Approximate Convex Decomposition and Transfer for Animated Meshes

DANIEL THUL, ETH Zurich
ĽUBOR LADICKÝ, ETH Zurich and Apagom AG
SOHYEON JEONG, Apagom AG
MARC POLLEFEYS, ETH Zurich and Microsoft

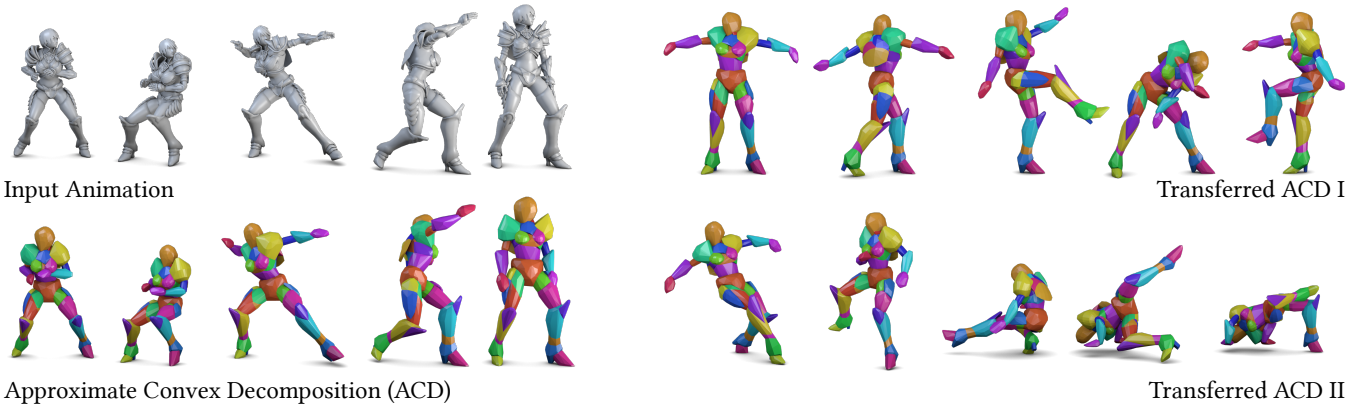


Fig. 1. Temporally coherent approximate convex decomposition of an animated mesh and transfers into new, previously unseen animations.

Many geometric quantities can be computed efficiently for convex meshes. For general meshes, methods for approximate convex decomposition have been developed that decompose a static, non-convex object into a small set of approximately convex parts. The convex hulls of those parts can then be used as a piecewise convex approximation to the original mesh.

While previous work was only concerned with static meshes, we present a method for decomposing animated 3D meshes into temporally coherent approximately convex parts. Given a mesh and several training frames—that is, different spatial configurations of its vertices—we precompute an approximate convex decomposition that is independent of any specific frame. Such a decomposition can be transferred in real-time to novel, unseen frames. We apply our method to a variety of pre-animated meshes as well as a 3D character interactively controlled by a user's body pose. We further demonstrate that our method enables real-time physics simulations to interact with animated meshes.

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

Additional Key Words and Phrases: convex decomposition, transfer

Authors' addresses: Daniel Thul, Department of Computer Science, ETH Zurich, daniel.thul@inf.ethz.ch; Ľubor Ladický, Department of Computer Science, ETH Zurich, Apagom AG, lubor.ladicky@inf.ethz.ch; SoHyeon Jeong, Apagom AG, sohyeon@apagom.com; Marc Pollefeys, Department of Computer Science, ETH Zurich, Microsoft, marc.pollefeys@inf.ethz.ch.

0730-0301/2018/11-ART226
https://doi.org/10.1145/3272127.3275029

ACM Reference Format:
Daniel Thul, Ľubor Ladický, SoHyeon Jeong, and Marc Pollefeys. 2018. Approximate Convex Decomposition and Transfer for Animated Meshes. *ACM Trans. Graph.* 37, 6, Article 226 (November 2018), 10 pages. https://doi.org/10.1145/3272127.3275029

1 INTRODUCTION

As hardware capabilities increase, modern graphical applications make use of ever larger and more detailed 3D models. Rendering methods reduce the computation time such large meshes would require by employing specialized acceleration structures such as octrees and bounding volume hierarchies (BVH) that efficiently cull irrelevant parts of the mesh. While this works well for ray-mesh intersection queries, other applications such as physics simulation need to answer more complex questions involving these meshes. They need to be able to efficiently determine whether a given point in space lies inside or outside of a mesh, whether two meshes intersect, how far they interpenetrate and so on. Answering these questions is non-trivial and computationally expensive for general meshes. The problems would suddenly become much simpler if one knew that all participating meshes were convex, since for convex meshes those questions turn into convex optimization problems that can be efficiently solved using greedy algorithms. This observation motivates the decomposition of meshes into a set of convex parts. Splitting the mesh into exactly convex parts [Chazelle 1981] generally produces too many pieces to be usable in practice and has been superseded by approximate convex decomposition [Ghosh et al. 2013; Lien and Amato 2007; Liu et al. 2016; Mamou 2016; Mamou and Ghorbel 2009], which decomposes the mesh into a small set of approximately convex parts, whose bounding hulls can be used as a piecewise convex approximation to the original mesh.

While prior work was concerned with decompositions of static meshes, we present an algorithm for the approximate convex decomposition of animated meshes. Our algorithm splits animated meshes into temporally coherent parts with tight bounding hulls. Different levels of detail arise naturally from the formulation of our algorithm and can be used as required by the problem at hand. Furthermore, once a decomposition has been pre-computed, it can be transferred in real-time to any pose of the mesh as long as that mesh's topology stays constant. This enables us to produce high-quality approximations of animated meshes for real-time and interactive settings such as games and virtual reality.

1.1 Related Work

There are various algorithms that focus on different aspects of mesh representation. This includes algorithms for the purpose of segmentation [Kalogerakis et al. 2010; Koschan 2003], mesh editing [Shlafman et al. 2002], structure extraction [de Goes et al. 2008; Shatz et al. 2006; Wu and Kobbelt 2005], recombination [Kreavoy et al. 2007] and shape approximation [Calderon and Boubekeur 2017] to name a few.

Of particular interest to us are algorithms that decompose a mesh into a set of convex parts, a problem that has been studied in the past. [Chazelle 1981] shows that a non-convex polyhedron can be decomposed into a set of convex polyhedra by iteratively cutting through carefully chosen edges of the polyhedron. Each cut splits the mesh into smaller, increasingly more convex parts until all parts are exactly convex. Decomposing a mesh into exactly convex parts is often impractical though due to the high number of parts this process can generate. Instead, approximate convex decomposition was developed to segment a mesh into a small number of approximately convex parts. The convex hulls of these parts can then serve as a convex approximation to the mesh. Two different approaches are used for finding approximate convex decompositions: surface based and volumetric methods.

Surface based methods cluster the faces of a mesh into nearly convex patches bounded by edge loops. ACD [Lien and Amato 2004, 2007] is one such method which was the first to focus on *approximate* convex decomposition. They measure concavity (the opposite of convexity) as the distance from a vertex on the mesh to a carefully chosen part of its convex hull. They then iteratively cut patches along the vertex with the highest concavity until a global concavity threshold has been reached. FACD [Ghosh et al. 2013] builds on this foundation and introduces the notion of relative concavity. Instead of optimizing a global concavity measure they use dynamic programming to find cuts which maximize the change in concavity induced by this cut. This leads to decompositions which can retain smaller mesh features like fingers and toes while at the same time keeping larger structures like head and torso intact. CoRise [Liu et al. 2016] is another method following the definition of concavity introduced in [Lien and Amato 2007]. It iteratively clusters vertices on the convex hull to form convex ridges respecting a given concavity threshold. The mesh is then split into components according to the convex ridges by finding appropriate cuts between them using a graph cut method. In [Au et al. 2012], concavity-sensitive scalar fields are computed across the surface to locate isolines that are then used as seams for subsequent cuts into surface patches. HACD [Mamou and Ghorbel 2009] works by iteratively collapsing edges of the mesh's dual graph. This simplification process is guided by a concavity measure and a regularization term that keeps the resulting components compact. Since surface-based methods do not consider a volumetric representation of the mesh they are not suitable for meshes of arbitrary shape: for example, the outside surface of a handle-less cup is a perfectly convex patch. Its convex hull however fills the inside of the cup and is as such an unsuitable approximation to the mesh.

In response to the shortcomings of surface based methods, V-HACD [Mamou 2016] has been developed as a volumetric method. In a first step it creates a binary voxelization of the mesh using a regularly sampled grid. The voxel representation is then iteratively split along the grid boundaries, minimizing a convex approximation error. The convex hulls of the resulting voxel clusters can be used as a convex approximation to the mesh. A different approach is to take a tetrahedralized mesh and create a hierarchical decomposition into polyhedra using a bottom-up clustering [Attene et al. 2008]. Other methods employ the pairwise inner visibility between surface points as an implicitly volumetric representation and do not require a voxelization or tetrahedralization of the mesh but a sampling of surface points instead [Asafi et al. 2013; Kaick et al. 2014]. Somewhere between surface and volume based approaches lies [Ren et al. 2011]. Like the previous method they also employ pairwise connections between surface points but compute perpendicular distances between those connections and the mesh surface instead of visibilities. This makes the method volumetric in 2D. In three dimensions, they use 2D projections of the mesh to compute this concavity information which makes the method not fully volumetric anymore.

All of these methods, whether surface-based or volumetric, work on a single static mesh. For animated meshes the decomposition would need to be recomputed each frame, rendering it too expensive for use in real-time and interactive settings. We develop a novel algorithm for the approximate convex decomposition of animated meshes instead.

Our algorithm is based on the idea of iteratively cutting through the animated mesh by choosing a suitable frame for the next cut and transferring this cut to all other frames of the animation. This iterative cutting approach has two advantages compared to existing methods for approximate convex decomposition: Firstly, unlike surface based methods which fail for certain shapes, it works for general polyhedral meshes. Secondly, we show that cuts through a polyhedral mesh are suitable for transfer from one mesh to another as long as those meshes share the same topology (e.g. between the frames of a mesh animation). Such a transfer would not be possible for a voxel method like V-HACD, since voxelizations of meshes of same topology do not necessarily correspond to one another.

Several related works address various decompositions of animated meshes, though none of them into approximate convex parts. [Wuhrer and Brunton 2010] find the near-rigid components of a mesh by building its dual graph and clustering the dual graph's vertices according to changes in the dihedral angles over the course of the animation. [Liao et al. 2012] first compute multi-scale high dimensional feature vectors on the mesh surface and then cluster those features to extract near-rigid parts. Similar to near-rigid component

extraction, [Lee et al. 2005] use the motion similarity of vertices and cluster them into components using a region growing approach. [Thiery et al. 2016] compute a multi-resolution approximation of the animated mesh based on a hierarchy of spheres which they use for collision detection and animation compression. This works well as a coarse approximation but has problems approximating objects with sharp edges and corners.

While our method is not itself a collision detection method, its results can be used as a basis for efficient collision detection with static or animated meshes in the narrow phase using efficient collision detection methods such as GJK [Cameron 1997; Gilbert et al. 1988]. In the broad phase of collision detection any of the many available acceleration structures can be used, most notably bounding volume hierarchies based on primitives like oriented boxes [Gottschalk et al. 1996], spheres [Hubbard 1995] or convex polytopes [Klosowski et al. 1998] as well as BVHs optimized for animated scenes [James and Pai 2004; Kavan and Žára 2005].

1.2 Overview

We introduce the notation that will be used throughout the paper:

Let $\mathcal{M}$ be a closed two-manifold mesh with N vertices and a set of edges and faces. A *static* mesh has a single set of vertex positions $\mathcal{V}$:

$$\mathcal{V} = \{v_1, \dots, v_N\} \qquad v_i \in \mathbb{R}^3 \tag{1}$$

An *animated* mesh has K sets of vertex positions $\mathcal{V}^1, \dots, \mathcal{V}^K$, called *frames*:

$$\mathcal{V}^j = \{v_1^j, \dots, v_N^j\} \qquad v_i^j \in \mathbb{R}^3 \tag{2}$$

When referring to the topological concept of a vertex as opposed to its position in space we write $\hat{v}_i$ to mean the i-th vertex of $\mathcal{M}$.

A *decomposition* $\mathcal{D} = \{\mathcal{P}_1, \dots\}$ of $\mathcal{M}$ is a set of submeshes $\mathcal{P}_i$, called *parts*, that make up $\mathcal{M}$.

Our goal is to find a decomposition of the mesh into parts such that each part is approximately convex across all frames. This will allow us to approximate the mesh by the convex hulls of those parts. We also want to be able to efficiently apply the precomputed approximate convex decomposition to novel unseen frames. This enables the use of our technique in interactive and real-time settings where not all frames are known in advance or where it is too expensive to store a decomposition for each frame.

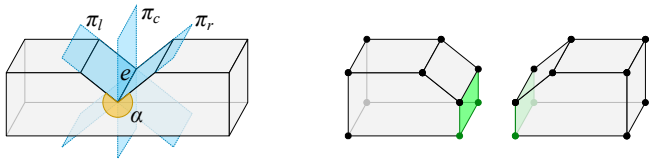
Section 2 shows how we compute an approximate convex decomposition for a static mesh.

Section 3 describes how we enable decompositions to be transferred from one frame of a mesh to another by choosing a suitable, frame-independent representation of the decomposition.

Section 4 then combines static approximate convex decomposition with the ability to transfer decompositions between frames to design an algorithm for approximate convex decomposition of animated meshes.

2 STATIC APPROXIMATE CONVEX DECOMPOSITION

Before we derive our algorithm for approximate convex decomposition of static meshes we lay the theoretical groundwork by first looking at exact convex decomposition.



(a) Different hyperplane orientations around concave edge e.

(b) The two parts after cutting along hyperplane π_c.

Fig. 2. Cut through a mesh along a concave edge. *(a)* The mesh before the cut. An infinite number of hyperplanes can be constructed that contain the concave edge e and split the dihedral outer angle α into two parts. Pictured are three possibilities: π_c, the hyperplane that exactly bisects the dihedral angle α as well as π_l and π_r, the hyperplanes with the most extreme rotation towards the left and right adjacent face respectively. Any rotation in between would also produce a valid hyperplane. *(b)* The mesh after the cut along hyperplane π_c. Open holes are closed by introducing new vertices, edges and faces along the cut, here shown in green. The previously concave edge e is present in both new parts but guaranteed to be convex now.

2.1 Exact Convex Decomposition

There exists a conceptually simple algorithm to produce an exact convex decomposition of a mesh [Chazelle 1981]. The idea is to iteratively cut the mesh along concavities into smaller parts until all parts are convex. This set of parts forms a non-unique, exact convex decomposition of the mesh. Figure 2 depicts this process.

Let us formalize this algorithm: We say that an edge (a vertex in two dimensions) is *concave* if the dihedral outer angle enclosed by its two adjacent faces is larger than 180 degrees. One can construct a hyperplane π_e that contains this edge e and splits the dihedral angle into two parts (see Figure 2a). Cutting the mesh along this hyperplane creates two new smaller parts in which the offending edge is not concave anymore. New vertices are introduced where edges of the original mesh intersect the hyperplane and open holes are closed by introducing new edges and faces along the cutting plane (see Figure 2b). Note that such a newly added vertex is a linear combination of the edge's end vertices, weighted by the position along the edge the new vertex was introduced at. The algorithm iteratively proceeds to cut the new smaller parts until there are no more concave edges left to cut. The output of the algorithm is the set of connected components of the generated parts. We refer to the original mesh as $\mathcal{M}$ while we call parts created by the cutting process $\mathcal{P}$. Algorithm 1 summarizes this process.

If the original mesh contained n concave edges Algorithm 1 can produce up to 2^n convex parts and is as such not suitable for real-world use. Further, the algorithm does not describe how to orient the hyperplanes and in which order to do the cuts. In the next section we will solve these problems and derive a method for approximate convex decomposition.

2.2 Approximate Convex Decomposition

In order to turn the algorithm for exact convex decomposition into a practical algorithm for approximate convex decomposition we

Algorithm 1: Exact convex decomposition.

Input : Static mesh $\mathcal{M}$
Output: Exact convex decomposition $\mathcal{D}$

$\mathcal{D} \leftarrow \emptyset$
activeParts $\leftarrow \{\mathcal{M}\}$
while activeParts $\neq \emptyset$ **do**
  $\mathcal{P} \leftarrow$ any element out of activeParts
  if $\mathcal{P}$ *does not contain a concave edge* **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{P}\}$
  else
    $\pi_e \leftarrow$ any hyperplane through e splitting outer angle
    $\mathcal{P}_1,\ \mathcal{P}_2 \leftarrow$ cut $\mathcal{P}$ at π_e
    activeParts $\leftarrow$ activeParts $\cup \{\mathcal{P}_1,\ \mathcal{P}_2\}$
$\mathcal{D} \leftarrow \bigcup_{\mathcal{P}\in\mathcal{D}}$ ConnectedComponents($\mathcal{P}$)

need to choose the cuts in such a way that already a small number of cuts—and thus a small number of parts—leads to a decomposition that is not exactly convex but is close enough.

To quantify the quality of each part of a decomposition, we define a *convex approximation error* $e(\mathcal{M})$ as a volume difference between a mesh and its convex hull:

$$e(\mathcal{M}) := \text{Volume}(\text{ConvexHull}(\mathcal{M})) - \text{Volume}(\mathcal{M}) \tag{3}$$

Note that convex objects by definition have zero convex approximation error. This measure is similar to other convexity measures like [Liu and Zhang 2007] but relies on absolute difference instead of relative difference to give more importance to bigger non-convex parts. Specialized convexity measures can be substituted but the one presented here works well for general application [Zunic and Rosin 2004].

Finding an optimal approximate convex decomposition $\mathcal{D}^*$ with M parts can now be expressed as minimizing the convex approximation errors of its parts:

$$\mathcal{D}^* = \arg\min_{\mathcal{D}} \sum_{\mathcal{P}\in\mathcal{D}} e(\mathcal{P}) \qquad \text{s.t. } |\mathcal{D}| = M \tag{4}$$

This optimization problem is in general intractable. Instead of solving for all M parts simultaneously, we iteratively add more parts to the decomposition. Given a decomposition with M' parts we greedily solve for the next best cut to produce a decomposition with $M' + 1$ parts. This process starts from the simplest possible decomposition—containing only the original mesh as a single part—and continues until the decomposition contains M parts.

Thus, given a current decomposition $\mathcal{D}$ with M' parts, we find a hyperplane π_e cutting through the part $\mathcal{P}^*$ that has the largest convex approximation error such that its error is minimized. Let $\mathcal{P}^*_{\pi_e,1}$ and $\mathcal{P}^*_{\pi_e,2}$ be the two parts that arise from cutting $\mathcal{P}^*$ at π_e. The optimal cutting plane π^*_e can then be defined as:

$$\pi^*_e = \arg\min_{\pi_e} \left(e(\mathcal{P}^*_{\pi_e,1}) + e(\mathcal{P}^*_{\pi_e,2}) - e(\mathcal{P}^*) \right) \tag{5}$$

where

$$\mathcal{P}^* := \arg\max_{\mathcal{P}\in\mathcal{D}} e(\mathcal{P}) \tag{6}$$

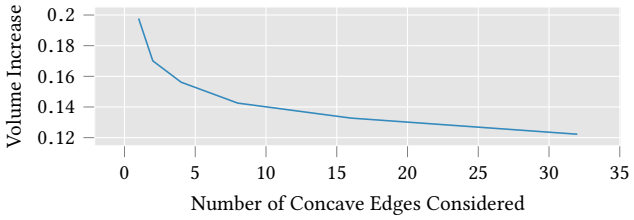


Fig. 3. Influence of the number of concave edges considered during decomposition. The horizontal axis shows the number of concave edges furthest from the convex hull taken into respect. The vertical axis shows the percentage of volume increase of the convex hulls of the decomposition with respect to the original mesh. The graph shown represents the numbers averaged across the V-HACD dataset.

Searching for cuts only in the part $\mathcal{P}^*$ with the largest convex approximation error as opposed to searching for cuts in all available parts $\mathcal{P} \in \mathcal{D}$ serves a dual purpose: firstly, it reduces the computational cost while still generally decreasing the optimization objective rapidly and secondly, it forces the algorithm to not ignore "difficult" parts. If for example $\mathcal{P}^*$'s convex approximation error cannot be reduced in a single cut, an algorithm looking at all parts would prefer to cut a different one even though $\mathcal{P}^*$ is responsible for the majority of the decomposition's total convex approximation error.

The new decomposition $\mathcal{D}'$ with $M' + 1$ parts is then the one where part $\mathcal{P}^*$ is replaced with its subparts $\mathcal{P}^*_{\pi^*_e,1}$ and $\mathcal{P}^*_{\pi^*_e,2}$:

$$\mathcal{D}' := \left(\mathcal{D} \setminus \mathcal{P}^*\right) \cup \left\{\mathcal{P}^*_{\pi^*_e,1},\ \mathcal{P}^*_{\pi^*_e,2}\right\} \tag{7}$$

Finding the hyperplane yielding the optimal cut (Equation 5) is still an intractable problem since there are infinitely many hyperplanes and the objective is highly non-convex. Fortunately, we know from Section 2.1 that we are guaranteed to converge to an exact convex decomposition if we only take hyperplanes containing a concave edge into account. This leaves us with a finite number of concave edges to construct hyperplanes from. We do not take all of those concave edges into account but only the subset of the C concave edges furthest from the convex hull. Figure 3 shows how this influences the decomposition result. In practice we choose $C = 8$.

We sample a fixed number of hyperplanes for each concave edge to make optimization feasible. In practice we construct $R = 5$ equiangularly rotated hyperplanes around each candidate edge (Figure 2a shows the setup for $R = 3$). If a part consists of linearly separable disconnected components, we construct additional hyperplanes splitting those components that are taken into account when choosing the optimal cut. Algorithm 2 summarizes the process.

We are now able to compute approximate convex decompositions for a static mesh (or a single frame of an animated mesh). In the next section we show how we enable decompositions to be transferred from one frame of a mesh to another. In Section 4 we then combine the static approximate convex decomposition with our ability to transfer decompositions between frames to design an algorithm for temporally coherent approximate convex decomposition and transfer for animated meshes.

Algorithm 2: Static approximate convex decomposition.

Input : Static mesh $\mathcal{M}$, target number of parts M, number of concave edges to sample C, number of hyperplane orientations R

Output : Approximate convex decomposition $\mathcal{D}$ with at most M parts

$\mathcal{D} \leftarrow \{\mathcal{M}\}$
while $|\mathcal{D}| < M$ *and there exists a concave edge* **do**
  $\pi_e^* \leftarrow$ nil
  minError $\leftarrow \infty$
  $\mathcal{P}^* \leftarrow \arg\max_{\mathcal{P}\in\mathcal{D}} e(\mathcal{P})$
  $E \leftarrow$ find the C concave edges $e \in \mathcal{P}^*$ furthest from ConvexHull($\mathcal{P}^*$)
  for *concave edge* $e \in E$ **do**
    $\Pi_e \leftarrow$ construct R hyperplanes π_e through e splitting the outer dihedral angle
    for *hyperplane* $\pi_e \in \Pi_e$ **do**
      $\mathcal{P}_1^*, \mathcal{P}_2^* \leftarrow$ cut $\mathcal{P}^*$ at π_e
      error $\leftarrow e(\mathcal{P}_1^*) + e(\mathcal{P}_2^*) - e(\mathcal{P}^*)$
      if error < minError **then**
        minError $\leftarrow$ error
        $\pi_e^* \leftarrow \pi_e$
  $\mathcal{P}_1^*, \mathcal{P}_2^* \leftarrow$ cut $\mathcal{P}^*$ at π_e^*
  $\mathcal{D} \leftarrow (\mathcal{D} \setminus \mathcal{P}^*) \cup \{\mathcal{P}_1^*, \mathcal{P}_2^*\}$

3 DECOMPOSITION REPRESENTATION AND TRANSFER FUNCTION

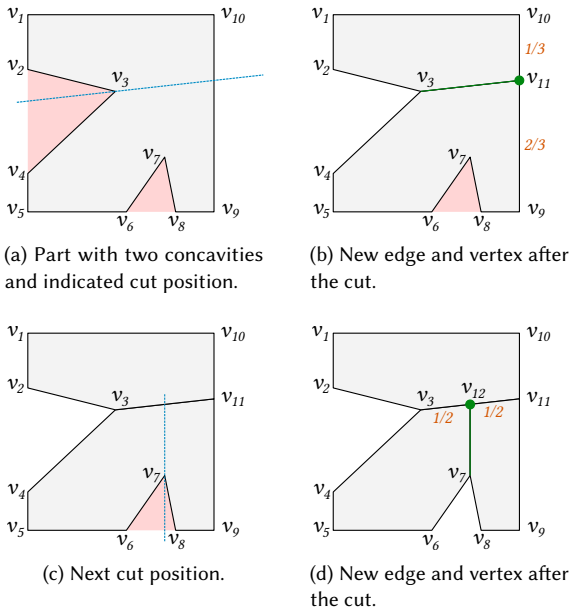
We now start looking at animated meshes that do not have a single set of vertex positions $\mathcal{V}$, but K sets of vertex positions $\mathcal{V}^j$ called *frames*. Parts will be written $\mathcal{P}_i^j$ to indicate that they belong to the j-th frame. $\mathcal{P}_i$ is now the set $\mathcal{P}_i := \{\mathcal{P}_i^1, \ldots, \mathcal{P}_i^K\}$ that contains the i-th part for every frame.

Let us consider a cut of a mesh $\mathcal{M}$ by hyperplane π_e in frame r into parts $\mathcal{P}_1^r$ and $\mathcal{P}_2^r$. Let $\mathcal{V}_1^r$ and $\mathcal{V}_2^r$ be the sets of original vertices of $\mathcal{M}$ belonging to each half space defined by π_e. The vertex sets of $\mathcal{P}_1^r$ and $\mathcal{P}_2^r$ after the cut will consist of:

$$\mathcal{P}_1^r = \mathcal{V}_1^r \cup \mathcal{V}' \quad (8)$$
$$\mathcal{P}_2^r = \mathcal{V}_2^r \cup \mathcal{V}', \quad (9)$$

where $\mathcal{V}'$ is a set of vertices, newly created by cutting each edge $\epsilon = (v_{\epsilon_1}^r, v_{\epsilon_2}^r)$, whose end points $v_{\epsilon_1}^r$ and $v_{\epsilon_2}^r$ belong to different half spaces (see Figure 2).

The set of triangles of a newly created part consists of those original triangles whose vertices were all in the same half space as the part. Additionally some original triangles will be split into a triangle and a quadrilateral by the hyperplane. The quadrilateral will be triangulated and all new triangles will be added to the triangle sets of the respective part depending of which side of the hyperplane they fall on. At last the parts' open holes along the hyperplane are closed by triangulation.



(a) Part with two concavities and indicated cut position.
(b) New edge and vertex after the cut.
(c) Next cut position.
(d) New edge and vertex after the cut.

Fig. 4. An exemplary decomposition of the top left mesh into three parts using two (green) cuts through the outer concave (red) angles. The first part consists of the points $\{v_1, v_2, v_3, v_{12}, v_{11}, v_{10}\}$, the second part consists of the points $\{v_3, v_4, v_5, v_6, v_7, v_{12}\}$, and the third part consists of the points $\{v_7, v_8, v_9, v_{11}, v_{12}\}$. Each newly created point is a pairwise convex combination of original or previously introduced vertices. Point v_{11} is described as a convex combination of vertices v_9 and v_{10} as $v_{11} = \frac{1}{3}v_9 + \frac{2}{3}v_{10}$, and similarly $v_{12} = \frac{1}{2}v_3 + \frac{1}{2}v_{11}$.

Each newly created vertex v'_ϵ can be uniquely described as a weighted sum of edge end points $v_{\epsilon_1}^r$ and $v_{\epsilon_2}^r$:

$$v'_\epsilon = \lambda_\epsilon v_{\epsilon_1}^r + (1-\lambda_\epsilon) v_{\epsilon_2}^r. \quad (10)$$

Figure 4 shows an example of this concept.

This allows us to transfer the split from reference frame r to any other frame j by copying the indices of the triangle sets for each part and defining a transfer function $T_e^j(v)$ for all original and new vertices:

$$T_e^j(v_i^r \in \mathcal{V}_1^r \cup \mathcal{V}_2^r) = v_i^j \quad (11)$$
$$T_e^j(v'_\epsilon \in \mathcal{V}') = \lambda_\epsilon v_{\epsilon_1}^j + (1-\lambda_\epsilon) v_{\epsilon_2}^j. \quad (12)$$

The transfer function applied to a vertex set $T_e^j(\mathcal{V})$ is defined as a transfer function applied to each vertex $v \in \mathcal{V}$. In the next section we will see how the ability to express parts in a frame-independent way can be used to design an algorithm for temporally coherent approximate convex decomposition.

4 TEMPORALLY COHERENT APPROXIMATE CONVEX DECOMPOSITION AND TRANSFER FOR ANIMATED MESHES

Instead of only taking a single frame into account when computing the decomposition, we will use several representative frames that capture the mesh's range of motion. This will make sure that the

decomposition does not "overfit" a particular frame but respects the whole range of possible spatial configurations of the mesh's vertices.

Following the static decomposition approach, we define the optimization objective as:

$$\mathcal{D}^* = \arg\min_{\mathcal{D}} \sum_{j=1}^{K} \sum_{\mathcal{P}\in\mathcal{D}} e(\mathcal{P}^j) \qquad \text{s.t. } |\mathcal{D}| = M, \tag{13}$$

where same parts $\mathcal{P}$ in different frames j have the same topological structure—corresponding vertices and triangles.

This cost function can be optimized iteratively by finding a hyperplane π_e cutting concave edge e minimizing:

$$\pi_e^* = \arg\min_{\pi_e} \sum_{j=1}^{K} \left(e(T^j(\mathcal{P}^{*r^*}_{\pi_e,1})) + e(T^j(\mathcal{P}^{*r^*}_{\pi_e,2})) - e(\mathcal{P}^j) \right). \tag{14}$$

where, analogous to Equation 6, $\mathcal{P}^{*r^*}$ is the part-frame combination with the highest convex approximation error:

$$\mathcal{P}^*,\ r^* := \arg\max_{\mathcal{P}\in\mathcal{D}, r\in\{1,\ldots,K\}} e(\mathcal{P}^r) \tag{15}$$

This optimal cut in frame r^* is then transferred into all frames using the transfer function $\mathcal{P}^j = T^j(\mathcal{P}^{r^*})$. The algorithm is described in detail in (Algorithm 3). Figure 5 gives a high level depiction of the process.

The vertex sets of resulting parts will consist of the vertices of the original mesh $\mathcal{M}$ and additional vertices, calculated as weighted sums of endpoints of edges split by the cuts. To reconstruct these points for an arbitrary new pose, and thus to transfer the convex decomposition, we just need to keep track of the indices of interpolated points and the weight λ. New points are calculated recursively in the same order as they were created during optimization, as the endpoints might already be interpolated points (see Figure 4).

The ability to transfer the decomposition allows to train using only a subset of frames (also called *keyframes*), and transfer the result to the remaining ones. Furthermore, the method can be applied for an animated mesh whose frames are not known in advance—such as a virtual reality avatar controlled by a user's body pose.

5 RESULTS AND EXPERIMENTS

We implemented the algorithm as a multi-threaded C++ application and compared it against state of the art methods for approximate convex decomposition of static meshes. We also present qualitative and quantitative results on animated meshes.

5.1 Quantitative Evaluation on Static Meshes

We evaluate our algorithm with respect to prior work by comparing static mesh performance. A straightforward quantitative measure on static meshes would be the ratio between the volume of the original mesh and the sum of volumes of the convex hulls of parts, as optimized in our cost function. However, for some of the methods we compare to, the resulting parts are not guaranteed to form a superset of the original mesh. Thus, we used the intersection over union (IoU) measure with additional penalization of overlapping

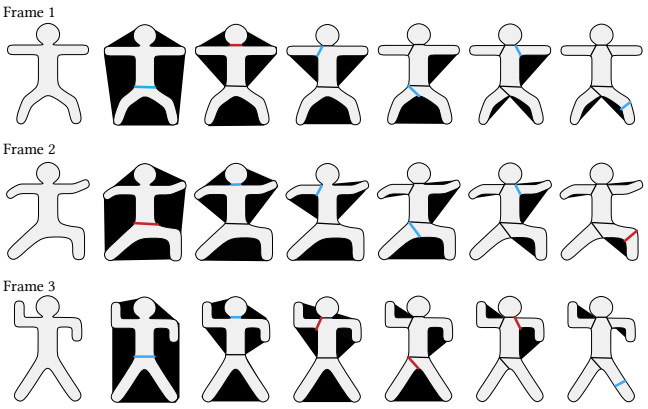


Fig. 5. Graphic depiction of our algorithm for temporally coherent approximate convex decomposition. Pictured are three (key)frames. Each column corresponds to one iteration of the cutting process. The leftmost column shows the initial frames. Each subsequent column indicates with a red line the cut in the reference frame that was chosen to execute the cut in and with a blue line the transfer of the cut result to the other two frames. The convex approximation errors—the difference between parts and their convex hulls—are highlighted as black areas and become smaller as the algorithm progresses.

Algorithm 3: Temporally coherent approximate convex decomposition.

Input : Animated mesh $\mathcal{M}$, target number of parts M, number of concave edges to sample C, number of hyperplane orientations R

Output : Approximate convex decomposition $\mathcal{D}$ with at most M parts

$\mathcal{D} \leftarrow \{\mathcal{M}\}$
while $|\mathcal{D}| < M$ *and there exists a concave edge* **do**
  $\mathcal{P}^r \leftarrow$ part-frame combination $\mathcal{P}^r$ with largest $e(\mathcal{P}^r)$
  $E \leftarrow$ find the C concave edges $e \in \mathcal{P}^r$ furthest from ConvexHull($\mathcal{P}^r$)
  $\pi_e^* \leftarrow$ nil
  minError $\leftarrow \infty$
  for *concave edge* $e \in E$ **do**
    $\Pi_e \leftarrow$ construct R hyperplanes π_e through e splitting the outer dihedral angle
    for *hyperplane* $\pi_e \in \Pi_e$ **do**
      $\mathcal{P}^r_1,\ \mathcal{P}^r_2, T \leftarrow$ cut $\mathcal{P}^r$ at π_e
      error $\leftarrow \sum_{j=1}^{K} e(T^j(\mathcal{P}^r_1)) + e(T^j(\mathcal{P}^r_2)) - e(T^j(\mathcal{P}^r))$
      if error < minError **then**
        minError $\leftarrow$ error
        $\pi_e^* \leftarrow \pi_e$
  $\mathcal{P}^r_1,\ \mathcal{P}^r_2,\ T \leftarrow$ cut $\mathcal{P}^r$ at π_e^*
  for *all frames* j **do**
    $\mathcal{P}^j_1,\ \mathcal{P}^j_2 \leftarrow T^j(\mathcal{P}^r_1),\ T^j(\mathcal{P}^r_2)$
  $\mathcal{D} \leftarrow (\mathcal{D} \setminus \mathcal{P}) \cup \{\mathcal{P}_1,\ \mathcal{P}_2\}$

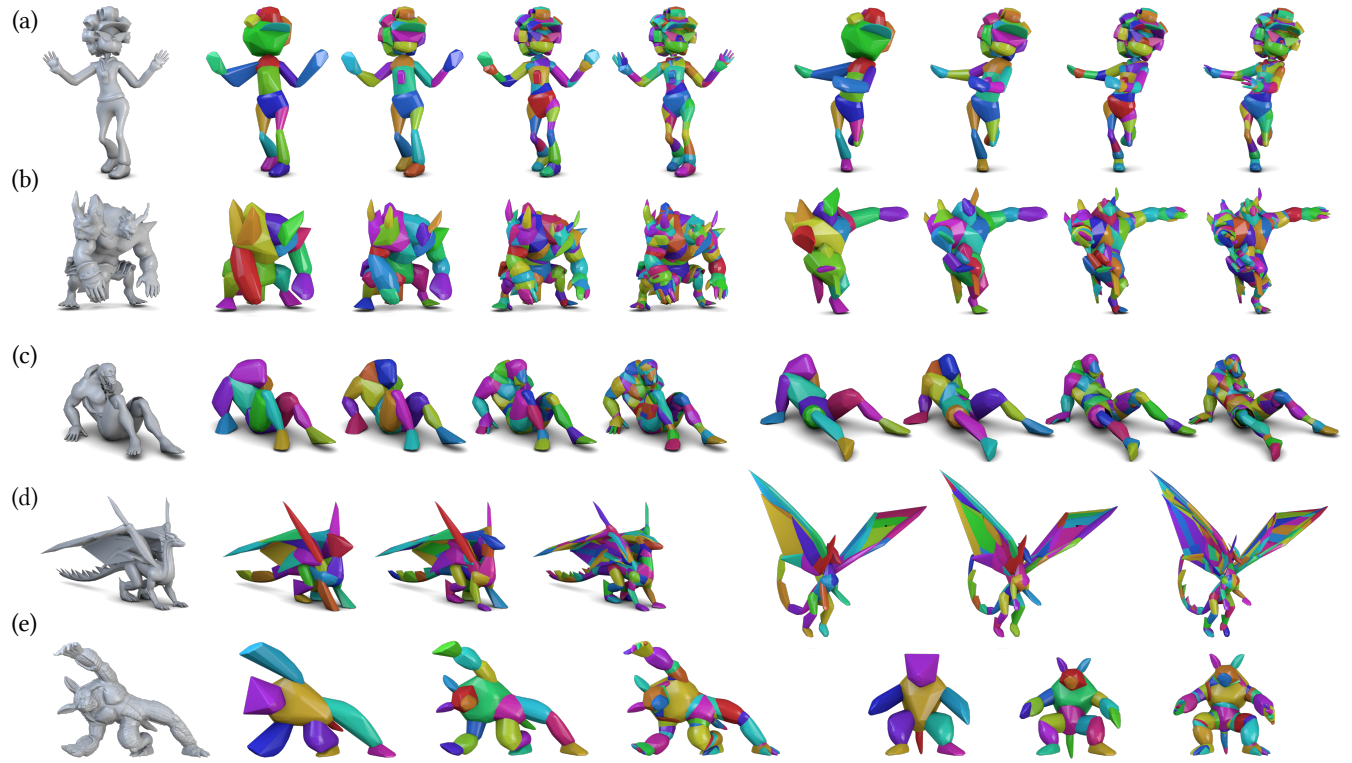


Fig. 6. Temporally coherent approximate convex decomposition for animated meshes a) sporty grandma, b) warrok, c) zombie, d) dragon and e) armadillo (a softbody simulation) at varying levels of detail, ranging from coarse body parts to the finest details capturing individual fingers. Our method managed to decompose the animations into temporally coherent parts using only 12 training keyframes.

Fig. 7. Temporally coherent approximate convex decomposition and transfer for an animated mesh. a) Input animation, b) Approximate Convex Decomposition, c) d) e) f) Transfers to new previously unseen animations. More transfer results in the teaser figure.

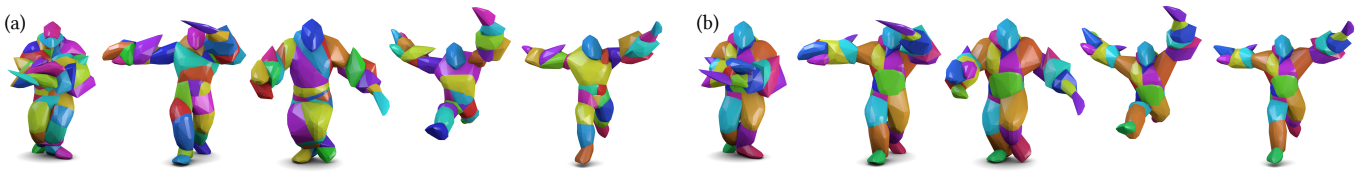


Fig. 8. Comparison of static approximate convex decomposition of individual frames (a) and decomposition computed across all frames of an animated mesh (b). The results are qualitatively and quantitatively very similar, however training on a subset of keyframes is much faster than computing an individual decomposition per frame and leads to temporally coherent parts, more suitable for physics simulations. Furthermore the static method is more likely to generate parts that do not have semantic meaning (see for example the third frame from the left in (a) where the ankles were merged).

Fig. 9. Difference between the decomposition of an animated mesh into 16 parts computed from only the first frame of the animation (red) and all frames (blue). The lower plot shows the approximation error throughout the 200 frames of the animation. The upper plot shows the histograms of those approximation errors. One can see that the decomposition computed from only the first frame "overfits" that frame and produces a low approximation error only for this and similar frames. The decomposition computed from all frames on the other hand has lower mean approximation error as well as a lower standard deviation in those errors.

hulls:

$$\text{score} = \frac{\text{Volume}(\mathcal{M} \cap \bigcup_{\mathcal{P}\in\mathcal{D}} \text{ConvexHull}(\mathcal{P}))}{\text{Volume}(\mathcal{M}') + \sum_{\mathcal{P}\in\mathcal{D}} \text{Volume}(\text{ConvexHull}(\mathcal{P}))} \tag{16}$$

where $\mathcal{M}' = \mathcal{M} \setminus \bigcup_{\mathcal{P}\in\mathcal{D}} \text{ConvexHull}(\mathcal{P})$ is the part of the mesh not covered by any convex hull. Note that for methods guaranteed to produce a superset of the original mesh (ours and V-HACD [Mamou 2016]), $\mathcal{M}' = \emptyset$ and $\mathcal{M} \cap \bigcup_{\mathcal{P}\in\mathcal{D}} \mathcal{P} = \mathcal{M}$. The error measure then simplifies to the ratio between the volume of the original mesh and the sum of volumes of the convex hulls of parts:

$$\text{score} = \frac{\text{Volume}(\mathcal{M})}{\sum_{\mathcal{P}\in\mathcal{D}} \text{Volume}(\text{ConvexHull}(\mathcal{P}))} \tag{17}$$

We also compared our results to the related works using a surface distance measure. We compute the average absolute distance from each point p on the mesh's surface $S(\mathcal{M})$ to the closest point on any of the generated convex hulls:

$$\text{distance} = \frac{\int_{S(\mathcal{M})} \min_{\mathcal{P}\in\mathcal{D}} \text{Distance}(p, \text{ConvexHull}(\mathcal{P}))\, dp}{\text{Area}(S(\mathcal{M}))} \tag{18}$$

Notice that such a measure does not reliably penalize falsely filled space.

The results of comparing our method to HACD [Mamou and Ghorbel 2009], V-HACD 2.0 [Mamou 2016] and CoRise [Liu et al. 2016] on the V-HACD dataset can be found in Table 11. It shows detailed results of the volumetric measure (Equation 16) as well as aggregated results for the surface measure (Equation 18) and runtime. To get a fair comparison, for each competing method we instructed our method to output the same number of parts. Our algorithm significantly outperforms existing approaches as judged by the volumetric score. Even when considering the surface based distance measure that our algorithm does not optimize for, we improve upon the state of the art. Our running times are competitive with the fastest related methods, taking for example approximately 7.2s on the Stanford bunny compared to 7.3s for HACD, 9.9s for CoRise and 108s for V-HACD.

5.2 Qualitative and Quantitative Evaluation on Animated Meshes

We tested our decomposition algorithm for animated meshes on publicly available animated meshes from https://www.mixamo.com and https://www.free3d.com as well as a soft body simulation created in Houdini. Each temporally coherent convex decomposition model was trained on a subset of 12 keyframes, uniformly sampled from the input animation. The qualitative results at varying levels of detail can be found in Figure 6. The results of the convex decomposition transfer to another animated mesh with the same topology (the same character) are shown in Figure 7. The difference between the decomposition on individual frames and on the whole animation is shown in Figure 8. The results are qualitative and quantitatively comparable, however a typical per frame decomposition can not take advantage of being evaluated across a multitude of frames. Furthermore, results obtained by our method for animated meshes are temporally coherent which makes them much more suitable for physics simulations than per-frame decompositions which can lead to "flickering". The algorithm turned out to be very robust and worked consistently for every animation. Tested animated meshes were slightly smaller in the number of vertices than the

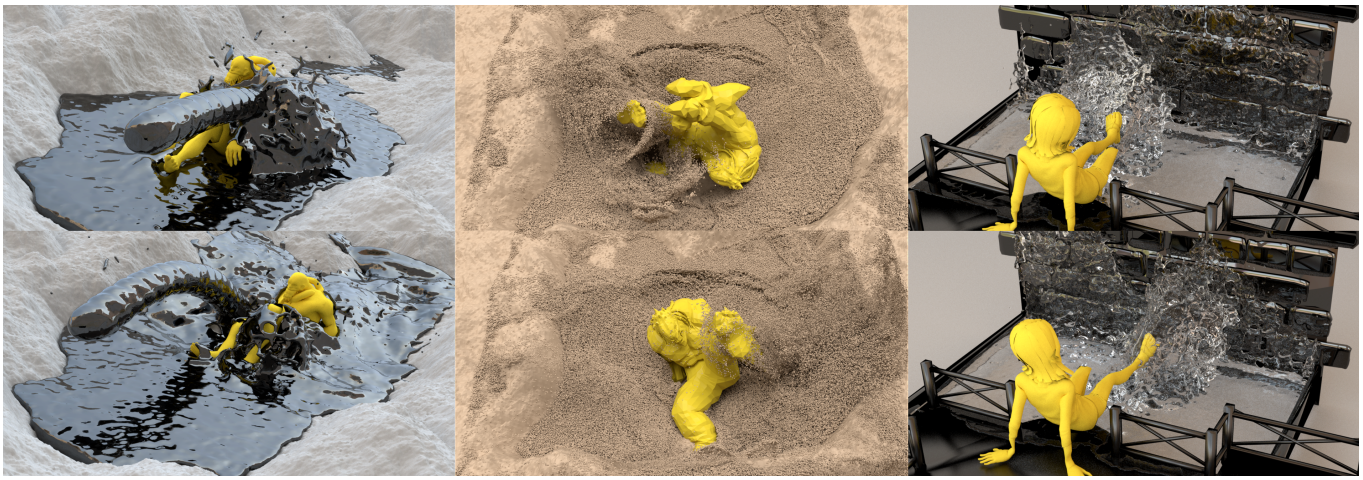


Fig. 10. Real-time fluid simulations using temporally coherent convex decompositions into 16 parts. We were able to calculate collisions and simulate up to 5 million particles in real-time. The resulting animations were rendered offline using Mitsuba [Jakob 2010] (fluid) and Blender (sand) renderers.

	HACD			VHACD			CoRise		
Mesh	#	theirs	ours	#	theirs	ours	#	theirs	ours
Block	19	0.636	**0.888**	36	0.360	**0.953**	7	0.309	**0.735**
Bull	14	0.730	**0.880**	28	0.920	**0.945**	26	0.171	**0.942**
Bunny	8	0.699	**0.876**	38	0.893	**0.961**	28	0.698	**0.952**
Camel	16	0.801	**0.851**	27	0.892	**0.923**	28	0.819	**0.927**
Cow	13	0.795	**0.863**	21	0.890	**0.913**	38	0.169	**0.949**
Dancing Figurine	7	0.548	**0.753**	45	0.808	**0.936**	13	0.742	**0.826**
Devilish Head	19	0.616	**0.949**	69	0.435	**0.972**	31	0.780	**0.961**
Dinosaur	13	0.762	**0.872**	29	0.872	**0.950**	8	0.795	**0.808**
Double Donut	13	0.581	**0.910**	19	0.929	**0.943**	27	0.343	**0.966**
Elk Toy	13	0.832	**0.860**	28	0.892	**0.958**	29	0.874	**0.959**
Fish	7	0.709	**0.880**	15	0.921	**0.941**	13	0.879	**0.932**
Foot	3	0.768	**0.914**	12	0.929	**0.974**	8	0.856	**0.967**
Genus	12	0.366	**0.838**	36	0.924	**0.959**	10	0.472	**0.785**
Girl	34	0.585	**0.869**	27	0.847	**0.860**	33	0.741	**0.870**
Hand	9	0.658	**0.883**	24	0.840	**0.963**	25	0.765	**0.965**
Head	9	0.824	**0.965**	64	0.210	**0.991**	33	0.764	**0.986**
Head Sculpture	2	0.893	**0.941**	10	0.958	**0.969**	24	0.911	**0.979**
Helix	9	**0.717**	0.704	19	0.842	**0.912**	18	0.863	**0.903**
Homer	18	0.840	**0.933**	21	0.923	**0.943**	27	0.170	**0.954**
Horse	10	0.643	**0.836**	23	0.901	**0.932**	25	0.759	**0.937**
Metal Casting	11	0.536	**0.776**	45	0.851	**0.923**	46	0.297	**0.925**
Moai	6	0.864	**0.921**	29	0.914	**0.972**	22	0.841	**0.968**
Octopus	31	0.607	**0.739**	31	0.649	**0.739**	45	0.743	**0.837**
Organic Fountain	18	0.436	**0.655**	44	0.715	**0.803**	18	0.538	**0.655**
Pig	9	0.676	**0.933**	17	0.936	**0.962**	25	0.843	**0.975**
Screwdriver	8	0.717	**0.832**	27	0.879	**0.920**	7	0.765	**0.819**
Sledge	14	0.463	**0.912**	24	0.922	**0.995**	15	0.079	**0.925**
Sword	4	**0.827**	0.814	73	0.812	**0.988**	4	0.703	**0.814**
Table	5	**1.000**	0.997	6	0.938	**0.998**	5	**1.000**	0.997
Torus	5	0.825	**0.885**	12	0.950	**0.977**	1	0.691	0.691
Venus	6	0.840	**0.920**	51	0.912	**0.987**	33	0.080	**0.981**
Winged Lion	25	0.643	**0.791**	42	0.765	**0.857**	47	0.655	**0.867**
median volumetric score		0.713	0.878		0.892	0.952		0.743	0.935
median surface distance measure		0.43	0.428		0.272	0.224		0.405	0.242
median runtime		8.5s	1.6s		362s	3.4s		2.6s	3.1s

Fig. 11. Quantitative comparison of our method to HACD, V-HACD 2.0 and CoRise. Each method resulted in different numbers of parts (#) for each mesh. We ran our method such that it matched the number of parts. The table shows detailed scores using the volumetric measure (Equation 16, higher is better) as well as aggregated scores using the surface distance measure in the second to last row (Equation 18, lower is better). The last row shows the median runtime each method took to compute a decomposition.

Fig. 12. The user controls the original mesh via a Kinect and a precomputed approximate convex decomposition is transferred in real-time to the novel frames.

static meshes, and thus training took only from 20 to 50 seconds per animation. The transfer for a new pose runs in real-time.

Figure 9 shows a typical quantitative difference between a decomposition computed from only the first frame of the animation and a decomposition computed from all frames of the animation. The decomposition computed only from the first frame overfits and achieves a low error score only on that specific frame while the average error across all frames of the animation is higher than for the decomposition that was computed based on the information from all frames.

5.3 Application in Real-time Simulations

We tested the convex decompositions in the real-time fluid simulation framework of [Ladicky et al. 2015]. Snapshots of the simulation results are shown in Figure 10. We decomposed the animated meshes into 16 temporally coherent approximate convex parts and used the convex hulls of those parts in the narrow phase of collision detection as well as simple axis aligned bounding boxes in the broad phase. This setup allowed us to simulate up to 5 million fluid particles interacting with an animated mesh in real-time. The simulations were rendered off-line using Mitsuba [Jakob 2010] (fluid) and Blender (sand).

5.4 Interactive Applications

As the popularity of virtual and augmented reality increases we expect approximate convex decomposition to become more important in these fields. Users want to interact with their virtual surroundings and a virtual physics representation of the user's body is thus needed. To showcase the applicability of our method to such scenarios we connected a Kinect device which is able to estimate a user's body pose with one of our animated meshes (Figure 12). A temporally coherent approximate convex decomposition was precomputed on the animated mesh and then transferred in real-time to the novel frames created by the user's body pose.

6 DISCUSSION

We presented a novel method for the fast and robust decomposition of animated meshes into temporally coherent approximately convex parts and showed how the decomposition model can be transferred to a new unseen animation with the same topology. Our method was shown to work robustly for a large variety of animation. The only artifacts appeared if the geometry of the training keyframes did not cover the range of deformation in the whole sequence or in the new unseen animation. This could have been observed for example, when the arm in the training simulation was always straight and ended up being one convex part in the decomposition, or when the training animation contained a character always holding a sword with both hands and fingers could not have been separated.

In the future we would like to explore variants of our algorithm that work on open, non manifold meshes. This could be used to compute robust signed distance fields or to repair topologically broken meshes by closely approximating them with a high number of convex hulls and merging those convex hulls back into a mesh that is two-manifold.

REFERENCES

Shmuel Asafi, Avi Goren, and Daniel Cohen-Or. 2013. Weak Convex Decomposition by Lines-of-sight. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing (SGP '13)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 23–31.

Marco Attene, Michela Mortara, Michela Spagnuolo, and Bianca Falcidieno. 2008. Hierarchical Convex Approximation of 3D Shapes for Fast Region Selection. In *Proceedings of the Symposium on Geometry Processing (SGP '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1323–1332.

O. Kin-Chung Au, Y. Zheng, M. Chen, P. Xu, and C. Tai. 2012. Mesh Segmentation with Concavity-Aware Fields. *IEEE Transactions on Visualization and Computer Graphics* 18, 7 (July 2012), 1125–1134.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding Proxies for Shape Approximation. *ACM Trans. Graph.* 36, 4 (July 2017), 57:1–57:13.

S. Cameron. 1997. Enhancing GJK: computing minimum and penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*, Vol. 4. 3112–3117 vol.4.

Bernard M. Chazelle. 1981. Convex Decompositions of Polyhedra. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC '81)*. ACM, New York, NY, USA, 70–79.

Fernando de Goes, Siome Goldenstein, and Luiz Velho. 2008. A Hierarchical Segmentation of Articulated Bodies. In *Proceedings of the Symposium on Geometry Processing (SGP '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1349–1356.

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

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

S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 171–180.

P. M. Hubbard. 1995. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics* 1, 3 (Sept. 1995), 218–230.

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Doug L. James and Dinesh K. Pai. 2004. BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models. *ACM Transactions on Graphics* 23 (July 2004).

Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 399–407.

Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-OR. 2014. Shape Segmentation by Approximate Convexity Analysis. *ACM Trans. Graph.* 34, 1 (Dec. 2014), 4:1–4:11.

Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D Mesh Segmentation and Labeling. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 102, 12 pages.

L. Kavan and J. Žára. 2005. Fast Collision Detection for Skeletally Deformable Models. *Computer Graphics Forum* 24, 3 (2005), 363–372.

J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. 1998. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics* 4, 1 (Jan. 1998), 21–36.

AF Koschan. 2003. Perception-based 3D triangle mesh segmentation using fast marching watersheds. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Vol. 2. IEEE, II–II.

V. Kreavoy, D. Julius, and A. Sheffer. 2007. Model Composition from Interchangeable Components. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. 129–138.

Lubor Ladicky, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 199.

Tong-Yee Lee, Ping-Hsien Lin, Shaur-Uei Yan, and Chun-Hao Lin. 2005. Mesh decomposition using motion information from animation sequences. *Computer Animation and Virtual Worlds* 16, 3-4 (2005), 519–529.

Bin Liao, Chunxia Xiao, Meng Liu, Zhao Dong, and Qunsheng Peng. 2012. Fast hierarchical animated object decomposition using approximately invariant signature. *The Visual Computer* 28, 4 (2012), 387–399.

Jyh-Ming Lien and Nancy M Amato. 2004. Approximate convex decomposition of polygons. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 17–26.

Jyh-Ming Lien and Nancy M Amato. 2007. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. ACM, 121–131.

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

Rong Liu and Hao Zhang. 2007. Mesh Segmentation via Spectral Embedding and Contour Analysis. *Computer Graphics Forum* 26, 3 (Sept. 2007), 385–394.

Khaled Mamou. 2016. Volumetric Hierarchical Approximate Convex Decomposition. In *Game Engine Gems 3*, Eric Lengyel (Ed.). A K Peters, 141–158.

Khaled Mamou and Faouzi Ghorbel. 2009. A simple and efficient approach for 3D mesh approximate convex decomposition. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 3501–3504.

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

Idan Shatz, Ayellet Tal, and George Leifman. 2006. Paper craft models from meshes. *The Visual Computer* 22, 9 (2006), 825–834.

Shymon Shlafman, Ayellet Tal, and Sagi Katz. 2002. Metamorphosis of polyhedral surfaces using decomposition. In *Computer graphics forum*, Vol. 21. Wiley Online Library, 219–228.

Jean-Marc Thiery, Émilie Guy, Tamy Boubekeur, and Elmar Eisemann. 2016. Animated mesh approximation with sphere-meshes. *ACM Transactions on Graphics (TOG)* 35, 3 (2016), 30.

Jianhua Wu and Leif Kobbelt. 2005. Structure recovery via hybrid variational surface approximation. In *Computer Graphics Forum*, Vol. 24. Wiley Online Library, 277–284.

Stefanie Wuhrer and Alan Brunton. 2010. Segmenting animated objects into near-rigid components. *The Visual Computer* 26, 2 (2010), 147–155.

J. Zunic and P. L. Rosin. 2004. A new convexity measure for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 7 (July 2004), 923–934.