VisACD: Visibility-Based GPU-Accelerated Approximate Convex Decomposition

Egor Fokin[1] Manolis Savva[1]

[1]Simon Fraser University
3dlg-hcvc.github.io/visacd

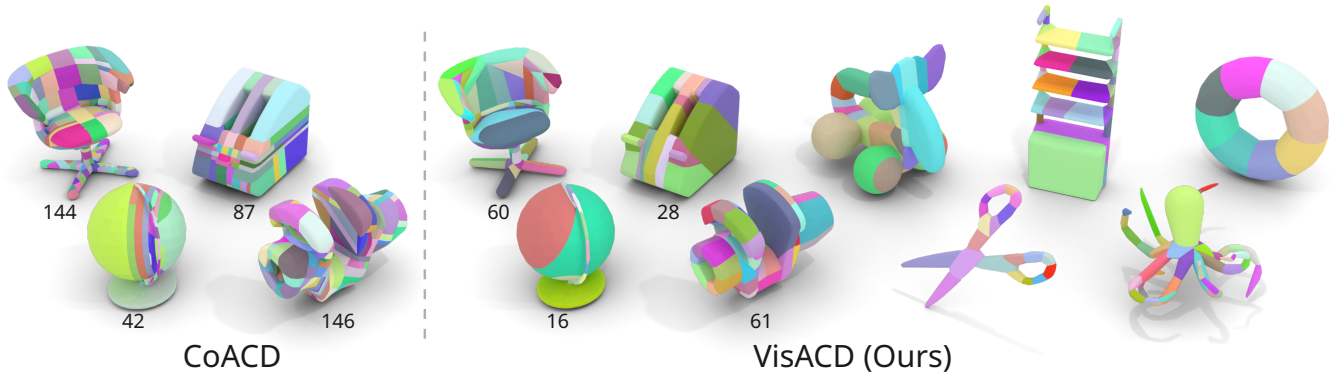

Figure 1: *We present VisACD, a visibility-based GPU-accelerated intersection-free approximate convex decomposition (ACD) algorithm. VisACD is rotation-equivariant and thus not sensitive to input mesh orientation. Our experiments show that VisACD outperforms prior methods producing accurate decompositions with fewer parts (indicated by numbers).*

Abstract

Physics-based simulation involves trade-offs between performance and accuracy. In collision detection, one trade-off is the granularity of collider geometry. Primitive-based colliders such as bounding boxes are efficient, while using the original mesh is more accurate but often computationally expensive. Approximate Convex Decomposition (ACD) methods strive for a balance of efficiency and accuracy. Prior works can produce high-quality decompositions but require large numbers of convex parts and are sensitive to the orientation of the input mesh. We address these weaknesses with VisACD, a visibility-based, rotation-equivariant, and intersection-free ACD algorithm with GPU acceleration. Our approach produces high-quality decompositions with fewer convex parts, is not sensitive to shape orientation, and is more efficient than prior work.

CCS Concepts

• ***Computing methodologies*** → *Collision detection; Mesh geometry models; Mesh models; Parallel algorithms;*

1. Introduction

Representing shapes as a set of convex hulls is an acceleration technique used widely in physics and game engines. This representation enables efficient algorithms such as checking whether a point lies inside the mesh [Sno17], computing distances between two objects [GJK02] or checking whether two meshes intersect [Ber99]. In practice, sets of convex hull 'colliders' are authored by content creators through fitting primitives such as boxes or capsules to a mesh asset. This is a time-consuming process, and the resulting colliders can differ substantially from the original mesh asset. Multiple algorithms were created to automate this process and produce more accurate convex decompositions.

Exact convex decomposition [Cha81] attempts to decompose a shape into convex hulls such that the shape of the decomposition is exactly the same as the input mesh. Such algorithms are slow and produce thousands of parts negating the intended efficiency gains. A different approach is Approximate Convex Decomposition (ACD) [LA04], which decomposes objects into parts that approximately match the initial shape. This relaxed condition allows algorithms that are faster and produce decompositions with orders of magnitude fewer parts.

In this paper, we present **a)** VisACD, a method that produces convex decompositions closer to the initial mesh and with lower part numbers, that is not sensitive to mesh orientation and that is

DOI: 10.2312/egs.20261014

more efficient, compared to baselines from prior work. **b)** A concavity metric tailored for efficient cutting plane computation. **c)** A parallelization of our algorithm using NVidia OptiX and CUDA.

2. Related Work

Applications for ACD methods. Recent works [LHG*24; GXL*23] use ACD to create colliders for different objects in robot grasping simulations. NERF-Texture [HCL*23] uses CoACD as volume-preserving smoothing. V-HACD [MLP16] was adapted by Unreal Engine 4 for automatic collider creation of 3D objects. The quality of the decompositions directly impacts the accuracy and efficiency in the mentioned applications.

Concavity measures for ACD. ACD algorithms use a concavity measure that they minimize and use in evaluation. Some methods use surface-based metrics to measure concavity. These metrics include distances between the mesh surface and its convex hull [LA04; GALL13], or between the shortest geodesic path on the mesh and the convex hull [LXL16]. HACD [MG09] projects the vertices onto the convex hull and measures the distances between the points and their projections. Wei et al. [WLLS22] show that these metrics fail to capture differences of volume properties. Other methods use volume-based metrics, such as volume ratio between a mesh and its hull [AMSF08; MLP16; TLJP18], and CoACD [WLLS22] uses a mixture of volume-based and surface-based metrics. Another family of metrics uses pairwise visibility of surface points (two points are considered visible to each other if the segment between them lies fully outside the mesh). Some works [KFK*14; AGC13] use a percentage of pairs that are mutually visible. Other works [LLL10; RYLL11] calculate the distance between those connections and the surface of the mesh. This metric can be easily computed and has useful properties that we discuss later. Our method uses an efficient visibility-based metric at its core, computing the total combined length of all segments between mutually visible vertices.

Prior work on ACD algorithms. Lien and Amato [LA04] introduced a surface-based approach that cuts meshes at regions of high concavity, with later works improving concavity measures [GALL13], cut paths [LXL16], or using triangle merging [MG09; KS24]. However, surface-based methods struggle to preserve volume, especially in meshes with holes or cavities. To address this, volumetric ACD methods were proposed, including tetrahedral merging [AMSF08], point-patch clustering [KFK*14], spectral clustering [AGC13], and Reeb-graph-based approaches [LLL10; RYLL11]. While these methods better preserve volume, they do not prevent convex hull intersections, limiting their applicability in animation and simulation. More recent works utilize cutting planes as a main tool of decomposition to prevent convex hull intersections. V-HACD [MG09] uses axis-aligned cutting planes to decompose a voxelized representation of the mesh. CoACD [WLLS22] improves on this by using an MCTS algorithm instead of a greedy one and by employing a regular mesh representation together with a volumetric concavity measure to preserve volume properties. Thul et al. [TLJP18] and Andrews [And24] additionally sample planes that contain edges of maximum concavity. Such methods evaluate the cutting planes by performing the cut and computing the concavity of resulting pieces, which is highly inefficient, limiting the number of planes that can be sampled at every iteration.

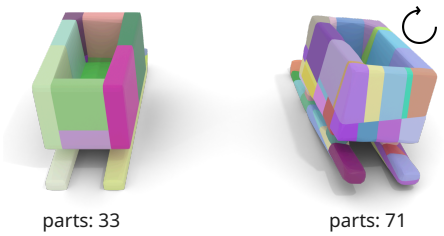

Figure 2: *Sensitivity of axis-aligned plane methods to orientation.*

Our approach also uses cutting planes. However, by using visibility-based concavity we can compute a plane *value* without cutting the mesh, which allows us to sample thousands of planes per iteration and achieve better results. Our plane sampling strategy also makes the method rotation-equivariant, eliminating sensitivity to mesh orientation in axis-aligned plane methods (Figure 2).

3. Method

3.1. Visibility-Based Concavity Metric

We use the definition of weak convexity proposed by Asafi et al. [AGC13] and define **visibility edges** e_i as edges between two mesh vertices that lie outside of the mesh and do not intersect it. An important property is that a convex mesh has zero visibility edges and the more concavities there are in a mesh, the more visibility edges.

One way to measure the concavity of the mesh is to use the number of such edges. However, not all visibility edges are equally important. A better way to compute the cost of a visibility edge was proposed by Ren et al. [RYLL11]. They take the maximum perpendicular distance between the edge and the mesh surface. Unfortunately, this is expensive to compute, especially in 3D. Instead, we use the length of an edge as the cost. This is motivated by the observation that edges farther away from the surface tend to have higher length. We compute the concavity of the mesh C^* using a combined length of all visibility edges e_i: $C^*(M) = \sum_i \|e_i\|_2$. We also define the concavity of the decomposition $D = \{M_1, M_2, ..., M_n\}$ as: $C^*(D) = \sum_{M \in D} C^*(M)$. Because this concavity measure is based on mesh vertices rather than the volume enclosed by the mesh, it depends on mesh topology and cannot be used to compare decompositions across different meshes. Therefore, for evaluation purposes we use the *collision-aware concavity measure* C by Wei et al. [WLLS22]. This metric takes the minimum between Hausdorff distance and scaled volume difference.

3.2. Visibility Edge Computation

Our main insight is that "point visibility" algorithms are highly parallelizable. Prior work does not leverage this and instead extrapolates from a subset of visibility edges [KFK*14] or uses visibility between Reeb graph vertices [LLL10; RYLL11] We implement a parallel GPU-accelerated algorithm to compute *all* visibility edges, leading to high precision without increasing computation time.

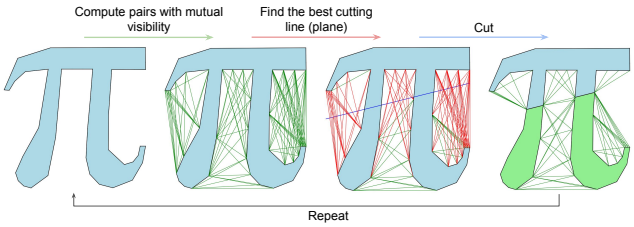

Figure 3: *We use pairwise point visibility to compute the **visibility edges** and approximate the concavity of the mesh. We sample k visibility edges and compute bisecting orthogonal planes for each. The value of the plane $Q_p(M,E)$ is the sum of cut visibility edge lengths. We select the plane with highest value to make the cut.*

To compute the visibility edges of a mesh M, we first construct a cage mesh M_c offset by $\varepsilon = 0.03$ from the surface of M. An edge e is classified as a *visibility edge* if and only if $(e \cap M_c \neq \varnothing) \wedge (e \cap M = \varnothing)$. In practice, these intersection tests are efficiently evaluated using ray–mesh queries implemented in NVIDIA OptiX. This method of computing visibility edges also filters out edges that are too close to the mesh and do not significantly impact the concavity. By changing ε, one can manipulate how many such edges are filtered out.

3.3. Plane Value

Given a mesh M, a cutting plane p splits it into two meshes M_1 and M_2. We want to find a plane p that minimizes $C^*(\{M_1, M_2\})$. Let $I_P(e_i) \in \{0,1\}$ be the indicator that p cuts the visibility edge e_i and e_i' be the new visibility edges created after the cut. Then:

$$C^*(\{M_1,M_2\}) = C^*(M) - \sum_i I_P(e_i)||e_i||_2 + \sum_i ||e_i'||_2 \quad (1)$$

The combined length of the newly created edges $\sum_{e_i'} ||e_i'||_2$ is complex to compute and, for high vertex densities, contributes little to the overall concavity. Therefore, we can simplify:

$$C^*(\{M_1,M_2\}) \approx C^*(M) - \sum_i I_P(e_i)||e_i||_2 \quad (2)$$

Then:

$$\begin{aligned}\arg\min_p C^*(\{M_1,M_2\}) &\approx \arg\min_p \left(C^*(M) - \sum_i I_p(e_i)\,\|e_i\|_2 \right) \\ &= \arg\max_p \sum_i I_p(e_i)\,\|e_i\|_2.\end{aligned}$$

From this, we get the value function $Q_p(M,E)$ of a plane p:

$$Q_p(M,E) = \sum_i I_p(e_i)\,\|e_i\|_2 \quad (3)$$

This value is interpretable as the combined length of all visibility edges that the plane p cuts.

3.4. Candidate Planes

Choosing plane candidates is a crucial step in ACD algorithms. CoACD [WLLS22] and V-HACD [MLP16] consider only axis-aligned cutting planes spaced by small epsilon values and Thul et al. [TLJP18] uses planes that intersect *concave* edges. The reason behind these choices is that the number of planes they can check at each step is quite low (60 for CoACD), which motivates setting strict limitations on the planes sampled. Our efficient value function enables sampling many more planes (1000+) at each step. Similarly to our concavity measure and value function, we sample cutting planes based on visibility edges. We pick k random edges and assign a plane to each of them such that each plane is orthogonal to the corresponding edge and bisects it. Additionally, the planes that correspond to the largest flat surfaces of the mesh are more likely to be cutting planes. For this reason, we extract these planes, add them to the set of candidates, and double their value.

Dataset	Method	Concavity ↓	Parts ↓
V-HACD	VisACD (Ours)	**0.043**	**28.4**
	CoACD [WLLS22]	0.048	31.6
	V-HACD [MLP16]	0.118	57.6
	Thul et al. [TLJP18]	0.069	34.4
PartNet-Mobility	VisACD (Ours)	0.046	**35.1**
	CoACD	0.046	35.6
Objaverse	VisACD (Ours)	0.047	**45.4**
	CoACD	0.047	58.3

Table 1: *Quantitative comparisons. VisACD outperforms baselines, especially on Objaverse where meshes have more irregular orientation and geometric structure.*

3.5. Greedy Algorithm

To arrive at the final decomposition D from the initial mesh M, we follow a simple greedy algorithm (Figure 3). At each step, we pick the part with the highest concavity (Both $C(M)$ and $C^*(M)$ can be used, but we find that using $C(M)$ produces better results). We compute all visibility edges for that part and sample k candidate cutting planes. Each sampled plane is evaluated according to the objective function $Q_p(M,E)$ (3), and the plane with the highest value is selected. The mesh is subsequently partitioned using this cutting plane. In cases where the cut yields more than two disconnected components, we further separate these components using triangle-level connectivity. The stopping criteria to terminate the decomposition process are: a) the target concavity threshold is reached; b) the prescribed number of parts has been obtained; or c) the current part contains no remaining visibility edges.

4. Results and Experiments

Task Definition. We decompose an input 3D mesh into a set of convex hulls using one of the ACD algorithms. We then report the quality of the decomposition through the number of convex hull parts, and concavity measuring distance from the input.

Setup. We evaluate on V-HACD [MLP16], PartNet-Mobility [XQM*20] and Objaverse [DSS*23]. For Objaverse we use a randomly sampled subset of 1,000 meshes. We compare results with CoACD [WLLS22], V-HACD[MLP16] and Thul et al. [TLJP18] (results reproduced from CoACD paper as source

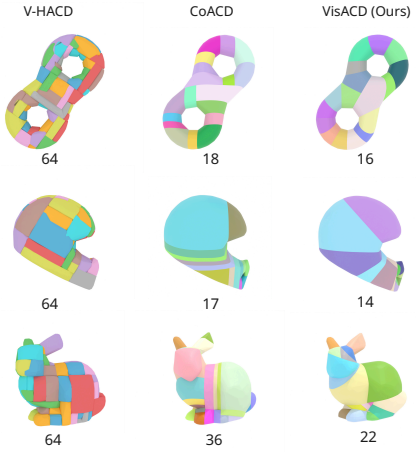

Figure 4: *Qualitative Comparisons. We see that VisACD is not limited by axis-aligned planes and can produce accurate decompositions with fewer parts (indicated in numbers).*

code is not publicly released). We do not evaluate methods that produce decompositions with intersecting convex hulls. We evaluate CoACD without the merging step, as merging produces intersecting convex hulls in 35% of cases. We do not use merging in our algorithm for the same reason. We use $C(M)$ for the evaluation. We also preprocess the meshes using SDF remeshing to make the meshes watertight, limit the number of vertices, and make vertex density more uniform.

Discussion. We present the results in Table 1. Our method outperforms all baselines on all datasets while having lower computation time (16.97 seconds on average per PartNet-Mobility model vs 36.31 for CoACD). We attribute the smaller differences on PartNet-Mobility to the prevalence of highly regular objects such as tables, chairs, and bookcases, that can be decomposed into nearly perfect convex components using only axis-aligned cutting planes. In contrast, the V-HACD and Objaverse datasets contain models with diverse, non-standard orientations and more organic shapes, including humans and animals, for which our method is well suited.

5. Conclusion and Future Work

We presented an approximate convex decomposition method that combines point visibility with cutting planes. It achieves a favorable trade-off between concavity and part count, while being significantly faster than prior work. At the same time, there are limitations to address. The current pipeline relies on a greedy algorithm that can produce sub-optimal solutions. Our algorithm is also sensitive to the topology of the original mesh, and requires remeshing for optimal performance. Future work may attempt to come up with an algorithm that considers the best possible decomposition not only on the current step, but also in the future (e.g. through Monte-Carlo Tree Search), while still utilizing the efficiencies of our approach.

References

[AGC13] ASAFI, SHMUEL, GOREN, AVI, and COHEN-OR, DANIEL. "Weak convex decomposition by lines-of-sight". *CGF*. 2013 2.

[AMSF08] ATTENE, MARCO, MORTARA, MICHELA, SPAGNUOLO, MICHELA, and FALCIDIENO, BIANCA. "Hierarchical convex approximation of 3D shapes for fast region selection". *CGF*. 2008 2.

[And24] ANDREWS, JAMES. "Navigation-driven approximate convex decomposition". *TOG*. 2024, 1–9 2.

[Ber99] BERGEN, GINO VAN DEN. "A fast and robust GJK implementation for collision detection of convex objects". *Journal of graphics tools* (1999) 1.

[Cha81] CHAZELLE, BERNARD M. "Convex decompositions of polyhedra". *ACM symposium on Theory of computing*. 1981 1.

[DSS*23] DEITKE, MATT, SCHWENK, DUSTIN, SALVADOR, JORDI, et al. "Objaverse: A universe of annotated 3d objects". *CVPR*. 2023 3.

[GALL13] GHOSH, MUKULIKA, AMATO, NANCY M., LU, YANYAN, and LIEN, JYH-MING. "Fast approximate convex decomposition using relative concavity". *Computer-Aided Design* (2013) 2.

[GJK02] GILBERT, ELMER G, JOHNSON, DANIEL W, and KEERTHI, S SATHIYA. "A fast procedure for computing the distance between complex objects in three-dimensional space". *IEEE Journal on Robotics and Automation* (2002) 1.

[GXL*23] GU, JIAYUAN, XIANG, FANBO, LI, XUANLIN, et al. "Maniskill2: A unified benchmark for generalizable manipulation skills". *arXiv preprint arXiv:2302.04659* (2023) 2.

[HCL*23] HUANG, YI-HUA, CAO, YAN-PEI, LAI, YU-KUN, et al. "NeRF-texture: Texture synthesis with neural radiance fields". *TOG*. 2023 2.

[KFK*14] KAICK, OLIVER VAN, FISH, NOA, KLEIMAN, YANIR, et al. "Shape segmentation by approximate convexity analysis". *TOG* (2014) 2.

[KS24] KUŞKONMAZ, ONAT ZEYBEK and SAHILLIOĞLU, YUSUF. "A surface-based approach for 3D approximate convex decomposition". *Turkish Journal of Electrical Engineering and Computer Sciences* 32.6 (2024), 774–789 2.

[LA04] LIEN, JYH-MING and AMATO, NANCY M. "Approximate convex decomposition". *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. 2004 1, 2.

[LHG*24] LI, XUANLIN, HSU, KYLE, GU, JIAYUAN, et al. "Evaluating real-world robot manipulation policies in simulation". *arXiv preprint arXiv:2405.05941* (2024) 2.

[LLL10] LIU, HAIRONG, LIU, WENYU, and LATECKI, LONGIN JAN. "Convex shape decomposition". *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010 2.

[LXL16] LIU, GUILIN, XI, ZHONGHUA, and LIEN, JYH-MING. "Nearly convex segmentation of polyhedra through convex ridge separation". *Computer-Aided Design* (2016) 2.

[MG09] MAMOU, KHALED and GHORBEL, FAOUZI. "A simple and efficient approach for 3D mesh approximate convex decomposition". *ICIP*. 2009 2.

[MLP16] MAMOU, KHALED, LENGYEL, E, and PETERS, A. "Volumetric hierarchical approximate convex decomposition". *Game engine gems* (2016) 2, 3.

[RYLL11] REN, ZHOU, YUAN, JUNSONG, LI, CHUNYUAN, and LIU, WENYU. "Minimum near-convex decomposition for robust shape representation". *ICCV*. 2011 2.

[Sno17] SNOEYINK, JACK. "Point location". *Handbook of discrete and computational geometry*. 2017 1.

[TLJP18] THUL, DANIEL, LADICKY, LUBOR, JEONG, SOHYEON, and POLLEFEYS, MARC. "Approximate convex decomposition and transfer for animated meshes." *TOG* (2018) 2, 3.

[WLLS22] WEI, XINYUE, LIU, MINGHUA, LING, ZHAN, and SU, HAO. "Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search". *TOG* (2022) 2, 3.

[XQM*20] XIANG, FANBO, QIN, YUZHE, MO, KAICHUN, et al. "Sapien: A simulated part-based interactive environment". *CVPR*. 2020 3.