. For our method, we use a concavity threshold of 0.05. For Animation, we let the number of decomposed components equal our results.

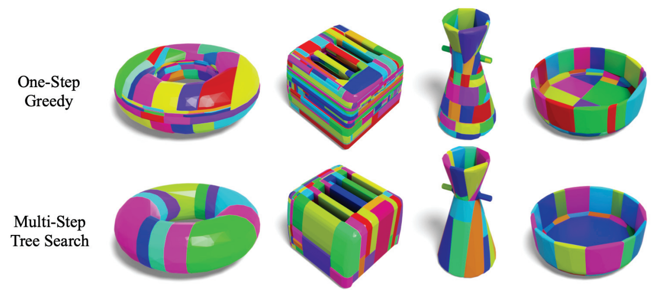


Fig. 12. First row: results from a one-step greedy strategy with our proposed concavity metric. Second row: results from our method with multi-step tree search. Both methods are tested with the same concavity threshold.

on their machine. We quantitatively compare our method and Animation only on the V-HACD dataset, since PartNet-Mobility dataset is too large.

As shown in Table 1, our method outperforms HACD and V-HACD in terms of the number of components on both datasets with a large margin. At the same time, the mean concavity scores of our results are even lower than that of the two methods. As for comparison with Animation, both methods generate almost the same number of components for each shape. However, we achieve lower concavity scores which indicate the finer decomposition results by our method. We run methods with a single CPU thread (except for Animation) and we share similar runtime with HACD and V-HACD. Animation is more time-efficient, since its implementation has been carefully optimized with industrial codes.

Figure 11 shows the qualitative comparison on the PartNet-Mobility dataset, where many shapes need fine-grained decomposition to enable downstream object interaction. For example, the inside ring of the scissors should be large enough, so that an agent can grab them, the slots of the toaster and the spouts of the kettles should not be filled so that they can work properly. However, decomposition results from existing methods may fail to preserve the original shape's functionality. Among the three baselines, HACD produces the worst results since it only considers the difference between the boundary surfaces while ignoring the interior structures. It fills the inner space for most shapes. V-HACD and Animation produce better results but still suffer from the hole filling issues to some extent. Both methods utilize the volume difference as the concavity metric and may ignore fine-grained structures or introduce some thin-planar components to fill the holes, since those errors do not receive a large penalty from the volume-based concavity. Instead, by leveraging our collision-aware concavity, our method keeps most structures of the input shapes. Also, the numbers of our decomposed components are smaller than those of HACD and V-HACD, which may speed up the downstream applications.

## 7.2 Ablation studies

***One-step greedy vs. multi-step tree search.*** To examine the benefit introduced by the multi-step tree search, we construct a counterpart greedy baseline which utilizes our proposed concavity metric and directly search for the best one-step cutting plane that leads to the minimum resulting concavity (Equation 7). We compare the one-step greedy baseline and our proposed method with the same concavity threshold 0.05. The quantitative results on the V-HACD dataset are shown in Table 2, where the one-step greedy baseline generates much more components than our multi-step tree search version. Moreover, since the multi-step tree search version reduces the number of rounds (fewer parts) and utilizes a simplified concavity calculation in the tree search, it is even faster than the greedy baseline.

Table 2. Quantitative comparison between a one-step greedy baseline and our method with multi-step tree search on the V-HACD dataset.

| One-Step Greedy | | Multi-Step Tree Search | |
|---|---|---|---|
| # component ↓ | runtime ↓ | # component ↓ | runtime ↓ |
| 49.9 | 271.7 | **34.5** | **229.8** |

As shown in Figure 12, the one-step greedy algorithm may be short-sighted and generate more components, while the results with multi-step tree search are more reasonable. For example, when searching for the first cutting plane of the torus (first from left), either vertically or horizontally cutting can lead to sub-parts with the same concavity score, and the greedy algorithm will thus randomly select the first cutting plane. However, horizontal cuttings will lead to more components in the final results. Similarly, for the bottle cap example (first from right), the one-step greedy algorithm will not cut off the bottom in the first step because it will even increase the concavity score. However, by leveraging the multi-step tree search, we can find that cutting off the bottom in the first step can avoid the bottom being divided into unnecessary parts.

***Impact of the concavity threshold $\epsilon$.*** Our method terminates when the concavities of all decomposed components are less than a pre-defined threshold $\epsilon$. The concavity threshold $\epsilon$ thus balances the level of details and the number of decomposed components. As shown in Figure 13a and Figure 14, when we decrease the concavity threshold $\epsilon$, the algorithm generates more components to preserve the details and the generated convex hulls are much closer to the original shape. When we increase the concavity threshold $\epsilon$, we generate fewer components to approximate the global structure of the original shape and may lose some of the details. The variation is more significant when the concavity is relatively small.

We also want to point out that compared to the volume-based concavity, our proposed concavity metric measures the distance. One can interpret the threshold as the degree to which the original shape becomes thicker, which may be more intuitive for users to adjust the threshold and achieve their desired decomposition. In contrast, the volume-based concavity may not correspond to such an intuitive interpretation, and the change caused by adjusting the threshold may be less predictable.

***Impact of hyper-parameters in the tree search.*** We study the impact of the hyper-parameters in the multi-step tree search by fixing other hyper-parameters and the concavity threshold. The ablation results are shown in the Figure 13. (i) We sample $m$ candidate cutting planes from each axis-aligned direction. By sampling more

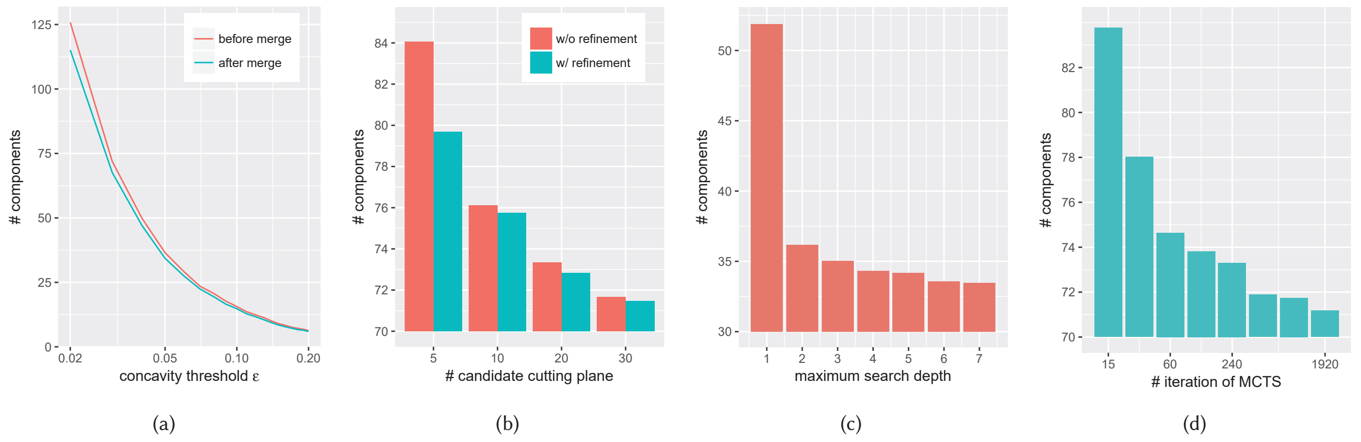


Fig. 13. Ablation studies: (a) the concavity threshold $\epsilon$ and the post-processing merge, (b) the number of sampled cutting plane candidates from each axis-aligned direction (i.e., $m$), (c) the maximum search depth $d$ in each MCTS, (d) the number of iterations in each MCTS (i.e., $t$). For all figures, the y-axis represents the number of decomposed component under each setting.

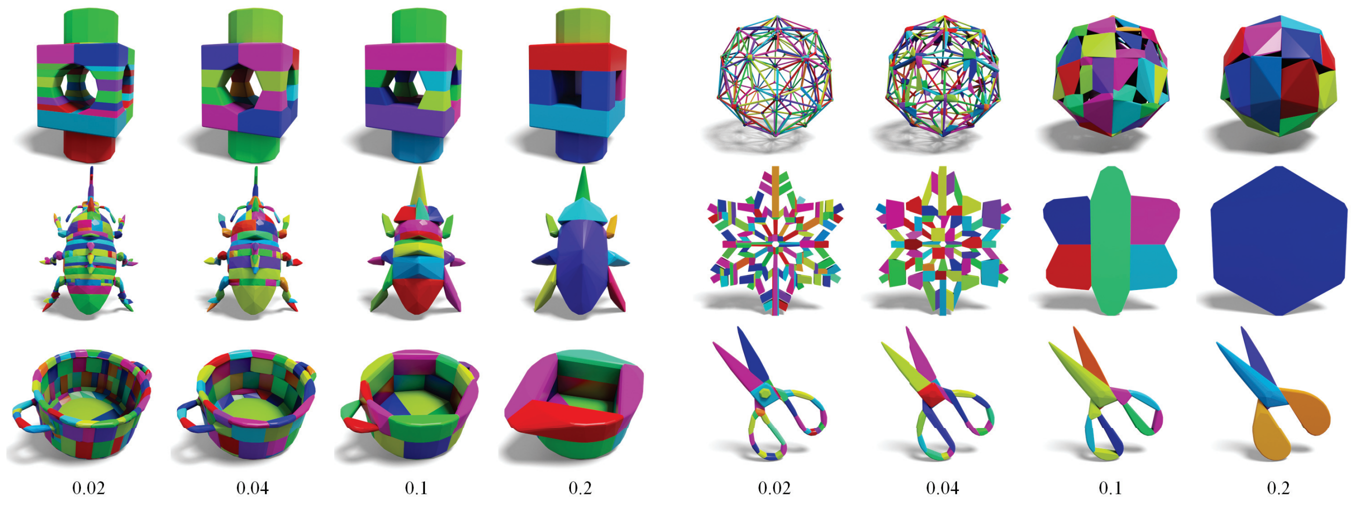


Fig. 14. Comparison of different concavity thresholds. For each example, we show the decomposition results under different concavity thresholds ranging from 0.02 to 0.2. Users can intuitively balance the level of detail and the number of components by adjusting the concavity threshold $\epsilon$.

candidate planes, we achieve more precise cuttings, which are much closer to the optimal location. As shown in Figure 13b, a larger $m$ thus leads to fewer components. (ii) We limit the maximum depth of the search tree to $d$ and evaluate each tree node by generating $d + 1$ components. A larger $d$ enables the algorithm to analyze cuttings in further steps and achieve a more precise tree node evaluation. As shown in Figure 13c, a larger $d$ leads to a better performance generally. Moreover, we find that seeing one step further (i.e., $d = 2$) introduces the most significant gain. (iii) As shown in Figure 13d, searching for more iterations leads to better solutions, since a larger number of iterations $t$ means expanding more nodes, exploring more cutting combinations, and more accurate evaluation for the tree nodes. However, increasing the three hyper-parameters causes a longer search time. As a result, there are trade-offs between the decomposition quality and the runtime.

***Refinement, Merging, and Cutting Directions.*** The cutting plane refinement aims to find a better position in the continuous local neighborhood of the searched discretized candidates, thus enabling a more precise cutting. As shown in Figure 13b, when the number of candidate cutting planes increases, the improvement brought by the refinement becomes smaller due to the narrower gap between two adjacent candidates.

After decomposition, we merge components as post-processing to further reduce the number of components. As shown in Figure 13a, when the concavity threshold is smaller, the shape is decomposed

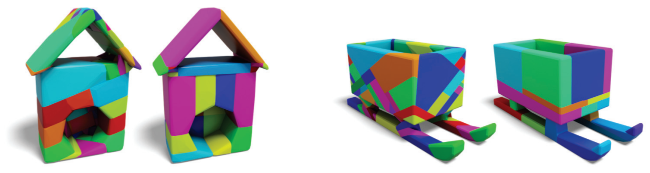


Fig. 15. Comparison of different cutting directions. In each pair, the left one indicates cutting with a set of random axes, while the right one indicates cutting with the principal axes computed by PCA.

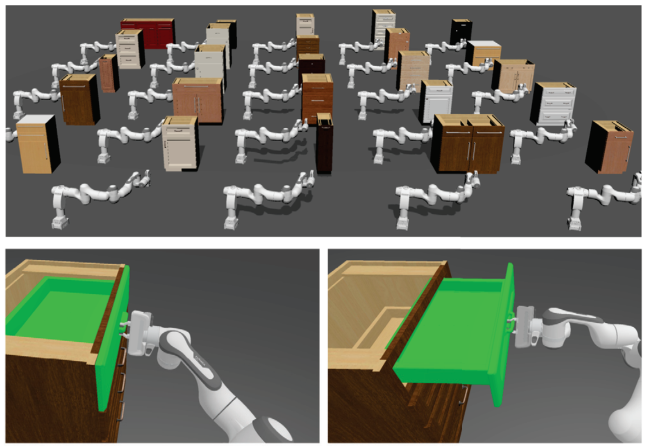


Fig. 16. Top: We train RL agents to open 49 drawers of 25 cabinets in a physics simulator. Bottom left: Using decomposition results of V-HACD as collision shapes. The collision shape of the drawer is highlighted in green, and the hole of the handle is filled (zoom in for details). Bottom right: Using our decomposition results as collision shapes. We preserve fine-grained details of the handle.

into more pieces, and the component merging can reduce more redundant divisions.

Since we sample cutting planes from three mutually orthogonal directions as V-HACD, the selection of the cutting directions may have a great influence on the final results in some cases, as shown in Figure 15. By specifying a set of good axes or computing principal axes by PCA, we may generate fewer components.

## 7.3 Application

An important application of convex decomposition is to provide collision shapes for physics simulators that perform extensive collision detection. On the one hand, we aim to approximate the shape with a small number of convex components, thereby speeding up the collision detection. On the other hand, we want the decomposed components to closely match the original shape, so that the functionality of the object is not compromised.

In this experiment, we compare two sets of collision shapes generated by our method and V-HACD. Specifically, we load 25 cabinets into SAPIEN [Xiang et al. 2020], a physics simulator. We utilize our method and V-HACD to generate a collision shape (i.e., an assembly of convex components) for each part (e.g., a drawer or a body), respectively. As shown in Figure 16, the collision shapes by our method preserve fine-grained details of the handles, while the collision shapes by V-HACD fill the holes of the handles even a tiny threshold is used.

Table 3. Results of the OpenCabinetDrawer task. We compare using different decomposition results as the collision shapes.

| | V-HACD | Ours |
|---|---|---|
| Successfully Opened Drawer | 49% | **80%** |

We train SAC [Haarnoja et al. 2018] (a reinforcement learning algorithm) agents to control a robot arm to open the drawers. Specifically, there are 49 drawers from the 25 cabinets. We train an individual SAC agent from scratch for each drawer with $10^6$ time steps per trial. Please refer to [Mu et al. 2021] for other training details. Since reinforcement learning algorithms are not guaranteed to converge to the optimum every run, if we open a drawer in 5 trials, we regard it as a success case. We report the result in Table 3.

By using more accurate collision shapes generated by our method, the RL agents achieve a much higher success rate. We observe that, when using our collision shapes, which preserve the fine-grained details (e.g., holes) of the handles, the robot arm is easier to form a shape-closure grasp, which is more robust. However, when using V-HACD's collision shapes, the robot arm easily slips off the handles, since they fill the holes.

## 8 DISCUSSION

We propose a novel approximate convex decomposition method that differs from prior approaches in three folds: (a) we introduce a novel collision-aware concavity metric that better examines the shapes from both the boundary and the interior. It preserves fine-grained structures of the input shape and enables delicate object interaction in downstream applications. (b) we decompose the shape by efficiently cutting the meshes. It ensures intersection-free components and avoids discretization artifacts. (c) we utilize multi-step tree search to find globally better cutting planes, leading to fewer decomposed components.

We currently adopt many simplifications due to the runtime consideration. In the future, we would like to optimize our implementation further (e.g., utilize parallelization). We may also employ deep neural networks to help evaluate a decomposition solution more efficiently and accurately. Moreover, we can explore a smarter way to pick the cutting directions and set adaptive thresholds for different parts to reduce the number of decomposed components.

## ACKNOWLEDGMENTS

This work is supported in part by gifts from Adobe, Qualcomm, and Kingstar. We would like to thank Lubor Ladicky and Daniel Thul for running the Animation code, thank Abdulaziz Almuzairee, Xiaoshuai Zhang, Jiayuan Gu, Fanbo Xiang, and Songfang Han for proofreading the manuscript.

## REFERENCES

Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.

Marco Attene, Michela Mortara, Michela Spagnuolo, and Bianca Falcidieno. 2008. Hierarchical convex approximation of 3D shapes for fast region selection. In *Computer graphics forum*, Vol. 27. Wiley Online Library, 1323–1332.

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.

Gino van den Bergen. 1999. A fast and robust GJK implementation for collision detection of convex objects. *Journal of graphics tools* 4, 2 (1999), 7–25.

Jose Luis Blanco and Pranjal Kumar Rai. 2014. nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees. https://github.com/jlblancoc/nanoflann.

Gareth Bradshaw and Carol O'Sullivan. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)* 23, 1 (2004), 1–26.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding proxies for shape approximation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Bernard Chazelle. 1984. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.* 13, 3 (1984), 488–507.

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

Bernard M Chazelle. 1981. Convex decompositions of polyhedra. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. 70–79.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 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/CVF Conference on Computer Vision and Pattern Recognition*. 31–44.

Andreas Fabri and Sylvain Pion. 2009. CGAL: The computational geometry algorithms library. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. 538–539.

Matheus Gadelha, Giorgio Gori, Duygu Ceylan, Radomir Mech, Nathan Carr, Tamy Boubekeur, Rui Wang, and Subhransu Maji. 2020. Learning generative models of shape handles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 402–411.

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

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

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

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).

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

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

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. 2019. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2652–2660.

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 2007 ACM symposium on Solid and physical modeling*. 121–131.

Jyh-Ming Lien and Nancy M Amato. 2008. Approximate convex decomposition of polyhedra and its applications. *Computer Aided Geometric Design* 25, 7 (2008), 503–522.

Jyh-Ming Lien, John Keyser, and Nancy M Amato. 2006. Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*. 219–228.

Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. 2016. Nearly convex segmentation of polyhedra through convex ridge separation. *Computer-Aided Design* 78 (2016), 137–146.

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

Minghua Liu, Minhyuk Sung, Radomir Mech, and Hao Su. 2021. DeepMetaHandles: Learning Deformation Meta-Handles of 3D Meshes with Biharmonic Coordinates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12–21.

Rong Liu, Hao Zhang, and James Busby. 2008. Convex hull covering of polygonal scenes for accurate collision detection in games.. In *Graphics Interface*. 203–210.

Khaled Mamou and Faouzi Ghorbel. 2009. A simple and efficient approach for 3D mesh approximate convex decomposition. In *2009 16th IEEE international conference on image processing (ICIP)*. IEEE, 3501–3504.

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

Brian Mirtich. 1998. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)* 17, 3 (1998), 177–208.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. 2019. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575* (2019).

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. 2021. ManiSkill: Learning-from-Demonstrations Benchmark for Generalizable Manipulation Skills. *arXiv e-prints* (2021), arXiv–2107.

Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2013. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

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

Despoina Paschalidou, Luc Van Gool, and Andreas Geiger. 2020. Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1060–1070.

Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. 2019. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10344–10353.

Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.

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

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Workshop on Applied Computational Geometry*. Springer, 203–222.

Dmitriy Smirnov, Matthew Fisher, Vladimir G Kim, Richard Zhang, and Justin Solomon. 2020. Deep parametric shape predictions using distance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 561–570.

Jack Snoeyink. 2017. Point location. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1005–1028.

Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. 2019. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Jean-Marc Thiery, Émilie Guy, and Tamy Boubekeur. 2013. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–12.

Daniel Thul, L'ubor Ladickỳ, Sohyeon Jeong, and Marc Pollefeys. 2018. Approximate convex decomposition and transfer for animated meshes. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–10.

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. 2017. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2635–2643.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–11.

René Weller. 2013. A brief overview of collision detection. *New Geometric Data Structures for Collision Detection and Haptics* (2013), 9–46.

Martin Wicke, Mario Botsch, and Markus Gross. 2007. A finite element method on convex polyhedra. In *Computer Graphics Forum*, Vol. 26. Wiley Online Library, 355–364.

Chuhua Xian, Hongwei Lin, and Shuming Gao. 2012. Automatic cage generation by improved obbs for mesh deformation. *The Visual Computer* 28, 1 (2012), 21–33.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. 2020. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11097–11107.

Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or. 2015. Generalized cylinder decomposition. *ACM Trans. Graph.* 34, 6 (2015), 171–1.

Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 2017. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 900–909.

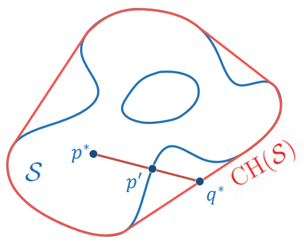


Fig. 17. Counter example of Case 2. $p^*$ must be on the boundary surface of $\mathcal{S}$.

## A PROOF FOR THEOREM 1

In the paper, we propose a surrogate term $\mathrm{R_v}(\mathcal{S})$ to accelerate the computation of $\mathrm{H_i}(\mathcal{S})$ and provide a theoretical guarantee:

Theorem 2. *For every solid shape $\mathcal{S}$, we have*

$$\sqrt{2}\max(\mathrm{H_b}(\mathcal{S}),\mathrm{R_v}(\mathcal{S})) \geq \max(\mathrm{H_b}(\mathcal{S}),\mathrm{H_i}(\mathcal{S}))$$

Here we give the detailed proof for the theorem. Recall that the Hausdorff distance for two point sets $A$ and $B$ is calculated as:

$$\mathrm{H}(A,B) = \max\{\sup_{a\in A} d(a,B), \sup_{b\in B} d(b,A)\} \tag{11}$$

where $d(x,Y) = \inf_{y\in Y} d(x,y)$ and $d(x,y)$ indicates the Euclidean distance between the two points.

When calculating $\mathrm{H_i}(\mathcal{S})$, the two point sets are sampled from the interior of the solid shape $\mathcal{S}$ and its convex hull $\mathrm{CH}(\mathcal{S})$. We denote them as $P$ and $Q$, respectively:

$$P = \mathrm{Sample}(\mathrm{Int}\,\mathcal{S}) \tag{12}$$

$$Q = \mathrm{Sample}(\mathrm{Int}\,\mathrm{CH}(\mathcal{S})) \tag{13}$$

In our proof, we assume that the interior doesn't exclude the boundary surface, which is slightly different from the usual definition. Also, we assume that Sample($T$) cover all points in $T$ (infinite sampled points).

Since $\mathcal{S}$ is contained by $\mathrm{CH}(\mathcal{S})$, $d(p,Q) = 0$ for all $p \in P$. We can thus simplify $\mathrm{H_i}(\mathcal{S})$ as:

$$\mathrm{H_i}(\mathcal{S}) = \sup_{q\in Q} d(q,P) \tag{14}$$

We know that there exists a pair of points $p^* \in P$ and $q^* \in Q$, such that $\mathrm{H_i}(\mathcal{S}) = d(q^*,P) = d(p^*,q^*)$. We prove the theorem by enumerating all possible locations of $p^*$ and $q^*$, which can be divided into four cases.

**Case 1: $q^*$ lies inside of $\mathcal{S}$.**

In this case, $\mathrm{H_i}(\mathcal{S}) = d(q^*,P) = 0$, and the theorem holds.

**Case 2: $p^*$ is not on the boundary surface of $\mathcal{S}$.**

This case is impossible. Since $q^*$ lies outside of $\mathcal{S}$ (not Case 1), there must exist another point $p'$ on the boundary surface of $\mathcal{S}$, such that $d(q^*,p') < d(q^*,p^*)$, which contradicts $d(q^*,P) = d(q^*,p^*)$. See Figure 17 for a illustration.

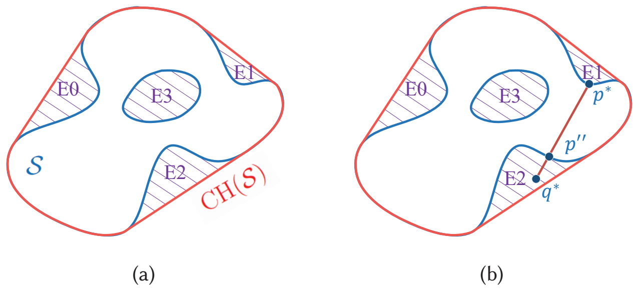


Fig. 18. Illustration of the interval space. Blue lines indicate the solid shape $\mathcal{S}$, and the red lines indicate its convex hull $\mathrm{CH}(\mathcal{S})$. (a) The shaded area shows the interval space, which consists of four connected regions $E_i$. (b) If $p^*$ and $q^*$ lie in different $E_i$, the segment connecting $p^*$ and $q^*$ will intersect with the boundary surface of $\mathcal{S}$ at another point $p''$, and $d(p'',q^*) < d(p^*,q^*)$.

**Case 3: $p^*$ is on the boundary surface of $\mathcal{S}$ and $q^*$ is on the boundary surface of $\mathrm{CH}(\mathcal{S})$.**

In this case, we have $\mathrm{H_i}(\mathcal{S}) = \mathrm{H_b}(\mathcal{S})$, and the theorem holds.

**Case 4: $p^*$ is on the boundary surface of $\mathcal{S}$ and $q^*$ is not on the boundary surface of $\mathrm{CH}(\mathcal{S})$.**

$q^*$ must lie within $\mathrm{CH}(\mathcal{S}) - \mathcal{S}$, the space outside of $\mathcal{S}$ but inside of $\mathrm{CH}(\mathcal{S})$. As shown in Figure 18a, we call the space between $\mathcal{S}$ and $\mathrm{CH}(\mathcal{S})$ as the **interval space**, which may consists of multiple connected regions $E_i$. We denote the connected region containing both $p^*$, $q^*$ as $E_*$.

Note that $(p^*,q^*)$ cannot locate on different $E_i$. Otherwise, as shown in Figure 18b, there must exist another point $p''$ on the boundary surface of $\mathcal{S}$, such that $d(q^*,p'') < d(q^*,p^*)$, which contradicts $d(q^*,P) = d(q^*,p^*)$.

Before we talk about the connection between $\mathrm{R_v}(\mathcal{S})$ and $\mathrm{H_i}(\mathcal{S})$, we first construct a **maximum inscribed sphere** within $E_*$ centered at $q^*$. We denote this sphere as $\Phi$ and its radius as $r$. We know that:

$$\mathrm{Vol}(\mathrm{CH}(\mathcal{S})) - \mathrm{Vol}(\mathcal{S}) \geq \mathrm{Vol}(E_*) \geq \mathrm{Vol}(\Phi) \tag{15}$$

Combined with the definition of $\mathrm{R_v}(\mathcal{S})$, we can infer that $\mathrm{R_v}(\mathcal{S}) \geq r$.

**Case 4.1: $\Phi$ does not intersect with the boundary surface of $\mathrm{CH}(\mathcal{S})$.**

In this case, $\Phi$ must intersect with the boundary surface of $\mathcal{S}$ at $p^*$. Otherwise, there exist another point $p'''$ on the boundary surface of $\mathcal{S}$, such that $d(q^*,p''') < d(q^*,p^*)$, which contradicts $d(q^*,P) = d(q^*,p^*)$. As a result, $d(q^*,p^*)$ is equal to the radius $r$. Since $\mathrm{R_v}(\mathcal{S}) \geq r$, we have:

$$r = d(p^*,q^*) = \mathrm{H_i}(\mathcal{S}) \leq \mathrm{R_v}(\mathcal{S}) \tag{16}$$

The theorem holds.

**Case 4.2: $\Phi$ intersects with the boundary surface of $\mathrm{CH}(\mathcal{S})$.**

In this case, $\mathrm{H_i}(\mathcal{S})$ does not equal to the radius $r$, since the maximum inscribed sphere $\Phi$ is bounded by the boundary surface of

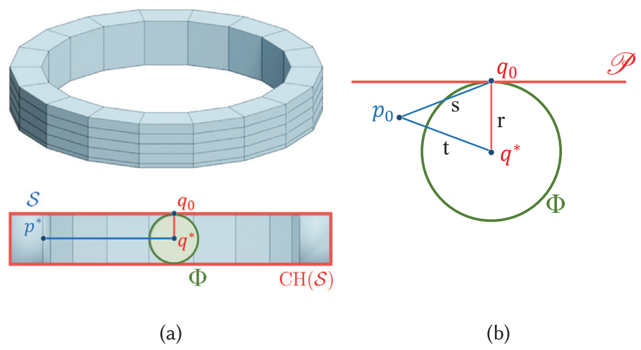


Fig. 19. (a) An example that the maximum inscribed sphere $\Phi$ is bounded by the boundary surface of $\mathrm{CH}(\mathcal{S})$ and $\mathrm{H_i}(\mathcal{S})$ does not equal to the radius $r$. (b) Illustration of the proof in Case 4.2.

Table 4. Quantitative comparison on the PartNet-Mobility dataset [Xiang et al. 2020]. "Better ratio" indicates the percentage of cases where our method outperforms the baseline.

| | HACD [Mamou and Ghorbel 2009] | | | | V-HACD [Mamou et al. 2016] | | | |
|---|---|---|---|---|---|---|---|---|
| | **# components** ↓ | | concavity ↓ | | **# components** ↓ | | concavity ↓ | |
| | theirs | ours | theirs | ours | theirs | ours | theirs | ours |
| average | 33.5 | **7.3** | 0.414 | **0.204** | 44.6 | **20.1** | 0.055 | **0.052** |
| median | 27 | **1** | 0.218 | **0.117** | 20 | **13** | 0.045 | **0.041** |
| better ratio | - | 90.85% | - | 89.18% | - | 96.50% | - | 96.90% |

CH($\mathcal{S}$) and it may fail to touch any point on the boundary surface of $\mathcal{S}$. Figure 19a shows such an example.

As shown in Figure 19b, we denote one of the intersection point as $q_0$. We know that $q_0$ is on the boundary surface of CH($\mathcal{S}$) and we have $d(q^*, q_0) = r$. We also find $q_0$'s nearest point on the boundary surface of $\mathcal{S}$ and denote it as $p_0$. We denote $d(p_0, q_0)$ as $s$ in the figure, and we know that $s \leq \mathrm{H_b}(\mathcal{S})$. We want to calculate the distance between $p_0$ and $q^*$, which is denoted as $t$ in the figure. To this end, we find the tangent plane of CH($\mathcal{S}$) at $q_0$, which is denoted as $\mathscr{P}$ in the figure. Due to the property of convex hulls, $q^*$, $q_0$, and $p_0$ should lie in the same side of the plane $\mathscr{P}$. Therefore, within $\triangle q^* q_0 p_0$, $\angle q^* q_0 p_0 \leq 90^\circ$, and we thus have:

$$r^2 + s^2 \geq t^2 \tag{17}$$

Since $r \leq \mathrm{R_v}(\mathcal{S})$ and $s \leq \mathrm{H_b}(\mathcal{S})$, we have:

$$t \leq \sqrt{r^2 + s^2} \leq \sqrt{\mathrm{R_v}(\mathcal{S})^2 + \mathrm{H_b}(\mathcal{S})^2} \leq \sqrt{2}\max(\mathrm{R_v}(\mathcal{S}), \mathrm{H_b}(\mathcal{S})) \tag{18}$$

Moreover, since $p^*$ is $q^*$'s nearest point on the boundary surface of $\mathcal{S}$, we know that $\mathrm{H_i}(\mathcal{S}) = d(p^*, q^*) \leq t$. The theorem thus holds.

## B DETAILED VERSION OF TABLE 1

We show the complete quantitative comparison (for all objects) of the V-HACD dataset [Mamou et al. 2016] in Table 5. Since the PartNet-Mobility dataset [Xiang et al. 2020] contains thousands of objects, we only report insightful statistics in Table 4. We compare each baseline algorithm with our method separately. For both HACD [Mamou and Ghorbel 2009] and V-HACD [Mamou et al. 2016], we let our method produce decomposition results with lower concavity scores and aim to compare the numbers of decomposed components. For Animation [Thul et al. 2018], we aim to match the numbers of decomposed components and compare the concavity scores. In addition to the average and median, we also calculate the percentage of cases where our method outperforms the baseline (denoted as "better ratio").

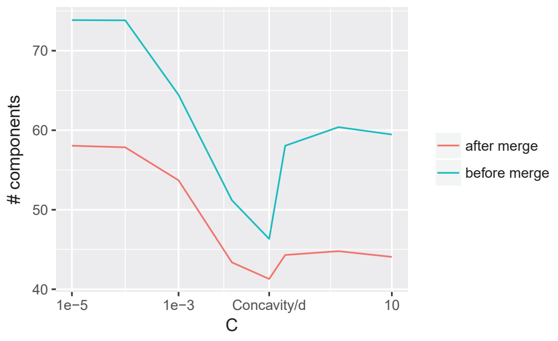


Fig. 20. The influence of the exploration parameter $c$.

## C DEFAULT VALUE OF HYPER-PARAMETERS

The default value of the hyper-parameters are $m = 20$, $t = 500$, $d = 4$, $k = 0.3$. We sample 3,000 points per unit area when computing $\mathrm{H_b}$. The hyper-parameters are consistent across datasets and experiments except for ablating the hyper-parameter. In experiments, all of our methods include the merging stage unless otherwise noted.

## D ABLATION STUDY OF THE EXPLORATION PARAMETER $c$

In the tree search, we use the UCB (Upper Confidence Bound) [Kocsis and Szepesvári 2006] term to select a tree node for expansion:

$$\mathrm{Q}(n) + c\sqrt{\frac{2\ln \mathrm{N}(n')}{\mathrm{N}(n)}} \tag{19}$$

where $c$ is the parameter balancing the exploration and exploitation. We study the influence of $c$ on the V-HACD dataset and report the results in Figure 20. As shown in the figure, when $c$ is set to 0, the UCB term only uses the existing value function $Q(n)$ to select a node, and no exploration occurs. In this case, MCTS almost degenerates into a one-step greedy search, and the resulting number of components increases a lot. In contrast, when $c$ is set to a large number, the UCB term ignores the influence of the quality function, and the MCTS degrades to an inefficient exhaustive search algorithm, which also leads to sub-optimal results. Instead, we empirically set $c$ to be $\widetilde{\mathrm{Concavity}}(\mathcal{S})/d$, where $\widetilde{\mathrm{Concavity}}(\mathcal{S})$ is the concavity score of each individual input component, and $d$ is the depth of the tree search. We find that it generates good results in general.

## E HOW WELL DOES $\mathrm{R_v}$ APPROXIMATES $\mathrm{H_i}$?

In addition to the theoretical proof, we experimentally verify that Theorem 1 holds for all shapes on both datasets. We empirically set

Table 5. Quantitative comparison on the V-HACD dataset [Mamou et al. 2016]. "Better ratio" indicates the percentage of cases where our method outperforms the baseline.

| | HACD [Mamou and Ghorbel 2009] | | | | V-HACD [Mamou et al. 2016] | | | | Animation [Thul et al. 2018] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | # **components** ↓ | | concavity ↓ | | # **components** ↓ | | concavity ↓ | | # components ↓ | | **concavity** ↓ | |
| | theirs | ours | theirs | ours | theirs | ours | theirs | ours | theirs | ours | theirs | ours |
| block | 52 | **2** | **0.307** | 0.316 | 19 | **18** | 0.043 | **0.030** | 18 | 18 | 0.066 | **0.035** |
| bunny | 58 | **53** | 0.050 | **0.033** | 21 | **12** | 0.100 | **0.078** | 52 | 52 | 0.083 | **0.051** |
| camel | 64 | **39** | 0.039 | **0.023** | 42 | **21** | 0.077 | **0.050** | 35 | 35 | 0.061 | **0.050** |
| casting | 86 | **4** | 0.312 | **0.267** | 59 | **52** | 0.064 | **0.037** | 68 | 69 | 0.066 | **0.056** |
| chair | 30 | **6** | 0.183 | **0.091** | 30 | **23** | 0.026 | **0.017** | 13 | 13 | 0.055 | **0.045** |
| cow1 | 66 | **45** | 0.038 | **0.022** | 33 | **20** | 0.062 | **0.045** | 27 | 27 | 0.059 | **0.049** |
| cow2 | **54** | 68 | 0.021 | **0.015** | 29 | **25** | 0.054 | **0.029** | 25 | 25 | **0.041** | 0.047 |
| crank | 90 | **12** | 0.190 | **0.097** | 114 | **12** | 0.187 | **0.097** | 80 | 80 | 0.186 | **0.050** |
| cup | 65 | **10** | 0.135 | **0.083** | 38 | **23** | 0.075 | **0.058** | 46 | 47 | 0.315 | **0.054** |
| dancer2 | **34** | 51 | 0.015 | **0.012** | 49 | **25** | 0.024 | **0.017** | 7 | 7 | 0.040 | **0.039** |
| deer_bound | 55 | **44** | 0.040 | **0.021** | 69 | **44** | 0.037 | **0.021** | 35 | 35 | **0.044** | 0.047 |
| dilo | **42** | 45 | 0.016 | **0.011** | 35 | **29** | 0.027 | **0.015** | 15 | 15 | 0.059 | **0.051** |
| dino | **51** | 70 | 0.019 | **0.013** | 32 | **30** | 0.048 | **0.024** | 25 | 25 | **0.039** | 0.053 |
| DRAGON_F | 90 | **29** | 0.085 | **0.052** | 76 | **42** | 0.064 | **0.040** | 52 | 52 | 0.061 | **0.058** |
| drum | 17 | **12** | 0.059 | **0.035** | 6 | **5** | 0.100 | **0.054** | 16 | 16 | 0.068 | **0.047** |
| egea | **34** | 52 | 0.032 | **0.023** | **6** | **6** | 0.102 | **0.074** | 26 | 26 | **0.048** | 0.050 |
| eight | 42 | **37** | 0.019 | **0.014** | 26 | **18** | 0.033 | **0.024** | 17 | 17 | 0.041 | **0.040** |
| elephant | 84 | **52** | 0.043 | **0.029** | 62 | **43** | 0.053 | **0.034** | 45 | 45 | 0.060 | **0.051** |
| elk | **42** | 46 | 0.052 | **0.032** | 40 | **28** | 0.098 | **0.058** | 52 | 52 | **0.045** | 0.050 |
| face-YH | 125 | **10** | 0.240 | **0.160** | 240 | **113** | 0.039 | **0.027** | 82 | 82 | 0.067 | **0.050** |
| feline | 91 | **27** | 0.086 | **0.054** | 87 | **29** | 0.079 | **0.051** | 54 | 54 | **0.051** | 0.052 |
| fish | 26 | **8** | 0.081 | **0.047** | 17 | **10** | 0.072 | **0.042** | 13 | 13 | **0.044** | 0.045 |
| foot | **30** | 32 | 0.018 | **0.015** | **6** | **6** | 0.040 | **0.034** | 5 | 5 | **0.041** | 0.050 |
| genus3 | 53 | **31** | 0.034 | **0.023** | 29 | **16** | 0.064 | **0.046** | 23 | 23 | 0.054 | **0.048** |
| greek_sculpture | 83 | **71** | 0.029 | **0.021** | 47 | **19** | 0.059 | **0.047** | 30 | 30 | 0.054 | **0.050** |
| Hand1 | 49 | **41** | 0.035 | **0.024** | 27 | **16** | 0.078 | **0.051** | 26 | 26 | 0.061 | **0.048** |
| hand2 | 49 | **41** | 0.038 | **0.024** | 27 | **17** | 0.077 | **0.048** | 25 | 25 | 0.060 | **0.047** |
| helix | 37 | **36** | 0.016 | **0.013** | **32** | **32** | 0.027 | **0.016** | 21 | 21 | **0.040** | 0.047 |
| helmet | 28 | **3** | 0.211 | **0.090** | 7 | **5** | 0.103 | **0.082** | 10 | 10 | **0.068** | 0.087 |
| hero | 118 | **13** | 0.144 | **0.097** | 228 | **51** | 0.060 | **0.044** | 78 | 78 | 0.090 | **0.052** |
| homer | **61** | 66 | 0.020 | **0.014** | 23 | **17** | 0.044 | **0.030** | 16 | 16 | 0.062 | **0.047** |
| hornbug | 114 | **51** | 0.059 | **0.038** | 120 | **38** | 0.076 | **0.048** | 67 | 67 | 0.059 | **0.050** |
| horse | 55 | **45** | 0.033 | **0.020** | 32 | **22** | 0.055 | **0.037** | 24 | 24 | 0.061 | **0.046** |
| maneki-neko | 130 | **1** | 0.394 | **0.299** | 516 | **278** | 0.038 | **0.027** | 190 | 191 | 0.096 | **0.051** |
| mannequin-devil | 50 | **13** | 0.085 | **0.062** | 8 | **3** | 0.190 | **0.116** | 36 | 36 | 0.100 | **0.051** |
| mannequin | **42** | **42** | 0.037 | **0.022** | 9 | **8** | 0.101 | **0.062** | 23 | 23 | 0.062 | **0.052** |
| mask | 112 | **6** | 0.259 | **0.178** | 183 | **77** | 0.036 | **0.026** | 50 | 50 | 0.069 | **0.060** |
| moaimoai | **53** | 63 | 0.022 | **0.016** | 8 | **6** | 0.083 | **0.066** | 17 | 17 | 0.103 | **0.048** |
| monk | 75 | **55** | 0.036 | **0.024** | 23 | **7** | 0.108 | **0.078** | 29 | 29 | 0.095 | **0.051** |
| octopus | 81 | **48** | 0.058 | **0.032** | 96 | **65** | 0.041 | **0.023** | 54 | 54 | 0.065 | **0.053** |
| pig | **46** | 60 | 0.022 | **0.016** | 13 | **11** | 0.074 | **0.053** | 18 | 18 | **0.046** | 0.050 |
| pinocchio_b | 150 | **4** | 0.319 | **0.249** | 330 | **140** | 0.048 | **0.032** | 132 | 132 | 0.069 | **0.050** |
| polygirl | 55 | **15** | 0.063 | **0.043** | 30 | **17** | 0.059 | **0.039** | 23 | 23 | 0.071 | **0.060** |
| rabbit | 36 | **5** | 0.118 | **0.075** | 11 | **7** | 0.076 | **0.055** | 15 | 15 | **0.048** | 0.050 |
| rocker-arm | 65 | **26** | 0.062 | **0.033** | 51 | **23** | 0.069 | **0.037** | 29 | 29 | 0.074 | **0.050** |
| screwdriver | 48 | **38** | 0.025 | **0.016** | 27 | **26** | 0.035 | **0.022** | 18 | 18 | **0.038** | 0.046 |
| shark_b | 84 | **7** | 0.098 | **0.056** | 336 | **80** | 0.013 | **0.010** | 16 | 16 | **0.040** | 0.052 |
| Sketched-Brunnen | 101 | **20** | 0.145 | **0.090** | 110 | **67** | 0.047 | **0.033** | 65 | 65 | 0.062 | **0.050** |
| sledge | 30 | **2** | 0.277 | **0.238** | 24 | **18** | 0.060 | **0.021** | 18 | 18 | 0.034 | **0.031** |
| squirrel | **44** | **44** | 0.048 | **0.034** | 14 | **10** | 0.113 | **0.079** | 46 | 46 | 0.058 | **0.053** |
| sword | **15** | 67 | 0.025 | **0.016** | 32 | **9** | 0.055 | **0.040** | 14 | 14 | **0.030** | 0.049 |
| table | **5** | 15 | 0.012 | **0.007** | **6** | 9 | 0.039 | **0.014** | 7 | 7 | 0.329 | **0.048** |
| Teapot | 83 | **7** | 0.240 | **0.216** | 61 | **23** | 0.152 | **0.091** | 63 | 63 | **0.085** | 0.097 |
| test2 | 27 | **1** | 0.651 | **0.521** | 15 | **13** | 0.058 | **0.043** | 21 | 21 | 0.075 | **0.046** |
| test | 8 | **1** | **0.534** | **0.534** | **2** | **2** | 0.058 | **0.029** | 3 | 3 | **0.011** | 0.013 |
| torus | 34 | **4** | 0.227 | **0.132** | 11 | **9** | 0.055 | **0.038** | 12 | 12 | **0.037** | 0.048 |
| tstTorusModel3 | 36 | **4** | 0.250 | **0.179** | **16** | **16** | 0.037 | **0.024** | 12 | 12 | **0.042** | **0.042** |
| tstTorusModel | 34 | **4** | 0.226 | **0.132** | 11 | **9** | 0.055 | **0.042** | 11 | 11 | **0.042** | 0.047 |
| tube1 | 8 | **4** | 0.223 | **0.031** | **4** | **4** | 0.050 | **0.031** | 4 | 4 | 0.042 | **0.032** |
| venus-original | **42** | 44 | 0.036 | **0.026** | 7 | **6** | 0.100 | **0.072** | 27 | 27 | 0.056 | **0.050** |
| venus | **46** | 52 | 0.024 | **0.017** | 13 | **10** | 0.075 | **0.050** | 20 | 20 | 0.056 | **0.048** |
| average | 57.6 | **29.6** | 0.118 | **0.084** | 60.2 | **29.8** | 0.067 | **0.044** | 34.4 | 34.5 | 0.069 | **0.049** |
| median | 51 | **31** | 0.058 | **0.033** | 29 | **18** | 0.059 | **0.040** | 25 | 25 | 0.059 | **0.050** |
| better ratio | - | 77.05% | - | 98.36% | - | 98.36% | - | 100.00% | - | 95.10% | - | 81.97% |

$k$ to 0.3 by calculating the average $\frac{\mathrm{H_i}}{\mathrm{R_v}}$. We find that $\max(\mathrm{H_b}, k\,\mathrm{R_v})$ approximates $\max(\mathrm{H_b}, \mathrm{H_i})$ well in practice, with a small absolute or relative error. Specifically, we compare two metrics on the PartNetM dataset (13,536 shapes), and find that for more than 94% shapes, we have: $|\max(\mathrm{H_b}, k\,\mathrm{R_v}) - \max(\mathrm{H_b}, \mathrm{H_i})| \leq 0.02$ or $0.8 \leq \max(\mathrm{H_b}, k\,\mathrm{R_v}) / \max(\mathrm{H_b}, \mathrm{H_i}) \leq 1.2$.