Computer-Aided Design 78 (2016) 137–146

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Nearly convex segmentation of polyhedra through convex ridge separation[☆]

Guilin Liu, Zhonghua Xi, Jyh-Ming Lien[*]

George Mason University, Fairfax, VA, USA

ARTICLE INFO

Keywords:
Segmentation
Convexity
Shape approximation

ABSTRACT

Decomposing a 3D model into approximately convex components has gained more attention recently due to its ability to efficiently generate small decompositions with controllable concavity bounds. However, current methods are computationally expensive and require many user parameters. These parameters are usually unintuitive thus adding unnecessary obstacles in processing a large number of meshes with various types and shapes or meshes that are generated online within applications such as video games. In this paper, we investigate an approach that decomposes a mesh P based on the identification of *convex ridges*. Intuitively, convex ridges are the protruding parts of the mesh P. Our method, called CoRiSe, extracts nearly convex components of P by separating each convex ridge from all the other convex ridges through the new concept of *residual concavity*. CoRiSe takes a single user parameter: concavity tolerance which controls the maximum concavity of the final components, as input, along with other two fixed parameters for encoding the smoothness term of graph cut. We show that our method can generate comparable (sometimes noticeably better) segmentations in significant shorter time than the current state-of-art methods. Finally, we demonstrate applications of CoRiSe, including physically-based simulation, cage generation, model repair and mesh unfolding.

1. Introduction

In interactive applications, it is necessary to create simplified representations of a mesh to support various computationally intensive operations. In this work, we are interested in obtaining such simplified representations via decomposition. Taking deformation as an example, a mesh that has been decomposed into visually meaningful parts (e.g. head, torso, limbs) eases the process of creating deformation at semantic level. On the other hand, if a mesh is caged and partitioned by a set of convex shells, artists can use these shells to perform physically-based deformation efficiently. In many situations, both of these higher-level (semantic) and lower-level (physical or geometric) deformations are required. While it is ideal to keep both representations, it is also desirable to have an unified representation. An unified representation not only reduces space requirement but also allows deformations created at various semantic levels to be applied to the original mesh through a single approximation. Therefore, we recognize the need to have decomposition methods, such as the one proposed in this paper, that provide users both visual saliency and bounded geometric properties.

The first type of decomposition above is called *shape decomposition*. In the past decade, significant progress has been made in producing high quality results [1–7]. Shape decomposition provides implicit shape approximation and is useful for shape analysis and recognition and semantic level shape editing and deformation. The second type of decomposition can be accomplished through the Approximate Convex Decomposition (ACD) [8]. ACD decomposes a 3D mesh into nearly convex parts. Unlike shape decomposition, ACD provides explicit approximation with bounded approximation errors and is therefore suitable for lower level processing. For example, Bullet physics library uses hierarchical ACD (HACD) [9] to speed up the collision detection and response computation of the non-convex shapes, and Muller et al. [10] proposed an interactive tool to generate approximate convex parts for speeding up dynamic fracture simulation.

The most challenging task in developing such a decomposition method stems from the fact that current convex decomposition and shape segmentation methods are computationally expensive and require different parameter settings for different shapes which are usually unintuitive and make processing a massive

[☆] This paper has been recommended for acceptance by Scott Schaefer and Charlie C.L. Wang.
[*] Corresponding author.
E-mail addresses: gliu2@gmu.edu (G. Liu), zxi@masonlive.gmu.edu (Z. Xi), jmlien@cs.gmu.edu (J.-M. Lien).

http://dx.doi.org/10.1016/j.cad.2016.05.015
0010-4485/© 2016 Elsevier Ltd. All rights reserved.

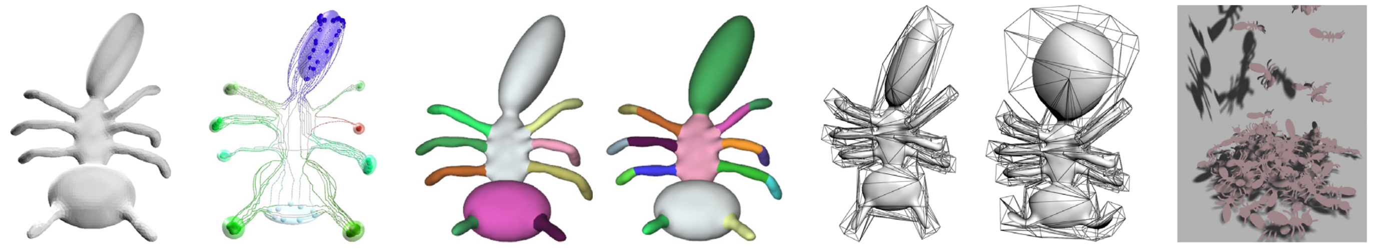
(a) Input.
(b) Convex ridges.
(c) Iterative decomposition.
(d) Cage-based deformation.
(e) Physically-based simulation.

Fig. 1. An example of CoRiSe with concavity tolerance $\tau = 0.05$. (a) Input mesh. (b) Ten convex ridges (shown as translucent ellipsoids) are connected by *deep valleys* (shown as the geodesic paths). Formal definition of convex ridge and valley can be found in Section 4. (c) Left is the decomposition result of the first iteration; right is final decomposition (the second iteration). Note that colors are just used to distinguish different parts. (d) CoRiSe used for automatic cage generation. (e) The convex hulls of CoRiSe components can be used to speedup the computation in physically-based simulation. (For interpretation of the references to color in this figure legend, the reader is referred to the electronic version of this article.)

number of meshes in applications difficult. Recently, learning based segmentation methods [4,5] are proposed to learn common parameters in an unsupervised or supervised manner, but the computation time of these methods is often intolerably high (minutes to hours), in particularly, for interactive applications.

Contribution. In this paper, we will describe an efficient decomposition method, called CoRiSe. The only user input parameter is a concavity tolerance which directly controls the approximation error. The novelty of CoRiSe comes from the idea of *convex ridge* which can be efficiently and robustly determined. Essentially, CoRiSe extracts nearly convex components of a 3D mesh P by separating each convex ridge from all the other convex ridges using graph cut. An example of convex ridge and CoRiSe decomposition is shown in Fig. 1. Formal definition of convex ridge can be found in Section 4. In addition, through the new concept of *residual concavity*, CoRiSe is insensitive to the parameter in graph cut and requires only a single user parameter: concavity tolerance. We will discuss residual concavity and graph cut in Section 5.

Using the Princeton Segmentation Benchmark, we show that CoRiSe generates noticeably better segmentation in significant shorter time than the existing methods [9,1]. Finally, we demonstrate applications of CoRiSe, including physically-based simulation, cage generation, model repair and mesh unfolding in Section 6. Fig. 1 shows two of these applications.

2. Related work

Many methods have been developed to decompose 3D mesh models. Comprehensive surveys can be found in [11–13]. In this section, we will review more recent works on shape segmentation and convex segmentation. After this short review, we will point out that the needs for developing a more intuitive and efficient method, such as CoRiSe.

Shape segmentation. Many of the existing single-shape segmentation methods are based on clustering mesh faces, such as k-means clustering [14], fuzzy clustering [15], mean-shift clustering [16], and spectral clustering [17]. Shape features, such as geodesic distance, local concavity, curvature, have significant impact on these clustering methods. More advanced features include shape diameter function [6] that is a measure of the diameter of the object's volume in neighborhood of a point, Mumford–Shah model [3] that measures the variation within a segment and continuous visibility feature [18] that uses more restricted visibility to better capture shape than the traditional line of sight visibility. Methods that are not clustering based do exist. For example, the method proposed by Wang et al. [19] uses the training segmentation of 2D projection images. Random cuts method [7] uses other different segmentation algorithms with various parameters to generate a collection of segmentations to find consistent cut positions.

Recently, we see more techniques using data-driven approach. These methods usually provide more consistent segmentations over a set of models [20] and produce segmentations that are more natural via machine learning approaches [4,5]. However, these methods are computationally intensive (require hours of computation) and are not suitable for interactive use.

Primitive segmentation. While many algorithms focus on decomposing a model into visually meaningful parts, other algorithms focuses on partitioning models into geometric primitives such as ellipsoids [21] and convex objects [8]. In other words, multiple primitives are used to jointly approximate the original shape.

In this paper, we are interested in producing nearly convex components. We found that many methods in the literature use again clustering (bottom-up) approach. Mamou and Ghorbel [9] proposed a method called Hierarchical Approximate Convex Decomposition (HACD) for 3D meshes. HACD iteratively clusters mesh facets by successively applying half-edge collapse decimation operations (see more detailed discussion in Section 6). Attene et al. [22] proposed to convert a model into tetrahedral mesh and then merge tetrahedra to form near convex components. However, this method requires human interaction, thus not suitable for segmenting a large number of models. The method proposed in [23] is based on a region growing approach controlled by convexity, compactness and part cost. Top-down approaches are rare. For example, Ghosh et al. [24] proposed a notion named *relative concavity* to model the concavity measure before and after every mesh cut. Nearly convex components are obtained by finding the cuts that have largest relative concavities via dynamic programming. In many of these methods, several parameters are needed to be set to balance various features.

We would like to note that there are also methods that use convexity and concavity but do not produce segmentation with bounded convexity or concavity. For example, Au et al. [2] used concavity-aware field to form potential cuts and the final cuts are achieved by maximizing the cut score. Even though concavity is used, it does not have direct control of concavity for final components and several parameters and thresholds need to be set. Asafi et al. [25] defined the convexity using the line-of-sight and applied spectral clustering on the visibility matrix to achieve segmentation. van Kaick et al. [1] applied merging on the initial result of [25] based on the Shape Diameter Function. However, since the pairwise visibility needs to be computed, the computation is time-consuming. Moreover, even though these two methods use convexity as clue for segmentation, they did not directly control the convex/concave geometric property for the final components.

Concavity and its counterpart, convexity, have shown to be valuable for various decomposition methods. Intuitively, concavity of a polyhedron P measures how different P is from a convex object, which is typically the smallest convex object enclosing P, i.e. the

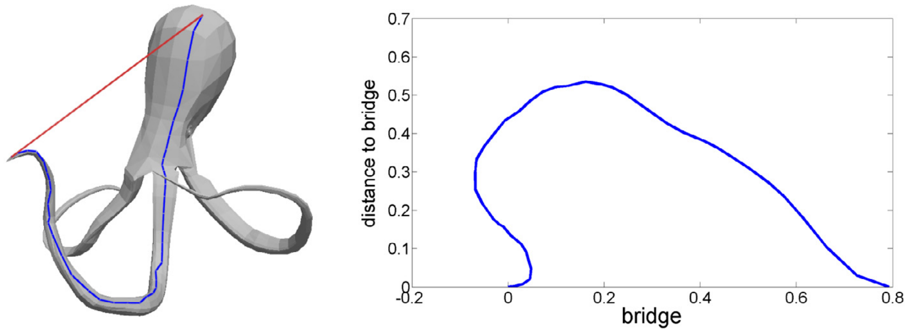

Fig. 2. The left figure shows one bridge and its valley in 3D. The right part shows its projected concavity polygon in 2D. Concavities are measured in this 2D polygon using the shortest-path distance.

convex hull $H(P)$ of P. Concavity can be measured globally from the difference between the volumes of P and $H(P)$ or the difference of their projections [26]. Concavity can also be measured locally for every point on the surface ∂P of P. Local concavity measures how far away a point on the surface of P is from the surface of $H(P)$, thus provides richer information to guide the decomposition process.

There have been several definitions of local concavity, such as curvature [2]. A more general definition of the concavity of a point p is the length of the shortest distances from $p \in \partial P$ to $H(P)$ without intersecting P. Due to the high computational complexity of 3D shortest-path problem, approximation is required. For example, Zimmer et al. [27] proposed to compute the shortest path by tessellating the space between ∂P and $\partial H(P)$ using constrained Delaunay triangulation (CDT). However, this solution required ∂P to be closed. Many other approaches resort to the Euclidean distance between the vertex and the convex hull in the outward normal direction of the vertex. It is clear that the association between ∂P and $\partial H(P)$ is usually complex and cannot be captured by the surface normals of P. This can result in inaccurate measurement of the concavity and lead to incorrect decomposition. Lien and Amato [8] proposed to determine the association by projecting the edges and then facets of $\partial H(P)$ to ∂P. Unfortunately, such an association between ∂P and $\partial H(P)$ is not always well defined as it usually depends on how P and $\partial H(P)$ are tessellated.

3. Overview

CoRiSe takes a single 3D mesh and a concavity tolerance parameter τ as input and decomposes this mesh into nearly convex components in a top-down fashion that includes three main steps:

Step 1. **Build bridges and compute concavities**. Bridges are the convex hull edges, and the shortest geodesic path on the mesh connecting the end vertices of a bridge is called *valley*. Then, the concavity of a point in the valley is measured as the shortest-path distance to the bridge. Exact shortest geodesic path and shortest-path distance are computationally expensive thus we will discuss how we approximate them. We say that a valley is *deep* if the maximum concavity in the valley is greater than τ, otherwise the valley is *shallow*. It is important to remember that, throughout the entire segmentation process, concavity is only defined for points in the valleys and undefined elsewhere.

Step 2. **Identify convex ridges**. In CoRiSe, a convex ridge consists of a group of convex-hull vertices connected *only* by shallow valleys. Intuitively, a convex ridge can be viewed as a *protruding* part of the mesh that is guaranteed to be convex enough. This implies that a continuous subset of the mesh containing two or more convex ridges must be too concave. Fig. 1(b) shows an example of 10 convex ridges.

Step 3. **Partition using convex ridges**. CoRiSe segments the mesh by ensuring that each component in the segmentation contains only a single convex ridge. The segmentation process is bootstrapped by splitting each deep valley into shallow ones at the *valley separators*. Then CoRiSe applies graph cut using the region defined by valley separators as data term and edge angle and length as smoothness term to find optimal cut loops near these valley separators.

The above three steps are repetitively applied to the segmented part whose concavity is greater than τ.

4. Bridge, valley and convex ridge

To approximate a mesh using convex shapes, we should establish relationships between the mesh and its convex hull. Let the input be a polyhedron $P = \{V_P, F_P, E_P\}$ composed of a list of vertices V_P and faces F_P connected by edges E_P. We assume that P consists of a single connected component. The convex hull of P is denoted as $H(P) = \{V_H, F_H, E_H\}$, where $V_H \subset V_P$. Let us consider a convex hull edge $e = \{a, b\} \in E_H$. Since e is the shortest Euclidean path connecting a and b, let us call e a *bridge*. Because a and b must also be the vertices of P, we can always find a path connecting a and b on ∂P. We call the shortest geodesic path connecting a and b on ∂P a *valley* underneath the bridge e_β.

A bridge e_β and its associated valley $e_\vee$ form a 3D polygon. We call this polygon the *concavity polygon*, which plays an important role in many stages of our algorithm. For example, we project the concavity polygon into a 2D space and only measure the concavity in the projected 2D space. More specifically, each point $p \in e_\vee = (a, b)$ is projected to

$$p' = (\overrightarrow{ap} \cdot \overrightarrow{ab}, \mathbf{d}(p, e_\beta)), \quad (1)$$

where $\overrightarrow{ap} \cdot \overrightarrow{ab}$ is the inner product of $\overrightarrow{ap}$ and $\overrightarrow{ab}$, and the function $\mathbf{d}$ defines the Euclidean distance between a point and a line segment. Fig. 2 shows an example of bridge, its valley and the projected 2D concavity polygon. Then, the concavity $C(p)$ of the point p is determined as the *shortest-path distance* between p' and the projected e_β. It is known that the shortest-path distance can be measured in $O(n)$ time for a polygon of size n.

It is possible that the projected polygon can self-intersect. However, self-intersection seldom happens; all projected polygons of models studied in this work are free of self intersection. However, when the projected polygon is self-intersecting or has large concave regions, we use a more sophisticated concavity measure proposed in [28], which decomposes the polygon into several near convex parts and determines a hierarchy of bridges. The concavity will be measured by the accumulated distance of the point-to-adjacent bridge distance and bridge-to-bridge distances. For example, in Fig. 3, the concavity of p will be the distance sum of the distance between p and β_1 and the distance between β_1 and ab.

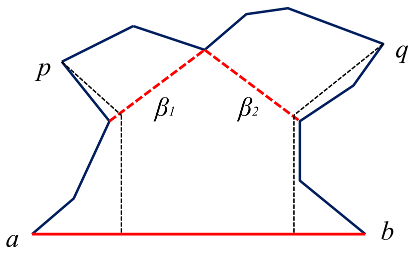

Fig. 3. The concavity of the projected polygon with highly concave regions or self-intersections.

The concavity of a valley $e_\vee$ (and its associated bridge e_β) is defined as the maximum concavity of its vertices, i.e., $C(e_\vee) = C(e_\beta) = \max_{p\in e_\vee}(C(p))$. Let us call $e_\vee$ *deep valley* if $C(e_\vee)$ is larger than the tolerance, otherwise $e_\vee$ is called a *shallow valley*. For the convenience of our future discussion, we also classify bridges into *high* and *low* bridges (as measured by the *deck height* of bridge) if their corresponding valleys are deep and shallow, respectively.

4.1. Convex ridge

Now let us formally define the convex ridge. A convex ridge R of P is a graph representing a subset of convex hull $H(P)$. We say a subset of $R \subset H(P)$ is a convex ridge if all bridges of R are shallow and there are no high bridges connecting two vertices in R. More specifically, a convex ridge R must satisfy the following two conditions:

$$\begin{aligned} \forall e \in E_R, \quad & C(e) < \tau, \text{ and} \\ \nexists e = (a \in V_R, b \in V_R), \quad & C(e) > \tau, \end{aligned} \tag{2}$$

where V_R and E_R are the vertices and edges of R, respectively. In short, a convex ridge can only have vertices connected by bridges whose valleys are shallow.

The convex ridges of a given mesh are constructed by iteratively greedy clustering the convex hull vertices while ensuring that the properties in Eq. (2) are maintained for each cluster. The clustering proceeds by collapsing low bridges sorted in an ascending order based on the length of the associated valleys. Each final cluster would become a convex ridge. The detailed procedure can be found in Algorithm 1 in the supplementary materials (Appendix A). Fig. 4 shows examples of convex ridges. We can also see that convex ridges are quite insensitive to surface noise in Fig. 4. In fact, as long as shallow valleys remain shallow, all convex ridges will be unaffected.

4.2. Residual concavity

Residual concavity measures the concavity of a valley $e_\vee$ after $e_\vee$ is split into two subpaths at a vertex of $e_\vee$. Let $e_\vee = \{v_0, v_1, \ldots, v_{n-1}\}$ be a deep valley, and let $e^k_\vee = \{v_0, v_1, \ldots, v_{k-1}\}$ be a prefix subset of $e_\vee$, where $k \leq n$. We further let $\hat{e}_\vee = \{v_{n-1}, v_{n-2}, \ldots, v_0\}$ be the reverse of $e_\vee$. The residual concavity of $e_\vee$ at the kth vertex is then defined as:

$$RC(e_\vee, k) = \max\left(C(e^k_\vee), C(\hat{e}^{(n-k+1)}_\vee)\right). \tag{3}$$

Recall that the concavity is always measured in the 2D concavity polygon projected using Eq. (1). It is important to note that once the valley $e_\vee$ is split, two or more new bridges must be formed to determine the concavity of $e_\vee$'s sub-valleys. Therefore, at the first glance, computing RC seems to be time consuming, but we have shown that RC can be computed in linear time.

Lemma 4.1. *The computation of residual concavity $RC(e_\vee, k)$ takes time linear to the size of $e_\vee$, i.e. $O(n)$ for $e_\vee$ with n edges.*

Proof. See details in the Appendix Proof 8 given in the supplementary material (Appendix A). □

4.3. Shape of a valley

The residual concavity gives us a way to estimate the shape of a valley. We say a valley $e_\vee$ is *V-shaped* if there exists a vertex v_k such that the residual concavity of $e_\vee$ is less than the tolerance τ. Otherwise $e_\vee$ is *U-shaped*. An example of a U-shaped valley can be found in Fig. 5. If $e_\vee$ is U-shaped, we can always find the bottom of $e_\vee$ as the sub-polygon bounded by the prefix and suffix of $e_\vee$ that have residual concavities less than τ. In Fig. 5, we show that the bottom of the U-shaped valley is defined as (v_i, v_j), where i and j are the maximum indices such that $C(e^i_\vee) < \tau$ and $C(\hat{e}^{(n-j)}_\vee) < \tau$, respectively; recall that $\hat{e}_\vee$ is the reverse of $e_\vee$. As we will discuss in Section 5, the shape of valley can be used to determine the number of vertices needed to separate each convex ridge in the wired representation to ensure that its concavity is less than τ.

5. Convex ridge separation

CoRiSe segments a mesh by ensuring that each component in the segmentation contains only a single convex ridge. Essentially, CoRiSe finds such segmentation in two steps: first, CoRiSe only focuses on the wireframe representation of the mesh that consists of vertices and edges of all convex ridges and deep valleys. An example of such a representation can be found in Fig. 1(b). For each deep valley, CoRiSe determines one or two key vertices (depending on whether it is a V- or U-shaped valley) that can separate each deep valley into at least two shallow valleys. In Section 5.1, we will discuss how CoRiSe decomposes this wireframe representation. Then, once the wireframe is decomposed, CoRiSe switches back to the original model and applies a two-way graph cut to find optimal cut loops near the valley separators. This step will be discussed in Section 5.2.

5.1. Valley separators

To separate the convex ridges in the wireframe presentation, CoRiSe finds *valley separators* for each deep valley and ensures that each convex ridge only connects the subset of the valley that has concavity less than the tolerance. For a V-shaped valley $e_\vee$, the valley separator is a single vertex that has the largest concavity in $e_\vee$. By splitting at the vertex with the largest concavity, we can guarantee that the two new sub-valleys of $e_\vee$ are shallow.

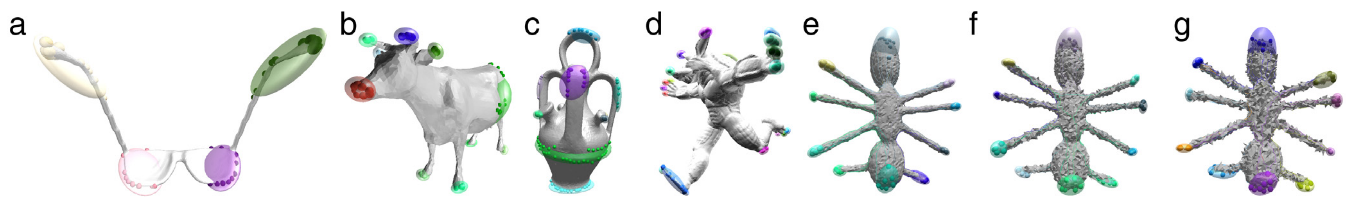

Fig. 4. Convex ridges found in these models. Each ellipsoid represents a convex ridge. Even though several convex ridges in these examples concentrate around small compact regions, the convex ridges in (c) manifest various ridge-like shapes (regions that locally have higher "elevation"). Fig. 4(e), (f) and (g) show the convex ridges on the ant model with random noise added. The convex ridges of Fig. 4(e), (f) and (g) are all generated using $\tau = 0.08$.

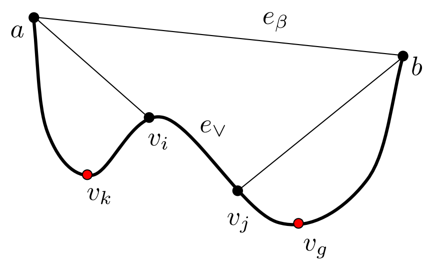

Fig. 5. A deep U-shaped valley $e_\vee$ and its valley separators v_k and v_g. The sub-valleys $e^i_\vee$ and $\hat{e}^j_\vee$ are subsets of $e_\vee$ whose residual concavity is less than the concavity tolerance τ. The valley separators $v_k \in e^i_\vee$ and $v_g \in \hat{e}^j_\vee$ minimize the residual concavities $RC(e^i_\vee, k)$ and $RC(\hat{e}^j_\vee, g)$.

For a U-shaped valley $e_\vee$, things are a bit more complex. Basically, CoRiSe splits $e_\vee$ into three sub-valleys and guarantees that two sub-valleys incident to two convex ridges are shallow even though the bottom of $e_\vee$ may not be. Let $e_\beta = (a, b)$ be a bridge whose valley $e_\vee$ is U-shaped. Using the residual concavity defined in Eq. (3) in Section 4.2, we can find two vertices v_i and v_j such that $e^i_\vee$, a sub-valley between a and v_i, and $\hat{e}^j_\vee$, a sub-valley between v_j and b, are shallow. See Fig. 5 for the illustration of $e^i_\vee$ and $\hat{e}^j_\vee$. Our goal is to define two valley separators in $e^i_\vee$ and $\hat{e}^j_\vee$, respectively. It is true that we can use v_i and v_j as the valley separators and still guarantee that $e^i_\vee$ and $\hat{e}^j_\vee$ are shallow. However, the locations of v_i and v_j are quite arbitrary in most cases. Therefore, the valley separators for a U-shaped valley $e_\vee$ are defined as a couple of vertices (v_k, v_g):

$$(v_k, v_g), \quad \text{where } k = \arg\min_k RC(e^i_\vee, k),\ g = \arg\min_g RC(\hat{e}^j_\vee, g).$$

From Fig. 5, we can see that the valley separators v_k and v_g of $e_\vee$ are identified at the locations of large concavity if the valley is split at v_i and v_j. Once the valley separators are identified for all deep valleys, all convex ridges are separated and only attached to shallow valleys. Next, CoRiSe switches back to the original representation and segments the mesh using graph cut.

5.2. Convex ridge separator

Graph cut [29] is a powerful optimization tool that has been proven to be successful in segmentation. In the previous section, we have found separators for deep valleys. And the valley separators have roughly defined where the cuts should be. However, since the valley separators are merely feature vertices capturing concavity constraints, separating convex ridges needs close cut loops. We formulate finding the cut loops as a simplified graph cut problem. We say simplified graph cut because the data term value is either infinity or zero, which corresponds must-link region and fuzzy region. The smoothness term is defined using dihedral angle and edge length. Fig. 6 shows the simplified graph cut formulation. In the rest of this section, we will discuss the data term and smoothness term in detail.

Data term. To formulate the graph cut problem, we introduce two terms: *must-link region* and *fuzzy region*. *Must-link region* would have infinite data term values associated with the *source* or *sink*. *Fuzzy region* would have zero data term values. For each iteration of graph cut, a facet belongs to either *must-link region* or *fuzzy region*. In other words, *must-link region* plus *fuzzy region* will cover the whole model. Intuitively, each convex ridge has a must-link region which is a set of faces that must be segmented with the convex ridge. To get the *must-link* region and *fuzzy region*, we will introduce another term named *potential region*. We call it *potential region* because a facet belonging to *potential region* has the chance to become either part of *must-link region* or part of *fuzzy region*. The ways to determine these regions are described as follows *Potential region*. We will collect a *potential region* for each convex ridge R_k. This potential region is the union of the facets collected by R_k's all the sub-valleys using geodesic distance. How each sub-valley collects its facets is described as follows:

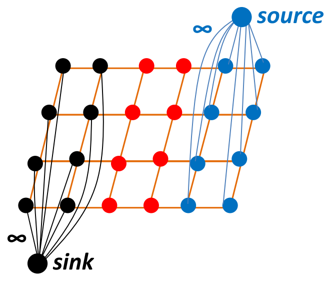

Fig. 6. Simplified graph cut: black nodes and blue nodes represent the must-link regions. They have infinite data term associated with source or sink. Red nodes represent the fuzzy region. (For interpretation of the references to color in this figure legend, the reader is referred to the electronic version of this article.)

1. Collecting using Geodesic Distance: for each sub-valley $e^i_\vee$ (recall that $e^i_\vee$ is created from valley separator as described in Section 5.1) incident to R_k, we let T_i be a set of facets whose geodesic distance to the end point of $e^i_\vee$ in R_k is less than the path length of $e^i_\vee$.
2. Union: from T_i, we can determine a set U_{R_k} of facets that forms a superset containing R_k. The set U_{R_k} is the *potential region* of convex ridge R_k. More specially,

$$U_{R_k} = \bigcup_{1 \le i \le n} T_i, \tag{4}$$

where n is the number of sub-valleys incident to R_k.

Potential region's overlapping issue. Two neighboring convex ridges R_k and R_j are likely to compete on certain facets. In this case, U_{R_k} and U_{R_j} overlap. The overlapping region of I_{R_k} with other convex ridges is then defined as:

$$I_{R_k} = \bigcup_{\forall j \neq k} \left(U_{R_k} \bigcap U_{R_j} \right). \tag{5}$$

Must-link and fuzzy regions. With U_{R_k} and I_{R_k}, we are ready to define the must-link and fuzzy regions. The must-link region C_{R_k} of R_k is simply U_{R_k} with facets in I_{R_k} removed,

$$C_{R_k} = U_{R_k} \setminus I_{R_k}. \tag{6}$$

Then the fuzzy region is the union of the facets that do not belong to any convex ridge's *must-link region*.

$$F_R = F \setminus \bigcup C_{R_k}, \tag{7}$$

where F is the facets of the mesh.

We give an example here. In the left part of Fig. 7, the black segments show the potential region of a convex ridge. The black vertices are the vertices of a convex ridge. The crossing vertices are the valley separators. The right part of Fig. 7 shows the must-link region and fuzzy region. The red segments represent the fuzzy region. Except the red, each of other colors stands for the must-link region for each convex ridge. There are three parts of fuzzy region (red regions). The left part corresponds to the area that is not covered by deep valley's potential region. The middle part is part of fuzzy region because it lies in the middle part of a U-shape valley and it is not within the potential region of either of end vertices from two convex ridges. The right part becomes fuzzy region due to the fact that it has been covered by two convex ridges.

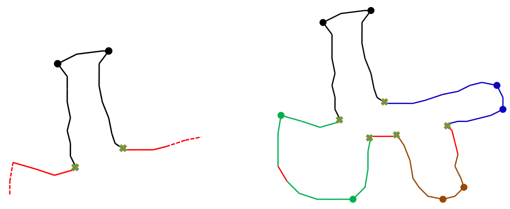

Fig. 7. Left: the potential region (including all the black and red regions). Right: must-link region and fuzzy region. In the right part, the convex ridges are differentiated by colors (4 in total). The colored circles (dots) are the end vertices of the deep valleys. The crossings are the detected valley separators. The red lines (region) are the fuzzy region. The colored lines attached to circles with same color are must-link region attached to corresponding convex ridge. (For interpretation of the references to color in this figure legend, the reader is referred to the electronic version of this article.)

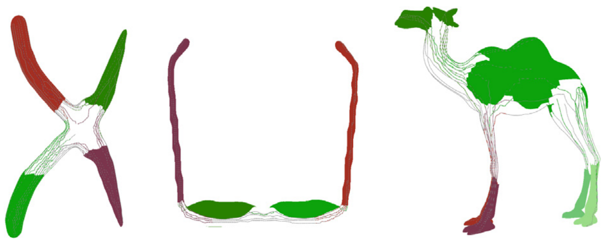

Fig. 8. The must-link regions connected by the deep valleys.

Fig. 8 shows the must-link regions on the 3D models. The filtered out regions are the fuzzy regions.

Smoothness term. The smoothness term evaluates the similarity/compatibility between two adjacent triangle facets. In this paper, we use the same definition of smoothness as that in [15].

A parameter α is introduced to balance the importance of dihedral angel and edge length. The definition is as below: Let v_i and v_j be two adjacent vertices, namely the two end vertices of an edge (denoted as e_{ij}) on the mesh. And let f_i and f_j be the two triangle facets sharing this edge. Let θ_{ij} be the dihedral angle between f_i and f_j, $|e_{ij}|$ be the length of e_{ij}. Let Θ_{ij} be a function of θ_{ij} such that Θ_{ij} is a positive small number (the second parameter, fixed to be 0.3) for concave edges and 1 for convex edges. Then the smoothness term defined in [15] is:

$$w_{ij} = \alpha\left(\frac{\Theta_{ij}}{\mu_\Theta}\right) + (1-\alpha)\frac{|e_{ij}|}{\mu_e}, \tag{8}$$

where μ_Θ and μ_e are the average values of Θ and edge length, respectively, of the entire mesh. In this paper, we set $\alpha = 0.8$ (the third parameter, fixed) in all experiments.

Segmentation using graph cut. We solve $n-1$ times (n is the number of convex ridges) binary graph cut optimization using the max-flow algorithm. The order of $n-1$ graph cut is determined by the bounding box's diameter of convex ridges' vertices. We separate one convex ridge at one time using the graph cut. At each graph cut iteration i, we will assign infinite data term values associated with *source* for the facets in its *must-link region* C_{R_i} and assign infinite data term values associated with *sink* for the facets in other convex ridges' *must-link region* $\bigcup_{\forall j\neq i} C_{R_j}$. Facets in *fuzzy region* do not have data term with neither *source* nor *sink*. The smoothness term will be defined for any pair of neighboring facets using the Eq. (8). We separate this convex ridge from the model and the rest of model will be used for applying region for next binary graph cut. Segmentation is performed iteratively until all components have smaller concavity than τ. Fig. 1(c) shows the first iteration decomposition result on the left and second (final) iteration decomposition result on the right.

Discussion. One may argue that there are three issues in the above formulation of graph cut problem. 1. Why use must-link region? The must-link region is defined using valley separators, which roughly defines where the cut should be and satisfied the concavity constraints. The must-link region can fully utilize the rough cut positions proposed by valley separators. 2. Why use $n-1$ binary graph-cut? Another option is to k-way graph cut. However, choosing a right k is important for k-way graph cut. A proper k would be the number of final components. However, the problem is the number of final components is unknown and the number of convex ridges is not the real number corresponding to the number components. Some components may have not been detected by the current convex hull. The real number should be $\geq n$. We apply $n-1$ times graph cut and postpone handling of the issue of undetected components in the further iteration of decomposition, because these components may be detected after we have already cut some components out of the model in previous iteration. 3. How to decide the order of cutting which convex ridge and how to deal with overlapping labeling issues? The order of applying graph cut is based on the heuristic that we handle the small components first (in terms of the bounding box's diameter of convex ridges' vertices). For each graph cut, if we apply on the same region, there may be one issue that a facet has been labeled as "source" for several times. In this paper, after we apply one graph cut, we separate one convex ridge from the model and the rest of the model would be used as the applying region for further graph cut. In this way, we would be able to avoid the issue of overlapping labeling issues.

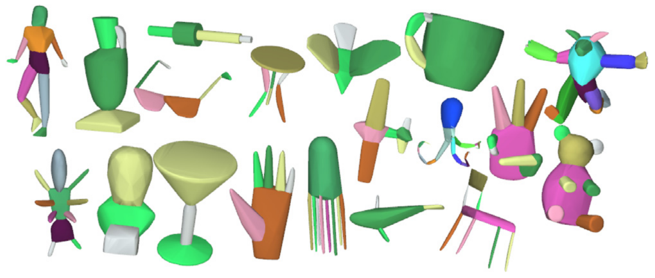

Fig. 9. Approximate shapes using the convex hulls of decomposed parts. The results in this figure are generated by CoRiSe using models in the Princeton Segmentation Benchmark.

6. Experimental results

We have implemented CoRiSe in C++. In order to evaluate CoRiSe, we compare CoRiSe extensively to existing approximate convex decomposition methods HACD [9] that is available in the Bullet Physics Library and WCSeg [1]. We also compare CoRiSe with shape decomposition methods using the Princeton 3D Mesh Segmentation Benchmark [13], which includes 380 surface meshes of 19 different object categories (see Fig. 9). All the models are scaled to have a 1.0 radius bounding sphere. The quality of the decomposition is measured via the Rand Index that estimates the similarity by measuring facet pairwise label difference between the segmentations generated by CoRiSe and those generated by humans and the state-of-art methods [6,7,3,1]. All experimental results are collected on a workstation with two Intel Xeon 2.30 GHz CPUs and 32 GB memory. HACD and CoRise are implemented using C++ and WCSeg uses MATLAB.

6.1. Comparison of approximation

Given a concavity tolerance, we compare CoRiSe with HACD [9] and WCSeg [1] based on (1) the number of decomposition components, (2) decomposition time, and (3) approximation accuracy. HACD first simplifies the model using Quadric Error Metrics (QEM) and then hierarchically merges facets and ensures

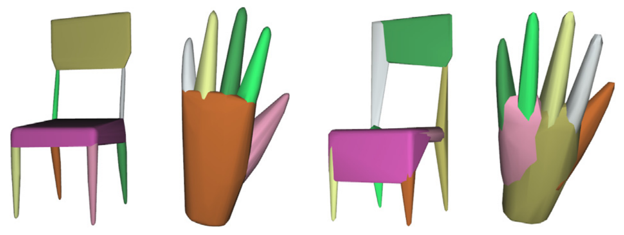
(a) CoRiSe. (b) HACD.

Fig. 10. Shape approximation using the convex hulls of all components in the decompositions from CoRiSe and HACD.

Table 1
Statistical results of the number of final components, computation time and the volume ratio which is approximations' volume compared to that of the original model on the Princeton 3D Mesh Segmentation Benchmark. All results are obtained using concavity tolerance $= 0.1$.

Category	# of components			Time (s)			Volume ratio		
	HACD	WCSeg	CoRiSe	HACD	WCSeg	CoRiSe	HACD	WCSeg	CoRiSe
Human	12.10	18.85	9.85	84.80	677.92	2.33	1.23	1.24	1.31
Cup	18.90	5.50	4.55	372.09	1362.30	1.25	2.66	3.02	3.15
Glass	6.40	8.90	5.53	433.84	416.09	0.63	1.69	1.90	1.75
Airplane	5.20	7.70	7.55	1103.85	431.91	1.07	1.30	1.25	1.37
Ant	13.45	11.25	14.50	223.99	413.63	1.76	1.11	1.17	1.14
Chair	9.50	16.35	10.64	297.72	1021.50	1.86	1.67	1.56	1.45
Octopus	14.40	9.10	14.45	903.44	599.90	1.98	1.23	1.83	1.27
Table	6.00	5.55	6.05	3385.97	2762.60	1.25	1.33	1.68	1.42
Teddy	6.80	7.45	7.05	1548.05	1026.30	2.16	1.05	1.05	1.07
Hand	9.10	9.20	9.00	205.92	670.00	1.67	1.14	1.16	1.27
Plier	5.20	6.50	5.89	393.18	368.90	0.88	1.30	1.37	1.33
Fish	5.05	6.25	5.20	2969.80	546.04	0.84	1.12	1.16	1.18
Bird	5.55	11.10	6.39	547.89	515.03	0.73	1.36	1.26	1.35
Armadillo	15.30	20.10	14.90	56.89	1914.80	5.46	1.12	1.12	1.19
Bust	9.30	8.70	7.80	118.18	2285.80	4.50	1.05	1.08	1.11
Mech	4.45	3.70	3.65	1286.43	2587.30	1.04	0.87	1.02	1.02
Bearing	5.47	3.00	1.58	1095.92	1188.90	0.25	0.95	1.08	1.17
Vase	9.80	5.65	5.15	1059.17	1247.60	1.63	1.06	1.10	1.12
Four-leg	12.95	14.05	12.25	61.85	614.97	1.86	1.20	1.23	1.25
Average	9.21	9.42	8.00	849.95	1086.90	1.75	1.29	1.38	1.36

that the concavities of all clusters are lower than the tolerance concavity. WCSeg creates initial segmentation using line-of-sight concavity/convexity measurement and then uses Shape Diameter Function [6] to merge the initial segmentations. Table 1 shows the decomposition size, computation time, and the ratio between the volume of the convex hull approximation and the volume of the original model (see Fig. 10).

From Table 1, we can observe that, on average, CoRiSe generates fewer components than HACD and WCSeg. In addition, CoRiSe is significantly faster than HACD and WCSeg. This is because HACD requires many concavity evaluations in the bottom-up approach clustering process. Note that even though it is not totally fair to directly compare the computation time of CoRiSe and WCSeg shown in Table 1 due to WCSeg's matlab implementation, doing spectral clustering that involves solving the eigen-decomposition of a large pairwise matrix (whose size is square to the number of facets) would still be a bottleneck using the faster C++ implementation. Without simplifying models, HACD can even take more than several hours to decompose a model with around 10,000 vertices.

Table 1 also shows the comparison of volume ratios. The volume ratio is defined as $\mathrm{vol}(\cup_i(CH_i))/\mathrm{vol}(P)$, where $\mathrm{vol}(\cup_i(CH_i))$ is the volume of the union of all convex hulls and $\mathrm{vol}(P)$ is the volume of the input mesh. From Table 1, HACD has smallest average volume ratios. However, the result is biased because HACD simplifies the input model before decomposition, thus these convex hulls created by HACD do not guarantee to enclose the original model. This may also produce noticeable penetration if these convex hulls are used in collision detection. We can also see that the approximation volume of CoRiSe is slightly smaller than WCSeg. However, because WCSeg produces more segments, we can say that CoRiSe is more effective (use less components to provide tighter approximation). This is especially true in the human and octopus category. Fig. 11 shows an example of the segmentations of a human model and an octopus model from both CoRiSe and WCSeg. The main differences between CoRiSe and WCSeg stem from how concavity is used for segmentation. CoRiSe cares about the approximation error, thus ignores the fingers of the human model because the concavity between two fingers is smaller than the concavity tolerance. On the other hand, the legs of the octopus model are bended and CoRiSe cuts the legs into several components to satisfy the concavity tolerance while WCSeg merges them together.

6.2. Comparison using benchmark dataset

We should first note that CoRiSe aims at resolving the concavity of a model and is not designed for semantic segmentation. For example, in Fig. 17, the bird is decomposed into over 30 parts which is much more than that of other shape segmentation. This will certainly influence the evaluation if CoRiSe is compared to segmentations created by human. However, we feel it is important to compare CoRiSe to shape segmentation methods since, after all, convexity is one of the important properties used in shape segmentation, and we also believe that a visually meaningful decomposition can provide visually convincing simulation.

Fig. 12 shows the Rand Index (RI) scores obtained from seven methods on the same benchmark. Although CoRiSe performs worse than the learning-based approach, CoRiSe outperforms some single-model based methods [6,7] that require more

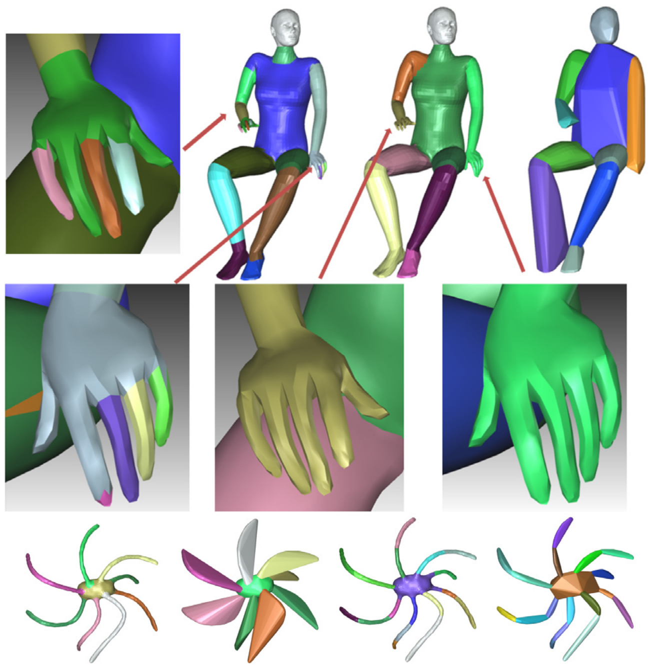

Fig. 11. WCSeg vs. CoRiSe. Top row from left to right: WCSeg decomposition, CoRiSe decomposition and convex hulls of CoRiSe components. Bottom row from left to right: WCSeg decomposition, convex hulls of WCSeg components, CoRiSe decomposition, and convex hulls of CoRiSe components.

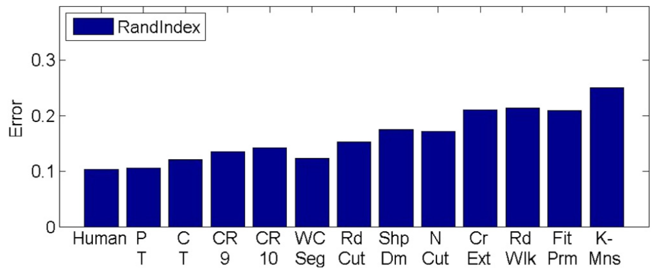

Fig. 12. Rand Index (RI) comparison on the Princeton Benchmark data. **human**: human cut; **PT**: CoRiSe with a τ for each model; **CT**: CoRiSe with a τ for each category; **CR9**: CoRiSe with $\tau = 0.09$; **CR10**: CoRiSe with $\tau = 0.1$; Additional comparisons evaluated using other metrics such as consistency error, cut discrepancy, Hamming distance all show similar trend as RI and can be found in the supplementary materials (see Appendix A).

user parameters, such as number of clusters. If CoRiSe can choose a concavity tolerance for each category, its performance is comparable to [1] which also has several parameters and thresholds. Moreover, if CoRiSe chooses concavity tolerance for each model individually, just as other methods selecting component size for each model, its RI score is lower than all single-model methods in Fig. 12.

Compared with other shape-segmentation methods, CoRiSe has three major advantages: First, CoRiSe provides the bounded convexity that can be used by broader applications. Second, CoRiSe has less number of parameters, while others require several user parameters for each model to specify component number, soft-clustering number, or smooth factors. Table 2 summarizes the number of parameters of each method. Finally, CoRiSe is efficient without simplification as pre-processing.

Although it is true that only a small subset of concavity tolerances produces semantic decomposition of a given shape, those that do create semantic decompositions reflect the geometric property of the shape. We encourage readers to look at our supplementary materials (see Appendix A) which show the RI, HD, CE and CD scores for a range of concavity tolerances.

Table 2
Number of parameters of segmentation methods (including fixed parameters). **WCSeg**: the parameters will be used for encoding pairwise matrix, computing Shape Diameter Function and post-merging; **RC**: rand cut uses the statistics of other approaches' decomposition results. The parameter number will be more the sum of those approaches; **SDF**: Shape Diameter Function method ≥5 parameters (number of rays per facet; cone angle; component number of GMM; two fixed parameters in encoding smoothness term same as CoRiSe; other parameters); **HACD**: many parameters in mesh simplification and thresholds for angles and edges during decomposition; **CORISE**: 3 parameters (the first is the concavity tolerance; the other two are the fixed parameters in encoding smoothness term).

WCSeg	RC	SDF	HACD	CoRiSe
>5	>5	≥5	7	3

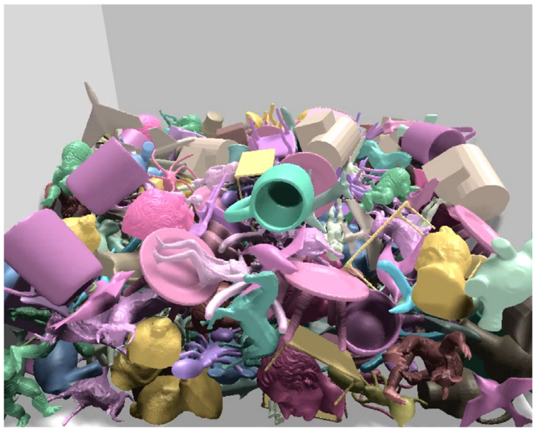

Fig. 13. Simulation created using the convex hulls of CoRiSe.

7. Applications

CoRiSe is designed to approximate a 3D shape with a set of convex objects. In this section, we demonstrate several applications using this type of approximation.

7.1. Physically-based simulation

Physics simulation libraries, such as Bullet and Box2D, use multiple convex shapes to approximate the original non-convex shape. Approximating a shape with bounded concavity error allows computations, such as collision response and penetration depth, become much more efficient. In these applications, we care about the number of approximation components and the approximation error. Exact convex decomposition has no approximation error, however, the number of components is usually prohibitively huge, thus the computation, e.g., penetration depth, is inherently expensive. It is therefore desirable to have smaller number of components while the approximation error is bounded. From the comparisons between HACD and CoRiSe shown in Table 1, CoRiSe provides following advantages: (1) CoRiSe guarantees to enclose the original model while HACD does not; (2) CoRiSe is much faster than HACD; (3) CoRiSe produces smaller segmentation given the same concavity tolerance.

We have successfully integrated CoRiSe with the Bullet library (see Fig. 13). This allows us to generate convincing physically-based simulations using only the convex hulls of CoRiSe decompositions. We encourage the readers to view the supplementary video (see Appendix A).

7.2. Cage-based deformation

Cage based deformation technique aims to be an easy-to-use tool for graphics modeling, texturing and animation. The

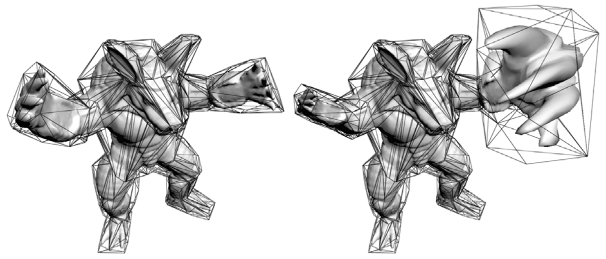

Fig. 14. A cage created from CoRiSe decomposition.

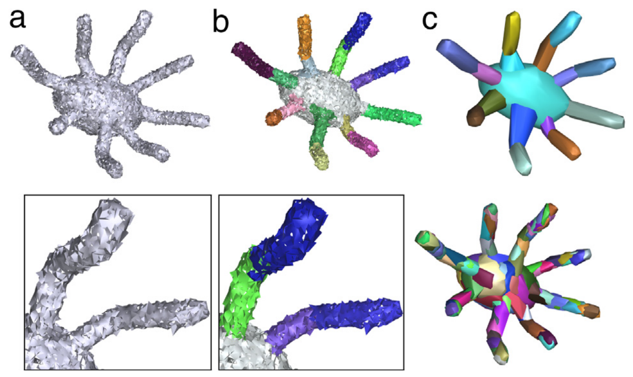

Fig. 15. (a) Noisy mesh with missing facets. The bottom figure provides a close-up view. (b) CoRiSe results. (c) Top: Convex hulls of 16 segmented parts in (b). Bottom: Convex hulls of HACD segmentation (138 components). All results are obtained using concavity tolerance 0.1.

advantage of cage-based space deformation is its simplicity, flexibility and efficiency. Most cages in cage-based deformation are created manually, and the process can be tedious. Xian et al. [30] proposed an automatic cage generation technique based on iteratively splitting the bounding boxes. However, their method requires mesh voxelization that is both time consuming and, more importantly, ignores important structural features of the input shape. For example, their cage usually misses vertices near the concave regions of the input mesh. Moreover, their method is not suitable for interactive applications as for a shape with only 10k vertices, their method can take several minutes. CoRiSe generates a cage in almost real-time. We first use CoRiSe to generate a nearly convex decomposition of the shape. Then convex hulls are computed for each part. These convex hulls are slightly expanded and simplified. The final cage is the union of the convex hulls. Fig. 14 shows a cage created for the Armadillo model and its deformation using Green coordinates. More examples can be found in the supplementary materials (see Appendix A).

7.3. Model repair

All the models used in Table 1 are watertight. However, many digitized models have degenerated features and may require mesh repair. The convex hulls of CoRiSe decomposition provide a convenient way to generate a watertight representation. In Fig. 15, we show a mesh that has 45% of the facets removed, and CoRiSe successfully decomposes the mesh and produces better approximation than HACD does. In fact, CoRiSe will be able to produce similar results as long as the mesh edge connectivity has not been significantly changed. For models with many holes, CoRiSe can get help from methods such as [31] that can be used to ensure that the projected geodesic path of the bridges is still well defined.

7.4. Origami

Origami has many applications in material engineering, manufacturing and packaging, such as solar panel packing and

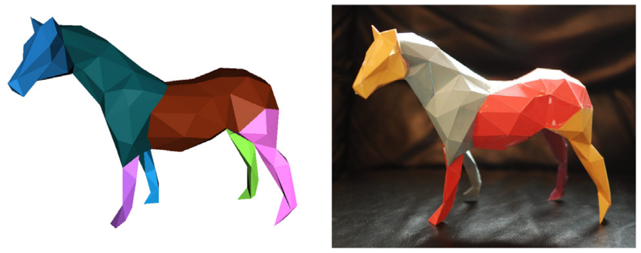

Fig. 16. Left: Segmentation result using CoRiSe. Right: Paper craft created by unfolding/folding each nearly convex component.

self-folding origami. Edge unfolding a convex polyhedron to a non-overlapping planer polygon (called a net) by cutting along its edges remains an open problem [32], in particular for non-convex shapes, which usually need to be segmented then a net can be found for each component [33,34]. Using CoRiSe to decompose a non-convex shape into nearly convex components can significantly reduce the computational time of finding the nets and make folding easier [35]. Fig. 16 shows an example of this approach.

8. Conclusion, limitations and future work

In summary, in this paper we proposed a 3D decomposition method that produces components with bounded concavity. Our method called CoRiSe meets the needs of real-time simulations and computer games better than the existing methods. We also showed that the decomposition results can produce semantically meaningful decomposition with low computation cost when proper concavity tolerances are given. The decomposition is achieved by identifying *convex ridges* and then solving a graph cut optimization problem formulated with *valley separators* and *must-link regions*. Comparing CoRiSe with other methods on the public benchmark dataset, we show that CoRiSe can generate results close to manual decomposition. Comparing CoRiSe with other approximate convex decomposition methods, we show that CoRiSe is significantly more efficient.

Known limitations. Each of the major step of CoRiSe is quite robust. For example, the convex ridges are always identified at expected locations for all the models that we tested (see the supplementary material, Appendix A for more examples). A main limitation of CoRiSe is that its semantic meaning of CoRiSe components is significantly influenced by the concavity tolerance τ. When τ is too small, small components are produced and the semantic meaning of these components diminishes. For example in Fig. 17(b) and (c), even though CoRiSe still provides a tight approximation of the initial model using a relatively small concavity tolerance (0.05), many small components do not have significant meanings.

Another limitation of CoRiSe is that CoRiSe may not decompose a cup-like shape if none of the valleys reach the bottom of the cup's concavity (see Fig. 17(d)). This is caused by the fact that a valley is the (approximated) shortest geodesic path. We believe that this can be addressed by ensuring that the vertex p with largest concavity of a valley has the local maximum concavity. If this is not the case, then we iteratively descend p until a point p' with local maximum concavity is reached. A new valley is then computed by enforcing it to pass through p'.

Acknowledgments

This work is in part supported by US National Science Foundation (NSF) IIS-096053, CNS-1205260, EFRI-1240459, AFOSR FA9550-12-1-0238 and in part by the Korean Federation of Science

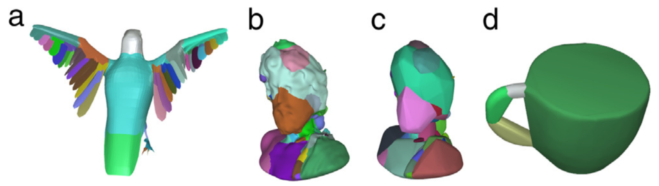

Fig. 17. (a) CoRiSe result of a bird. (b) CoRiSe result using $\tau = 0.05$. (c) The corresponding nearly convex approximation of (b). (d) CoRiSe result of a cup.

and Technology Societies (KOFST) grant funded by the Korean government.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cad.2016.05.015.

References

[1] van Kaick O, Fish N, Kleiman Y, Asafi S, Cohen-Or D. Shape segmentation by approximate convexity analysis. ACM Trans Graph 2014;34(1): Article 4, 11 pages.
[2] Au O-C, Zheng Y, Chen M, Xu P, Tai C-L. Mesh segmentation with concavity-aware fields. IEEE Trans Vis Comput Graphics 2012;18(7):1125–34.
[3] Zhang J, Zheng J, Wu C, Cai J. Variational mesh decomposition. ACM Trans Graph (TOG) 2012;31(3):21.
[4] Kalogerakis E, Hertzmann A, Singh K. Learning 3d mesh segmentation and labeling. ACM Trans Graph (TOG) 2010;29(4):102.
[5] Huang Q, Koltun V, Guibas L. Joint shape segmentation with linear programming. In: ACM transactions on graphics (TOG), Vol. 30. ACM; 2011. p. 125.
[6] Shapira L, Shamir A, Cohen-Or D. Consistent mesh partitioning and skeletonisation using the shape diameter function. Vis Comput 2008;24(4):249–59.
[7] Golovinskiy A, Funkhouser T. Randomized cuts for 3d mesh analysis. In: ACM transactions on graphics (TOG), Vol. 27. ACM; 2008. p. 145.
[8] Lien J-M, Amato NM. Approximate convex decomposition of polygons. Comput Geom, Theory Appl 2006;35(1):100–23. http://dx.doi.org/10.1016/j.comgeo.2005.10.005.
[9] 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.
[10] Müller M, Chentanez N, Kim T-Y. Real time dynamic fracture with volumetric approximate convex decompositions. ACM Trans Graph (TOG) 2013;32(4): 115.
[11] Attene M, Katz S, Mortara M, Patané G, Spagnuolo M, Tal A. Mesh segmentation—a comparative study. In: IEEE international conference on shape modeling and applications, 2006. SMI 2006, IEEE; 2006. p. 7–18.
[12] Shamir A. A survey on mesh segmentation techniques. In: Computer graphics forum, Vol. 27. Wiley Online Library; 2008. p. 1539–56.
[13] Chen X, Golovinskiy A, Funkhouser T. A benchmark for 3d mesh segmentation. ACM Trans Graph 2009;28(3): Article 73, 12 pages.
[14] Shlafman S, Tal A, Katz S. Metamorphosis of polyhedral surfaces using decomposition. In: Computer graphics forum, Vol. 21. Wiley Online Library; 2003. p. 219–28.
[15] Katz S, Tal A. Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Trans Graph 2003;22(3):954–61. http://dx.doi.org/10.1145/882262.882369.
[16] Yamauchi H, Lee S, Lee Y, Ohtake Y, Belyaev A, Seidel H. Feature sensitive mesh segmentation with mean shift. In: 2005 International conference Shape modeling and applications. IEEE; 2005. p. 236–43.
[17] Liu R, Zhang H. Mesh segmentation via spectral embedding and contour analysis. In: Computer graphics forum, Vol. 26. Wiley Online Library; 2007. p. 385–94.
[18] Liu G, Gingold Y, Lien J-M. Continuous visibility feature. In: 28th IEEE conference on computer vision and pattern recognition. (CVPR), Boston, MA: IEEE; 2015. p. 1182–90.
[19] Wang Y, Gong M, Wang T, Cohen-Or D, Zhang H, Chen B. Projective analysis for 3d shape segmentation. ACM Trans Graph (TOG) 2013;32(6):192.
[20] Golovinskiy A, Funkhouser T. Consistent segmentation of 3d models. Comput Graph 2009;33(3):262–9.
[21] Lu L, Choi Y, Wang W, Kim M. Variational 3d shape segmentation for bounding volume computation. Computer graphics forum, Vol. 26, Wiley Online Library; 2007. p. 329–38.
[22] Attene M, Mortara M, Spagnuolo M, Falcidieno B. Hierarchical convex approximation of 3d shapes for fast region selection. In: Computer graphics forum, Vol. 27. Wiley Online Library; 2008. p. 1323–32.
[23] Kreavoy V, Julius D, Sheffer A. Model composition from interchangeable components. In: 15th pacific conference on computer graphics and applications, 2007. PG'07, IEEE; 2007. p. 129–38.
[24] Ghosh M, Amato NM, Lu Y, Lien J-M. Fast approximate convex decomposition using relative concavity. Comput-Aided Des 2013;45(2):494–504.
[25] Asafi S, Goren A, Cohen-Or D. Weak convex decomposition by lines-of-sight. In: Computer graphics forum, Vol. 32. Wiley Online Library; 2013. p. 23–31.
[26] Lian Z, Godil A, Rosin PL, Sun X. A new convexity measurement for 3d meshes. In: 2012 IEEE conference on computer vision and pattern recognition. (CVPR), IEEE; 2012. p. 119–26.
[27] Zimmer H, Campen M, Kobbelt L. Efficient computation of shortest path-concavity for 3d meshes. In: 2013 IEEE conference on computer vision and pattern recognition. (CVPR), IEEE; 2013. p. 2155–62.
[28] Liu G, Xi Z, Lien J-M. Dual-space decomposition of 2d complex shapes. In: 27th IEEE conference on computer vision and pattern recognition. (CVPR), Columbus, OH: IEEE; 2014.
[29] Boykov Y, Veksler O, Zabih R. Fast approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 2001;23(11):1222–39.
[30] Xian C, Lin H, Gao S. Automatic cage generation by improved obbs for mesh deformation. Vis Comput 2012;28(1):21–33.
[31] Crane K, Weischedel C, Wardetzky M. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. ACM Trans. Graph. 2013;32.
[32] O'Rourke J. Unfolding polyhedra (2008).
[33] Straub R, Prautzsch H. Creating optimized cut-out sheets for paper models from meshes, 2011.
[34] Takahashi S, Wu H-Y, Saw SH, Lin C-C, Yen H-C. Optimized topological surgery for unfolding 3d meshes. In: Computer graphics forum, Vol. 30. Wiley Online Library; 2011. p. 2077–86.
[35] Xi Z, Lien J-M. Continuous unfolding of polyhedra—a motion planning approach, in: 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS), Hamburg, Germany, 2015, pp. 3249–3254.