DOI: 10.1111/cgf.70411
EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Convex Primitive Decomposition for Collision Detection

Julian Knodt and Xifeng Gao

Lightspeed Studios, Bellevue, Washington, USA

Abstract

Creation of collision objects for 3D models is a time-consuming task, requiring modelers to manually place primitives such as bounding boxes, capsules, spheres, and other convex primitives to approximate complex meshes. While there has been work in automatic approximate convex decompositions of meshes using convex hulls, they are not practical for applications with tight performance budgets such as games due to slower collision detection and inability to manually modify the output while maintaining convexity as compared to manually placed primitives. Rather than convex decomposition with convex hulls, we devise an approach for bottom-up decomposition of an input mesh into convex primitives specifically for rigid body simulation inspired by quadric mesh simplification. This approach fits primitives to complex, real-world meshes that provide plausible simulation performance and are guaranteed to enclose the input surface. We test convex primitive decomposition on over 60 models from Sketchfab, showing the algorithm's effectiveness. On this dataset, convex primitive decomposition has lower one-way mean and median Hausdorff and Chamfer distance from the collider to the input compared to V-HACD and CoACD, with less than one-third of the complexity as measured by total bytes for each collider. On top of that, rigid-body simulation performance measured by wall-clock time is consistently improved across 24 tested models.

1. Introduction

Collision objects are an important component of 3D games, defining the interactions between the player and the world. The current flow for game developers and artists for constructing collider meshes can be slow and time-consuming, requiring manual placement of primitives such as capsules, boxes, and spheres, or construction of convex hulls and back-and-forth tuning of performance. Prior work in this area focuses on convex decomposition of input meshes [WLLS22, LA07, MG09, TLJP18] into convex hulls, and there has not been much emphasis on fitting primitives such as boxes, capsules, and other convex primitives shapes to approximate an input mesh. On the other hand, support within physics engines such as PhysX [Nvi17] for collision with primitives (boxes, capsules, spheres) is heavily optimized, and is considered faster than convex hulls. Because of this belief, artists will manually build colliders despite the effort required to do so, especially compared to the minimal effort required to use prior fully-automatic collider mesh generation approaches.

Previous approaches for automatic collision object construction focus mostly on convex hull decomposition [WLLS22, MG09, TLJP18, And24] for static and animated meshes. These approaches generate approximately convex components for an input mesh based on concavity metrics which measure how concave each component is. Prior work generates a small number of convex components which look good at first glance, but suffer from a few practical problems. First, the decomposition may hide complexity that slows simulation, as each convex component can have an arbitrary number of faces and vertices. Second, decomposition is difficult to control, sometimes producing poor approximations in concave regions, filling holes in empty space, and cutting up planar surfaces. Finally, the output primitives are not similar to artist created colliders, due to reliance on plane cutting, asymmetry of output shapes, and over-complexity of each component. While this final reason may not appear a problem, it prevents artists from easily modifying colliders using tools such as Blender [Ble18] or Unreal Engine [Epi22]. For these reasons, we diverge from the prior approach of approximate convex decomposition and focus instead on fitting a subset of parametric convex primitives to represent an input mesh, which can be easily manipulated with existing tools and closely matches artist's manual workflow.

Our approach takes inspiration from Quadric Mesh Reduction [GH97, Hop99, GH98] and Spherical Quadric Error Metrics [TGB13, TGBE16], with each mesh face representing a primitive. These primitives are then greedily combined together bottom-up, producing a simplified representation. This simplified representation, consisting of a set of primitives that cover the surface of an input mesh, is similar to Oriented Bounding Box Trees [GLM96], while allowing for more diversity in primitives.

Our approach is efficient and can compute an arbitrary number of primitives for meshes that have millions of faces. The meshes produced are suitable for accurate and efficient collision detection, which we verify using an off-the-shelf collision simulation.

We collect and test our approach on a dataset of models from Sketchfab [Ske22], showing our approach is efficient and

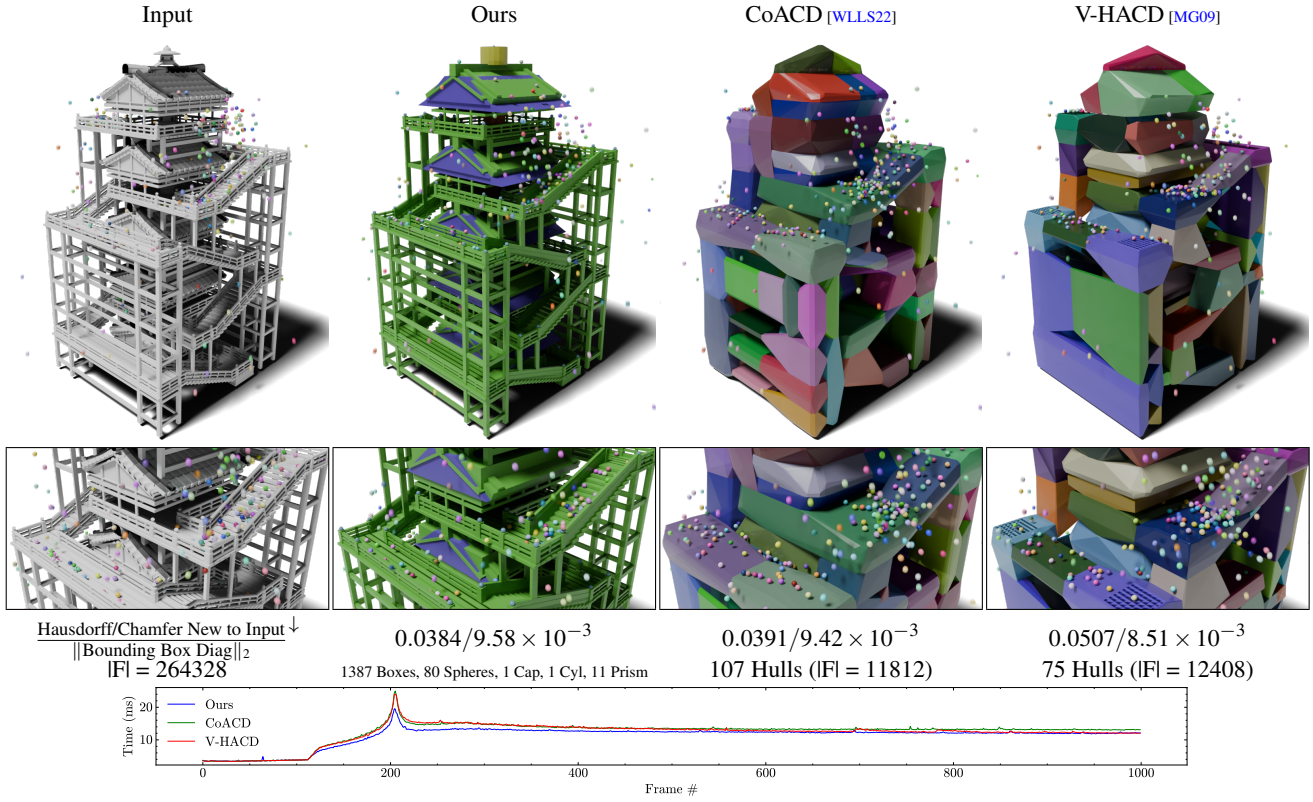


Figure 1: *Convex primitive decomposition on a complex non-manifold mesh with boundaries. Output primitives tightly adhere to the input, and is more efficient in simulation than convex hulls. To test performance, we drop spheres on each collider and measure frame times↓. Our approach has the closest simulation to the original mesh (i.e. balls in the stairwell), but with higher efficiency than CoACD or V-HACD shown in the plot, where x-axis indicates the frame number, y-axis indicates time taken. Green in our approach indicates a bounding box, yellow a cylinder, dark blue a trapezoidal prism, light blue a sphere, and red a capsule. For CoACD and V-HACD, colors are randomly assigned per convex component. Rendering artifacts on V-HACD are due to flipped faces.*

robust in practice. We compare our approach to prior approximate convex decomposition algorithms V-HACD [MG09] and CoACD [WLLS22] on distance from the input mesh, complexity of output collider, and performance in downstream collision detection simulation.

In summary, we hope to demonstrate that convex primitive decomposition serves to fill the gap between research in creating efficient colliders and actual artistic modeling of collider meshes for rigid body simulation. Our tests validate that our approach is suitable for collision detection and more closely adheres to the input mesh than prior work while at the same time has reduced complexity and better performance.

2. Related Work

2.1. Quadric Mesh Reduction

Our approach is loosely based on Quadric Mesh Reduction [GH97, GH98, Hop99, TK20], which reduces the number of triangles through edge collapse [LT98, HDD*93] by representing vertices of triangles as linear operators of the form $Q_v = \Sigma pp^\top$, with $p \in \mathbb{R}^4$ defined as the equation of the plane of each face containing vertex $v \in \mathbb{R}^3$. These linear operators can be efficiently combined as $Q^* = Q_1 + Q_2$, and the optimal position of a combination of vertices can be computed by solving:

$$\begin{bmatrix} Q_{00} & Q_{01} & Q_{02} & Q_{03} \\ Q_{10} & Q_{11} & Q_{12} & Q_{13} \\ Q_{20} & Q_{21} & Q_{22} & Q_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

Because of the simplicity of combining Q due to its linearity, this approach scales well as the size of the input mesh increases. Edges are merged greedily, based on the error $\mathbf{v}^\top (Q_{v_0} + Q_{v_1})\mathbf{v}$, and are used to build progressively coarser representations of the input mesh, similar in spirit to [GH97, SGG*00, TGB13]. Quadric mesh reduction is used in industry and research as a way to reduce the number of elements in complex meshes. It has also been extended to handle other attributes on meshes, including vertex colors, normals, and texture coordinates. Quadrics have been applied to many problems, including handling topology computations that rely on mass [HLW24], shape reconstruction [ZBCS*23], spectral simplification [LLT*20], and vertex positions in dual contouring [JLSW02].

Quadrics have also been adapted to handle approximation of meshes as a linear interpolation of spheres [TGB13, TGBE16]. Instead of the standard quadric metric, a newly proposed quadric

metric is based on the signed distance to a sphere: $d = \mathbf{n}^\top(\mathbf{p} - \mathbf{c}_{\text{sphere}}) - r_{\text{sphere}}$. From this formulation, an approach that solves for the center $\mathbf{c}_{\text{sphere}}$ and radius r_{sphere} is derived, and elements can be merged together in a similar way as standard quadrics. Unlike standard quadrics though, the cost function is replaced with the sum of squared oriented distances of each sphere to the faces of the vertices that it subsumes. Spherical Quadrics (SQEM) lead to a coarse approximation of the input mesh as an interconnected series of spheres, which can then be used for animation. Because of its simplicity and similarity to our goal of abstracting an input with coarse shapes, we draw inspiration for our approach from Spherical Quadric Error Metrics. Note that SQEM is not comparable to our method in rigid body simulation, since the parameters of spheres are linearly interpolated along mesh faces and edges and thus it cannot be represented as discrete components for collision.

Our approach builds on the edge collapse operator, replacing the vertices usually merged together for mesh simplification with primitives for each face that are progressively merged together to form a larger primitive. Our formulation is also simpler than normal quadrics, as we only need to maintain a single matrix in $\mathbb{R}^{3\times3}$.

A few mesh simplification approaches in theory could be applied to collision detection [CB17, SGG*00], but in practice triangle meshes used directly in rigid body simulation often lead to objects clipping through faces and poor performance.

2.2. Shape Abstraction

Another approach to simplifying meshes is abstraction using simpler geometric primitives. Geometric primitives, often represented as parametric models such as signed distance functions, quadrics, or explicit primitives, are used in constructive solid geometry for computer aided design [Lys07] along with parametric curves [MZL*09]. There are a number of different approaches to recovering shapes from input meshes such as random sampling (RANSAC) [SWK07, BBN*20] and data-driven approaches [SZTL19, PUG19, LWRC23, LLY*23]. These approaches primarily work on simple inputs and cannot preserve high frequency details. One work related to our approach is [YLW06], which fits quadrics to a surface with Lloyd's algorithm and a Euclidean distance metric. Their fitting also requires the computation of eigenvectors to fit quadrics, distinct from our approach which uses them for alignment only. A number of works stemmed from that work [LCWK07]. Another similar work is [AFS06] which seeks to perform shape approximation by merging faces, similar to our approach. In contrast though, this prior work only shows 3 shapes with custom cost functions per shape (whereas this work shows 6 with the same cost function), and is not guaranteed to enclose the input shape. Furthermore, that work does not have a specific target application and tested on a much smaller dataset, whereas this work has a large emphasis on validation specifically for collision detection.

There are also many works using clustering for shape abstraction. For such approaches, there are a few design choices. First is choice of clustering algorithm, usually Lloyd's algorithm/iterative region growing [YLW06, YWLY12, JKS05, LCWK07, MDKK07, WHX*22], a greedy approach which merges elements together bottom up (similar to our approach) [GWH01, GH97, YJ20, AFS06], or splitting approaches [PS24, ZLGW15]. Prior work may target different outputs, such as clustering faces into primitives represented by quadrics [YLW06, YWLY12, BBN*20], a small non-general set of primitives [BK02, YJ20, LD17, ZLGW15, LCWK07], oriented bounding boxes [GLM96, PS24, WHX*22, LCWK07, BH11], and ellipsoids [LCWK07, BK02]. Our approach is close to prior work that produces primitives, but our approach is more general as any parametric shape that satisfies a simple interface can be used. This includes, spheres, capsules, boxes, frustums, trapezoidal prisms and cylinders, which no previous approach can produce together. Furthermore, prior work relies on primitive specific features for fitting. Our work introduces the use of the eigendecomposition for the primitive's orientation, which can be used to fit other parameters using the enclosed points, with less reliance on primitive specific features. Our work also focuses on the specific problem of rigid body simulation. For a comprehensive analysis on prior work, see a recent survey on geometric primitive fitting [KYZB19].

2.3. Approximate Convex Decomposition

Prior work on convex decomposition for collision detection has revolved around approximate convex decomposition [KJS07, MG09, And24, LA07, TLJP18, KFK*15, AMSF08]. Approximate convex decomposition relies on splitting an input mesh into convex hulls, approximating the shape of the input mesh using a concavity metric to determine whether to a split a shape into subparts. Many of these approaches remesh or voxelize the input to make it manifold, then partition the manifold mesh top-down along cutting planes, tightly matching the input shape. What we found when investigating whether this approach is suitable for game development is that convex hulls are more complex than the count of hulls indicates, can occasionally be imprecise, and are not easy to modify within existing DCC tools or engines while maintaining convexity. This motivates our divergence from convex hull decomposition.

One component of prior work (and specifically CoACD) is their use of plane-cutting of hulls into separate components, relying on Monte Carlo search to find a good cutting plane. Due to the infinite number of cutting plane candidates, prior approaches cut on a limited set of axes, leading to excessive and poor cuts. We evade this problem entirely by performing bottom-up merging of primitives. This allows us to take into account local coordinate frames, ridding ourselves of suboptimal axis-aligned constraints.

We also note that our approach's error metric using volume differences is built on prior works such as [TLJP18] which uses volume to determine when to split a hull. One key difference is that prior work often requires watertight meshes to compute the volume of the input mesh. Our approach does not require a watertight input, since we use the volume of the already computed primitives rather than the mesh itself.

3. Method

Given an input mesh $\mathbb{M} = (V, F), \mathbf{V}_i \in \mathrm{R}^3, F_i \subseteq V, |F_i| \geq 3$, with each face a polygon of three or more vertices, we output a set of primitives $P, |P| \leq N$ where $N \in \mathbb{Z}_+$ is a user-defined target positive number of primitives. In our implementation, P can be a capped

cylinder, a capsule, a sphere, an isosceles trapezoidal prism, a frustum, or an oriented bounding box. Our approach follows traditional quadric mesh reduction [GH97] and constructs a linear operator, in our case a 3×3 matrix, corresponding to each primitive. Linear operators support addition, $(f+g)(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$, and scalar multiplication, $kf(\mathbf{x}) = f(k\mathbf{x}), k \in \mathbb{R}$. **Bold text** indicates a vector in $\mathbb{R}^3$.

We outline our approach, starting from initializing linear operators (Sec. 3.1) for each face of the input mesh, corresponding to one primitive per face, and describe how to convert linear operators into primitives in Sec. 3.2. We then describe our greedy approach to combine topologically adjacent linear operators, using the input mesh's face adjacency as topology. Akin to QEM's edge-collapse, we use a minimal excess volume cost function to select primitives to merge (Sec. 3.3), and terminate once there are either no more elements to simplify or a user-defined criteria is met (Sec. 3.3). We discuss the implementation details in Sec. 3.4, and the full algorithm for our approach is given in Alg. 1.

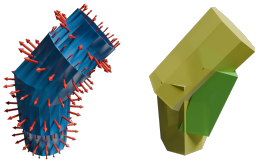
Algorithm 1 Convex Primitive Decomposition

Input: Mesh = $V \in \mathbb{R}^3, F_i \subseteq V$, # Prims N$\in \mathbb{Z}_+$
Volume Threshold $M \in \mathbb{R}_+ \overset{\text{default}}{=} \inf$
Output: Primitives P s.t. $|P| \leq$ N
Preprocessing ▷ Remove overlapped vertices
$\mathbf{n}_i = \text{normal}(F_i)$, $\mathbf{t}_i = \text{tangent}(F_i)$
$Q_i = \text{area}(F_i)(\mathbf{n}_i\mathbf{n}_i^\top + \epsilon\mathbf{t}_i\mathbf{t}_i^\top)$, $P_i = \text{Prim}(Q_i)$, $\text{Vol}(f_i) = \text{Vol}(P_i)$
pq : Priority Queue
for adjacent faces $f_0, f_1 \in$ F **do** ▷ Initialize Priority Queue
  $P^* = \text{Prim}(f_0 + f_1)$
  if Vol(P^*) - (Vol(f_0) + Vol(f_1)) > M **then continue**
  pq.push(priority = Vol(P^*) - (Vol(f_0) + Vol(f_1)), (P^*, f_0, f_1))
while !pq.empty() **and** $|P| >$ N **do** ▷ Greedily Merge
  (P^*, f_0, f_1) = pq.pop()
  Vol(f_0) = Vol(f_1) = Vol(P^*)
  $P_{f_0} = P_{f_1} = P^*$ ▷ Set both faces to new primitive
  Update Costs of Adjacent Primitives
Postprocessing ▷ Remove primitives enclosed by other primitives
return Unique Primitives P_i

3.1. Linear Operator

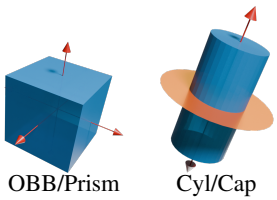
Our linear operator corresponding to a primitive is a 3×3 area-weighted matrix per face of the input mesh, Q. We define $Q = \mathbf{n}\mathbf{n}^\top$, $\mathbf{n}^\top\mathbf{n} = 1, \mathbf{n} \in \mathbb{R}^3$. $\mathbf{n}\mathbf{n}^\top \in \mathbb{R}^{3\times3}$ is the outer product of the normal of a face with itself, which characterizes each face's plane. Q is a subset of QEM's metric, $\text{QEM}(\mathbf{x}) = \mathbf{x}^\top(\mathbf{n}\mathbf{n}^\top)\mathbf{x} - 2\mathbf{n}^\top(\mathbf{p}^\top\mathbf{x})\mathbf{x} + (\mathbf{p}-\mathbf{x})^\top(\mathbf{p}-\mathbf{x})$, with the linear and constant terms dropped since they identify position. We do not need positions since we use the vertices within each primitive to compute positions directly. To derive a primitive from Q, we use Q's eigendecomposition $Q = W\Lambda W^{-1}, \Lambda = \text{diag}(\begin{bmatrix}\lambda_2 & \lambda_1 & \lambda_0\end{bmatrix}), W = \begin{bmatrix}\mathbf{w}_2 & \mathbf{w}_1 & \mathbf{w}_0\end{bmatrix}^\top, |\lambda_2| \geq |\lambda_1| \geq |\lambda_0|$: for a single face, Q's largest eigenvector $Q\mathbf{w}_2 = \lambda_2\mathbf{w}_2$ is the face's area-weighted normal. When adding another face $Q_a + Q_b$, the largest eigenvector $\mathbf{w}_{2,a+b}$ corresponds to the largest shared direction of both faces' normals, and the second $\mathbf{w}_{1,a+b}$ will be the shared component orthogonal to the first. When adding faces, the largest eigenvector corresponds to the largest area-weighted direction that the faces are oriented towards, and the others will be orthogonal. The orthogonal basis of eigenvectors $\mathbf{w}_2, \mathbf{w}_1, \mathbf{w}_0$ defines an oriented bounding box which bounds the corresponding set of faces. We provide a visualization of Q defined per input face and the eigendecomposition of output primitives in the inset figure. An analysis of this operators eigendecomposition and its relation to principal curvature is provided in [GH99].

A visualization of the area weighted Q of an input mesh (shown in blue), and our approach's output primitives. The output primitives from our approach are aligned with the faces they enclose. In the output, cylinders are yellow, oriented boxes are green, and eigenvectors of the input Q for each face are red arrows.



Input |F| = 216 | Output 1 Box, 2 Cyl.

We arrived upon this generalization while crafting an operator for cylinders. Instead of more parameters, the smallest eigenvalue's eigenvector closely aligns with the cylinder's axis. We then observed other eigenvectors align with reasonable bounding boxes for shapes and can serve as orientations. This operator identifies an orientation that aligns closely with the largest area-weighted tangent and normal of the enclosed faces. This operator is not globally optimal, but gives good approximations. An illustration of the eigenvectors that correspond to primitives is shown in the inset below.



OBB/Prism | Cyl/Cap

For OBBs and isosceles trapezoidal prisms eigenvectors align with faces. For capsules and cylinders, one eigenvector is aligned with the cylinder's direction, and other eigenvectors will lie orthogonal to this axis.

Face Normal Computation For a triangle $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, the normal is given by the cross-product of the edges: $\mathbf{n}_{\text{tri}} = (\mathbf{v}_2 - \mathbf{v}_0) \times (\mathbf{v}_1 - \mathbf{v}_0)$. For quads (faces with four vertices) the face's normal is not well-defined if the quad is non-planar. To compute an approximate normal for both planar and non-planar quads, we define the normal of a quad $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ as the cross-product of the diagonals: $\mathbf{n}_{\text{quad}} = \frac{(\mathbf{v}_0-\mathbf{v}_2)\times(\mathbf{v}_1-\mathbf{v}_3)}{\|(\mathbf{v}_0-\mathbf{v}_2)\times(\mathbf{v}_1-\mathbf{v}_3)\|_2}$. For planar quads, this is exactly correct. For polygons with more than four vertices, we create a triangle fan rooted at the 0th vertex with separate normals per triangle.

3.2. Primitive Construction from Linear Operators

To convert Q into a primitive, we first decompose it into its eigenvectors: $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^3, \mathbf{w}_i^\top\mathbf{w}_j = 0$ iff $i \neq j, \mathbf{w}_i^\top\mathbf{w}_i = 1$, $Q\mathbf{w}_i = \lambda_i\mathbf{w}_i, |\lambda_0| \leq |\lambda_1| \leq |\lambda_2|$, implemented using the eigendecomposition described in [MST*11]. With these eigenvectors as the axes,

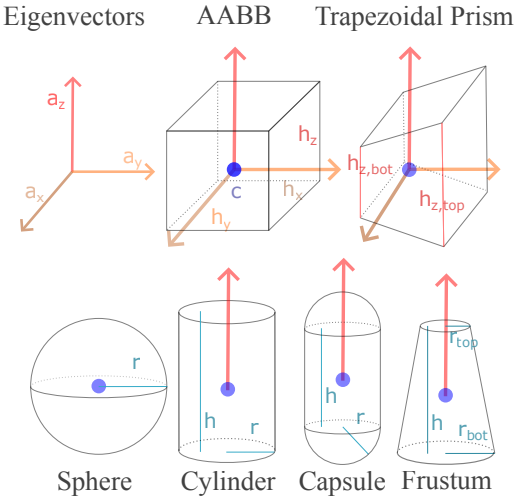


Figure 2: *Primitives supported in our approach. First, axes are computed as the eigendecomposition of a quadric, a_z corresponds to the minimum eigenvalue's eigenvector, a_x the max. These axes are then used to compute an oriented bounding box (OBB). The OBB's center c and the axes are then used to fit all other primitives.*

we fit all additional parameters for each primitive by expanding them to fully enclose the corresponding mesh faces.

Our approach supports arbitrary primitives which satisfy an interface. Primitives must have efficient volume computation, be constructible as zero-volume elements with a given orientation and position, and support strictly increasing their parameters to enclose an unordered set of points. For our implementation, we use oriented bounding boxes, spheres, capped cylinders, capsules, frustums, and isosceles trapezoidal prisms, shown in Fig. 2. We outline our implementation of the interface for each primitive below. For each primitive, the points it subsumes are denoted as $\mathbf{p}_i \in \mathbb{R}^{n\times 3}$. We informally define *subsuming* as when a primitive *must* enclose a point. This does not include points primitives enclose but are not required to.

Oriented Bounding Boxes Oriented bounding boxes are the foundation of all other primitives since they do not require an initial position and their position can be used for other primitives. The bounds ($u_x, u_y, u_z, l_x, l_y, l_z \in \mathbb{R}$) of OBBs are computed per axis as $u_i = \max_{\mathbf{p}\in P}(\mathbf{v}_i^\top \mathbf{p})$ and $l_i = \min_{\mathbf{p}\in P}(\mathbf{v}_i^\top \mathbf{p}), i \in \{x,y,z\}$. The half-extent along each axis is $h_i = \max(\frac{1}{2}(u_i - l_i), 1\times 10^{-3})$, where 1×10^{-3} is included to prevent degenerate volumes. We apply the same clamping to parameters for other shapes. The center of the OBB is $c_i = \frac{1}{2}(u_i + l_i)$, the volume is $8\Pi_{i=1}^{3} h_i$, with a factor of 8 since h_i are half-extents. OBBs are fit first, as other primitives initialize their position to the center of the OBB.

Spheres Spheres are the simplest primitive, starting from a fixed center $\mathbf{c}$ (from the previously computed OBB), the radius is $r = \max_{\mathbf{p}\in P}(\|\mathbf{c}-\mathbf{p}\|_2)$. The volume of a sphere is $\frac{4}{3}\pi r^3$. Spheres are the cheapest primitive for collision detection, and are used despite their coarse shape.

Capped Cylinders For capped cylinders with a fixed axis $\mathbf{a}, \mathbf{a}^\top\mathbf{a} = 1$ and a point on the axis, $\mathbf{p}_{\text{cyl}}$, we derive the height and radius:

$$\begin{aligned} \mathrm{r} &= \max_{\mathbf{p}\in P}(\|(\mathrm{I}-\mathbf{a}\mathbf{a}^\top)(\mathbf{p}-\mathbf{p}_{\text{cyl}})\|_2) \\ \mathrm{h} &= \max_{\mathbf{p}\in P}(\mathbf{a}^\top(\mathbf{p}-\mathbf{p}_{\text{cyl}})) - \min_{\mathbf{p}\in P}(\mathbf{a}^\top(\mathbf{p}-\mathbf{p}_{\text{cyl}})) \end{aligned} \tag{2}$$

The cylinder's volume is $\pi r^2 h$. We compute 1 cylinder per axis, and choose the cylinder with minimal cost.

Capsules Capsules are similar to cylinders, with a fixed axis $\mathbf{a}$ such that $\mathbf{a}^\top\mathbf{a} = 1$ and point on $\mathbf{a}$, $\mathbf{p}_{\text{cap}}$, the radius computation is the same as a capped cylinder. There is a slight difference for the height, taking into consideration that the ends of the capsules are spheres, and the equation for the height is adjusted as follows:

$$\begin{aligned} r(\mathbf{p}) &= \|(\mathrm{I}-\mathbf{a}\mathbf{a}^\top)(\mathbf{p}-\mathbf{p}_{\text{cap}})\|_2 \\ \mathrm{h}(\mathbf{p}) &= \mathbf{a}^\top(\mathbf{p}-\mathbf{p}_{\text{cap}}) - \sqrt{r^2 - r(\mathbf{p})^2} \\ \text{height} &= \max_{\mathbf{p}\in P}(\mathrm{h}(\mathbf{p})) - \min_{\mathbf{p}\in P}(\mathrm{h}(\mathbf{p})) \end{aligned} \tag{3}$$

Capsules are supported in downstream physics applications due to their ease of computing distance, which is why we include them. Similar to cylinders, we compute one capsule per axis.

Frustum To capture conical structures, we also implement frustums. Frustums are initialized from the axis of the minimum cost cylinder, with a position on the base $\mathbf{p}$, and axis $\mathbf{a}$. The height is set identically to cylinders. The radius of the top and bottom faces are set using the algorithm shown in. Alg. 2. The volume of the output frustum is given by $\frac{\pi h}{3}(r_{\text{top}}^2 + r_{\text{top}} r_{\text{bot}} + r_{\text{bot}}^2)$.

Isosceles Trapezoidal Prisms Finally, we implement isosceles trapezoidal prisms. No prior work uses explicit trapezoidal prisms, but we found buildings often have gable roofs that resemble a triangular prism and are poorly modeled by other primitives. We found isosceles triangular prisms to be numerically unstable because of the singularity (zero-width side) for subsuming sets of points. Instead, isosceles trapezoidal prisms can closely approximate a triangular prism and are numerically stable.

Given the orthogonal basis $\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z$ and center $\mathbf{c}$, we check all 6 possible orderings of axes and compute the half-extents ($h_x, h_y, h_{zt}, h_{zb} \in \mathbb{R}$), where h_{zt}, h_{zb} are along the axis $\mathbf{a}_z$ on $+\mathbf{a}_y, -\mathbf{a}_y$ respectively. h_x and h_y are computed identically to OBBs as $\max(\mathbf{a}_x^\top(\mathbf{p}-\mathbf{c}))$ and $\max(\mathbf{a}_y^\top(\mathbf{p}-\mathbf{c}))$. Pseudocode for computing h_{zt}, h_{zb} is given in the Appendix, Alg. 3. The volume of the resulting trapezoidal prism is $4h_x h_y(h_{zt} + h_{zb})$, with the factor of 4 as $h_\bullet$ are half-extents.

Our construction of isosceles trapezoidal prisms is fast because it uses an unordered stream of points without allocation, but is not optimal. We chose this tradeoff as it is recomputed every collapse.

An implementation can be extended with additional shapes depending on what is accepted in the downstream application. Our approach does not require that primitives be optimal, since only

relative costs between primitives matters. Furthermore, for specific kinds of shapes it is possible that more optimal algorithms can be used, such as exact OBBs and spheres [O'R85], but for the other primitives exact algorithms are unknown. Since only a few primitives have exact algorithm our approach uses a more general fit.

Face-Based Mesh Reduction [GH97] and [TGB13] rely on merging vertices to approximate positions on the input mesh, and the surface is defined by the edges and faces connecting these vertices, but our approach uses discrete primitives to represent the input's surface. Allowing each vertex to be covered by only one primitive does not cover the surface, leaving holes in the output. For example, consider the output primitives if no edges are contracted: the output would be small primitives around each vertex, a poor approximation of the input shape. To remedy this, primitives subsume *faces* of the mesh, similar to [GWH01]. Each primitive covers all vertices of each face it subsumes, allowing vertices to be covered by multiple primitives and the union of all primitives covers the mesh's surface. We show the poor output of using one primitive per vertex in Sec. 5, Fig. 11.

3.3. Optimal Primitive Selection

Given possible primitives that enclose a point set, we must decide which primitive is best. We measure best in two ways: the geometric tightness of the primitive, as measured by the additional volume introduced when merging two shapes and an abstract cost function for downstream applications, such as the cost of collision detection. For our implementation, we select parameters that balance geometric similarity and support in physics engines.

Collapse Cost Function Our end-goal is to approximate the input mesh as closely as possible for collisions; we want collision primitives to be as tight-fitting to the input surface as possible. To measure this, we minimize the excess volume introduced by each primitive merge. This leads to the following cost function for merging two primitives p_0, p_1:

$$C(p_0,p_1) = V(\text{merge}(p_0,p_1)) - (V(p_0)+V(p_1)) \tag{4}$$

where $V(p)$ is the volume defined by the primitive p. This cost function penalizes *excess* volume introduced by merging two primitives, and promotes removing primitives which overlap. When this cost-function is 0, there is no penalty to merge two primitives together, and when it is negative it will reduce the total volume of the shape. Usually, this cost-function will be positive, indicating an increase in volume due to merging. Like QEM, we collect all edges between mesh elements in one priority queue, and iteratively take the minimum cost contraction. For each mesh element, we store the volume of its primitive and update the volume after each collapse.

Overlapping Primitives Eq. 4 does not account for double-counted volume of intersecting primitives, which would instead lead to the following cost function:

$$C(p_0,p_1) = V(\text{merge}(p_0,p_1)) - (V(p_0)+V(p_1)-V(p_0 \cap p_1)) \tag{5}$$

This primary reason we do not include the volume of the intersection is because such a computation is expensive, especially if computed exactly using mesh booleans, and we experimentally find little quality increase when adding in this term. Even an approximate approach, such as rejection sampling, incurs a high penalty. In practice, primitives do not overlap much, especially for clean meshes designed for games. Despite Eq. 4's simplicity, it performs well in practice, and we leave exploration of alternatives to future work. We ablate Eq. 4 against Eq. 5 using rejection sampling in Sec. 5, Fig. 16, and find minuscule improvement but high computational cost, at least 30× the wall-clock time.

Handling Different Primitive Costs. In downstream applications, geometric fit is not the only consideration, as primitive variants incur different costs. For example, physics engines have primitives for boxes, capsules, and spheres giving them better performance. When merging primitives based only on volume though, prisms often have lower volume than OBBs, a subset of trapezoidal prisms with $h_{zt} = h_{zb}$, which may lead to lower simulation performance. To incorporate preferences over shape variants, we combine our cost function with a user-provided weighting: $V'(p) = k(p)V(p)$. $k(p)$ depends on the type of primitive, and can be set per target application. We use a weight of 1.05 for cylinders, 1.4 for trapezoidal prisms, 1.0 for capsules, spheres and bounding boxes, and 2.1 for frustums. Our weighting heavily prefers bounding boxes, capsules and spheres due to their support in physics engines. We penalize cylinders, as when they have small radius and large height there is little visible difference between cylinders and capsules, but capsules are more performant. Depending on use case, this weighting can be tuned, but these values strike a good balance between performance and geometric accuracy. We ablate our choice of weights as compared to a uniform and alternative weighting in Sec. 5, Fig. 14.

The choice of primitives in downstream applications is dependent on use case. For example, in a platformer or first-person shooter which requires precise control it may be better to use expensive colliders to keep accuracy. For cases such as a puzzle game, it may be fine to use coarser boxes. We believe that currently there is no one correct answer for which primitives to use in all cases, and that consideration must be taken case by case.

Termination Termination happens when there are no more collapsible edges or the target number of primitives is reached. Since our approach always decreases the number of primitives, our approach is guaranteed to terminate. After termination, we output parameters for all primitives or quantize primitives to a mesh.

Like QEM [GH97], our approach relies on the user providing the target number of primitives, which affects the ability to represent the input mesh.

Excess Volume Thresholding Manually tuning the number of primitives gives good results, but may provide insufficient control over order of merges. Furthermore, some merges introduce more volume than merging two disconnected components. To allow better control of order, we add an optional user-defined *excess volume threshold*, relative to the volume of the axis-aligned bounding box of the input. If combining two primitives increases the volume above this threshold, that merge is prevented. This can be used to prune undesirable merges with less manual tuning.

3.4. Implementation Details

Removing Redundant Primitives After termination, we observe that sometimes the points subsumed by one primitive are fully enclosed by a larger primitive, and there is no point to maintain nested primitives since they incur computation without functionality. These primitives can be culled after our approach, by checking if all points subsumed for each primitive are entirely within another primitive. We perform a concurrent pairwise check after simplification and cull internal primitives, with negligible impact to the total execution cost.

Coplanar Vertices For coplanar faces, the quadric may have degenerate eigenvectors, a problem inherited from the original QEM. To fix this, we add a quadric in the tangent space. For quads, we define a tangent basis that follows the directions of the quad. To handle general quads including non-planar ones, we use the following as the tangent direction $\mathbf{t}$ given vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$: $\mathbf{t} = \mathbf{v}_0 - \mathbf{v}_2 + \mathbf{v}_1 - \mathbf{v}_3$. This is the halfway vector between the quad's two diagonals, which follow the direction of the quad's edges if it is regular and planar, and gives a reasonable value otherwise. Quads are specially handled due to their prevalence in game assets.

We treat triangles t with counterclockwise edges $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2 \in \mathbb{R}^3, \|\mathbf{e}_0\|_2 < \|\mathbf{e}_1\|_2 < \|\mathbf{e}_2\|_2$ as half of a regular quad, where $\mathbf{e}_2$ corresponds to one of the quads diagonals, by computing the equivalent of the above equation using the triangle's edges: $\mathbf{t}_0 = \frac{1}{2}(\mathbf{e}_0 - \mathbf{e}_1 + \mathbf{e}_2)$. We flip the sign of the tangent based on the orientation of the vertex $\mathbf{v} \notin e_2$: $\mathbf{t} = (-\mathbf{t}_0)$ if $(\mathbf{v} - \mathbf{v}_{\in e_2}) \cdot (\mathbf{e}_2 \times \mathbf{n}) < 0$ else $\mathbf{t}_0$, where n is the normal of the corresponding face and $\mathbf{v}_{\in e_2}$ is one of the vertices of e_2. This is designed for quad meshes which were triangulated, such as when a mesh is imported into UE [Epi22].

We add the weighted outer product of $\mathbf{t}$ to Q, to help improve stability: $Q' = Q + \epsilon \mathbf{t}\mathbf{t}^\top$. This reduces error in ambiguous cases, but may reduce adherence to the input in other cases. We show an ablation of one example where it improves the output quality in Sec. 5, Fig. 10. We decide per mesh whether to include this factor.

Vertex Deduplication A common pitfall we find when running our approach on uncleaned meshes is that vertices may be overlapped. This can happen due to kitbashing or modeling of parts which are later combined. We show deduplicating such vertices can change the output result in Sec. 5, Fig. 11. Merging these duplicate vertices by distance is similar to non-edge contractions in [GH97] (also known as virtual edge collapses). A recent preprint [LZY24] explored virtual edge collapses between distinct components, and we leave exploration of alternative virtual edge collapses to future work.

Pairwise Component Merging After performing reduction based on the connectivity of primitives, there may be no remaining edges even if the target number of primitives is not met due to disconnected components. To meet the target number of primitives, edges are added to create a fully-connected graph and decimation is continued. Theoretically, this is costly since the number of edges is $\frac{1}{2}|C|(|C|-1) \approx O(C^2)$, where $|C|$ is the number of components in the input. In practice, we observe the number of components for many meshes is small, and all pairs can be computed quickly. In cases where it is intractable, edges can be culled based on excess volume thresholding from Sec. 3.3.

We find if the input mesh has separate components, this may help our approach, as separate components often delineate distinct convex objects which should not be merged, and note that constraining collapses to existing topology has not hindered prior work [Hop99,TGB13,SGG*00,GWH01]. Vertex deduplication and pairwise merges also mitigate effects of input topology.

Collapsible Primitive Data Structure Triangle mesh reduction requires bookkeeping for each faces' indices, but our implementation does not need to retain faces. Instead, we must manage which faces are subsumed by which primitive, for which we use a forest of cyclic linked-lists. We initialize one list per primitive, with each primitive subsuming a single face. When merging two primitives, we select an arbitrary node from each list, and swap pointers to their next element, merging the two lists together. This allows for $O(1)$ merging with efficient iteration of groups at the cost of pointer chasing. We also use Disjoint Set Unions [TvL84] for quickly identifying which faces have been merged together, allowing for fast pruning of deleted edges in the priority queue.

4. Results

To evaluate our approach, we focus on two primary metrics: similarity of simulation of the new mesh to the original (**correctness**), and **efficiency** measured first-and-foremost by timing in simulation, then secondarily the complexity of colliders measured in bytes used, and the number and kind of each component. For correctness, our primary metric is qualitative appearance in simulation. If the simulation plausibly resembles the original, our approach is good enough for downstream usage, since there are no good measures for similarity of simulation. We also measure one-directional distance from points on the surface of the new mesh to the original, which has been used in prior work such as [TLJP18], but we note that other prior works have ignored geometric distance [WLLS22,And24]. We choose to measure the one-directional distance from the new collider mesh to the original, as it penalizes falsely filling in holes and concave regions.

To measure rigid body simulation efficiency, we test collision performance directly in simulation by dropping 5000 spheres on each mesh and measure the frame duration in the first 1000 frames of simulation. Frame duration measurements reflect two things: different outcomes of collision detection and different performance of colliders. We include our rigid body simulator in the supplemental as an executable, with convex decompositions from our approach. We do not use frame-rate (frames per second), because it changes non-linearly with performance, and frame time is easier to reason about when budgeting time while containing equivalent information to frame-rate.

We test our approach on meshes from Sketchfab, which have a mix of triangle, quad, and polygonal faces, and some other data shown in the appendix. We test on props, buildings, characters, plants, and levels/environments. Unlike prior work, our approach handles non-manifold, non-watertight meshes directly without preprocessing. Our focus is primarily on meshes for games which have

clean topology and a mix of tris and quads, but we also test on a few noisy scanned models. For each model, we manually decide how many primitives to keep.

4.1. Efficiency of Collision Objects

To demonstrate the usability of our approach in rigid body simulation, our primary metric is in-simulation wall-time benchmarking to validate collider performance and correctness. We caveat that performance of any simulation is heavily dependent on hardware and physics engine, but trends should be similar across systems. To determine if a collision is similar to the input, we manually inspect the behavior, similar to prior work on simulation.

Simulation details We run all methods on a 8-core AMD Ryzen 7800X3D processor with all processing done on the CPU, using Rapier. While most physics engines support boxes, capsules, and spheres, support for cylinders is mixed, and no engines known to the authors support trapezoidal prisms or frustums. Rapier supports cylinders, otherwise they can be discretized and represented as convex hulls, which is the approach we took for trapezoidal prisms and frustums.

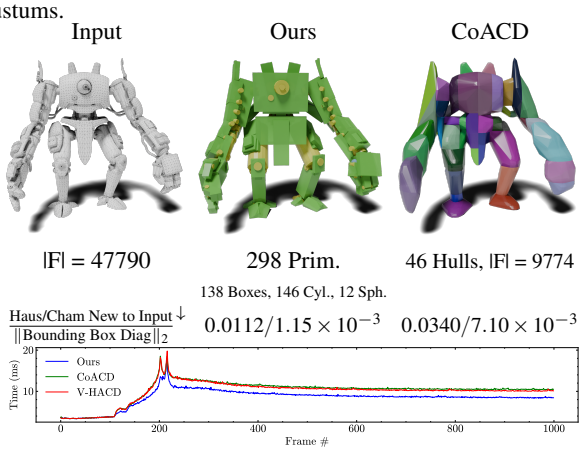


Figure 3: *The Robot Vera model, tested in collision simulation by dropping 5000 spheres on top of it. A snapshot of our test simulation is shown on the right, with the model from our approach. Our approach is closer to the input model than CoACD, while having better performance as shown in the plot above. More model comparisons are in Fig. 20.* JohnMesplay.

We evaluated collision time on 24 meshes such as the robot shown in Fig. 3, with more results in Fig. 20. When comparing our approach with CoACD and V-HACD on the two key factors of simulation similarity and wall-clock time, our approach has more similar collisions as compared to the input with better performance. For example, for the mesh from Fig. 1 our approach holds spheres inside the stairwell and on platforms. On the other hand, CoACD and V-HACD produce coarse outputs, causing spheres to roll off the mesh. This boosts their performance since there are fewer collisions but with inaccurate behavior. Despite our approach being more similar to the input than CoACD and V-HACD, our approach has better performance, often by at least one to two milliseconds per frame. When dealing with a performance budget of 16 milliseconds per frame (60 fps), that saved time is meaningful, giving more time for other computations.

4.2. Comparing Complexity/Costs of Colliders

It is difficult to provide an apples-to-apples comparison of complexity of our approach and CoACD/V-HACD, since their fundamental components are convex hulls with vertices and faces, versus our parametric primitives. Even worse, since capsules, spheres and cylinders do not have a finite number of faces, it is not possible to compare face counts. To demonstrate that we are using fewer resources than CoACD/V-HACD, even though the primitive count is higher than the hull count, we use the lowest common denominator, the number of bytes of each approach. We compute the number of bytes for all methods, showing our approach is less complex, while at the same time is geometrically closer to the input with better simulation wall-clock times. To measure complexity, we count the number of bytes per component as per Tab. 1, with total costs for all models shown in the Appendix, Tab. 5. Note we underestimate the cost of convex hulls by assuming that integers fit in two bytes (`uint_16t`), when in some rarer cases it is necessary to use four bytes (`uint_32t`). When comparing aggregate statistics on our dataset, convex primitive decomposition, CoACD, and V-HACD uses on average **22523.7**, 93809.3, and 68934.1 bytes, and a median of **6362**, 76572, and 44592 bytes respectively. Clearly, convex primitive decomposition creates colliders with fewer resources than approximate convex hull decomposition, with better simulation frame durations than convex hulls. Since each primitive is cheaper than each convex hull, even though our approach has a larger number of primitives compared to the number of hulls, overall it is cheaper. Furthermore, even with an equal number of primitives and hulls, we show primitives have lower wall-clock simulation time in Sec. 5, Fig. 13.

Primitive Kind	Minimum Floats Required	Total	Engine
Oriented Box	3 position, 3 length, 4 orientation	10	Yes
Capsule	3 start-point, 3 end-point, 1 radius	7	Yes
Sphere	3 center, 1 radius	4	Yes
Cylinder	3 start-point, 3 end-point, 1 radius	7	Some
Frustum	3 start-point 3 end-point, 2 radii	8	Quantized
Prism	3 position, 4 length, 4 orientation	11	As Hull
Convex Hull	3 floats per vertex, 3 ints per tri	-	Yes

Table 1: *Memory costs required for each primitive and for convex hulls. Orientations are stored as quaternions. For cylinders, some physics engines have direct support. Frustums are not supported, but can be quantized for usage, and prisms can be represented as convex hulls.*

Implementations of Rigid-Body Collision To understand the performance of rigid body simulation with many primitives/convex hulls, it is useful to understand collision detection's implementation. Collision detection happens in two phases, a broad phase followed by a narrow phase. In the broad phase, many comparisons are culled using coarse checks such as axis-aligned bounding boxes and axis overlap checks. The broad phase is independent of the complexity of each collider, and thus is more agnostic to primitives versus hulls. In the narrow phase, precise algorithms such as GJK [GJK88], or primitive-to-primitive checks are used. Due to

broad phase culling, it is difficult to predict performance, as usually collision detection skips most checks, and there are only a few expensive comparisons. In the worst case though, objects may collide with all primitives in the scene, nullifying the broad phase. In our case, since collision with primitives is cheaper than collision with convex hulls and only a small set of expensive checks are used, convex primitive decomposition is faster than approximate convex hull decomposition.

4.3. Measuring Similarity of Collision Objects

To demonstrate that our approach is robust at producing faithful decompositions, we show results on a variety of cases, including inputs with holes, multiple components, and environments.

We compare our approach's preservation of sharp edges on a fractal shape as compared to CoACD in Fig. 4. On this model, our approach is two orders of magnitude closer to the input than CoACD, since CoACD rounds the edges of the input shape. We also compare the simulation frame duration, showing that even with a more precise decomposition, our approach is faster.

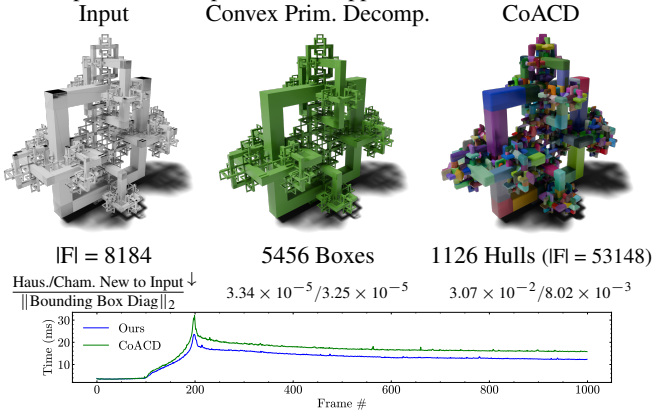


Figure 4: *Our approach decomposes the input mesh into a number of primitives which more closely adhere to the input mesh than CoACD, while allowing faster collision detection. Dark regions on the input mesh are due to overlapping non-manifold faces. Frame duration comparisons for simulation of collision with 5000 dropping balls are shown below; our approach has faster simulation than CoACD on all frames.* CC BY NC SA *Dixbit.*

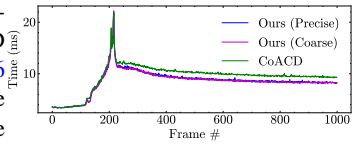
We also demonstrate our approach's generation of colliders with holes by running on a maze while preserving traversability in Fig. 5. To compare fairly to alternatives, we increase CoACD's resolution from 30 units to 80 units, and set the concavity to its minimum value of 0.01. For V-HACD, we use the off-the-shelf settings. While CoACD preserves some holes, the gaps are thinned and edges rounded, whereas our approach preserves most holes. V-HACD closes most holes, making it unsuitable for replacing the original mesh. We visualize differences in simulation, by dropping 500 balls in Blender [Ble18] (5000 is used in simulation, 500 is only for visualization). The balls pass through our output and the original mostly unfettered, whereas CoACD and V-HACD prevent many from going through.

We then demonstrate our approach on environments. For game levels, convex decomposition is overkill since floors and walls are replaced with box colliders. Yet, current tools cannot automatically fit boxes, requiring artists to place them and balance trade-offs in precision versus runtime, as some applications need details such as height changes in the floor and others use one box for performance. To reflect the ability to control this trade-off, we show two granularity of output on an environment in Fig. 6, and one output on the Bistro [Lum17] scene in the Appendix, Fig. 22. Our outputs are visually and quantitatively faithful to the original mesh at both resolutions, and preserve items such as the barrels and boxes in Fig. 6, and wine glasses and utensils in Fig. 22. The variation in the floors and walls show how our approach can be used to tune precision. By comparison, convex decomposition smooths over the props and floor with arbitrary cuts, giving implausible simulation.

Summary Statistics	Ours	CoACD	V-HACD
#Models w/ Lowest 1-way Hausdorff↑	44	20	4
#Models w/ Lowest 1-way Chamfer↑	43	3	22
Mean 1-way Chamfer↓	6.95×10^{-3}	9.91×10^{-3}	8.82×10^{-3}
Median 1-way Chamfer↓	5.40×10^{-3}	9.41×10^{-3}	7.60×10^{-3}
Mean 1-way Hausdorff↓	0.0445	0.0514	0.0710
Median 1-way Hausdorff↓	0.0346	0.0383	0.0593

Table 2: *Our approach has more lower 1-way distances on our dataset from the generated mesh to the input mesh as compared to CoACD [WLLS22] and V-HACD [MG09].*

We measure frame duration of simulation of CoACD and our approach from Fig. 6 shown in the inset. Our coarse and precise meshes both have better performance than CoACD, with better geometric similarity.



4.4. Distance to Original Mesh

We measure geometric similarity by the one-way Hausdorff and Chamfer distances from the collider to the input mesh, like [TLJP18]. The key motivation for choosing this metric is that each point on the collider should be as close to the input as possible. Complete results are provided in Tab. 4. Across our tested meshes, our approach has the lowest Hausdorff distance for 44 meshes, and lowest Chamfer distance for 43, shown in Tab. 2. The average one-way hausdorff distance on our dataset for convex primitive decomposition, CoACD, and V-HACD are **0.0445**, 0.0514, and 0.0710, and median **0.0346**, 0.0383, and 0.0593 respectively. The average chamfer distances are $6.95 \times 10^{-3}, 9.91 \times 10^{-3}, 8.82 \times 10^{-3}$, and median $5.40 \times 10^{-3}, 9.41 \times 10^{-3}, 7.60 \times 10^{-3}$ respectively. This shows that convex primitive decomposition consistently has closer geometry to the original, with less total complexity, measured by byte count. This also shows how CoACD's optimization for max distance does so at the cost of average distance, whereas our approach is closer on both metrics, even though it optimizes for volume.

Level of Detail Collision Objects To demonstrate our approach for generating different levels of detail for colliders, we create multiple resolutions for one mesh in Fig. 7, all tested in simulation. At each level our approach maintains the similarity of balls rolling

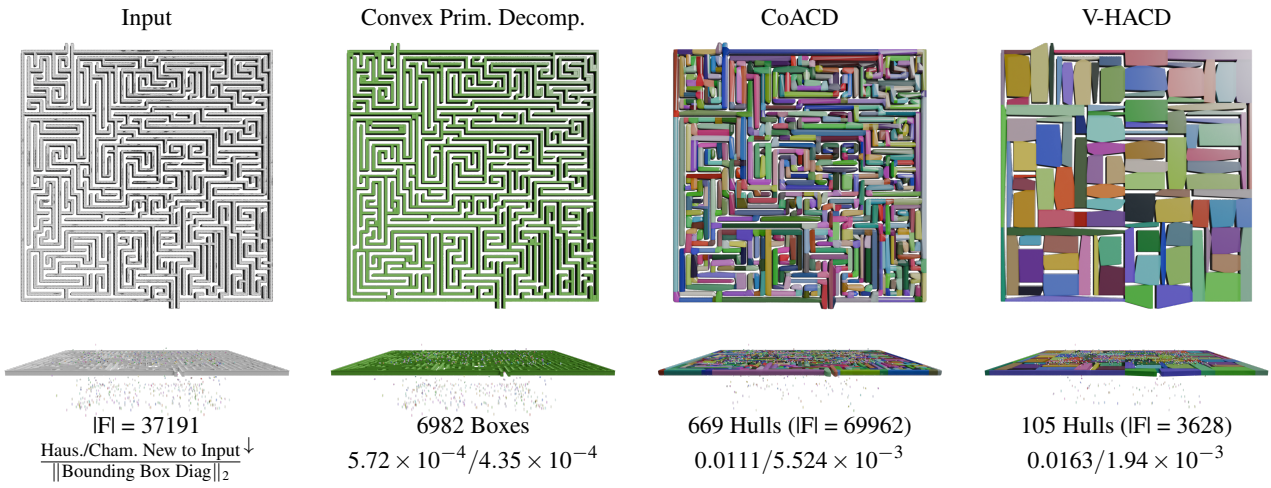


Figure 5: *Our approach can cleanly preserve holes in the non-manifold, non-watertight input mesh, maintaining its traversability as shown by the number of balls that can pass through in the second row. Prior convex decomposition approaches reduce the size of holes due to voxelization and preprocessing needed to make the mesh manifold and watertight, changing the collision behavior of the output collider.* ©Talaei-dev.

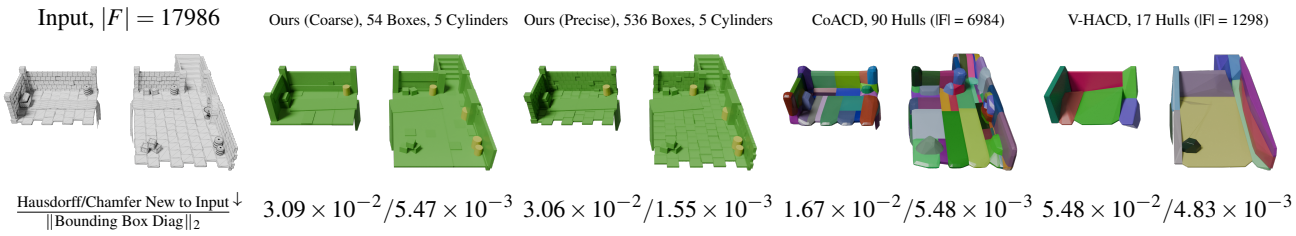


Figure 6: *Our approach can automatically fit colliders for environments at multiple resolutions, while more closely adhering to the input as compared to CoACD and V-HACD, maintaining sharp features such as the boxes, barrels, and chest, and can be edited easily.* ©*Karthik Naidu.*

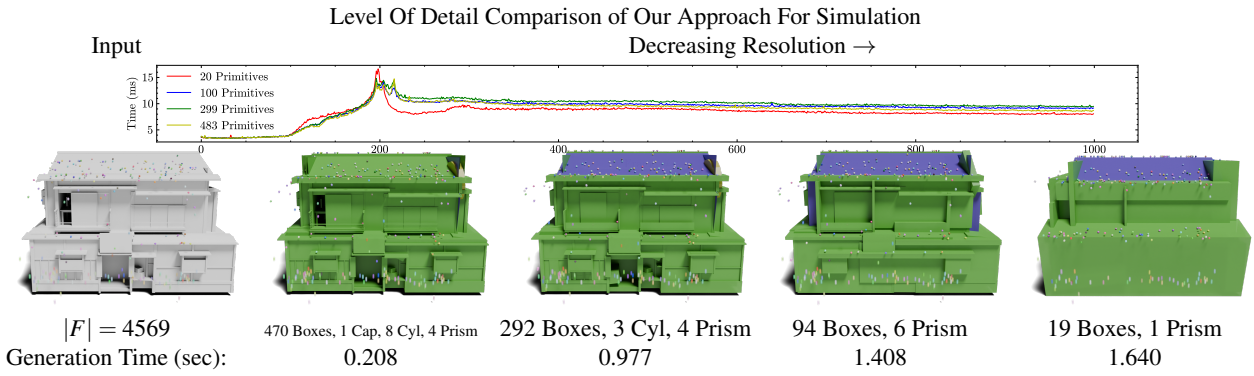


Figure 7: *We compare our approach at multiple levels of detail for a single mesh, varying only the target number of output primitives. Our approach gracefully degrades from a high-resolution approximation of the input to a lower resolution approximation. At different resolutions, the simulation using our collider still resembles the input mesh. Rigid-body collision time for each resolution is shown in the plot. The more precise meshes are generated faster than coarse meshes, as our approach is bottom up.* ©*Mikail Karaca.*

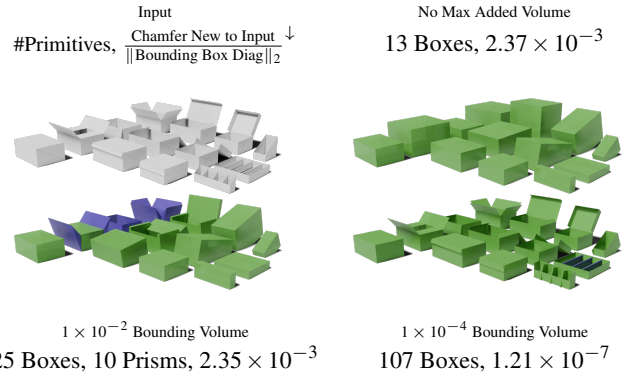
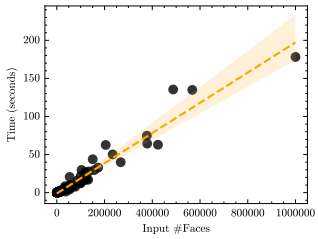


Figure 8: *Our approach can be controlled using the maximum allowed volume added when merging two primitives. If merging two primitives together exceeds some (See label above images for this example's values) fraction of the input mesh's axis-aligned bounding box's volume, merging is forbidden. This allows closer adherence to the original mesh.* CC BY *jellystuff.*



9: *The wall-clock runtime of our implementation scales linearly with the input mesh's number of faces, which allows our approach to scale, processing meshes with millions of faces. In the worst case, performance may be $O(|V|^2)$ if one primitive contains nearly all vertices, but we never observe this in practice.*

off the roof, demonstrating our approach's versatility. The simulation run-time at each level also shows that varying the resolution provides a way to tune performance versus geometric similarity.

Limiting Excess Volume To control how close the collection of primitives should adhere to the the input mesh, users can control the output mesh's adherence to the original using a maximal excess volume. In Fig. 8, the input mesh is a collection of cardboard boxes with flaps in various positions. By limiting the excess volume introduced per collapse, the output has higher geometric similarity, but more primitives. Depending on the user's goal, excess volume can be tuned to balance the trade-off between precision and efficiency.

Timing Our implementation efficiently decomposes meshes and even though the theoretical time complexity of collapse is at least $O(n \log n)$ due to reordering the priority queue the wall-clock time scales linearly with the size of the input mesh. We visualize the time to decompose an input mesh in Fig. 9, with exact time per mesh in Tab. 3. For small meshes, our approach completes in under a second. This is acceptable as the approach is designed to process meshes offline. In the worst case, our algorithm may be slow if one primitive repeatedly merges and no others do, as it will have many neighbors and enclose a large number of points with $O(|V|^2)$ complexity. In practice this case does not appear, since it would lead to a high volume error.

5. Ablations

In the following sections, we ablate some of our design choices.

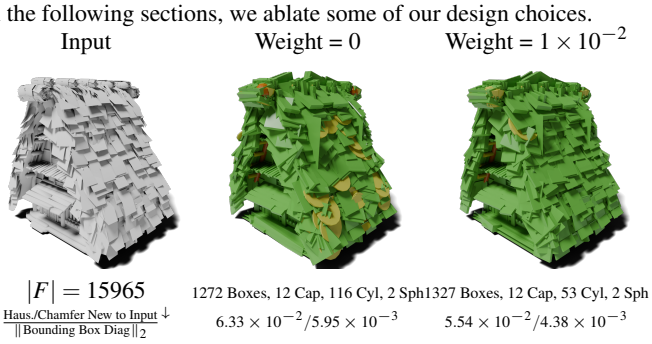


Figure 10: *Comparison of our approach with and without tangent weights for each input face's quadric. Without tangents, primitives may not properly align with planar faces, resulting in worse approximations as shown by the cylinders. With an added quadric in the longest edge's direction, primitives align more closely with each face.* CC BY *Lokomoto.*

Coplanar Vertices Tangent Quadric To demonstrate adding a tangent weight from Sec. 3.4, we visualize outputs with and without tangent edge weights in Fig. 10. Many coplanar faces in the input are converted to cylinders due to ambiguity with no weight, but with edge-weights these primitives more closely align with the input mesh's faces, allowing them to be tighter OBBs.

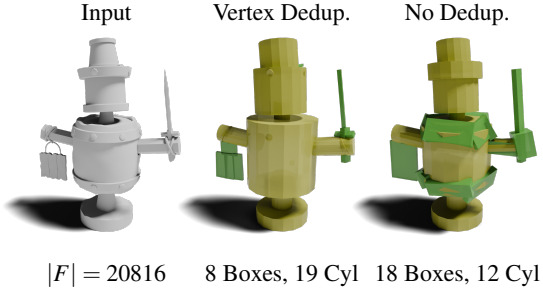


Figure 11: *Comparison of our approach with and without vertex deduplication and the same number of target primitives. The deduplicated model has additional primitives enclosed, leading to a lower final number of primitives. Deduplicating vertices changes the topology of the input mesh, leading to different results.* CC BY *"FinBeenWhere?".*

Vertex Deduplication We test our approach with and without vertex deduplication in Fig. 11. Deduplicating vertices can change results since it increases possible edge collapses. Without deduplicating, components such as barrel hoops are not merged with the body, but other regions such as the hat are preserved differently. For most cases, we find exactly overlapped vertex merging improves the output quality of the result.

Vertex vs. Face Merging To show each vertex must correspond to multiple primitives, we show our approach on a lekythos with primitives per vertex or face in Fig. 12. Vertex merging leaves much of the input surface uncovered, unsuitable for use as a collider, showing the importance of using faces to represent primitives.

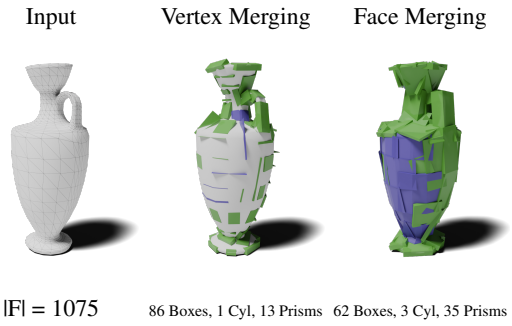


Figure 12: *Our approach using per-vertex merging and per-face merging. Per-vertex merging leaves large portions of the input (shown in white) uncovered, whereas merging faces covers the whole surface.* *Global Digital Heritage and GDH-Afrika.*

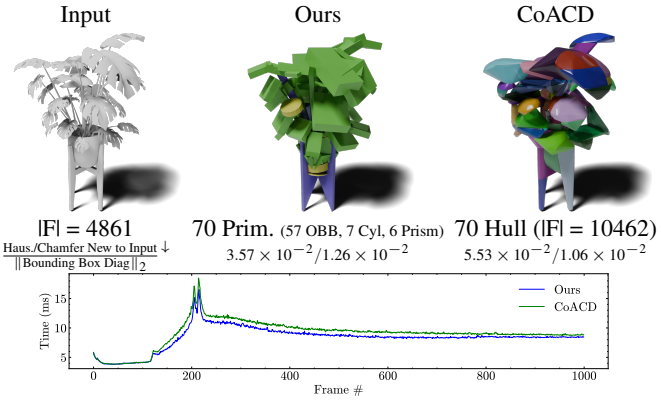


Figure 13: *Our approach compared to CoACD with an identical number of primitives and convex hulls, and similar geometric similarity. Our approach has faster simulation than CoACD, as the collision with hulls depends on the number of vertices and faces.* *Giora.*

Comparing Primitive & Hull Counts We show hull and primitive counts do not reflect cost by comparing simulation with an equal number of primitives and hulls in Fig. 13. Despite matching counts, primitives are faster in simulation, as the number of hulls masks the number of faces and vertices used in GJK.

Primitive Variant Cost Ablation We ablate our set of costs for each variant of primitive, by comparing our weighting to uniform weighting (all weights set to 1) and a weighting with only spheres, capsules, and OBBs in Fig. 14. When using uniform weighting, many prisms are output. We also test a weighting that only uses the common primitives, where spheres, capsules and OBBs have weights of 0.5, 0.75, and 1.5 respectively. The output of this weighting is also reasonable, but is coarser in some regions like the legs, compared to our weighting. We plot the simulation time for each weighting, to highlight the trade-off of geometric similarity and simulation performance. As the uniform weighting has prisms that are converted to hulls, it is slower than our weighting. On the other hand, using only directly supported primitives leads to faster simulation time, but with less geometric similarity. Our weighting balances the two approaches.

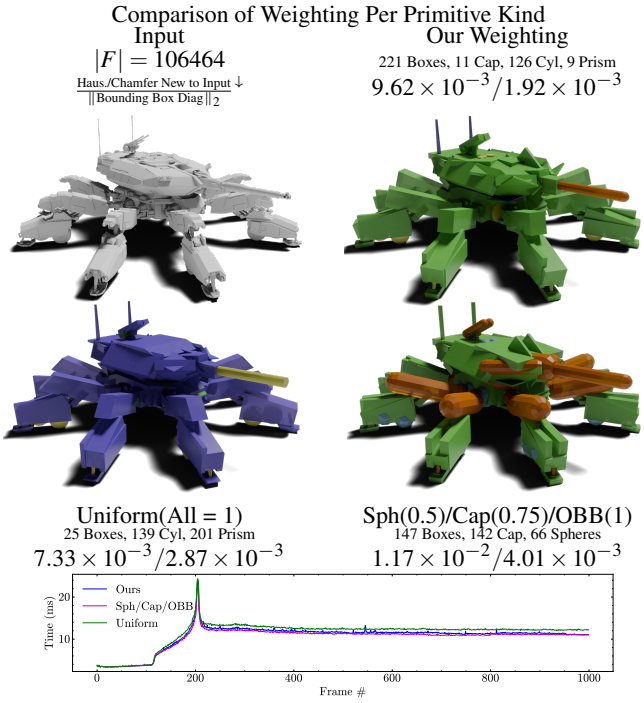


Figure 14: *Our approach with different sets of weights. Uniform weighting outputs more isosceles trapezoidal prisms, reducing performance in simulation as shown in the plot. We also compare a weighting which heavily favors spheres (0.5), capsules (0.75), and OBBs (1) with everything else disabled, and observe that this can improve simulation performance but with less geometric similarity. OBBs are shown in green, capsules in orange, cylinders in yellow, prisms in blue, and spheres in light blue.* *Adoni.*

Scanned Data To demonstrate our approach on noisy data, we benchmark our approach on a 3D scan of the Wat Benchamabophit in Fig. 15. This scan contains 999956 faces, with holes on the bottom. Our approach still accurately decomposes the input mesh, and preserves holes even with the high density, while maintaining small features such as columns of the inner gate and chofa on the roof.

Intersecting Primitive Ablation We ablate the cost function from Eq. 4 versus Eq. 5 in Fig. 16, by approximating the intersection's volume using dense rejection sampling. We sample points from inside the smaller primitive, and estimate the intersection's volume as the fraction of points contained in the other primitive times the smaller primitive's volume. We do not see significant gains from Eq. 5, but sampling even with a small number of points is 30× slower. Since there is little benefit with large cost, we do not use Eq. 5.

6. Discussion

Our approach decomposes an input mesh into a number of convex primitives, suitable for rigid-body collision, drastically and ro-

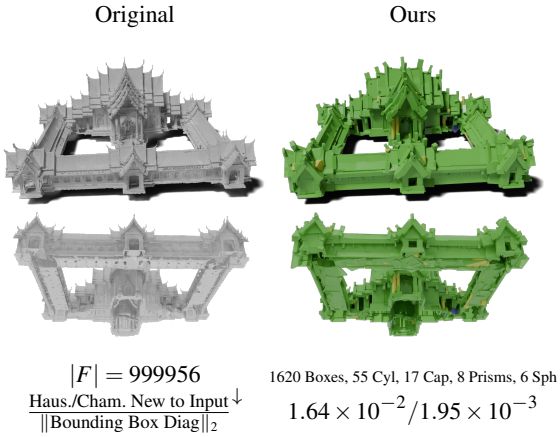


Figure 15: *Our approach on a noisy scanned 3D model with an open bottom. Despite the input's density, our approach adheres to its structure including small details such as columns and chofa.* CC BY NC SA *GoodScan 3D.*

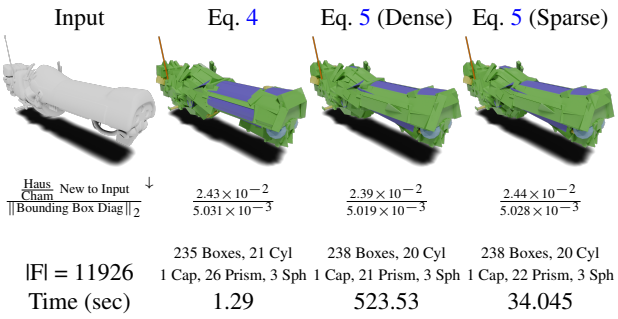


Figure 16: *We compare the cost function from Eq. 4 with an approximation of Eq. 5 using rejection sampling. For dense sampling, we sample 1,000,000 points for every pair of primitives, and for sparse sampling we sample 50,000 points. In practice there is little benefit using Eq. 5, with no visible improvement, and the cost of computation makes it intractable for real usage.* CC BY NC *Tasha.Lime.*

bustly simplifying complex real-world meshes. Convex primitive decomposition scales with mesh size and can handles millions of triangles.

While previous approaches focus on tight decompositions using convex hulls, they are impractical due to slower simulation and imprecision due to voxelization and planar cuts. We flip the top-down approach with bottom-up merging, removing the complexity of cutting while producing efficient decompositions.

Unlike prior approaches in primitive abstraction, we do not use deep learning, RANSAC, or differentiability, making our approach robust and fast for quick artistic iteration. Our approach can be built into existing tooling for artists to quickly generate collision objects which can then be directly manipulated.

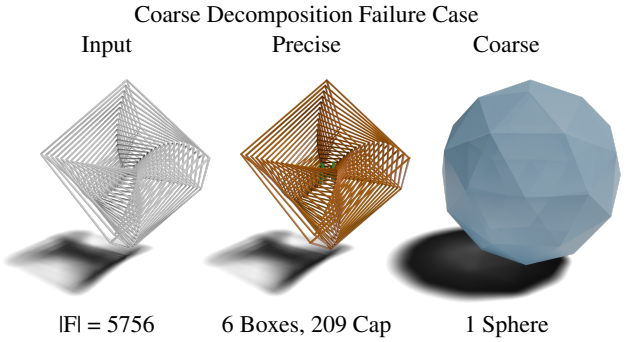


Figure 17: *While our approach can decompose each component of the input well separately (middle), when combining them it does not recover a tight bounding box (right). This is because internal components unnecessarily affect the orientation of the output primitive.* CC BY *deslancer.*

7. Limitations

One limitation of our approach is meshes with internal components will have coarser bounding primitives, because normals of enclosed faces affect the eigendecomposition of each primitive. We show an example of this failure case in Fig. 17 on a cube with twisted cubes inside of it. It is clear that the tightest bounding box only needs to fit the largest cube, but our approach does not recover that due to the influence of the other cubes.

Another limitation is that primitives that subsume coplanar faces may be degenerate and depend on the input mesh's orientation. This can be mitigated by adding the tangent direction (Sec. 5), but does not always remove the influence of the input orientation. Furthermore, perfectly coplanar faces may be overeagerly merged together. This is because these faces have no width along their shared plane, introducing near-zero volume when they are merged.

One limitation due to the coarseness of primitives is that they may be less suitable for high frequency details. For organic objects which often have more detail and curvature, primitive decomposition must output more primitives to maintain geometric similarity. As one future direction, this approach could be extended with curved primitives, or for those portions of the mesh convex hulls can be used to preserve detail.

Finally, it may be considered a limitation that convex primitive decomposition relies on the topology of the input. Small modifications in the topology of the input mesh can change the output mesh as can be seen by vertex deduplication in Fig. 11. We do not find this to be a problem, and perform vertex deduplication and pairwise collapse to mitigate the effects of topology. Often, topology also serves as a good indicator as to where to split the mesh into different components, so it can be considered a way for artists to indicate where to separate colliders.

Future Work We are able to handle cases where disconnected components of the mesh are fused together, using something similar to non-edge contraction from Garland and Heckbert [GH97]. In collision generation though, disconnected components may be more amenable to merging as compared to mesh simplification.

We leave it to future work to explore edge collapses not defined by the input topology, using techniques such as the one from Liu et al. [LZY24].

8. Conclusion

In summary, we develop a concise and robust algorithm for decomposing an arbitrary mesh into a number of convex primitives usable in downstream applications such as collision detection. Our approach produces a small number of tight-fitting primitives designed for efficiency at runtime. We hope that our approach motivates further work into simple algorithms that closely align with the processes artists take, and that this work is adapted into game engines and physics/collider generation tooling.

References

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual computer. 22*, 3 (2006). URL: https://doi.org/10.1007/s00371-006-0375-x, doi:10.1007/s00371-006-0375-x. 3

[AMSF08] ATTENE M., MORTARA M., SPAGNUOLO M., FALCIDIENO B.: Hierarchical convex approximation of 3d shapes for fast region selection. *Computer Graphics Forum 27*, 5 (2008), 1323–1332. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01271.x, doi:https://doi.org/10.1111/j.1467-8659.2008.01271.x. 3

[And24] ANDREWS J.: Navigation-driven approximate convex decomposition. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, 2024), SIGGRAPH '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3641519.3657479, doi:10.1145/3641519.3657479. 1, 3, 7

[BBN*20] BIRDAL T., BUSAM B., NAVAB N., ILIC S., STURM P.: Generic primitive detection in point clouds using novel minimal quadric fits. *IEEE Transactions on Pattern Analysis and Machine Intelligence 42*, 6 (2020), 1333–1347. doi:10.1109/TPAMI.2019.2900309. 3

[BH11] BAO H., HUA W.: *Variational OBB-Tree Approximation for Solid Objects*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 281–293. URL: https://doi.org/10.1007/978-3-642-18342-3_6, doi:10.1007/978-3-642-18342-3_6. 3

[BK02] BISCHOFF S., KOBBELT L.: Ellipsoid decomposition of 3d-models. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission* (2002), pp. 480–488. doi:10.1109/TDPVT.2002.1024103. 3

[Ble18] BLENDER ONLINE COMMUNITY: *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org. 1, 9

[CB17] CALDERON S., BOUBEKEUR T.: Bounding proxies for shape approximation. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017) 36*, 5 (july 2017). URL: https://doi.org/10.1145/3072959.3073714, doi:10.1145/3072959.3073714. 3

[Epi22] EPIC GAMES: Unreal engine 5, 2022. URL: https://www.unrealengine.com/en-US/unreal-engine-5. 1, 7

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. URL: https://doi.org/10.1145/258734.258849, doi:10.1145/258734.258849. 1, 2, 3, 4, 6, 7, 13

[GH98] GARLAND M., HECKBERT P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of the Conference on Visualization '98* (Washington, DC, USA, 1998), VIS '98, IEEE Computer Society Press, p. 263–269. doi:10.1109/VISUAL.1998.745312. 1, 2

[GH99] GARLAND M., HECKBERT P.: *Quadric-based polygonal surface simplification*. PhD thesis, USA, 1999. AAI9950005. 4

[GJK88] GILBERT E., JOHNSON D., KEERTHI S.: 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. doi:10.1109/56.2083. 8

[GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, Association for Computing Machinery, p. 171–180. URL: https://doi.org/10.1145/237170.237244, doi:10.1145/237170.237244. 1, 3

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2001), I3D '01, Association for Computing Machinery, p. 49–58. URL: https://doi.org/10.1145/364338.364345, doi:10.1145/364338.364345. 3, 6, 7

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1993), SIGGRAPH '93, Association for Computing Machinery, p. 19–26. URL: https://doi.org/10.1145/166117.166119, doi:10.1145/166117.166119. 2

[HLW24] HAFNER C., LY M., WOJTAN C.: Spin-it faster: Quadrics solve all topology optimization problems that depend only on mass moments. *ACM Trans. Graph. 43*, 4 (jul 2024). URL: https://doi.org/10.1145/3658194, doi:10.1145/3658194. 2

[Hop99] HOPPE H.: New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years* (Washington, DC, USA, 1999), VIS '99, IEEE Computer Society Press, p. 59–66. 1, 2, 7

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-Charts: Quasi-Developable Mesh Segmentation. *Computer Graphics Forum* (2005). doi:10.1111/j.1467-8659.2005.00883.x. 3

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Trans. Graph. 21*, 3 (jul 2002), 339–346. URL: https://doi.org/10.1145/566654.566586, doi:10.1145/566654.566586. 2

[KFK*15] KAICK O. V., FISH N., KLEIMAN Y., ASAFI S., COHEN-OR D.: Shape segmentation by approximate convexity analysis. *ACM Trans. Graph. 34*, 1 (Dec. 2015). URL: https://doi.org/10.1145/2611811, doi:10.1145/2611811. 3

[KJS07] KREAVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (USA, 2007), PG '07, IEEE Computer Society, p. 129–138. URL: https://doi.org/10.1109/PG.2007.43, doi:10.1109/PG.2007.43. 3

[KYZB19] KAISER A., YBANEZ ZEPEDA J. A., BOUBEKEUR T.: A Survey of Simple Geometric Primitives Detection Methods for Captured 3D Data. *Computer Graphics Forum* (2019). doi:10.1111/cgf.13451. 3

[LA07] LIEN J.-M., AMATO N. M.: Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2007), SPM '07, Association for Computing Machinery, p. 121–131. URL: https://doi.org/10.1145/1236246.1236265, doi:10.1145/1236246.1236265. 1, 3

[LCWK07] LU L., CHOI Y.-K., WANG W., KIM M.-S.: Variational 3d shape segmentation for bounding volume computation. *Computer Graphics Forum 26*, 3 (2007), 329–338.

URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01055.x, doi:https://doi.org/10.1111/j.1467-8659.2007.01055.x. 3

[LD17] LE T., DUAN Y.: A primitive-based 3d segmentation algorithm for mechanical cad models. *Computer Aided Geometric Design 52* (2017), 231–246. URL: https://doi.org/10.1016/j.cagd.2017.02.009, doi:10.1016/j.cagd.2017.02.009. 3

[LLT*20] LESCOAT T., LIU H.-T. D., THIERY J.-M., JACOBSON A., BOUBEKEUR T., OVSJANIKOV M.: Spectral mesh simplification. *Computer Graphics Forum (Proc. of EUROGRAPHICS 2020) 39*, 2 (2020), 315–324. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13932, doi:https://doi.org/10.1111/cgf.13932. 2

[LLY*23] LI Y., LIU S., YANG X., GUO J., GUO J., GUO Y.: Surface and edge detection for primitive fitting of point clouds. In *ACM SIGGRAPH 2023 Conference Proceedings* (New York, NY, USA, 2023), SIGGRAPH '23, Association for Computing Machinery. URL: https://doi.org/10.1145/3588432.3591522, doi:10.1145/3588432.3591522. 3

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *Proceedings of the Conference on Visualization '98* (Washington, DC, USA, 1998), VIS '98, IEEE Computer Society Press, p. 279–286. doi:10.1109/VISUAL.1998.745314. 2

[Lum17] LUMBERYARD A.: Amazon lumberyard bistro, open research content archive (orca), July 2017. http://developer.nvidia.com/orca/amazon-lumberyard-bistro. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 9, 16, 23

[LWRC23] LIU W., WU Y., RUAN S., CHIRIKJIAN G.: Marching-primitives: Shape abstraction from signed distance function. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2023). doi:10.1109/CVPR52729.2023.00847. 3

[Lys07] LYSENKO M.: Realtime constructive solid geometry. In *ACM SIGGRAPH 2007 Sketches* (New York, NY, USA, 2007), SIGGRAPH '07, Association for Computing Machinery, p. 7–es. URL: https://doi.org/10.1145/1278780.1278789, doi:10.1145/1278780.1278789. 3

[LZY24] LIU H.-T. D., ZHANG X., YUKSEL C.: Simplifying triangle meshes in the wild, 2024. URL: https://arxiv.org/abs/2409.15458, arXiv:2409.15458. 7, 14

[MDKK07] MIZOGUCHI T., DATE H., KANAI S., KISHINAMI T.: Quasi-optimal mesh segmentation via region growing/merging. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2007), vol. 48078, pp. 547–556. doi:10.1115/DETC2007-35171. 3

[MG09] MAMOU K., GHORBEL F.: A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE International Conference on Image Processing (ICIP)* (2009), pp. 3501–3504. doi:10.1109/ICIP.2009.5414068. 1, 2, 3, 9, 16, 19, 20, 24

[MST*11] MCADAMS A., SELLE A., TAMSTORF R., TERAN J., SIFAKIS E.: Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations. URL: https://api.semanticscholar.org/CorpusID:18183079. 4

[MZL*09] MEHRA R., ZHOU Q., LONG J., SHEFFER A., GOOCH A., MITRA N. J.: Abstraction of man-made shapes. *ACM Trans. Graph. 28*, 5 (dec 2009), 1–10. URL: https://doi.org/10.1145/1618452.1618483, doi:10.1145/1618452.1618483. 3

[Nvi17] NVIDIA CORPORATION: Physx, 04 2017. URL: https://developer.nvidia.com/physx-sdk. 1

[O'R85] O'ROURKE J.: Finding minimal enclosing boxes. *Int. J. Parallel Program. 14*, 3 (1985), 183–199. URL: https://doi.org/10.1007/BF00991005, doi:10.1007/BF00991005. 6

[PS24] PARK C., SUNG M.: Split, merge, and refine: Fitting tight bounding boxes via over-segmentation and iterative search. In *2024 International Conference on 3D Vision (3DV)* (2024), pp. 1468–1477. doi:10.1109/3DV62453.2024.00146. 3

[PUG19] PASCHALIDOU D., ULUSOY A. O., GEIGER A.: Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (June 2019). doi:10.1109/CVPR.2019.01059. 3

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 327–334. URL: https://doi.org/10.1145/344779.344935, doi:10.1145/344779.344935. 2, 3, 7

[Ske22] SKETCHFAB: The best 3d viewer on the web, 2022. URL: https://sketchfab.com/. 1

[SWK07] SCHNABEL R., WAHL R., KLEIN R.: Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (2007), 214–226. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01016.x, doi:https://doi.org/10.1111/j.1467-8659.2007.01016.x. 3

[SZTL19] SUN C., ZOU Q., TONG X., LIU Y.: Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Transactions on Graphics (SIGGRAPH Asia) 38*, 6 (2019). URL: https://doi.org/10.1145/3355089.3356529, doi:10.1145/3355089.3356529. 3

[TGB13] THIERY J.-M., GUY E., BOUBEKEUR T.: Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia 2013) 32*, 6 (2013), Art. No. 178. URL: https://doi.org/10.1145/2508363.2508384, doi:10.1145/2508363.2508384. 1, 2, 6, 7

[TGBE16] THIERY J.-M., GUY E., BOUBEKEUR T., EISEMANN E.: Animated mesh approximation with sphere-meshes. *ACM Trans. Graph. 35*, 3 (May 2016), 30:1–30:13. URL: http://doi.acm.org/10.1145/2898350, doi:10.1145/2898350. 1, 2

[TK20] TRETTNER P., KOBBELT L.: Fast and Robust QEF Minimization using Probabilistic Quadrics. *Computer Graphics Forum* (2020). doi:10.1111/cgf.13933. 2

[TLJP18] THUL D., LADICKÝ L., JEONG S., POLLEFEYS M.: Approximate convex decomposition and transfer for animated meshes. *ACM Trans. Graph. 37*, 6 (dec 2018). URL: https://doi.org/10.1145/3272127.3275029, doi:10.1145/3272127.3275029. 1, 3, 7, 9

[TvL84] TARJAN R. E., VAN LEEUWEN J.: Worst-case analysis of set union algorithms. *J. ACM 31*, 2 (mar 1984), 245–281. URL: https://doi.org/10.1145/62.2160, doi:10.1145/62.2160. 7

[WHX*22] WANG R., HUA W., XU G., HUO Y., BAO H.: Variational hierarchical directed bounding box construction for solid mesh models, 2022. URL: https://arxiv.org/abs/2203.10521, arXiv:2203.10521. 3

[WLLS22] WEI X., LIU M., LING Z., SU H.: Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–18. URL: https://doi.org/10.1145/3528223.3530103, doi:10.1145/3528223.3530103. 1, 2, 7, 9, 19, 20

[YJ20] YANG X., JIA X.: Simple primitive recognition via hierarchical face clustering. *Computational Visual Media 6* (2020), 431–443. doi:10.1007/s41095-020-0192-6. 3

[YLW06] YAN D.-M., LIU Y., WANG W.: Quadric surface extraction by variational shape approximation. In *Proceedings of the 4th International Conference on Geometric Modeling and Processing* (Berlin, Heidelberg, 2006), GMP'06, Springer-Verlag, p. 73–86. URL: https://doi.org/10.1007/11802914_6, doi:10.1007/11802914_6. 3

[YWLY12] YAN D.-M., WANG W., LIU Y., YANG Z.: Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design 44*, 11 (2012), 1072–1082. doi:https://doi.org/10.1016/j.cad.2012.04.005. 3

[ZBCS*23] ZHAO T., BUSÉ L., COHEN-STEINER D., BOUBEKEUR T., THIERY J.-M., ALLIEZ P.: Variational shape reconstruction via quadric error metrics. In *ACM SIGGRAPH 2023 Conference Proceedings* (New York, NY, USA, 2023), SIGGRAPH '23, Association for Computing Machinery. URL: https://doi.org/10.1145/3588432.3591529, doi:10.1145/3588432.3591529. 2

[ZLGW15] ZHANG H., LI C., GAO L., WANG G.: Hierarchical mesh segmentation based on quadric surface fitting. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)* (2015), IEEE, pp. 33–40. doi:10.1109/CADGRAPHICS.2015.26. 3

Meshing of Cylinders and Capsules Our approach outputs cylinders, capsules, spheres, and frustums modeled as parametric functions. For some applications though (such as rendering), all inputs must be given as a mesh, so we convert these primitives to quantized meshes. For cylinders and frustums, we output prisms capped by one face on each end. For spheres, we use a UV sphere approximation. For capsules, we output prisms capped with UV spheres on each end. This makes our approach suitable for any downstream applications.

Appendix A: Additional Results

We show additional results from our approach and provide pseudocode for fitting frustums in Alg. 2 and isosceles trapezoidal prisms in Alg. 3. Then, we plot frame durations for collisions with 5000 spheres for many meshes in Fig. 20, and Fig. 21. We show outputs on buildings, characters, props, and environments in Fig. 18. We include more comparisons to CoACD and V-HACD in Fig. 19. Complete statistics for the primitives generated on our dataset are in Tab. 3, with all 1-way distances in Tab. 4. We show byte counts for all models in Tab. 5. We show our approach on a complex scene [Lum17], to generate an initial decomposition, then automatically delete thin bounding boxes to get the output shown in Fig. 22. We then show our approach on two organic objects in Fig. 23, and one complex example in Fig. 24. Finally, we show additional examples with collision detection comparisons from the V-HACD dataset [MG09] in Fig. 26.

Algorithm 2 Frustum Point Subsuming

Input: Points $\in \mathbb{R}^{N\times 3}$, Base Point **c**, Axis **a**
Output: height h, top and bottom radius $r_{\text{top}}, r_{\text{bot}}$
$h = 0, r_{\text{bot}} = 0, r_{\text{top}} = 0$
for p $\in$ Points **do** $h = \max(h, |(p-c)^\top a|)$
$r^*_{\text{top}} = 0, y^*_{\text{top}} = 1, r^*_{\text{bot}} = 0, y^*_{\text{bot}} = 0$ ▷ Approx. constraints
procedure Y(p): **return** $\frac{(p-c)^\top a}{h}$
procedure R_{SIDE}(r, y, r_{opp}): **return if** side **is** bot
$|\frac{r - r_{\text{opp}} y}{1-y}|$ **else** $|\frac{r - r_{\text{opp}}(1-y)}{y}|$
for p $\in$ Points **do**:
  side, opp = **if** $Y(p) \le 0.5$ { bot, top } **else** { top, bot }
  $r = \|(p-c) - h(p-c)^\top a\|_2$
  next = $R_{\text{side}}(r, Y(p), r_{\text{opp}})$
  if next $> r^*_{\text{side}}$ **then**
    $r^*_{\text{side}} = r, y^*_{\text{side}} = Y(p), r_{\text{side}} =$ next
    $r^*_{\text{opp}} = R_{\text{opp}}(r^*_{\text{opp}}, y^*_{\text{opp}}, r^*_{\text{side}})$
Similar FixSide(...) Procedure as Alg. 3
return $h, r_{\text{top}}, r_{\text{bot}}$

Algorithm 3 Isosceles Trapezoid Point Subsuming

Input: Points $\in \mathbb{R}^{N\times 3}$, Center c, Axes a_x, a_y, a_z
Output: Half-radii h_x, h_y, h_{zt}, h_{zb}
$h_x = 0, h_y = 0, h_{zt} = 0, h_{zb} = 0$
for p $\in$ Points **do** ▷ Compute h_x, h_y
  $h_x = \max(h_x, |(p-c)^\top a_x|)$, $h_y = \max(h_y, |(p-c)^\top a_y|)$
$z^*_{zt} = 0, y^*_{zt} = 1, z^*_{zb} = 0, y^*_{zb} = 0$ ▷ Current approx. constraints
for p $\in$ Points **do**:
  procedure UPDATECONSTRAINT(side $\in$ {zt, zb})
    opp = **if** side **is** zt { zb } **else** { zt }
    next = $h_{\text{side}}(z(p), y(p), h_{\text{opp}})$ ▷ h_{zt} or h_{zb}
    if next $> h_{\text{side}}$ **then**
      $z^*_{\text{side}} = z(p), y^*_{\text{side}} = y(p), h_{\text{side}} =$ next
      $h_{\text{opp}} = |h_{\text{opp}}(z^*_{\text{opp}}, y^*_{\text{opp}}, h_{\text{side}})|$
  UPDATECONSTRAINT(**if** $y(p) \le 0.5$ { zb } **else** { zt })
procedure FIXSIDE(top $\in$ bool)
  ▷ Fix one side, find min. that subsumes all points for other.
    ▷ Guarantees that all input points are subsumed.
  $v, o =$ **if** top { zt, zb } **else** { zb, zt }
  $h_v = 0$
  for p $\in$ Points **do**: $h_v = \max(h_v, h_v(z(p), y(p), h_o))$
FixSide($h_{zt} < h_{zb}$) ▷ Fix larger side and set smaller
FixSide($h_{zt} \ge h_{zb}$) ▷ Then vice-versa
return h_x, h_y, h_{zt}, h_{bt}

Additional Results

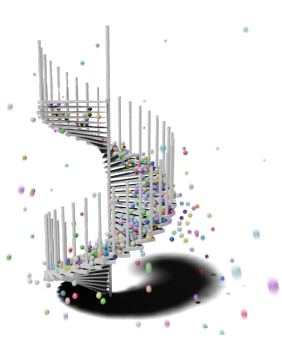
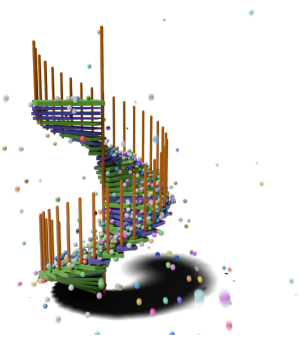
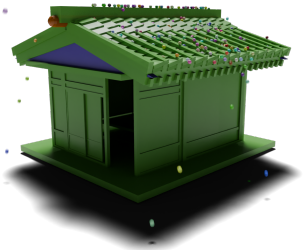
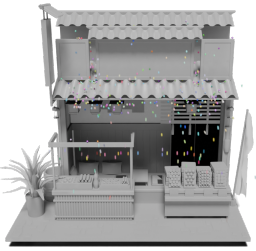
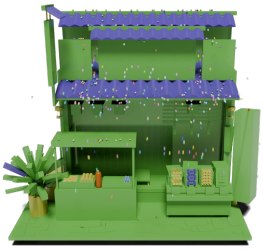
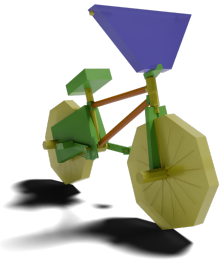
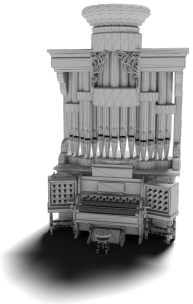
Input Mesh	Convex Prim. Decomp.	Input Mesh	Convex Prim. Decomp.
			
$\|F\| = 3404$	1 Box, 2 Cylinders, 3 Frustums	$\|F\| = 6358$	24 boxes, 40 Capsules, 16 Prisms
			
$\|F\| = 233965$	409 Boxes, 1 Cap, 2 Prism	$\|F\| = 119639$	645 Boxes, 22 Cap, 145 Cyl, 118 Prisms, 22 Sph
			
$\|F\| = 54995$	50 Boxes, 34 Cap, 22 Cyl, 2 Prism	$\|F\| = 20875$	94 Boxes, 17 Cap, 67 Cyl, 2 Sph
			
$\|F\| = 6280$	77 Boxes, 6 Capsules, 40 Cylinders	$\|F\| = 267267$	3675 Boxes, 223 Cap, 131 Cyl, 2 Prisms

Figure 18: *Additional results from running our approach on a variety of input meshes. Green indicates bounding boxes, yellow for cylinders, blue for trapezoidal prisms, and red for capsules, light blue for spherres. Our approach greatly cuts down on the number of input elements.* *BlenderFace,* *RubaQewar,* *neos_nelia,* *Deerex,* *Hoody468,* *Nathan Sioui,* *neih.D,* *Scritta.*

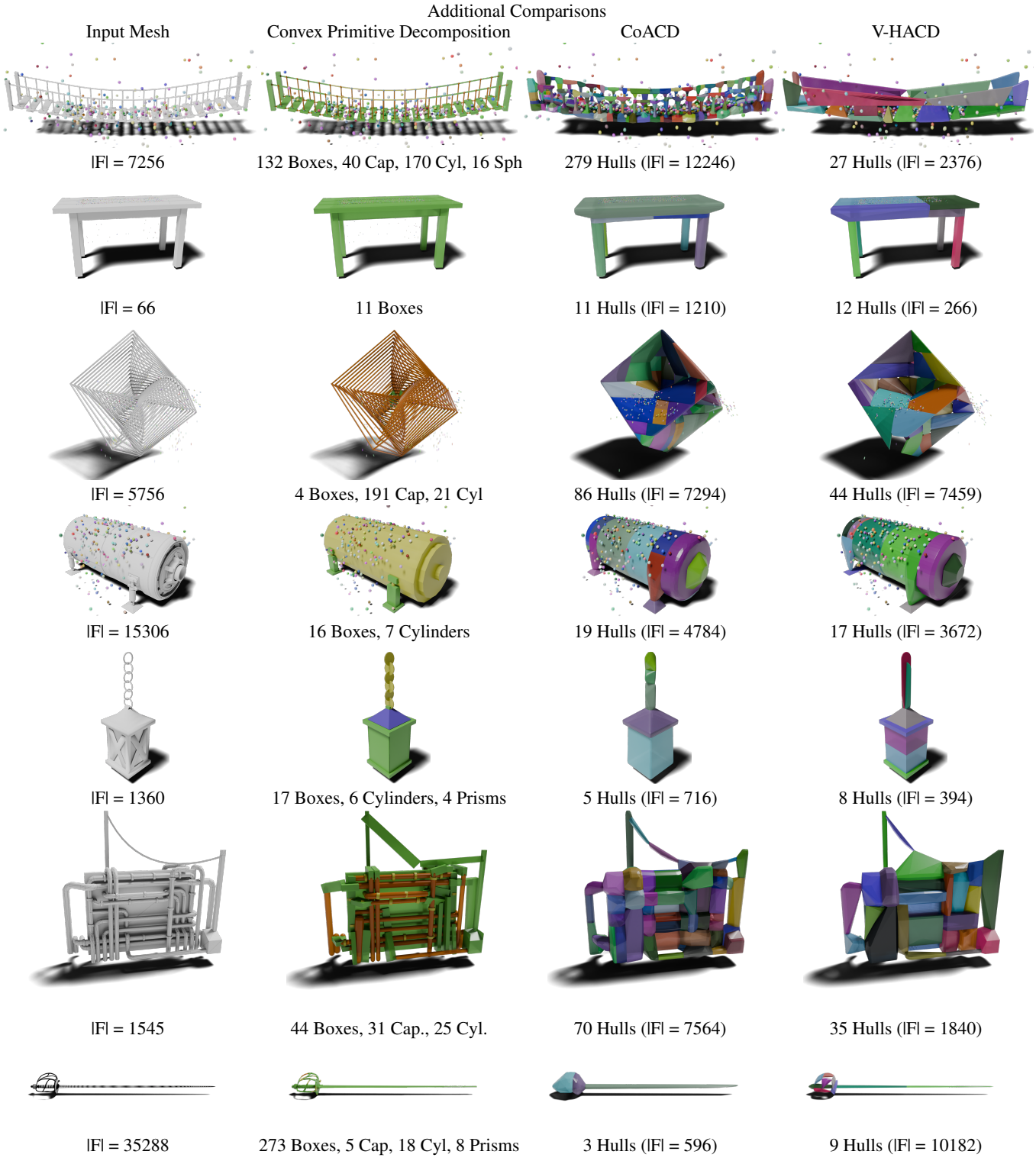


Figure 19: *Additional comparison from running our approach on a variety of input meshes. Green indicates bounding boxes, yellow cylinders, blue trapezoidal prisms, red capsules, light blue spheres, and light orange frustums. Our approach more closely adheres to the input mesh while containing fewer elements than approximate convex decomposition approaches. Our approach also uses the fewest bytes on all models shown.* ⓒⓘ*lucq,* ⓒⓘ*toAflame,* ⓒⓘ*maxpsr,* ⓒⓘ*Blenderolokos,* ⓒⓘ*Aerial_Knight,* ⓒⓘ*RitorDP.*

	Input				Ours							CoACD		V-HACD	
Name	#F	#V	Mani.	WT	#Box	#Cap	#Sph	#Cyl	#Fru	#Pri	Time (s)	#F	#Hull	#F	#Hull
Armadillo	99976	49990	✓	✓	195			4		1	12.303	9632	24	11275	15
Armored Charizard	4649	3273	✓	×	144	2		10		32	0.417	7170	41	2402	26
Bell	10686	10578	✓	×	47	4		2			1.418	11546	63	6198	22
Bicycle	54995	102354	✓	×	50	34		22		2	9.373	4904	30	9106	23
Canon At1	9624	13094	✓	×	8			4			1.531	7484	28	3174	15
Cardboard Boxes	119	202	×	×	34					10	0.006	5568	65	496	40
Casio Keyboard	2804	3730	✓	×	16						0.511	440	1	658	6
Chimney Pipe	3404	4864	✓	×	1			2	3		0.641	5354	27	2334	30
Chitinous Knight	87258	89239	✓	×	47	1		1			16.084	4316	16	13019	14
Chuo House	173163	209450	×	×	1023	10		22		18	32.894	3198	29	5200	27
Church Organ	20875	26902	✓	×	94	17	2	67			2.392	5550	54	3122	39
Cinema Scan	423713	351719	✓	×	475	1		17		5	62.922	8384	87	6030	37
Cube	5756	7401	✓	×	4	191		24			0.516	7286	87	7076	44
Cyberpunk Atm	6566	10447	✓	×	63			24			0.659	3912	25	1458	16
Cyberpunk Bike	31916	37040	✓	×	34	3	4	16		1	4.379	4614	15	9626	42
Dojo	4569	4133	✓	×	470	1		8		4	0.208	9866	149	2354	50
Dungeon (Precise)	17986	59852	✓	×	532			9			1.680	6984	90	1298	17
Espresso Machine	19738	29059	✓	×	60	16	1	101			2.500	10878	93	4668	35
Fantasy Asian House	267627	289886	✓	×	3675	223		131		2	39.965	3548	28	10940	35
Fps-Hands	4076	4600	✓	×	145			5			0.533	2472	12	2440	14
Fractal	8184	7816	×	×	5434						0.168	53148	1126	3010	52
French Halfbasket	35288	41255	✓	×	273	5		18			5.383	596	3	10182	9
Greek Vase	12545	14464	×	×	379						0.713	18624	62	12790	51
Gun	10256	8994	✓	×	112			13		8	1.203	3354	17	2308	20
Hover Bike	9327	7025	✓	×	66			6			1.011	3924	16	2850	19
Jpn Corridor	200	308	✓	×	33						0.011	4754	34	542	20
Jpn House	119620	141206	✓	×	645	22	22	145		118	17.247	6332	75	4010	34
Jpn House 2	233965	235330	×	×	409	1				2	50.214	3876	39	2184	24
Jpn House 3	41462	42444	✓	✓	645			1		3	4.282	1970	13	1404	13
Jpn Paddle Crab	567566	550825	✓	×	1146			14		3	134.896	11484	64	16579	22
Shinto Shrine	6280	6638	✓	×	77	6		40			0.780	7116	69	2270	26
Kagiya Edo Tokyo	22819	18820	×	×	668	1		207		19	2.400	1088	7	652	8
Lantern	1360	1862	✓	×	17			6		4	0.148	716	5	392	8
Lekythos	1075	1148	✓	×	110			3		7	0.106	9056	40	3498	42
Lethe	4782	8219	✓	×	112	1	13	62		10	0.490	4172	25	2140	18
Marble Track	32558	47029	✓	×	106	1	4	117		10	5.460	7594	50	3946	18
Maze	37191	40663	✓	×	7000						1.450	70674	684	3628	105
Mech	8732	15771	✓	×	42	2		5		4	1.130	5618	29	2632	22
Melon Pallet	45016	45342	✓	×	38						6.015	9790	54	4900	23
Militech Robot	52373	29422	✓	×	116			68			5.925	13050	66	5784	25
Modular Fence Corner	4642	3257	✓	×	3			5			0.592	5240	13	1602	10
Old Tree	4226	4921	✓	×	98	1				1	0.476	6204	38	3079	15
Pipe Wall	1545	1791	✓	×	44	31		25			0.131	7564	70	1840	35
Pipes	1200	2154	✓	×	21			22		7	0.127	5666	27	3032	42
Potted Plant	4861	6830	✓	×	57	2		8		5	0.535	10560	70	4600	46
Raspberry Plant	487445	427594	✓	×	245	3		5		1	135.427	8724	76	9640	34
Road	306	522	✓	×	5						0.044	132	1	66	6
Rope Bridge	7256	7624	✓	×	132	40	16	170			0.515	12254	279	2376	27
Shinto Watchtower	131367	134770	×	×	1387	1	80	1		11	16.909	12968	122	12408	75
Shirakawago House	15965	21329	✓	×	1299	11	2	80			1.487	11424	100	3009	41
Snowflake Orb	77418	38395	✓	✓	1273			156		66	7.822	22768	78	39433	39
Sony Pc	15512	24533	✓	×	171	6		42		5	2.341	3440	30	3864	18
Space Fighter	34965	55060	✓	×	158		4	23			8.472	4512	16	5593	39
Spacestation V3	35488	89668	✓	×	1667	8	91	181			3.808	6170	165	2132	28
Speeder Bike	11926	15152	✓	×	239	1	3	20		20	1.305	3924	20	6096	42
Spider Tank	106464	204581	✓	×	244	11	3	125		2	15.844	7560	41	4104	19
Spiral Staircase	6358	7437	×	×	24	40				16	0.861	3556	35	4502	30
Guard Rail	14744	14559	✓	×	16	13		11			1.985	2428	29	2836	14
Table	66	154	✓	×	11						0.005	1210	11	266	12
Tank	15306	21338	✓	×	16			7			2.636	4784	19	3672	17
Teacup	1465	2360	✓	×	117			3			0.076	10670	37	3554	39
Training Dummy	20816	17483	✓	×	8			19			2.981	4586	14	4108	17
Turtle Castle	378627	405918	✓	×	384	1		56		6	64.410	7852	49	7942	24
Robot Vera	47790	66243	✓	×	138		12	146			5.829	9774	46	7874	24
War Tank	18477	26601	×	×	57	3		5			2.474	7992	98	2236	29
Wat Benchamabophit	999956	1115136	✓	×	1621	17	6	54		8	178.146	6376	56	6096	44
White Headed Vulture	115998	65304	✓	×	198					2	16.866	4006	15	6488	18
Yeahright	377344	377084	✓	✓	761	13	4	22			70.787	4860	47	38432	30

Table 3: *Primitive counts of our approach on our dataset from Sketchfab, compared to CoACD [WLLS22] and V-HACD [MG09]. Empty cells indicate 0. Non-manifoldness indicates if there are any non-manifold edges, and (WT) watertightness indicates if there are no boundary edges. Convex primitive decomposition's output varies heavily with the structure of the input mesh, sometimes favoring one kind of primitive over others.*

1-way (New → Input) Dist.↓	Ours		CoACD		V-HACD	
Name	Hausdorff	Chamfer	Hausdorff	Chamfer	Hausdorff	Chamfer
Armadillo	0.10065	0.01314	0.08252	0.01008	**0.07888**	**0.00848**
Armored Charizard	**0.01523**	**0.00071**	0.07099	0.01056	0.07472	0.01009
Bell	0.06347	0.01062	**0.03188**	**0.00663**	0.09992	0.01308
Bicycle	**0.01196**	**0.00221**	0.03536	0.00935	0.04943	0.00718
Canon At1	0.12554	0.02809	**0.12344**	0.01619	0.12567	**0.01438**
Cardboard Boxes	**0.01531**	**0.00236**	0.03449	0.00758	0.05353	0.00563
Casio Keyboard	**0.01388**	**0.00296**	0.02015	0.00997	0.04523	0.00490
Chimney Pipe	0.13351	0.02427	**0.08515**	0.01136	0.08925	**0.00826**
Chitinous Knight	0.07392	0.01502	**0.06393**	0.01201	0.06716	**0.00719**
Chuo House	0.10229	**0.00340**	**0.08021**	0.01410	0.08105	0.01114
Church Organ	**0.03531**	0.00746	0.03630	0.00946	0.03745	**0.00493**
Cinema Scan	0.07168	**0.00214**	**0.03566**	0.00906	0.05282	0.00793
Cube	**0.00260**	**0.00025**	0.02967	0.00810	0.04041	0.00707
Cyberpunk Atm	**0.04043**	**0.00746**	0.07036	0.01175	0.05880	0.00847
Cyberpunk Bike	0.07587	0.01264	**0.03430**	0.00912	0.04883	**0.00599**
Dojo	0.11107	0.01811	**0.03127**	0.00792	0.09150	**0.00762**
Dungeon Level (Precise)	**0.00570**	**0.00294**	0.01758	0.00548	0.05483	0.00483
Espresso Machine	**0.02850**	**0.00691**	0.07415	0.01195	0.06583	0.01048
Fantasy Asian House	0.06878	**0.00547**	0.05872	0.01137	**0.05344**	0.00748
Fps-Hands	**0.01239**	**0.00383**	0.03822	0.00944	0.04366	0.00614
Fractal	**0.00003**	**0.00003**	0.02123	0.00506	0.03494	0.00619
French Halfbasket	**0.00596**	**0.00065**	0.03245	0.00838	0.01955	0.00204
Greek Vase	**0.00669**	**0.00176**	0.03433	0.00998	0.11744	0.01602
Gun	**0.03779**	**0.00673**	0.05785	0.01105	0.05903	0.00723
Hover Bike	**0.05884**	**0.00960**	0.06923	0.01254	0.07398	0.01110
Jpn Corridor	**0.00770**	**0.00003**	0.02189	0.00832	0.12483	0.00890
Jpn House	0.09144	0.02295	**0.03314**	0.00817	0.25382	**0.00716**
Jpn House 2	**0.01592**	**0.00195**	0.03046	0.00715	0.03925	0.00452
Jpn House 3	**0.01791**	**0.00021**	0.12074	0.01888	0.12718	0.01599
Jpn Paddle Crab	**0.02933**	**0.00562**	0.07334	0.00910	0.08255	0.00969
Shinto Shrine	0.04818	0.00908	**0.04068**	0.00880	0.05948	**0.00786**
Kagiya Edo Tokyo	**0.00642**	**0.00176**	0.10794	0.02253	0.10737	0.01558
Lantern	0.11232	0.01439	**0.10980**	0.01612	0.11202	**0.01331**
Lekythos	**0.04035**	0.00738	0.04216	0.01160	0.06151	**0.00709**
Lethe	0.07164	**0.00751**	**0.04467**	0.00822	0.09024	0.00923
Marble Track	**0.01486**	**0.00053**	0.02851	0.00630	0.10007	0.00621
Maze	**0.00057**	**0.00043**	0.00701	0.00247	0.01625	0.00194
Mech	0.08841	0.01407	0.06509	0.00984	**0.05552**	**0.00670**
Melon Pallet	**0.01604**	**0.00334**	0.05276	0.00656	0.05174	0.00426
Militech Robot	**0.00808**	**0.00221**	0.04393	0.00707	0.05294	0.00760
Modular Fence Corner	0.25632	0.02719	**0.02813**	0.00839	0.06559	**0.00594**
Old Tree	0.08825	0.00616	**0.03537**	**0.00410**	0.06434	0.01040
Pipe Wall	**0.00977**	**0.00240**	0.03148	0.00733	0.05593	0.00590
Pipes	**0.01968**	0.00718	0.03984	0.00967	0.04104	**0.00656**
Potted Plant	0.05526	0.01271	**0.05503**	**0.01049**	0.08425	0.01146
Raspberry Plant	0.06606	**0.00817**	**0.03220**	0.00833	0.04783	0.00877
Road	**0.00153**	0.00066	0.01749	0.01075	0.00344	**0.00006**
Rope Bridge	0.05778	0.01795	**0.01827**	0.00678	0.03575	**0.00648**
Shinto Watchtower	**0.03843**	0.00958	0.03871	0.00944	0.05066	**0.00851**
Shirakawago House	**0.06299**	**0.00490**	0.12086	0.01269	0.12261	0.01411
Snowflake Orb	**0.01533**	**0.00449**	0.02962	0.00795	0.03527	0.00569
Sony Pc	0.05983	**0.00839**	**0.04663**	0.01307	0.11451	0.01695
Space Fighter	0.04089	0.00609	0.03990	0.00673	**0.03338**	**0.00309**
Spacestation V3	**0.02719**	**0.00248**	0.02855	0.00641	0.11113	0.03307
Speeder Bike	**0.05146**	0.00897	0.05253	0.00956	0.05333	**0.00728**
Spider Tank	**0.00962**	**0.00192**	0.03645	0.00769	0.05948	0.00760
Spiral Staircase	0.13282	0.01846	**0.03745**	0.00972	0.05840	**0.00797**
Guard Rail	**0.00502**	**0.00173**	0.03474	0.00778	0.04421	0.00988
Table	**0.00207**	**0.00098**	0.03840	0.01053	0.03449	0.00234
Tank	**0.02078**	**0.00533**	0.16587	0.02053	0.16880	0.02079
Teacup	**0.01361**	**0.00415**	0.03497	0.00972	0.08308	0.00593
Training Dummy	**0.03855**	0.01072	0.09124	0.01129	0.05419	**0.00929**
Turtle Castle	**0.05321**	**0.00784**	0.09978	0.01941	0.10281	0.01814
Robot Vera	**0.01120**	**0.00115**	0.03402	0.00710	0.04165	0.00605
War Tank	0.04930	0.00802	**0.04800**	0.00721	0.05458	**0.00662**
Wat Benchamabophit	**0.01643**	**0.00196**	0.03528	0.00801	0.08985	0.00699
White Headed Vulture	**0.03381**	**0.00180**	0.09739	0.01401	0.09997	0.01310
Yeahright	**0.00333**	**0.00092**	0.03685	0.00939	0.06662	0.01072

Table 4: *Full set of comparisons of our approach on our dataset. Convex primitive decomposition is on average closer to the input mesh than CoACD [WLLS22] and V-HACD [MG09] on both the chamfer and hausdorff distance. Distances are normalized as* $\frac{\textit{Hausdorff/Chamfer New} \rightarrow \textit{Input}}{||\textit{Bounding Box Diag}||_2}$.

Model Name	Ours		CoACD			V-HACD		
	Floats	Total	Floats	Ints	Total	Floats	Ints	Total
Bytes Used	4	Bytes	4	2	Bytes	4	2	Bytes
Armadillo	1989	**7956**	14592	28896	116160	17835	33825	138990
Armored Charizard	1876	**7504**	11001	21510	87024	3810	7206	29652
Bell	512	**2048**	17697	34638	140064	9726	18594	76092
Bicycle	914	**3656**	7536	14712	59568	14268	27318	111708
Canon At1	108	**432**	11394	22452	90480	4959	9522	38880
Cardboard Boxes	450	**1800**	8742	16704	68376	987	1488	6924
Casio Keyboard	160	**640**	666	1320	5304	1065	1974	8208
Chimney Pipe	48	**192**	8193	16062	64896	3813	7002	29256
Chitinous Knight	484	**1936**	6570	12948	52176	20706	39057	160938
Chuo House	10652	42608	4971	9594	**39072**	8142	15600	63768
Church Organ	1536	**6144**	8649	16650	67896	5019	9366	38808
Cinema Scan	4931	**19724**	13098	25152	102696	9882	18090	75708
Cube	1545	**6180**	11451	21858	89520	11013	21228	86508
Cyberpunk Atm	798	**3192**	6018	11736	47544	2346	4374	18132
Cyberpunk Bike	500	**2000**	7011	13842	55728	15357	28878	119184
Dojo	4807	**19228**	15693	29598	121968	3894	7062	29700
Dungeon (Precise)	5383	21532	11016	20952	85968	2127	3894	**16296**
Espresso Machine	1423	**5692**	16875	32634	132768	7389	14004	57564
Fantasy Asian House	39250	157000	5490	10644	**43248**	16524	32820	131736
Fps-Hands	1485	**5940**	3780	7416	29952	3813	7320	29892
Fractal	54340	217360	86478	159444	664800	4845	9030	**37440**
French Halfbasket	2891	11564	912	1788	**7224**	16737	30546	128040
Greek Vase	3790	**15160**	28308	55872	224976	19779	38370	155856
Gun	1299	**5196**	5133	10062	40656	3627	6924	28356
Hover Bike	702	**2808**	5982	11772	47472	4491	8550	35064
Jpn Corridor	330	**1320**	7335	14262	57864	936	1626	6996
Jpn House	9005	**36020**	9948	18996	77784	6360	12030	49500
Jpn House 2	4119	**16476**	6048	11628	47448	3552	6552	27312
Jpn House 3	6490	25960	3033	5910	23952	2316	4212	**17688**
Jpn Paddle Crab	11591	**46364**	17610	34452	139344	27912	49737	211122
Shinto Shrine	1092	**4368**	11088	21348	87048	3660	6810	28260
Kagiya Edo Tokyo	8345	33380	1674	3264	13224	1068	1956	**8184**
Lantern	256	**1024**	1104	2148	8712	636	1176	4896
Lekythos	1198	**4792**	13824	27168	109632	5598	10494	43380
Lethe	1723	**6892**	6408	12516	50664	3420	6420	26520
Marble Track	2012	**8048**	11691	22782	92328	6438	11838	49428
Maze	70000	280000	330	212022	425364	6105	10884	**46188**
Mech	513	**2052**	8601	16854	68112	4185	7896	32532
Melon Pallet	380	**1520**	15009	29370	118776	8127	14700	61908
Militech Robot	1636	**6544**	19971	39150	158184	9144	17352	71280
Modular Fence Corner	65	**260**	7938	15720	63192	2472	4806	19500
Old Tree	998	**3992**	9534	18612	75360	4878	9237	37986
Pipes	441	**1764**	8661	16998	68640	4872	9096	37680
Pipe Wall	832	**3328**	11766	22692	92448	3069	5520	23316
Potted Plant	695	**2780**	16260	31680	128400	7437	13800	57348
Raspberry Plant	2517	**10068**	13542	26172	106512	16023	28920	121932
Road	50	**200**	204	396	1608	135	198	936
Rope Bridge	2854	**11416**	20055	36762	153744	3891	7128	29820
Shinto Watchtower	14325	**57300**	20184	38904	158544	19698	37224	153240
Shirakawago House	13635	54540	17736	34272	139488	4854	9027	**37470**
Snowflake Orb	14548	**58192**	34620	68304	275088	67422	118299	506286
Sony Pc	2101	**8404**	5340	10320	42000	6336	11592	48528
Spacestation V3	18357	73428	10245	18510	78000	3432	6396	**26520**
Space Fighter	1757	**7028**	6864	13536	54528	8802	16779	68766
Speeder Bike	2769	**11076**	6006	11772	47568	9642	18288	75144
Spider Tank	3426	**13704**	11586	22680	91704	6510	12312	50664
Spiral Staircase	696	**2784**	5544	10668	43512	7110	13506	55452
Guard Rail	328	**1312**	3816	7284	29832	4473	8508	34908
Table	110	**440**	1881	3630	14784	471	798	3480
Tank	209	**836**	7290	14352	57864	5781	11016	45156
Teacup	1191	**4764**	16227	32010	128928	5676	10662	44028
Training Dummy	213	**852**	6963	13758	55368	6375	12324	50148
Turtle Castle	4305	**17220**	12072	23556	95400	12660	23826	98292
Robot Vera	2450	**9800**	14937	29322	118392	12339	23622	96600
War Tank	626	**2504**	12576	23976	98256	3564	6708	27672
Wat Benchamabophit	16819	**67276**	9900	19128	77856	9624	18288	75072
White Headed Vulture	2002	**8008**	6099	12018	48432	10590	19464	81288
Yeahright	7871	**31484**	7572	14580	59448	59139	115296	467148

Table 5: *Comparison of memory cost for each model in our dataset, measured in bytes. Our approach on whole uses less memory than the other approaches. For floats, we treat them as 4 bytes (single-precision* `float` *in C), and for integers we treat them as unsigned 2 byte integers (*`uint16_t` *in C), even though in some cases it would overflow.*

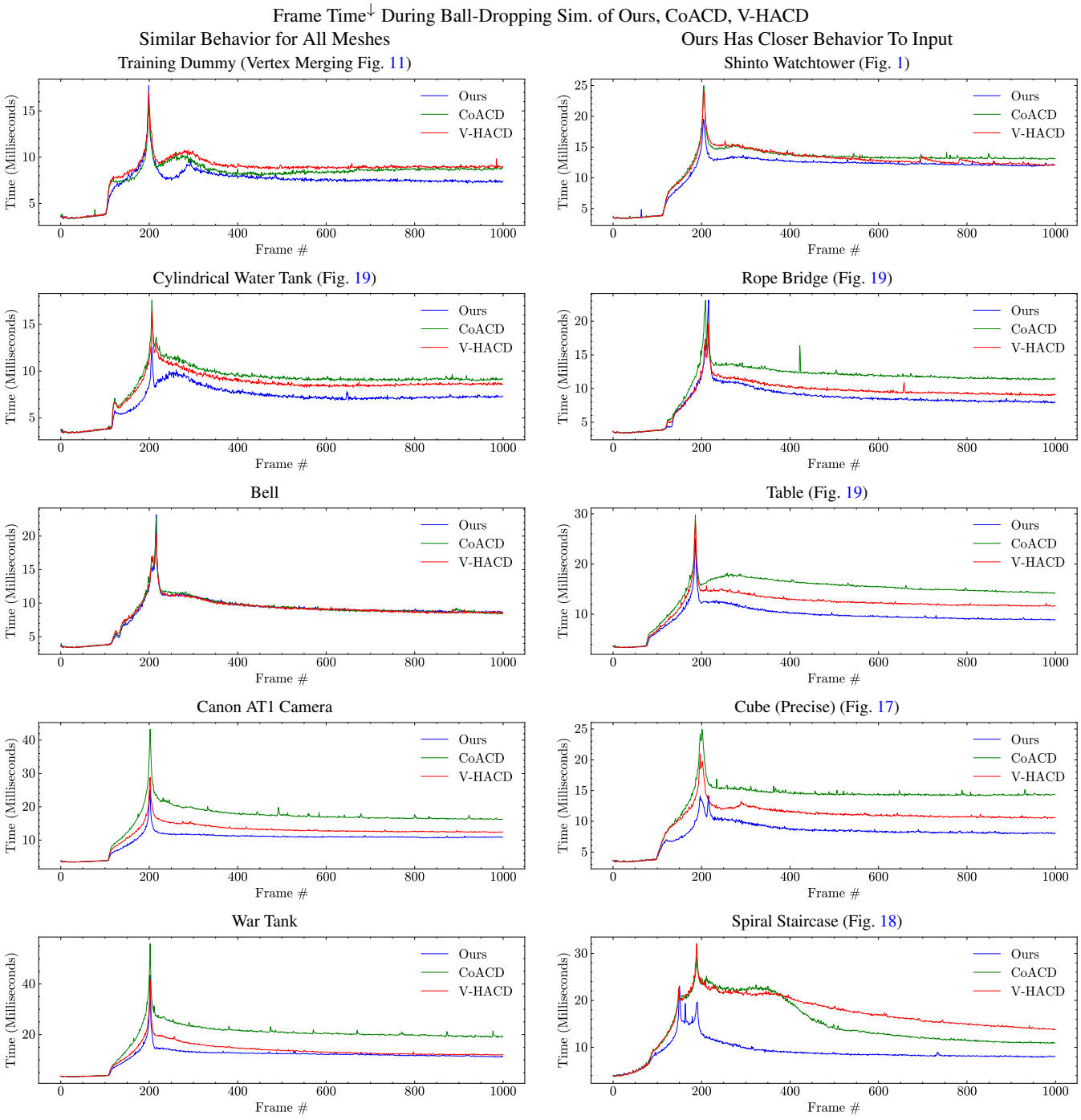


Figure 20: *Comparison of frame duration for collision detection when dropping 5000 spheres on different meshes, where the left-column is meshes where the behavior of all colliders is relatively similar, and the right column is the performance when our collider has visually more similar performance to the input. In both cases, our approach has comparable or better performance than CoACD and V-HACD.*

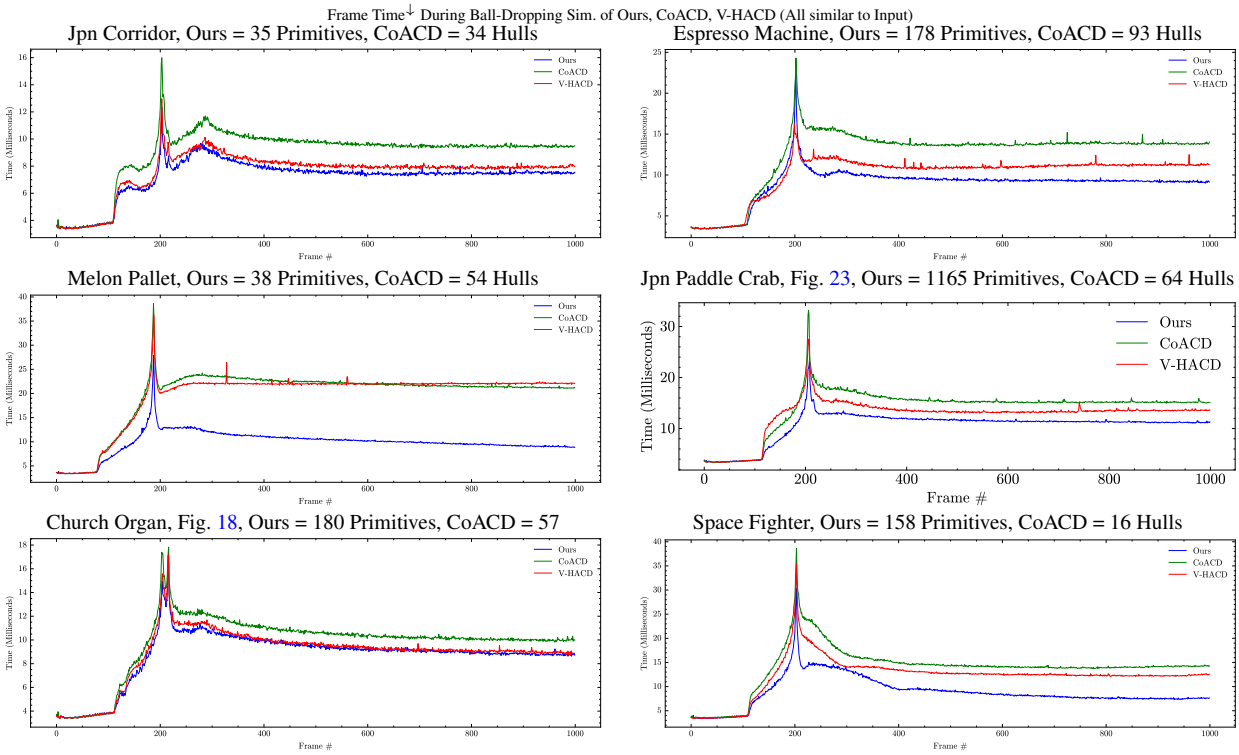
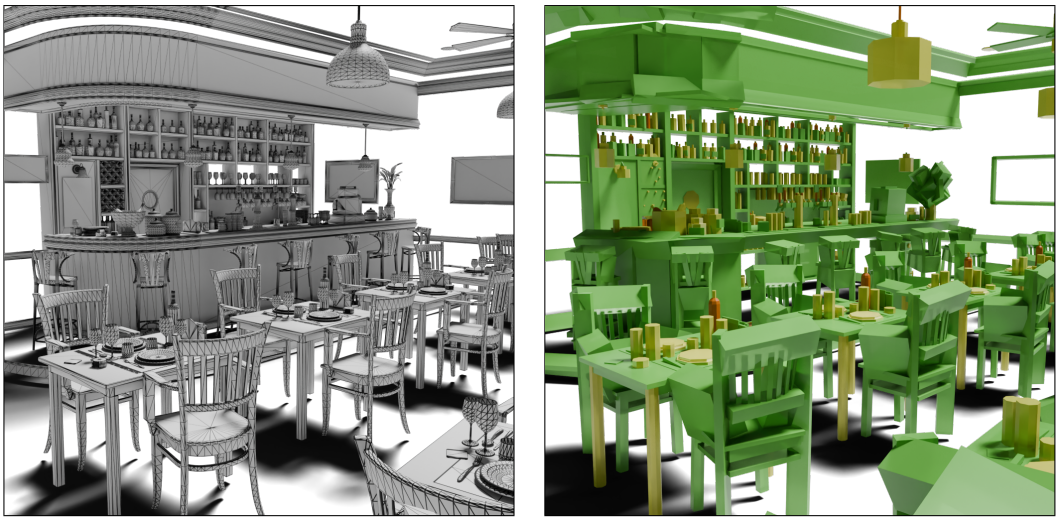


Figure 21: *Comparison of our approach against CoACD and V-HACD, considering the number of components in our approach versus CoACD. Our approach is always faster or equal to approximate convex decomposition, regardless of the number of components.*



|F| = 570588 | 2343 Boxes, 115 Capsules, 1343 Cylinders

Figure 22: *An additional example of our approach on a dense, complex scene [Lum17]. The entire scene is processed in one pass in under 2 minutes, and naturally decomposes objects into constituent parts. We use our approach to generate an initial decomposition, then automatically delete any extremely thin bounding boxes which have radius on any axis $\leq 1 \times 10^{-4}$. This postprocessing is included because around the placemats there are a number of frills which contribute a large number of primitives but have essentially 0 volume. Furthermore, many walls are entirely planar but may not be rectangular, leading to regions jutting out. Removing thin bounding boxes fixes both problems. Green indicates bounding boxes, yellow for cylinders, and red for capsules.*

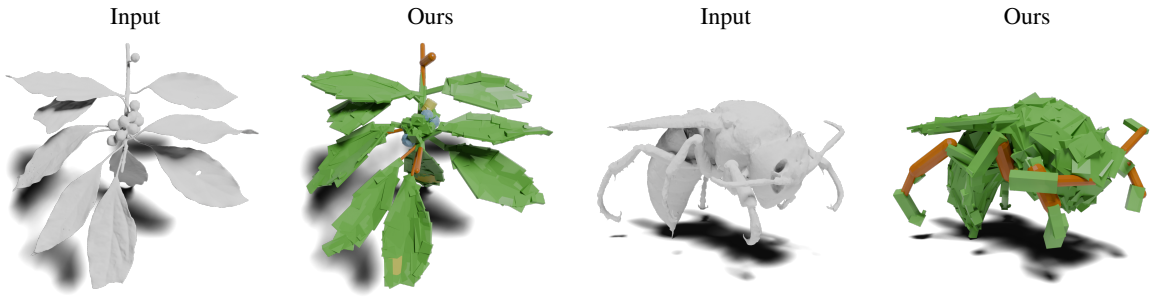


Figure 23: *We show our approach on organic meshes, which are not represented heavily in our dataset. Our approach is primarily designed for artist created meshes for games, but can capture coarse details on organic meshes. A larger number of primitives are used, as each primitive alone cannot capture curved surfaces well. For these models, we change weights so that spheres are 0.7, capsules are 0.85, and isosceles prisms are disallowed. The current approach for these kinds of mesh is manual creation, convex hulls, or coarse approximations. ffish.asia / floraZia.com.*

Figure 24: *Our approach on a particularly challenging mesh, "Yeahright". Our approach can preserve thin structures and holes. Keenan Crane.*

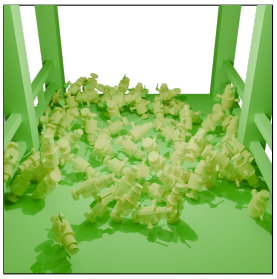


Figure 25: *Visualization of many training dummies from Fig. 11, colliding inside of "Jpn. Corridor Middle".*

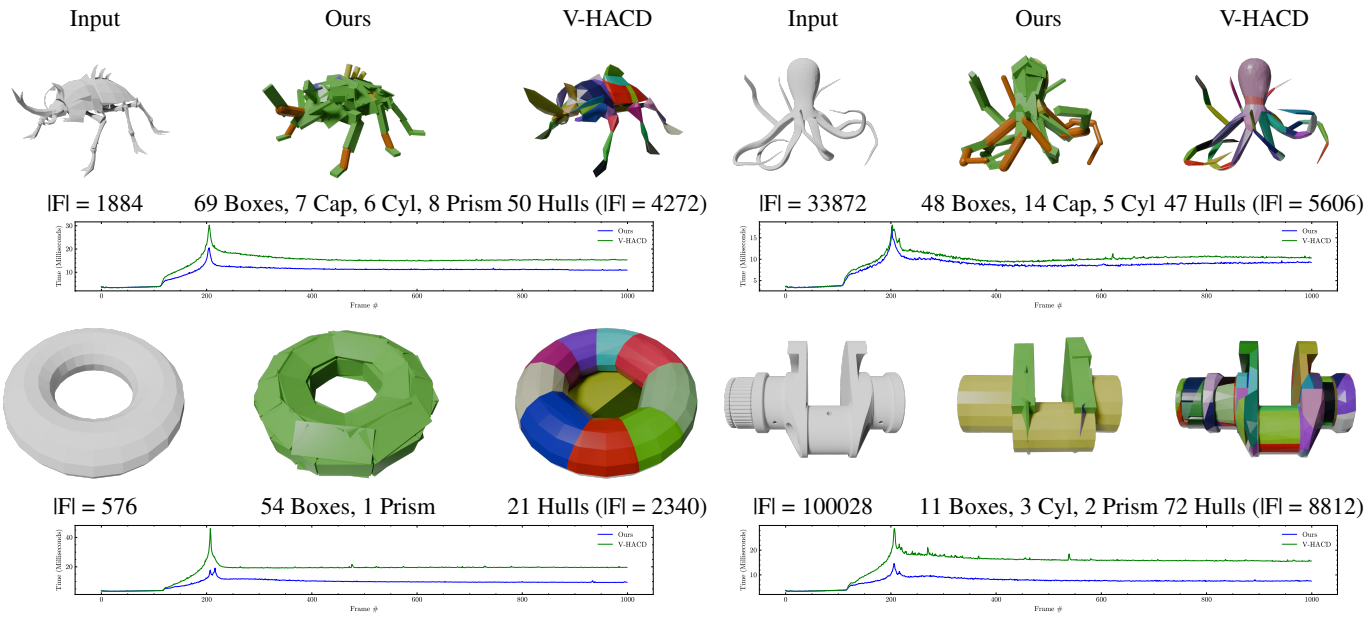


Figure 26: *Comparison of our approach to V-HACD [MG09] on a few models from their dataset. Our approach's decomposition often breaks up the input mesh in similar positions to V-HACD, while improving collision performance.*