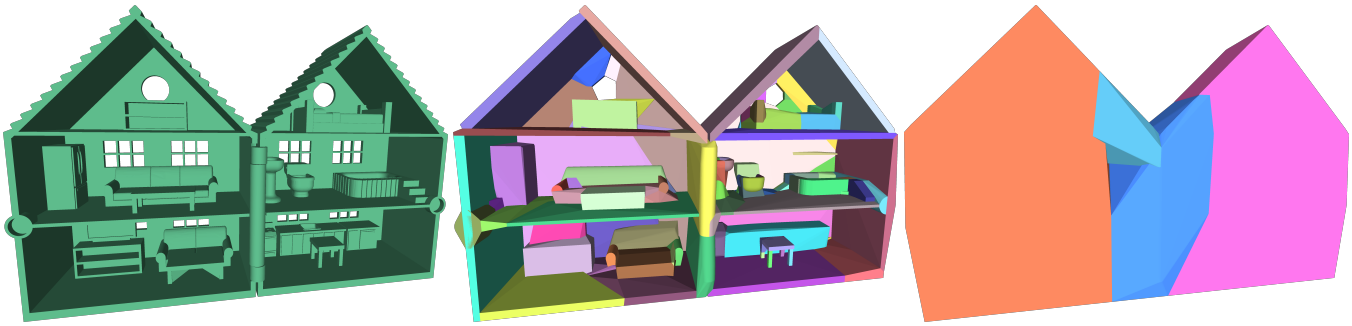
Navigation-Driven Approximate Convex Decomposition

James Andrews
jimmy.andrews@epicgames.com
Epic Games
Cary, NC, United States



(a) A furnished dollhouse (b) 102 collision shapes for small characters (c) 4 collision shapes for large characters

Figure 1: A shape like this furnished dollhouse (a) has different collision accuracy requirements depending on the expected scale of characters or objects interacting with it. We find the *navigable space* where characters of a given size could actually fit, and use that space to generate an appropriate collision representation – e.g., (b) a 102-part approximate convex decomposition, appropriate for characters as small as 2% of the house's width (computed in 5.6 seconds), or (c) a 4-part decomposition for characters at least as wide as the house (computed in 0.9 seconds).

ABSTRACT

Approximate convex decomposition – approximating a shape by a set of convex hulls – is a popular approach to creating efficient collision representations for games and simulations. Existing algorithms to construct such decompositions are typically driven by general surface- or volume-based error metrics that can't ignore unreachable internal surfaces nor provide local control over the results. We introduce the problem of *navigable* approximate convex decomposition: First, define a *navigable space* for the input shape which other objects in the game or simulation must be able to move through, then find a decomposition which does not overlap that space. We show how to automatically find such navigable space, how to customize it, and we introduce an approximate convex decomposition algorithm that protects it. Our results demonstrate that this approach can generate decompositions that meet application requirements faster and with fewer convex hulls than previous methods, while providing a new level of flexibility in defining what those requirements are.

SIGGRAPH Conference Papers '24, July 27–August 01, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07
https://doi.org/10.1145/3641519.3657479

CCS CONCEPTS

• **Computing methodologies** → **Shape analysis**.

KEYWORDS

Convex Hull, Convex Decomposition, Collision, Geometry

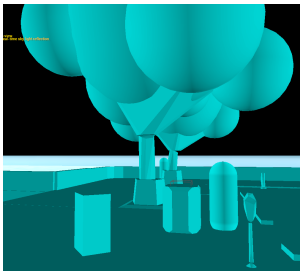
ACM Reference Format:
James Andrews. 2024. Navigation-Driven Approximate Convex Decomposition. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27–August 01, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3641519.3657479

1 INTRODUCTION

Video games and real-time physically-based simulations often use a simplified collision representation for the objects that they need to simulate or collide against: a virtual car, for example, may be represented by thousands of triangles for rendering, but only four cylinders and a box for collision. Especially in games, this is a ubiquitous industry standard: Almost every character, vehicle and environment object in a modern AAA game will have associated simple collision shapes like those visualized in Fig. 2b. These collision representations are often carefully fine-tuned by artists to use the fewest, simplest, collision primitives possible while still remaining accurate enough for the game or simulation to work as expected.

In addition to spheres, boxes and capsules, convex hulls are a popular choice for collision representation in these performance-critical applications: They are expressive in the range of shapes

(a) Sample 'game-ready' assets



(b) Their collision shapes

Figure 2: (a) Sample game assets from Unreal Engine for Fortnite, by Epic Games, Inc, used with permission. (b) Their corresponding collision shapes, demonstrating industry best-practices: Shapes are coarsely approximated by spheres, boxes, capsules and convex hulls.

they can represent, while still being relatively efficient to query for collision thanks to methods such as GJK [den Bergen 1999].

Approximate convex decomposition – the problem of automatically building such a collision representation with convex hulls – is a well-studied problem [Lien and Amato 2004, 2007; Mamou 2016; Mamou and Ghorbel 2009; Thul et al. 2018; Wei et al. 2022], and the V-HACD [Mamou 2016] method is directly supported by popular game engines as a collision generation method [Epic Games 2023].

The question of whether a given approximate convex decomposition is 'good enough' for a given simulation or game is application-specific, and artists will often manually tune the parameters of an automatic convex decomposition algorithm for each object that needs collision shapes. The criteria for this tuning is a mix of aesthetics (e.g., the same level of inaccuracy may look worse on a floor, where it makes the character appear to float, compared to a ceiling) and functional requirements (e.g., characters must be able to pass through a doorway). The parameters of existing approximate convex decomposition algorithms can only indirectly address these criteria, by controlling the number of generated parts, or the maximum allowed difference between the shape and the decomposition according to some 'concavity' metric (typically based on volume differences and/or surface distances).

Our contributions in this work are:

- We introduce the concept of *navigable space* to automatic collision shape generation, to directly capture the application-specific acceptability criteria for our collision shapes.
- We introduce a method to automatically find navigable space for characters of a given size, and also show how to locally customize that space.
- We show how our navigable space representation can be used to guide an approximate convex decomposition algorithm to automatically generate reliable, functional, and efficient collision representations with flexible support for locally-targeted art direction where needed.

2 RELATED WORK

2.1 Approximate Convex Decomposition

The many previous approaches to approximate convex decomposition can be largely divided by three considerations: (1) the 'concavity' metric used to decide when an approximate decomposition is good enough, (2) the method to choose where to split parts that are not good enough and/or which smaller parts to merge together, and (3) whether to perform preprocessing or discretization of the input.

2.1.1 Concavity metrics. Previous concavity metrics take some measure of the difference between the mesh surface and the convex hull, but there is a wide variety of approaches to do so.

Surface-based metrics attempt to characterize the difference between samples on the mesh surface and the hull, such as the difference along the mesh normal direction [Mamou and Ghorbel 2009], or the distance of a path from the surface to the hull [Lien and Amato 2004]. These metrics are often unable to see non-local error, which can cause the method of Mamou and Ghorbel [2009] to cover the opening of a cup with hulls generated from the cup's outer surface.

Volume-based methods [Mamou 2016; Thul et al. 2018] instead compare the volume of parts of the mesh to the volumes of the corresponding convex hulls; this is much better at avoiding non-local error, but can miss large thin features.

A **hybrid** method of Wei et al. [2022] introduced a metric based on both the Hausdorff distance and (re-scaled) volume differences, allowing both non-local and thin features to be better captured. However, it cannot ignore *unreachable* surface detail – like the doll furniture in Fig. 1.

2.1.2 Spitting and merging. Strategies to split too-concave parts often work by proposing a set of cutting planes and choosing the plane that best minimizes their concavity metric. Mamou [2016]; Wei et al. [2022] sample axis-aligned planes, Thul et al. [2018] sample planes through concave edges (inspired by the exact method of Chazelle [1981]), and Lien and Amato [2007] use a more global analysis of concave features to place cut planes. Wei et al. [2022] use a Monte Carlo tree search to globally optimize the combined set of cutting planes. After cutting the mesh into smaller parts, Mamou [2016]; Wei et al. [2022] then merge parts when doing so will not raise the concavity metric too much. The method of Mamou and Ghorbel [2009] only uses merges – starting with the individual triangles as parts, and locally merging based on their concavity metric.

2.1.3 Preprocessing. To reliably compute volumes of an input shape, or cut it with planes, many methods require a clean, watertight mesh or some other volumetric representation. V-HACD [Mamou 2016] addresses this by voxelizing the input, guaranteeing a solid input for decomposition at the cost of some accuracy. Other work assumes the user has provided a clean input, or that some preprocess can be run to fix a bad input: The published implementation of Wei et al. [2022], for example, runs an implicit surface reconstruction on problematic inputs – creating a slightly offset version of bad inputs, while still being more precisely accurate on clean inputs.

2.1.4 Additional applications. Approximate convex decomposition is not limited to the problem of representing a single, static collision mesh: for example, Thul et al. [2018] introduced an efficient solution for decomposition of an animated mesh, and learning-based methods [Chen et al. 2020; Deng et al. 2020] have used convex decomposition as a generative model – for example finding a 3D decomposition given only a 2D image as input.

2.2 Path Finding

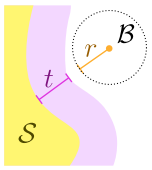
The idea of building an explicit model of navigable space is often used to enable path finding for game AI and robot navigation – for example, defining a projected grid of walkable space [Bandi and Thalmann 1998], a 'Nav Mesh' of walkable surfaces [Snook 2000], or a 'Configuration Space' of robot positions [de Berg et al. 2008; Lozano-Pérez 1990]. Many representations of such spaces have been proposed; our approach was inspired by the implicit fast marching navigation method of Sethian [1999].

3 NAVIGABLE SPACE

We define a *navigable space* of a shape $\mathcal{S}$ as a space around $\mathcal{S}$ that any other object in a given application should be able to freely pass through, and a *navigable approximate convex decomposition* as an approximate convex decomposition that doesn't overlap (*satisfies*) a given navigable space. Note a decomposition that satisfies a conservatively over-estimated navigable space also satisfies any more accurate navigable space contained within.

3.1 Automatic Construction

We construct a conservative navigable space using two parameters: First, a radius r defining a ball $\mathcal{B}$. We assume that all collision-relevant objects are large enough to fully contain this ball (for example, all characters in a game having capsule colliders with at least this radius). Second, we introduce a tolerance distance t at which it is acceptable to approximate $\mathcal{S}$. Any space that $\mathcal{B}$ can reach while staying at least distance t away from $\mathcal{S}$ is considered navigable space.



We start by defining a *center-navigable space* which ball $\mathcal{B}$'s center can reach. The ball center can be placed at any position p that satisfies:

$$\mathrm{SDF}(\mathcal{S}, \mathbf{p}) \geq r + t. \tag{1}$$

where $\mathrm{SDF}(\mathcal{S}, \mathbf{p})$ is the signed distance function (Fig. 3a).

We refine the center-navigable space to only consider *collision-relevant* space, near enough to the shape that ball $\mathcal{B}$ could overlap the shape's convex hull, $\mathrm{SDF}(\mathrm{Hull}(\mathcal{S}), \mathbf{p}) \leq r$ where $\mathrm{Hull}(\mathcal{S})$ is an exact convex hull (note the convex parts cannot extend beyond the overall convex hull, so tolerance t isn't needed here), and the *center-reachable* space, which ball $\mathcal{B}$ could arrive at by traversing a contiguous path, starting from outside the convex hull (Fig. 3b). Because internal 'air-pockets' of $\mathcal{S}$ are not reachable, we can also replace the signed-distance function $\mathrm{SDF}(\mathcal{S}, \mathbf{p})$ with an unsigned distance, $\mathrm{DF}(\mathcal{S}, \mathbf{p})$, which is simpler to compute and doesn't require $\mathcal{S}$ to form a solid. This gives the definition of relevant, center-navigable space as:

$$\mathrm{Relevant}(\mathbf{p}) = \mathrm{SDF}(\mathrm{Hull}(\mathcal{S}), \mathbf{p}) \leq r \wedge \mathrm{DF}(\mathcal{S}, \mathbf{p}) \geq r + t. \tag{2}$$

We find the boundary of center-reachable, relevant space by using Marching Cubes [Lorensen and Cline 1987] to mesh the true/false boundary of Eq. (2), using grid cells of size t. To ensure we only include the reachable portion, we specifically use the continuation method [Bloomenthal 1994]: Given seed points and a fixed grid resolution, output the marching cubes surfaces at the grid cells containing the seed points, and at the grid cells that can be found by locally following the implicit surface from those cells. We choose seed points on the outer-most relevant surfaces of the space by sampling the faces of $\mathrm{Hull}(\mathcal{S})$ and offsetting by r in the face normal direction. The continuation method then constructs *only* the reachable portion of the relevant, center-navigable space of $\mathcal{S}$. Finally, the boundary of center-navigable space can be offset by radius r to find the navigable space (Fig. 3c).

3.2 Representation

We need a representation of navigable space that can be quickly queried during construction of the convex decomposition. Motivated by the relative simplicity and speed of sphere-vs-convex overlap tests, we represent navigable space with a set of spheres. A sphere at position $\mathbf{p}$ has radius determined by the tolerance, t, and distance function, $\mathrm{DF}(\mathcal{S}, \mathbf{p})$:

$$\mathrm{Radius}(\mathbf{p}) = \mathbf{max}(0, \mathrm{DF}(\mathcal{S}, \mathbf{p}) - t). \tag{3}$$

We convert the Marching Cubes-generated mesh of center-navigable space to a set of spheres representing the navigable space by adding a sphere on each mesh vertex – letting the spheres cover the offset from center-navigable to navigable space. Note this representation is approximate, with error based on the marching cubes grid size.

3.3 Customization

It is easy to customize the navigable space by adding new spheres at any position, with radius set by Eq. (3). Such manual additional sphere placements can be used to require more details at specific locations in the decomposition without any need to tweak global parameters (Fig. 4).

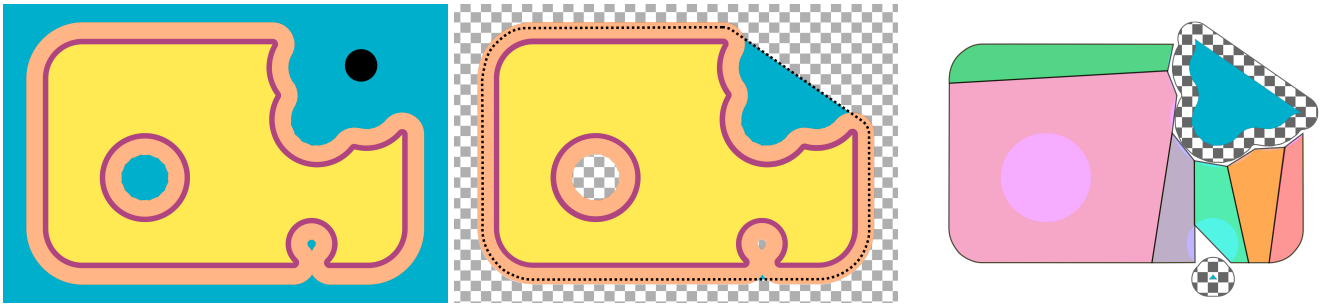
Our implicit definition of center-navigable space can also be customized to support user-specified obstacles – blocking off would-be navigable space can allow simpler decompositions. To do so, we update Eq. (2) with an optional, user-specified signed-distance function, BlockSDF:

$$\mathrm{CustomRelevant}(\mathbf{p}) = \mathrm{BlockSDF}(\mathbf{p}) \geq r \wedge \mathrm{Relevant}(\mathbf{p}). \tag{4}$$

For example, an impassable floor can be specified by setting:

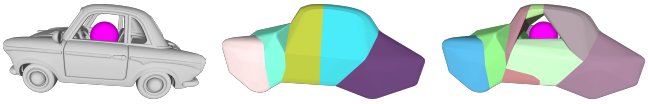
$$\mathrm{BlockSDF}(\mathbf{p}) = \mathbf{p}.z - \mathit{FloorHeight}, \tag{5}$$

where Z is up, which can allow our decomposition algorithm to automatically ignore concavities that would only be navigable if the object was approached from below – great for objects that will always be placed on a solid floor (Fig. 5).



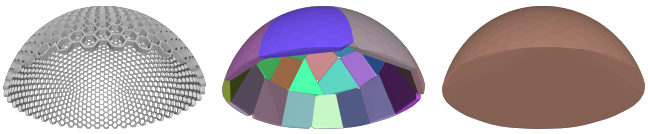
(a) Potential 'center-navigable space' (blue) (b) With unreachable/irrelevant removed (c) Expanded to navigable space (checkered)

Figure 3: The process to find navigable space, visualized. a) The potential 'center-navigable space' (blue) for ball $\mathcal{B}$ (black) is separated from shape $\mathcal{S}$ (yellow) by tolerance t (purple) plus $\mathcal{B}$-radius r (orange). b) The actual center-navigable space, in blue, after discarding *irrelevant* space outside the convex hull of $\mathcal{S}$ offset by r (dashed line) and *unreachable* space not connected to that offset hull. c) The center-navigable space is offset by radius r to find the navigable space (blue + checkered). We show an example 6-part convex decomposition of $\mathcal{S}$ that does not cover the navigable space.



(a) Input (b) Automatic (c) Custom

Figure 4: (a) A car and a custom navigable-space sphere (magenta). (b) An automatic, 9-part decomposition for large-scale characters, fully covering the inside of the car. (c) The same automatic settings, but with the custom sphere added, gives a 15 part decomposition that leaves space for the sphere.



(a) Input (b) Without floor (c) With floor

Figure 5: (a) A dome of small, tiled hexagons. (b) An automatic decomposition finds the interior navigable from below, and models it with 27 parts. (c) Using Eq. (5) to add a solid floor allows a single-part decomposition.

Algorithm 1: Navigable Convex Decomposition($\mathcal{S}, \mathcal{N}$)

Input: Shape mesh $\mathcal{S}$, and navigable spheres $\mathcal{N}$
Output: Convex decomposition $\mathcal{D}$

$\mathcal{D} \leftarrow \emptyset$ // Decomposition
$\mathcal{G} \leftarrow \emptyset$ // Graph of connections between parts
// Only track nav. spheres overlapping Hull($\mathcal{S}$)
$\mathcal{N} \leftarrow \{S \mid S \in \mathcal{N}, S \cap \text{Hull}(\mathcal{S}) \neq \emptyset\}$
$Q \leftarrow \{(\mathcal{S}, \mathcal{N})\}$ // Queue of (part, nav. spheres) tuples
// Cut parts until $\mathcal{N} \cap \mathcal{D} = \emptyset$
while $(Part, Spheres) \leftarrow Q$.Dequeue() **do**
  if $Spheres = \emptyset$ **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup Part$
  else
    $Plane \leftarrow$ChooseCutPlane($Part$) // Sec. 4.1.1
    $P_1, P_2 \leftarrow$Cut($Part, Plane$)
    MergeGraphNodes($\mathcal{G}, Part, P_1, P_2$) // Sec. 4.1.2
    // Filter relevant navigable spheres per part
    $N_1 \leftarrow \{S \mid S \in Spheres, S \cap \text{Hull}(P_1) \neq \emptyset\}$
    $N_2 \leftarrow \{S \mid S \in Spheres, S \cap \text{Hull}(P_2) \neq \emptyset\}$
    Q.Enqueue((P_1, N_1))
    Q.Enqueue((P_2, N_2))
  end if
end while
return MergeParts($\mathcal{D}, \mathcal{G}, \mathcal{N}$) // Alg. 2

4 NAVIGABLE CONVEX DECOMPOSITION

4.1 Decomposition Algorithm

To create an approximate convex decomposition $\mathcal{D}$ that doesn't overlap a given navigable space $\mathcal{N}$, we repeatedly cut shape $\mathcal{S}$ into smaller parts until no parts overlap $\mathcal{N}$ (Alg. 1). We then simplify $\mathcal{D}$ by repeatedly merging adjacent parts wherever the resulting merged part will not overlap $\mathcal{N}$ (Alg. 2).

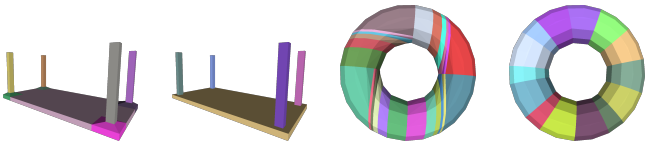
We optimize this process in two ways: (1) To reduce the cost of testing against navigable space during the split phase, we keep track of the subset of navigable spheres overlapping each part, and only test against those subsets – which tend to shrink quickly in size as the parts are cut. (2) To reduce the number of merges we need to consider, we maintain a connection graph – a graph tracking which parts are approximately in contact as we cut and merge – and only consider merges between parts that are connected in that graph.

4.1.1 Choice of Cut Planes. Alg. 1 will always find a decomposition that satisfies the navigable space as long as the cuts sufficiently reduce the size of the convex parts: A convex part with bounding box diagonal length less than t can never be farther than t from the input shape, so cannot overlap the navigable space, which is by construction at least distance t from the input shape.

Algorithm 2: MergeParts($\mathcal{D}, \mathcal{G}, \mathcal{N}$)

Input: Convex decomposition $\mathcal{D}$, graph of connections between parts $\mathcal{G}$, and navigable spheres $\mathcal{N}$
Output: Updated convex decomposition $\mathcal{D}$

$Q_{pri} \leftarrow \mathcal{G}$.Edges() // Priority queue of edges in $\mathcal{G}$
// Merge parts where doing so would not overlap $\mathcal{N}$
while $(P_1, P_2) \leftarrow Q_{pri}$.DequeueBest() **do**
  $MergedPart \leftarrow$ Merge(P_1, P_2)
  if Hull($MergedPart$) $\cap \mathcal{N} = \emptyset$ **then**
    // Replace P_1, P_2 w/ $MergedPart$ in $\mathcal{D}$, $\mathcal{G}$, Q_{pri}
    $\mathcal{D} \leftarrow \mathcal{D} \cup MergedPart - P_1 - P_2$
    $\mathcal{G}$.CollapseEdge(P_1, P_2)
    Q_{pri}.Remove(*Edges w/* P_1 *or* P_2)
    Q_{pri}.Enqueue(*MergedPart's edges*)
  end if
end while
return $\mathcal{D}$



(a) *Edges* = 0 **(b)** *Edges* = 4 **(c)** *Edges* = 0 **(d)** *Edges* = 4

Figure 6: Sampling concave edges for cutting planes is especially helpful for low-poly shapes, like this flipped table and torus. (a) Table without concave edges (9 hulls). (b) Table using 4 concave edges (5 hulls). (c) Torus without concave edges (24 hull). (d) Torus using 4 concave edges (14 hulls).

Table 1: Average part counts and runtimes for the V-HACD dataset meshes (Sec. 6.1.1) when varying the number of concave edges sampled to generate candidate cutting planes, with navigable space parameters $r = 5\%, t = 1\%$, where r, t are specified relative to the longest bounding box dimension.

# Concave Edges Sampled	# Parts	Runtime (s)
0	48.0	2.8
2	40.8	3.1
4	40.1	3.9
8	40.2	4.6
32	39.4	10.0
64	38.6	17.3

To guarantee the part width can be reduced enough to protect the navigable space, we propose candidate cutting planes bisecting the part along each major axis where the part's bounding extent is at least $t/2$. To heuristically improve the results, we also include candidate planes sampled through concave edges of the source geometry – that is, edges whose adjacent faces form a local concavity. Specifically, we take four concave edges (prioritized by midpoint depth inside the part's convex hull) and sample three cutting planes through each edge (one bisecting the edge's dihedral angle, and two aligned to the edge's adjacent faces). These planes through concave edges are especially useful on relatively simple input meshes, like those shown in Fig. 6. Table 1 explores the cost/benefit trade-off of sampling concave edges.

Of our candidate cutting planes, we heuristically choose the plane that creates parts with the smallest convex hull volumes:

$$\begin{aligned} \text{argmin}_{Plane}\ &\text{Volume}(\text{Hull}(P_1)) + \text{Volume}(\text{Hull}(P_2)), \\ &\text{where } P_1, P_2 \leftarrow \text{Cut}(Part, Plane). \end{aligned} \tag{6}$$

Note that in some cases, like the initial cut of a torus, Eq. 6 will score all cut planes equally. To nudge our algorithm toward more evenly-divided results in these cases, we apply a slight bias (scaling score by 0.99) in favor of the axis-aligned plane that cuts through the longest bounding box side.

4.1.2 Choice of Parts to Merge. Alg. 2 greedily reduces the number of parts while ensuring the navigable space remains protected. To avoid the need to consider the complete graph of all possible part merges, we maintain a *connection graph* of part adjacency. We update this during the cutting process: When a part is split in two, we update edges that were connected to the original part, and add an edge connecting the two new parts if their axis-aligned bounding boxes are closer than a small tolerance distance. During the merge phase, we only consider merges between parts that are neighbors in the connection graph, and we keep the connection graph up to date by collapsing the corresponding edge after each merge.

We heuristically choose to merge the (edge-connected) parts where the volume of the merged convex hull is closest to the volumes of the individual hulls of the original parts:

$$\text{argmin}_{A,B}\ \text{Volume}(\text{Hull}(\bigcup_{i \in A,B} P_i)) - \sum_{i \in A,B} \text{Volume}(\text{Hull}(P_i)), \tag{7}$$

where P_i are original parts generated by the cutting phase, and merge candidates A, B are edge-connected parts composed of original parts. Note that we sum the volumes of the original parts, and not the combined part volumes, to avoid double-counting volume in regions where convex hulls overlap after merging.

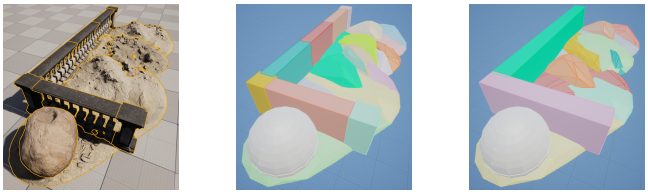
4.1.3 Robustness. Our algorithm can be made to run robustly on any 'triangle soup' input mesh, regardless of artifacts such as self-intersections, open boundaries, or non-manifold geometry, provided that: (1) for co-planar parts on which a 3D convex hull algorithm would fail, we slightly thicken the input by duplicating vertices with a small offset along the degenerate axis or axes, and (2) we detect if the input shape is not a well-defined solid (e.g., has artifacts like open boundaries) and use non-solid plane cuts in that case. The non-solid plane cut simply splits triangles where they cross the plane, creating open mesh boundaries.

5 COLLISION SHAPE MERGING

The merging phase of our convex decomposition algorithm can be adapted to solve a variant of the convex decomposition problem: Simplifying an existing *general* collision shape representation, where the collision may be represented by multiple primitives such as spheres, oriented boxes, and capsules, as well as convex hulls. This is especially useful when generating collision for a model that

(a) Input (b) $r = 5\%, t = 1\%$ (c) $r = 10\%, t = 1\%$

Figure 7: (a) A one-sided mesh of a bottle, with a hole at the top, from the PartNet-Mobility dataset [Xiang et al. 2020]. (b) Decomposition using navigable space parameters $r = 5\%, t = 1\%$. The interior is navigable. (c) Decomposition using navigable space parameters $r = 10\%, t = 1\%$. The radius of the hole is smaller than $r + t$, so the interior is filled aside from a small, still-navigable indent at the top. (Note: r, t are specified relative to the longest bounding box dimension.)



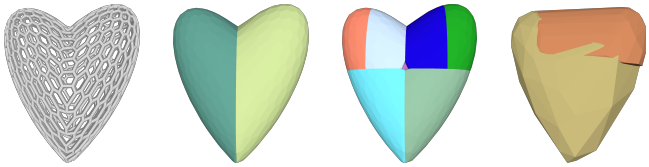
(a) Input (b) Input Collision (c) Merged

Figure 8: (a) A shape built from premade parts (each outlined in yellow). (b) Each premade part already has simple collision defined. (c) The result of our collision shape merging.

was created by combining (or 'kitbashing') instances of preexisting models, as we can leverage the collision that was created for the pre-existing models in isolation. For example, the small scene in Fig. 8 has a railing built of 7 instanced parts, each with box collision. A top-down convex decomposition would not know that box collision is acceptable for those shapes, but the bottom-up algorithm can directly use those boxes – merging them to represent the combined railing with just 2 shapes.

Our collision shape merging algorithm is a straightforward application of Alg. 2, with the following adjustments: (1) We convert all input parts to convex hulls to run the algorithm, and on completion we convert any parts that were not merged back to their original representation. For the parts that were merged, we can optionally attempt to further simplify the representation by fitting a minimum volume sphere and oriented box to each part [Eberly 2020], and replacing the hull with the best-fit shape if it is within a tolerance distance of the convex hull. (2) We generate the required connection graph by connecting all parts with bounding boxes closer to each other than a user-specified distance.

Note that Eq. (7) assumes the input parts do not overlap, which is true by construction for Alg. 1, but may not be true here. When parts overlap, the metric is biased in favor of overlapping pairs because it double counts the volume of their intersection. While an alternate metric could be used, we find that merging overlapping parts early typically still gives acceptable results, so we still use this metric.



(a) Input (b) Ours (c) CoACD (d) V-HACD

Figure 9: Decompositions of (a) this hollow mesh heart. (b) Our algorithm with $r = 5\%, t = 1\%$ fills the (non-navigable) inside of the heart (2 hulls). (c) Even with $t_c = 20\%$, CoACD avoids filling the heart (12 hulls). (d) V-HACD with $hulls = 2$ loses the outer shape of the heart (2 hulls).

6 EVALUATION

6.1 Comparison to Prior Work

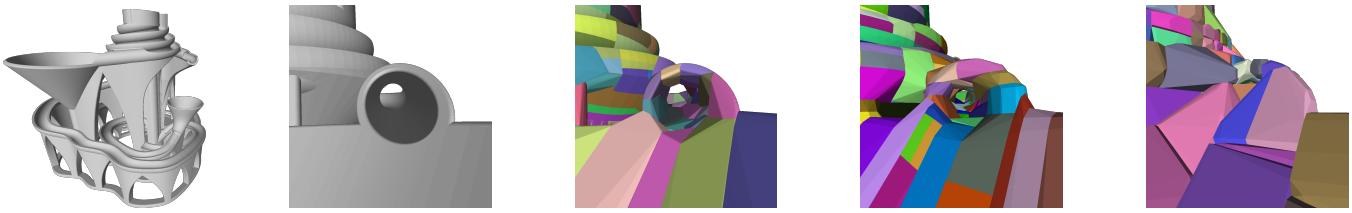
We run our decomposition algorithm, as well as V-HACD [Mamou 2016] and the 'Collision-Aware' method of Wei et al. [2022], CoACD, on two datasets of meshes to evaluate how the methods perform. All evaluations are run on the same machine, with a 64-core, 2.7 GHz AMD Threadripper processor and 256 GB of RAM. Note that all distance-based parameters (r, t for our algorithm, and concavity tolerance t_c for CoACD) are specified in terms of a percentage of the longest side of each shape's bounding box.

6.1.1 Datasets. We use two sets of meshes: First, a set of 61 meshes distributed with V-HACD, which have often been used as a test set for approximate convex decomposition algorithms. This dataset includes a variety of shape categories, including complete scans, open/partial scans, humans, animals, and various objects. It does not include arrangements of many parts to form larger shapes – as may be created by 'kitbashing' – so to consider that case, we also use the 2346 assemblies of meshes in the PartNet-Mobility dataset [Xiang et al. 2020]. Note that these assemblies are parametric, with many possible poses; for our evaluation we take the default pose for each assembly.

6.1.2 Quantitative Evaluation Overview. Previous approximate convex decomposition algorithms do not have a concept of navigable space; instead, V-HACD and CoACD are controlled by parameters such as the maximum requested hull count and a tolerance for a 'concavity' metric. To compare the results of previous methods to our own, we evaluate a range of parameters for each algorithm, and report the timings, number of parts used, and how closely the results can satisfy two different navigable spaces. Specifically, for CoACD we vary the concavity tolerance parameter, t_c, which approximates the distance between the convex decomposition and the input shape. For V-HACD we vary the number of hulls requested, because directly specifying a number of hulls is the current best-supported usage pattern [neemoh and Ratcliff 2023]. To favor accuracy, we run V-HACD with a relatively-high resolution of 1,000,000 voxels. The two navigable spaces we evaluate against are parameterized by (1) a *coarse* navigable space, $r = 2.5\%, t = 5\%$, and (2) a *fine-grained* navigable space, $r = 5\%, t = 1\%$. Note that for the fine-grained space

Table 2: Evaluation of approximate convex decompositions of shapes in the V-HACD and PartNet-Mobility datasets. We list how much each decomposition overlaps two navigable spaces generated from different r, t parameters. The '% Overlap' column lists the percent of decompositions that overlap the navigable space, and the 'Max Overlap' column lists how deeply any navigable sphere overlaps any decomposition as a fraction of the sphere radius, with a maximum measured value of 1 indicating a navigable sphere's center was inside a convex hull. The runtime is an average time per part, in seconds.

Dataset	Method	Parameters	# Parts	Runtime (s)	$r = 2.5\%, t = 5\%$		$r = 5\%, t = 1\%$	
					% Overlap	Max Overlap	% Overlap	Max Overlap
V-HACD	V-HACD	$hulls = 10$	10.0	4.7	70%	1	97%	1
		$hulls = 25$	25.0	4.6	26%	1	95%	1
		$hulls = 100$	98.2	4.5	**0%**	**0**	95%	0.73
	CoACD	$t_c = 10\%$	13.9	25.3	7%	0.24	93%	1
		$t_c = 5\%$	31.6	34.1	**0%**	**0**	93%	0.44
		$t_c = 1\%$	245.3	328.7	**0%**	**0**	38%	0.09
	Ours	$r = 2.5\%, t = 5\%$	**8.4**	**1.1**	**0%**	**0**	93%	1
		$r = 5.0\%, t = 1\%$	40.1	3.9	**0%**	**0**	**0%**	**0**
PartNet-Mobility	V-HACD	$hulls = 10$	10.0	5.4	37%	1	93%	1
		$hulls = 25$	25.0	5.1	17%	1	90%	1
		$hulls = 100$	99.5	5.0	1%	1	85%	1
	CoACD	$t_c = 10\%$	15.0	49.0	6%	0.63	95%	1
		$t_c = 5\%$	33.7	64.9	**0%**	**0**	90%	0.55
	Ours	$r = 2.5\%, t = 5\%$	**6.4**	**0**.5	**0%**	**0**	95%	1
		$r = 5.0\%, t = 1\%$	24.4	1.7	**0%**	**0**	**0%**	**0**



(a) Input **(b) Close-up view** **(c) Ours** **(d) CoACD** **(e) V-HACD**

Figure 10: (a) A marble machine. (b) Close-up view of marble tunnel. (c) Our algorithm, run with parameters $r = 1\%, t = .5\%$ (chosen so $r + t < 1/2$ tunnel width), preserves the tunnel (1276 hulls). (d) CoACD, run with its (best supported) tolerance $t_c = 1\%$, preserves the tunnel albeit with added thickness (1898 hulls). (e) V-HACD, run with 10,000,000 voxels and $hulls = 2000$, fails to preserve the tunnel (2000 hulls).

we use a larger r to explore a case where navigability is a more significant factor. We run our own algorithm on both navigable spaces.

6.1.3 Quantitative Analysis. We report results on each dataset in Table 2. Our algorithm always satisfies its own navigable space. Neither previous algorithm can reliably satisfy the fine-grained space, though CoACD almost does so (with only small overlaps) for its most-precise tolerance setting, at the cost of using a huge amount of time and number of convex hulls. Both V-HACD and CoACD can generally satisfy the coarse navigable space, however both would require a manual parameter search for each input mesh to do so with the fewest possible convex hulls. Note that CoACD uses many more hulls than needed when its concavity tolerance t_c matches the $t = 5\%$ value of the coarse navigable space: for $\geq 93\%$ of both datasets, CoACD still satisfies the coarse navigable space even when we double parameter t_c to 10%. (This is likely due to nature of the volumetric approximation used to define t_c, which generally over-estimates the error.) Overall, these results demonstrate the power of our method to quickly find a smaller number of hulls that satisfy our requirements.

6.1.4 Qualitative Analysis. We visualize the differences between these algorithms on two illustrative test cases. First, a marble machine (Fig. 10) has the strict requirement that the marble tunnels must remain open wide enough for marbles to pass through. By specifying marble-sized r, t parameters, our algorithm reliably does so. V-HACD is not capable of this level of accuracy, and CoACD with its most precise tolerance setting $t_c = 1\%$ keeps the tunnels open but thickens them. Second, we illustrate the value of navigable space for a hollow shape (Fig. 9). Our algorithm reliably fills in the hollow shape, but CoACD has no notion of unreachable surfaces, and tends to preserve the interior even at large tolerance values.

V-HACD likewise has no way to distinguish the unimportant interior volume, so does not generate ideal results either. We show additional comparisons to CoACD in Fig. 11 and 12, and to V-HACD in Fig. 13.

6.2 Case Study

We integrated our collision shape merging algorithm into the internal content pipeline of a AAA video game, LEGO® Fortnite, which heavily used assets built from combinations of smaller LEGO pieces. Our method was used to automate generation of collision shapes for over 1400 assets, each of which needed multiple sets of collision shapes with different accuracy requirements (one set for character collision, and another for a game-specific construction mechanic). Our method's ability to protect the navigable space consistently across all examples enabled the game's developers to tune parameters visually for a subset of example cases and trust the system to generate similar results for the whole set.

7 DISCUSSION

Our choice to represent the navigable space with spheres, and to characterize all interacting shapes by a minimal-radius sphere, was motivated by games which often use sphere- or capsule-based colliders for characters. While the sphere-based representation, especially with custom sphere placements, should be able to represent any desired level of accuracy, future work could explore tailoring the navigable space search, and navigable space representation, to differently-shaped character colliders.

In this work we focused on proving the viability of explicitly representing and protecting navigable space. Our approximate convex decomposition algorithm could be improved in other ways, for example by using the random tree search for the best cutting planes suggested by Wei et al. [2022], and perhaps adding a similar, search-based approach to the merging part of the algorithm as well. The resampling-based hull simplification step proposed by Mamou [2016] could also be applied to reduce the number of vertices used by each convex hull.

ACKNOWLEDGMENTS

Thanks to Jack Oakman for being the patient first user of these algorithms and providing invaluable advice and feedback throughout development, and to Rinat Abdrashitov, David Hill, Ryan Schmidt, and the anonymous reviewers for their helpful comments on earlier drafts of this paper. For 3D models used to demonstrate our algorithms, thanks to: RMTCTRL for the Tiny House model (Fig. 1; CC-BY), Slava_Z for the Pony Toy Car model (Fig. 4 and 13j; CC-BY), dav88 for the Hex Dome model (Fig. 5 and 13g; CC-BY), mroek for The Cyclone model (Fig 10; CC-BY), pmoews for the Heart model (Fig. 9; CC-BY), the Stanford Graphics Lab for their classic Bunny model (Fig. 12a and 13a), Mamou [2016] for the V-HACD dataset and Xiang et al. [2020] for the PartNet-Mobility dataset.

REFERENCES

Srikanth Bandi and Daniel Thalmann. 1998. Space Discretization for Efficient Human Navigation. *Computer Graphics Forum* 17, 3 (1998), 195–206. https://doi.org/10.1111/1467-8659.00267

Jules Bloomenthal. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, Paul S. Heckbert (Ed.). Academic Press Professional Inc., San Diego, CA, 324–349.

Bernard M. Chazelle. 1981. Convex Decompositions of Polyhedra. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing* (Milwaukee, Wisconsin, USA) *(STOC '81)*. Association for Computing Machinery, New York, NY, USA, 70–79. https://doi.org/10.1145/800076.802459

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. BSP-Net: Generating Compact Meshes via Binary Space Partitioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 42–51. https://doi.org/10.1109/CVPR42600.2020.00012

Mark de Berg, Otfried Cheong, Marc Kreveld, and Mark Overmars. 2008. *Robot Motion Planning*. Springer Berlin Heidelberg, Berlin, Heidelberg, 283–306. https://doi.org/10.1007/978-3-540-77974-2_13

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. https://doi.org/10.1080/10867651.1999.10487502

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. 2020. CvxNet: Learnable Convex Decomposition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 31–41. https://doi.org/10.1109/CVPR42600.2020.00011

Dave Eberly. 2020. *Robust and Error-Free Geometric Computing*. CRC Press, Boca Raton, FL, USA.

Epic Games. 2023. *Add a Collision Hull to a Static Mesh Using the Auto Convex Collision Tool*. https://docs.unrealengine.com/5.3/en-US/add-a-collision-hull-to-a-static-mesh-using-the-auto-convex-collision-tool-in-unreal-engine/

Jyh-Ming Lien and Nancy M. Amato. 2004. Approximate convex decomposition of polygons. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) *(SCG '04)*. Association for Computing Machinery, New York, NY, USA, 17–26. https://doi.org/10.1145/997817.997823

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* (Beijing, China) *(SPM '07)*. Association for Computing Machinery, New York, NY, USA, 121–131. https://doi.org/10.1145/1236246.1236265

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

Tomás Lozano-Pérez. 1990. Spatial Planning: A Configuration Space Approach. In *Autonomous Robot Vehicles*, Ingemar J. Cox and Gordon T. Wilfong (Eds.). Springer New York, New York, NY, 259–271. https://doi.org/10.1007/978-1-4613-8997-2_20

Khaled Mamou. 2016. Volumetric hierarchical approximate convex decomposition. In *Game Engine Gems 3*, Eric Lengyel (Ed.). A K Peters / CRC Press, Boca Raton, FL, USA, 141–158.

Khaled Mamou and Faouzi Ghorbel. 2009. A simple and efficient approach for 3D mesh approximate convex decomposition. In *Proceedings of the 16th IEEE International Conference on Image Processing* (Cairo, Egypt) *(ICIP'09)*. IEEE Press, 3465–3468.

neemoh and John W. Ratcliff. 2023. *V-HACD GitHub issue 149: V4.1 Always producing m_maxConvexHulls*. https://github.com/kmammou/v-hacd/issues/149

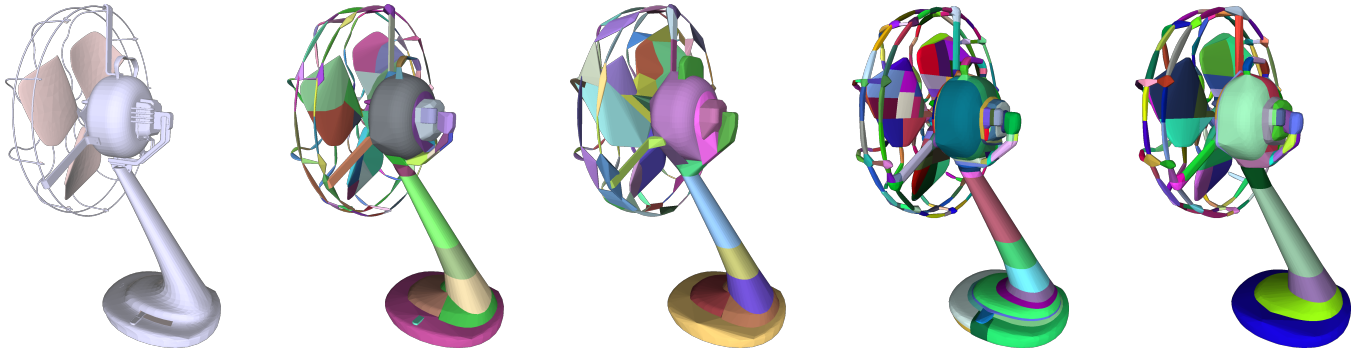
James A. Sethian. 1999. Optimality and First Arrivals. In *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 286–315.

Greg Snook. 2000. Simplified 3D Movement and Pathfinding Using Navigation Meshes. In *Game Programming Gems*, Mark DeLoura (Ed.). Charles River Media, 288–304.

Daniel Thul, L'ubor Ladický, Sohyeon Jeong, and Marc Pollefeys. 2018. Approximate convex decomposition and transfer for animated meshes. *ACM Trans. Graph.* 37, 6, Article 226 (dec 2018), 10 pages. https://doi.org/10.1145/3272127.3275029

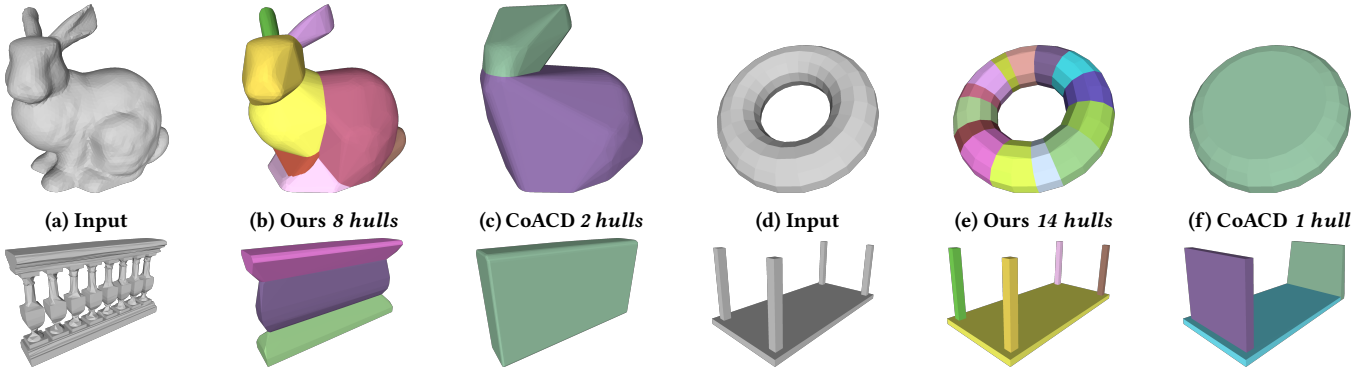
Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. 2022. Approximate convex decomposition for 3D meshes with collision-aware concavity and tree search. *ACM Trans. Graph.* 41, 4, Article 42 (jul 2022), 18 pages. https://doi.org/10.1145/3528223.3530103

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. 2020. SAPIEN: A SimulAted Part-Based Interactive ENvironment. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 11094–11104. https://doi.org/10.1109/CVPR42600.2020.01111



(a) Input (b) Ours $r = 0.5\%, t = 1\%$ (c) Ours $r = 5\%, t = 1\%$ (d) CoACD $t_c = 1\%$ (e) CoACD $t_c = 2\%$

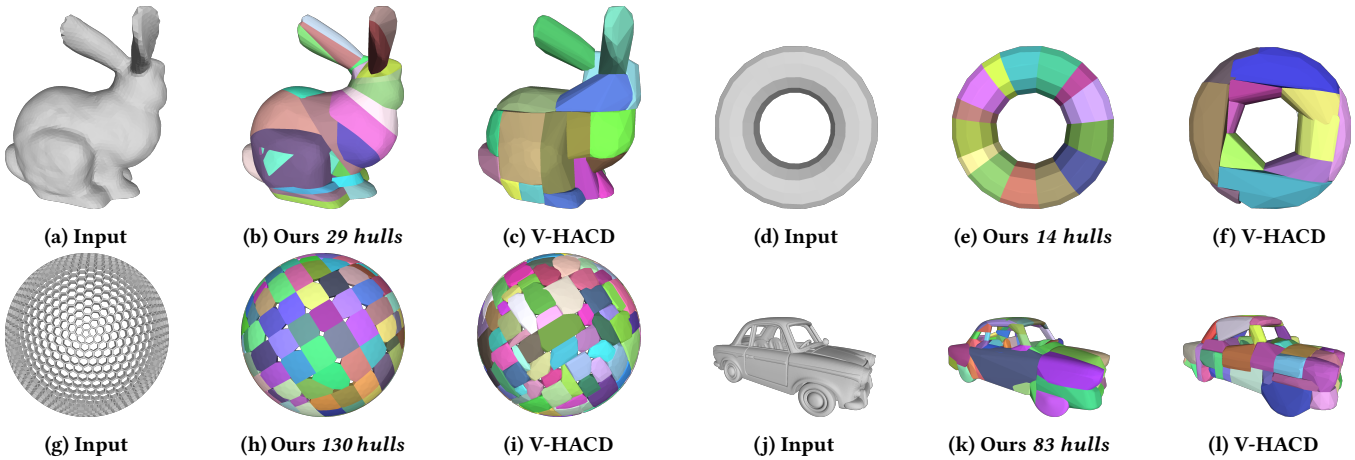
Figure 11: (a) A fan model from the PartNet-Mobility dataset [Xiang et al. 2020]. (b) Our algorithm, run with $r = 0.5\%, t = 1\%$, captures fine details (182 hulls). (c) Our algorithm, run with $r = 5\%, t = 1\%$, can use fewer hulls around small, inaccessible details (108 hulls). (d) CoACD, run with tolerance $t_c = 1\%$ (328 hulls). (e) CoACD, run with tolerance $t_c = 2\%$ (190 hulls).



(a) Input (b) Ours *8 hulls* (c) CoACD *2 hulls* (d) Input (e) Ours *14 hulls* (f) CoACD *1 hull*

(g) Input (h) Ours *3 hulls* (i) CoACD *1 hull* (j) Input (k) Ours *5 hulls* (l) CoACD *3 hulls*

Figure 12: Our method's r parameter enables us to tailor collision to larger-scale characters, while still using a low tolerance value. We show results of our method run with $r = 50\%, t = 1\%$ (b,e,h,k), compared to results from CoACD with $t_c = 50\%$ (c,f,i,l). Note that increasing CoACD's t_c parameter to match our r parameter loses large-scale features, such as the torus center and space between table legs. Also note that our method's torus result at this large r is the same as for a much smaller r (Fig. 13e).



(a) Input (b) Ours *29 hulls* (c) V-HACD (d) Input (e) Ours *14 hulls* (f) V-HACD

(g) Input (h) Ours *130 hulls* (i) V-HACD (j) Input (k) Ours *83 hulls* (l) V-HACD

Figure 13: We show results from our method and V-HACD (with resolution of 1,000,000 voxels) for the same hull counts. Our method was run with $r = 5\%, t = 1\%$ (b,e,h,k), then we set V-HACD to generate decompositions with matching hull counts (c,f,i,l).