Approximate Convex Decomposition of Polyhedra

Jyh-Ming Lien[*]
George Mason University

Nancy M. Amato[†]
Texas A&M University

Abstract

Decomposition is a technique commonly used to partition complex models into simpler components. While decomposition into convex components results in pieces that are easy to process, such decompositions can be costly to construct and can result in representations with an unmanageable number of components. In this paper we explore an alternative partitioning strategy that decomposes a given model into "approximately convex" pieces that may provide similar benefits as convex components, while the resulting decomposition is both significantly smaller (typically by orders of magnitude) and can be computed more efficiently. Indeed, for many applications, an approximate convex decomposition (ACD) can more accurately represent the important structural features of the model by providing a mechanism for ignoring less significant features, such as surface texture. We describe a technique for computing ACDs of three-dimensional polyhedral solids and surfaces of arbitrary genus. We provide results illustrating that our approach results in high quality decompositions with very few components and applications showing that comparable or better results can be obtained using ACD decompositions in place of exact convex decompositions (ECD) that are several orders of magnitude larger.

CR Categories: I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: concavity measurement, convex decomposition

1 Introduction

One common strategy for dealing with large, complex models is to decompose them into components that are easier to process. Many different decomposition methods have been proposed – see, e.g., Chazelle and Palios [1994] for a brief review of some common strategies. Of these, decomposition into convex components has been of great interest because many algorithms, such as collision detection and mesh generation, perform more efficiently on convex objects. Convex decomposition of polygons is a well studied problem and has optimal solutions under different criteria; see [Keil 2000] for a good survey. In contrast, convex decomposition in three-dimensions is far less understood and, despite the practical motivation, little research on convex decomposition of polyhedra has gone beyond the theoretical stage [Chazelle et al. 1995].

A major reason that convex decompositions of polyhedra are not used more extensively is that they are not practical for complex models – an *exact convex decomposition (*ECD*)* can be costly to construct and can result in a representation with an unmanageable number of components. This is true for both *solid* decompositions, which consist of a collection of convex volumes whose union equals the original polyhedron, and *surface* decompositions, which partition the surface of the polyhedron into a collection of convex surface patches. For example, a solid ECD of the Armadillo model has more than 726,240 components (see Figure 1). Similar statistics for additional models are show in Table 1 in Section 6.

[*]Department of Computer Science, e-mail:jmlien@cs.gmu.edu

[†]Department of Computer Science, e-mail:amato@cs.tamu.ed

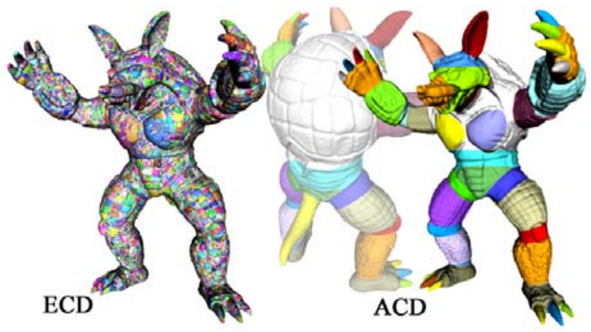


Figure 1: The approximate convex decompositions (ACD) of the Armadillo model consists of a small number of nearly convex components that characterize the important features of the models better than the exact convex decompositions (ECD) that have orders of magnitude more components. The Armadillo model (500K edges, 12.1MB) has a solid ACD with 98 components (14.2MB) that can be computed in 232 seconds while the solid ECD has more than 726,240 components (20+ GB) and could not be completed because the disk space was exhausted after nearly 4 hours of computation.

Our Approach. In this work, we explore a partitioning strategy that decomposes a polyhedron into *"approximately convex"* pieces. Our motivation is that for many applications, the approximately convex components of this decomposition provide similar benefits as convex components, while the resulting decomposition is both significantly smaller (typically by several orders of magnitude) and can be computed more efficiently. These advantages have been proven theoretically and experimentally for planar polygons by Lien and Amato [2004]. In this paper we show that, unlike ECD, it is feasible to apply the concept of approximate convex decomposition (ACD) to three-dimensional polyhedra. In particular, we describe

- practical methods for computing a solid or surface ACD of a polyhedron of arbitrary genus.

Our general strategy is to iteratively identify the most concave feature(s) in the current decomposition, and then to partition the polyhedron so that the concavity of the identified features is reduced. This process continues until all components in the decomposition have acceptable concavity, i.e., until they are convex 'enough,' which is a tunable parameter. While this follows the general approach used successfully for polygons, there are several operations that were straight forward for polygons but which become nontrivial for polyhedra. The main challenges include computing the concavity of a feature for a polyhedra and resolving concave features to generate small and high quality decomposition. To deal with these technical challenges in 3D, we introduce a new technique:

- *approximate feature grouping*, that enables sets of features to be processed together, which is both more efficient and produces better results.

We demonstrate the feasibility of our approach by applying it to

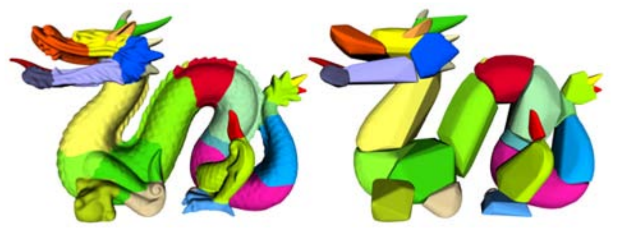


Figure 2: ACD provides a simpler representation of the dragon model using the convex hulls (slightly separated) of its components.

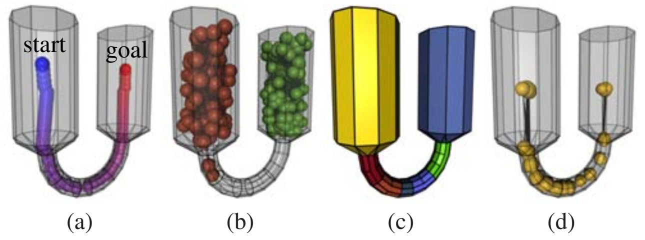


Figure 3: A difficult motion planning problem (a) in which the robot is required to pass through a narrow passage to move from the start to the goal. In (b), a uniform sampling of 200 collision-free configurations fails to connect the start to the goal. In contrast, in (d), placing 200 samples around the openings of the ACD of the environment (c) successfully connects the start to the goal. The solution path is shown in (a). See 'Motion planning' in Section 7 for detail.

a number of complex models. In general, even for very complex models, the ACDs have very few components, typically several orders of magnitude fewer than the ECDs. The size (memory) and computational time are also significantly less, particularly for the solid ACDs; see Figure 1.

We would like to emphasize that ACD aims to provide an approximate representation of the original shape using a set of convex components. Thus, unlike the part-based segmentations using automatic [Rom and Medioni 1994; Wu and Levine 1997; Mangan and Whitaker 1999; Li et al. 2001; Dey et al. 2003; Katz and Tal 2003; Goswami et al. 2006; Lai et al. 2006] or (semi-)interactive [Funkhouser et al. 2004; Lee et al. 2005; Liu et al. 2006] approaches, the main goal of ACD is in fact closer to that of the work on shape approximation [Wu and Levine 1994; Cohen-Steiner et al. 2004; Yamauchi et al. 2005]. While most shape approximations focused on meshes, ACD provides both solid and surface approximations.

Applications of ACD. In many applications, the detailed features of the model are not crucial and in fact considering them could serve to obscure important structural features and add to the processing cost. In such cases, an approximate representation of the model, such as our proposed ACD, that captures the key structural features would be preferable. For example, the ACD of the Armadillo model in Figure 1 identifies anatomical features much better than the ECD. Other applications of ACD include shape approximation (Figure 2), motion planning (Figure 3), mesh generation (Figure 4), and point location (Figure 5).

2 Preliminaries

A model P in $\mathbb{R}^2$ or $\mathbb{R}^3$ is represented by a set of boundaries ∂P. The *convex hull* of a model P, CH_P, is the smallest convex set enclosing P. P is said to be *convex* if $P = CH_P$. Features of P (vertices in $\mathbb{R}^2$ and edges in $\mathbb{R}^3$) are *notches* (non-convex features) if they have internal angles greater than $180°$. We say P_i is a component of P if $P_i \subset P$. A set of components $\{P_i\}$ is a *decomposition* of P if their union is P and all P_i are interior disjoint, i.e., $\{P_i\}$ must satisfy:

$$\mathrm{D}(P) = \{P_i \mid \cup_i P_i = P \text{ and } \forall_{i \neq j} P_i^\circ \cap P_j^\circ = \emptyset\}, \qquad (1)$$

where P_i° is the open set of P_i. A *convex decomposition* of P is a decomposition of P that contains only convex components.

For some applications, considering only the surface of a model is of interest. We say P_i is a *convex surface patch* of P if $P_i \subset \partial P$ and lies entirely on the surface of its convex hull H_{P_i}, i.e., $P_i \subset \partial H_{P_i}$ [Chazelle et al. 1995]. A *convex surface decomposition* of P is a decomposition of ∂P that contains only convex surface components.

Saliency. ACD decomposes a model by prioritizing salient features. Curvature is known to be the most popular tool to evaluate feature saliency, e.g., for non-photorealistic rendering [DeCarlo et al. 2003], texture mapping [Lévy et al. 2002], and shape segmentation [Funkhouser et al. 2004]. However, estimating curvature of an entire model is difficult. Expensive preprocessing, such as mesh smoothing, simplification [Katz and Tal 2003] and function approximation [Ohtake et al. 2004], or post-processing, such as Hysteresis thresholding [Hubeli and Gross 2001], are generally required. Despite its ability to identify *surface* features, e.g., crest, we believe that curvature, by itself, is not sufficient to identify *structural* features. Thus, ACD uses *concavity* to identify salient features.

Concavity. In contrast to measures like area and volume, concavity does not have a well accepted definition. A few methods have been proposed that attempt to define and measure the concavity of poly-

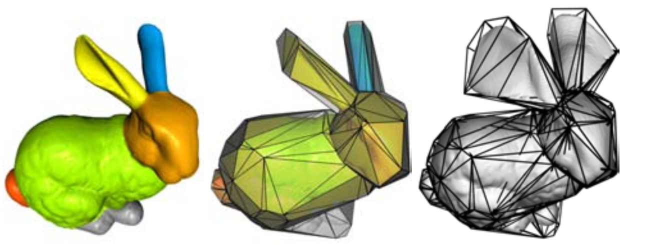


Figure 4: A tetrahedral mesh is generated from the (simplified) convex hulls of ACD components. The rightmost figure shows a deformation using this mesh.

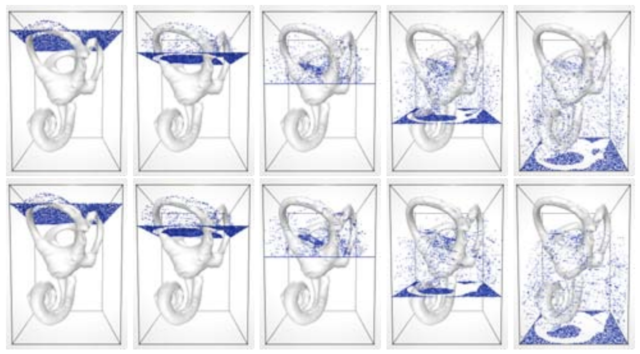


Figure 5: Snap shots of a system of 10,000 particles using the full model and the convex hulls of the ACD components. In this simulation, using ACD (lower row) is 2 times faster than using the full model (upper row) without introducing evident errors.

gons [Sklansky 1972; Lien and Amato 2004]. To our knowledge, no concavity measure has been proposed for polyhedra.

Although ACD is not restricted to a particular measure, all the measures we consider in this work define the concavity of a model P as the maximum concavity of its boundary points, i.e.,

$$\text{concavity}(P) = \max_{x \in \partial P} \{\text{concavity}(x)\} ,$$

where x are the vertices of P. An important consequence of this decision is that now we can use points with maximum concavity to identify important features where decomposition can occur. This would not be the case if we choose to sum concavities or use the *convexity* measurement in [Zunic and Rosin 2002], where the convexity of a model P is defined as $\frac{\text{volume}(P)}{\text{volume}(H_P)}$.

Concavity can be combined with other measures, e.g., curvature or convexity, to provide more sophisticated saliency identification. For example, ACD can combine concavity and convexity to focus on both deep and large features, e.g., to ignore wide but shallow or deep but narrow tunnels in a model. As we will see later (Section 4), we combine concavity and curvature for better feature grouping.

Measuring Concavity. Intuitively, one can think of the concavity measurement as the length of the path traveled by a point $x \in \partial P$ during the process of inflating a balloon of the shape of P until the balloon assumes the shape of CH_P. Although a physically based simulation of this *balloon expansion* [Kent et al. 1992] can be expensive, we will show later that x's traveling distance can be efficiently approximated.

In particular, our concavity measures use the concepts of *bridges* and *pockets*. Bridges are convex hull facets that connect non-adjacent vertices of ∂P, i.e., $\text{BRIDGES}(P) = \partial CH_P \backslash \partial P$. Pockets are the portion of the boundary ∂P that is not on the convex hull boundary ∂CH_P, i.e., $\text{POCKETS}(P) = \partial P \setminus \partial CH_P$.

Because concave features, i.e., notches, can only be found in pockets we measure the concavity of a notch x by

- associating each bridge with a unique pocket, and
- computing the distance from x to its associated bridge β_x, i.e., $\text{concavity}(x) = \text{dist}(x, CH_P) = \text{dist}(x, \beta_x)$.

For polygons, there is a natural one-to-one bridge/pocket matching that can be obtained easily. Also, in this case, Lien and Amato [2004] proposed two practical methods to compute the concavity: SL- and SP-concavity. SL-concavity is the straight-line distance to the bridge. SP-concavity is the length of the shortest path to the bridge without intersecting the polygon.

However, the techniques used for polygons do not extend easily to three-dimensions. In particular, there is no trivial one-to-one bridge/pocket matching. In addition, while SL-concavity can still be computed efficiently, the best known methods for computing shortest paths on polyhedra require exponential time [Sharir and Schorr 1986]. We will address these issues later in this paper.

3 Approximate Convex Decomposition

The goal of approximate convex decomposition (ACD) is to generate decompositions whose components are approximately convex. We estimate how convex a component is using the concavity of the component. For a given model P, P is said to be *τ-approximate* convex if $\text{concavity}(P) < \tau$, where $\text{concavity}(\rho)$ denotes the concavity measurement of ρ and τ is a tunable parameter denoting the non-concavity tolerance of the application. A *τ-approximate* convex decomposition of P, $\text{ACD}_\tau(P)$, is defined as a decomposition that contains only *τ-approximate* convex components; i.e.,

$$\text{ACD}_\tau(P) = \{P_i \mid P_i \in \text{D}(P) \text{ and } \text{concavity}(P_i) \le \tau\}. \quad (2)$$

Thus, an ACD_0 is simply an exact convex decomposition.

An ACD is generated by recursively removing (*resolving*) concave features in order of decreasing significance, i.e., concavity, until all remaining components have concavity less than some desired bound. This strategy is outlined in Algorithm 1.

Algorithm 1 ACD(P, τ)

```
Input. A model, $P$, and tolerance, $\tau$.
Output. A decomposition, $\{P_i\}$, such that
    $\max\{\text{concavity}(P_i)\} \le \tau$.
1: if $\text{concavity}(P) < \tau$ then                  ▷ see Sections 4.1 and 5
2:     return $P$
3: else
4:     Let $x$ be a feature (notch) realizing $\text{concavity}(P)$
5:     $\{P_i\} = \text{resolve}(P, x)$                  ▷ see Sections 4.2 and 4.3
6:     for each component $\{P_i\}$ do
7:         ACD($P_i$,$\tau$)
```

The two main operations required in Algorithm 1 for ACD are:

- measuring the concavity of a feature(s), and
- resolving specified concave feature(s).

3.1 Measuring Concave Features

ACD measures the concavity as the distance from a feature to its associated bridge. Unfortunately, unlike polygons, there is no trivial one-to-one bridge/pocket matching for polyhedra. The problem of obtaining the bridge/pocket relationship is closely related to the problem of spherical [Praun and Hoppe 2003] and simplical [Khodakovsky et al. 2003] parameterization. However, mesh parameterization is costly to compute. Polyhedron realization [Shapiro and Tal 1998] that transforms a polyhedron P to a convex object H can be computed efficiently, but H is generally not the convex hull of P and cannot be determined before performing the transformation.

In addition, while SL-concavity can still be computed efficiently, the best known methods for computing shortest paths on polyhedra require exponential time [Sharir and Schorr 1986] and even methods [Choi et al. 1997] that approximate the shortest paths are too inefficient to be used in our approach. We use only SL-concavity in this paper.

3.2 Resolving Concave Features

Notch-cutting [Chazelle 1981] is a strategy that splits a polyhedron with a cut plane can be used to resolve notches in Algorithm 1. The details of this notch-cutting strategy are discussed in [Bajaj and Dey 1992]. Figures 6(a)(b) illustrate an ACD using cut planes that bisect dihedral angles.

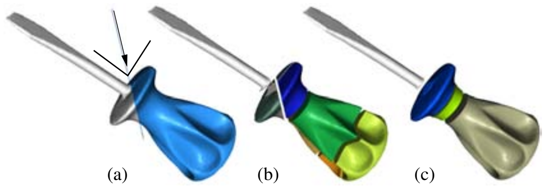


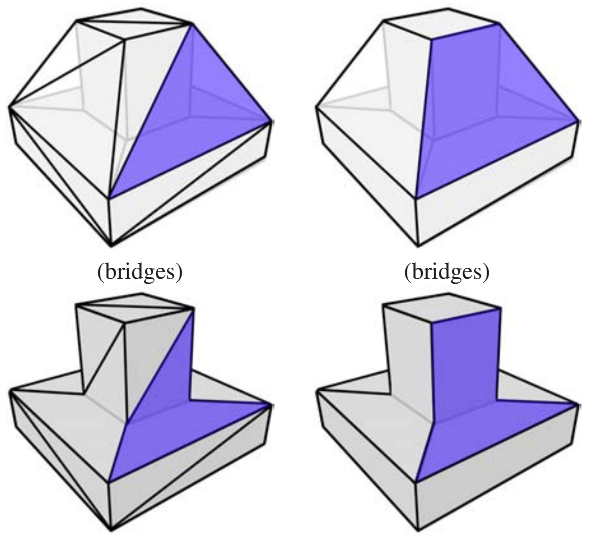
Figure 6: Resolving concavity (a) using a cut plane that bisects a dihedral angle results in (b) a decomposition with 10 components with concavity ≤ 0.1. In contrast, (c) carefully selected cut planes generate only 4 components with concavity ≤ 0.1.

A difficulty of this approach is selecting "good" cut planes. For example, in Figure 6(c), carefully selected cut planes can generate fewer components than cut planes that simply bisect the dihedral angles of notches. Unfortunately, good strategies for finding such good cut planes are not well known. Joe [1994] proposed an approach to postpone processing notches whose resolution would produce small components, but this strategy still produces many small components with sharp edges for large models, especially for more complicated models that are commonly seen nowadays.

3.3 General Strategy: Feature Grouping

For both measuring and resolving concavities, we use a technique we call *feature grouping* to collect sets of similar and adjacent features that can be processed together.

For measuring concavity, by allowing bridges to be formed from convex hull patches instead of a single convex hull facet, we can both dramatically reduce the number of bridges as well as decrease the cost of computing the pocket to bridge matching. Figure 7 shows an example of the bridge/pocket relationship with and without grouping. As we will see in Section 4.1, bridge patches can be used to provide a conservative measure of concavity.



without grouping **with grouping**

Figure 7: The bridges and the pockets with and without bridge grouping (clustering).

Resolution of concavity can also be improved by considering feature sets rather than individual features and by forcing the cut plane to be defined with respect to a feature set. Unlike the existing curvature-based methods [Hubeli and Gross 2001; DeCarlo et al. 2003; Ohtake et al. 2004; Rusinkiewicz 2004; Yoshizawa et al. 2005], our feature grouping is based on concavity.

4 ACD of Polyhedra without Handles

We first discuss our strategy for computing an ACD of a genus zero polyhedron. This strategy will be extended to handle polyhedra with non-zero genus in the next section.

4.1 Measuring Concave Features

Recall that we define the concavity of a vertex x as the distance from ∂P to the convex hull boundary. Since there is no unambiguous mapping from notches to convex hull facets in 3D as there was in 2D, we first must define one.

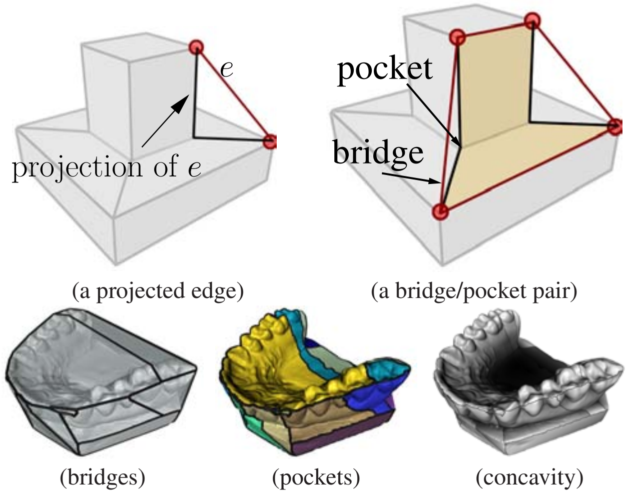


Figure 8: Top: An identified bridge/pocket pair. Bottom: Bridge/pocket pairs from the teeth model. The rightmost model is shaded so that darker areas indicate higher concavity.

Our strategy to match bridges with pockets is to identify pockets by *projecting* convex hull edges to the polyhedron's surface. The "projection" of a convex hull edge e is a path on the polyhedron's surface ∂P connecting the end points of e; we compute the paths on ∂P using Dijkstra's algorithm. After the convex hull edges are projected, the set of all (connected) polyhedral facets bounded by the projected edges forms a pocket. See Figure 8. After matching bridges with pockets, we measure the concavity of x in pocket ρ as the straight line distance to the tangent plane of ρ's associated bridge β.

Extension 1: Feature grouping – a conservative estimation. Finding pockets for all facets in ∂CH_P can be costly for large models. It turns out we can reduce this cost and still provide a conservative estimate of concavity by grouping clusters of 'nearly' coplanar and contiguous facets to form a *bridge patch* (or simply a *bridge*) on ∂CH_P. We then designate a *"supporting" plane* that is tangent to ∂CH_P as a representative plane for all facets in the bridge and compute the concavity of a vertex as the distance to the supporting plane of its bridge; see Figure 9. The bridge patches can be selected so that the distance from all faces in the bridge patch to the supporting plane will be guaranteed to be below some tunable threshold ϵ. For example, when $\epsilon = 0.05$, only 20 bridges are identified for the model in Figure 8 which has 4,626 facets on its convex hull.

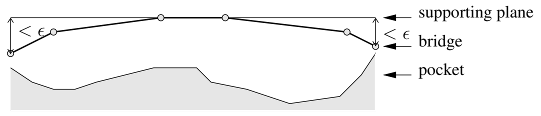


Figure 9: A bridge patch and its supporting plane.

One way to compute bridge patches is from an outer approximation of a polyhedron. Here we use *Lloyd's* clustering algorithm adapted from [Cohen-Steiner et al. 2004] to identify bridges and to ensure that the maximum distance from the included facets to the supporting plane is less than ϵ. Our clustering process is composed of the following two main steps:

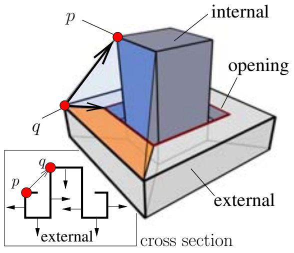
1. estimating the number k of the required bridges, and
2. grouping the convex hull facets into k clusters.

In the first step, we estimate the required bridge size for a given threshold ϵ by incrementally creating bridges and assigning convex hull facets to the bridges until all the convex hull facets are assigned. We say that a facet can be assigned to a bridge if the distance between them is less than ϵ. Our estimation process is outlined in Algorithm 2 in Appendix A.

In the second step, after we know the upper bound of the number of bridges required, we can approximate the convex hull boundary. This can be solved using *Lloyd's* clustering algorithm introduced in [Cohen-Steiner et al. 2004], which iteratively assigns all convex hull facets to the best bridges using a priority queue.

It is important to note that, as stated in Observation 4.1, the estimated concavity measurement computed this way is always greater than or equal to the concavity measured as convex hull facets are projected individually. Therefore, the estimated concavity is an upper bound for the actual concavity.

Observation 4.1. *The estimated concavity measurement is always greater than, in an amount less than ϵ, or equal to the concavity measured as convex hull facets are projected individually.*



Extension 2: Polygonal surface. In most cases, the previously mentioned concavity measure can handle surfaces with *openings* naturally. The case that requires more attention is when a surface "exposes" its internal side to the surface of the convex hull, e.g., the surface on the right. The internal side of a surface is exposed to the convex hull surface *if and only if* at least one of the convex hull vertices is *concave*. A convex hull vertex p is concave if its outward normals on the convex hull and on the surface are pointing in opposite directions. The point p (resp., q) in the figure above is concave (resp., convex).

Now, we can compute the pocket of a bridge β from the projection of β's boundary $\partial\beta$. Let e be an edge of $\partial\beta$. If e's vertices are

- *both convex*, then project e as before,
- *both concave*, then e has no projection,
- *one convex and one concave* (e.g., the edge $\overline{pq}$ in the figure), then e's projection is the path connecting the convex end to the opening.

4.2 Feature Grouping: Global Cuts

When resolving concave features, the concept of feature grouping allows us to better prioritize concave features for resolution and also results in a smaller and more meaningful decomposition. We first describe our method for grouping features, and then show how the groups are used to select cut planes to partition the model.

Our strategy of grouping concave features is a *concavity-based* bottom-up approach in which critical points, called "*knots*", on the boundary of each pocket are connected into local feature sets, called "*pocket cuts*", which are then grouped to form global feature sets, called "*global cuts*". Our approach is illustrated in Figure 10 and sketched below.

1. *Identifying knots*. Knots are critical points on a pocket boundary $\partial\rho$ identified as notches of the simplified $\partial\rho$ using the Douglas-Peucker (DP) algorithm [Hershberger and Snoeyink 1992] with simplification threshold δ, $0 \le \delta \le \tau$.
2. *Computing pocket cuts*. A pocket cut is a chain of consecutive edges in a pocket ρ whose removal will bisect ρ. Here, pocket cuts are paths connecting pairs of knots, and we consider all knot pairs for ρ.
3. *Weighting cuts*. The weight of a cut determines the quality of the cut. We compute the weight of each pocket cut κ as $\mathrm{W}(\kappa) = \omega(\kappa)/\gamma(\kappa)$, where $\omega(\kappa) = |\kappa|/\sum_{v\in\kappa}\mathrm{concavity}(v)$ is the reciprocal of the *mean concavity* of κ and $\gamma(\kappa)$ is the accumulated curvature of the edges in κ. The curvature of an edge e is measured using the *best fit polynomial* [Hubeli and Gross 2001].
4. *Connecting pocket cuts into global cuts*. Our strategy is to organize the knots and pocket cuts in a graph G_K whose vertices are knots and edges are pocket cuts. The cycle with the minimum weight in G_K will be the global cut.

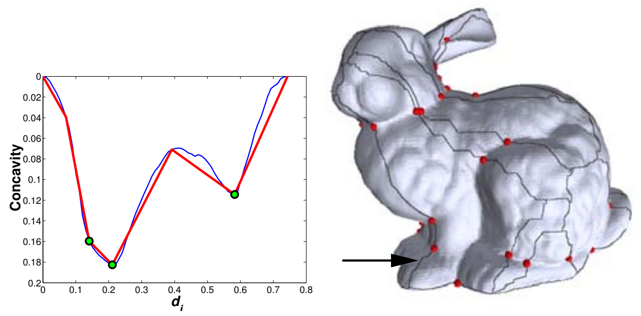


Figure 11: The thin (blue) line in the plot is a pocket boundary of the Stanford Bunny (indicated by an arrow) in concavity domain. Its simplification is shown in a thicker (red) line and identified knots are marked as dots.

Essentially, this bottom-up approach identifies and groups the knots on the projected bridge edges. It is natural to ask why knots are of interest. As knots are the critical points of a projected bridge edge π_e, we also consider a projected bridge edge as a critical representation of a polyhedral boundary. Note that the end points of π_e are both vertices of the convex hull. Intuitively, the vertices of π_e are *samples* of ∂P and therefore encode important geometric features related to concavity over the traversal from one *peak* to another *peak* i.e., π_e is an evidence that shows how the convex hull vertices are connected on ∂P.

Next, we will provide more implementation details and justify the choices of the steps mentioned above. The reader may first skip the details and proceed to Section 5 to focus on this work's high level strategy.

4.2.1 Step 1: Identifying Knots

We use the Douglas-Peucker (DP) line approximation algorithm to identify knots because DP can reveal critical points [White 1985] and resembles the concept of ACD. A critical point in DP of a polyline π is a farthest point from the line segment connecting the end points of π and, similarly, a knot in ACD is a farthest point from the bridge boundary. This provides an explanation of why we can use DP to extract important concave features.

Given a pocket boundary $\pi_e(i)$, knots are critical points on $\pi_e(i)$ found by the DP algorithm. To identify knots on $\pi_e(i)$, we first transform $\pi_e(i)$ in $\mathbb{R}^3$ into a two dimensional line $\pi_e^*(i)$ in the *concavity space* using the following function:

$$\pi_e^*(i) = \big(d_i, \mathrm{concavity}(\pi_e(i))\big),\ 0 \le i \le 1, \qquad (3)$$

where $d_i = i \cdot |e|$ and $|e|$ is the length of e. Then $\pi_e^*(i)$ is simplified using the DP algorithm [Hershberger and Snoeyink 1992]. We call a vertex a "knot" if it is a *notch* in $\pi_e(i)$ with concavity larger than δ, $0 \le \delta \le \tau$. The threshold δ controls the size of knots, i.e., a smaller δ implies more concave features will be identified; in this paper, we experimentally set δ between $\frac{\tau}{10}$ and $\frac{\tau}{100}$.

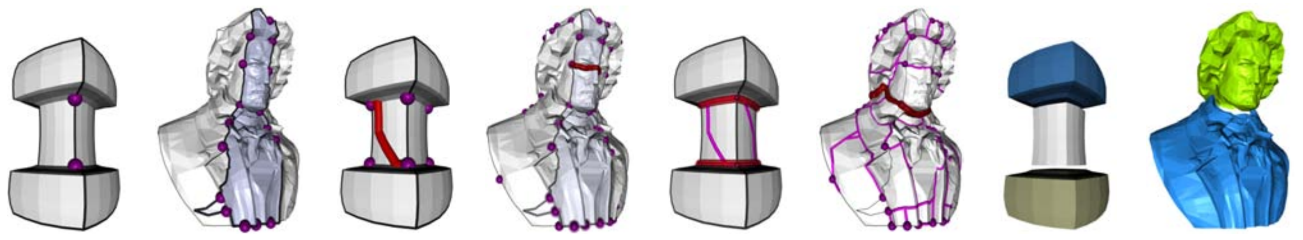

(a) identifying knots (b) computing pocket cuts (c) extracting global cuts (d) splitting the model

Figure 10: The process of grouping and resolving concave features. (a) Knots (marked by spheres) from one of the pockets. (b) Knots from all pockets and a pocket cut (shown in thick lines) connecting a pair of knots. (c) Global cuts (thick lines) and the graphs G_K. (d) Solid (left) and surface (right) decompositions using the identified global cuts.

An example of $\pi^*_e(i)$ and identified knots are shown in Figure 11. We note that these pocket boundaries have similar functionality as the *exoskeleton* that connects critical points on ∂P coded with *average geodesic distance* [Zhang et al. 2003].

4.2.2 Step 2: Computing Pocket Cuts

A pocket cut is a chain of consecutive edges in a pocket ρ whose removal will bisect ρ. In fact, any path in ρ that connects any two knots is a pocket cut. For a given pair of knots, we form a pocket cut by computing a path using Dijkstra's algorithm (w.r.t. a weight function W defined in Step 3). Figure 12(a) and (b) shows a pocket with its knots on the boundary and all of its pocket cuts, respectively.

A pocket with n_k knots has $O(n_k^2)$ pocket cuts. Not all of these $O(n_k^2)$ pocket cuts in ρ are interesting to us. In fact, we only need to consider $O(n_k)$ pocket cuts. This reduction is based on the following observation.

Observation 4.2. *Let N_{ρ_i} be a set of knots on the boundary between ρ and one of its neighboring pockets ρ_i. Pocket cuts between each pair N_{ρ_i} and N_{ρ_j} in ρ form a non-crossing minimum (weight) bipartite matching.*

We say two pocket cuts κ_ρ and κ'_ρ cross each other if κ'_ρ will become disconnected after ρ is separated by κ_ρ; see Figure 12(c). We also restrict a knot to be connected to only one knot from a neighboring pocket. The result of this restriction is that the pocket cuts between two boundaries form a bipartite matching of their knots and only $O(n_k)$ pocket cuts need to be considered when connecting them into global cuts; Figure 12(d) shows a result using the minimum weight bipartite matching (w.r.t. a weight function W).

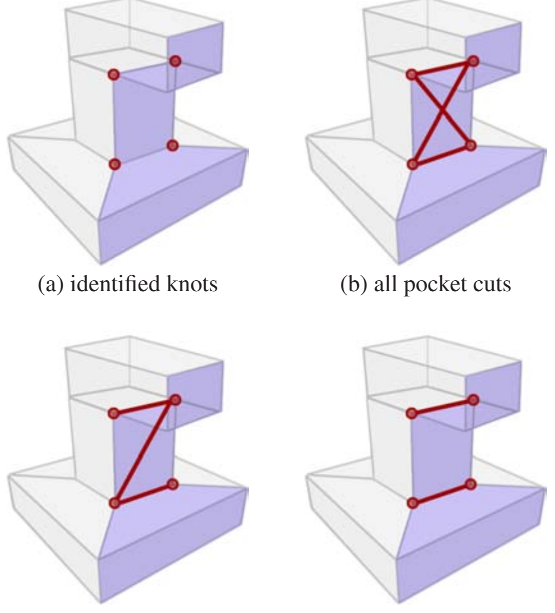
Cup-shape pocket. Because knots are identified on the boundary of a pocket ρ, we cannot find any pocket cut if the boundary of ρ is near its bridge β, e.g., a cup shape pocket. Indeed, decomposing a cup shaped model into meaningful components is known to be difficult. In our case, this problem can be solved by simply subdividing β and ρ into smaller bridges and pockets and forcing the new pocket boundary to pass the maximum concavity of ρ, as illustrated in Figure 13.

4.2.3 Step 3: Weighting a Cut

The weight of a cut determines the quality of the cut. As mentioned in Section 2, we believe that curvature, which has been extensively used to identify *surface* features, is not sufficient to identify *structural* features. Thus, we define the weight of a cut as:

$$\mathrm{W}(\kappa) = \frac{\omega(\kappa)}{\gamma(\kappa)} , \tag{4}$$

where $\omega(\kappa) = |\kappa|/\text{concavity}(\kappa)$ is the reciprocal of the *mean concavity* of a cut κ and $\gamma(\kappa)$ is the accumulated curvature of the edges in κ. The curvature of an edge e is measured using the *best fit polynomial* [Hubeli and Gross 2001] of the intersection of the model and the plane bisecting e. Since curvature is only measured on cuts, instead of on the entire model, the computation is less expensive.


(a) identified knots (b) all pocket cuts

(c) non-crossing pocket cuts (d) bipartite matching pocket cuts

Figure 12: (a) Identified knots of a pocket shown in dark circles. (b) All pocket cuts that connect all pairs of knots in the pocket. (c) Non-crossing pocket cuts. (d) Pocket cuts from bipartite matchings between pairs of boundaries.

4.2.4 Step 4: Extracting Cycles from Graph G_K

Recall that G_K is a graph whose vertices and edges are the knots and the selected pocket cuts. An example of G_K is shown in Figure 14. Each cycle in G_K represents a possible way of decomposing the model. The process of extracting cycles from G_K used here is similar to that of constructing a minimum spanning tree (MST) T_K on G_K by greedily expanding the most promising branch into all its neighboring pockets in each iteration. A cycle is identified when two growing paths of T_K meet. With this high level idea in mind, we are going to discuss technical details next.

Let κ_ρ be a pocket cut to be resolved, e.g., the pocket cut that con-

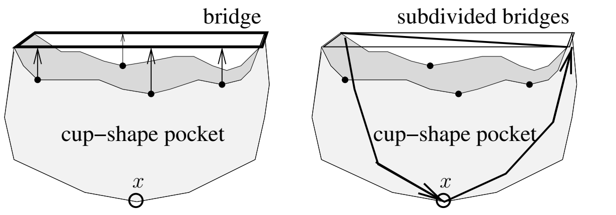


Figure 13: Left: A cup-shape pocket and its bridge. The boundary of the pocket are very close to the bridge. Right: The bridge is subdivided and the new pocket boundary is forced to pass the most concave feature x.

tains the most concave vertex. To find cycles that include κ_ρ, we extract T_K rooted at κ_ρ from G_K. T_K is constructed so that a path from the root κ_ρ to a leaf will consist of concave features that can be resolved together.

The process of building a tree T_K from G_K is similar to that of constructing a minimum spanning tree on G_K. An exception is that we also dynamically create new pocket cuts after each MST iteration. These new pocket cuts are simply the shortest (geodesic distance) paths connecting the current leaves of T_K to the pocket boundaries without knots, e.g., κ' in Figure 14. Thus, T_K can explore low concavity or even convex areas without using knots. A MST that is built directly on vertices and edges of a polyhedron has been used for feature extraction, e.g., [Pauly et al. 2003]. However, unlike T_K which is built on knots and pocket cuts, their MST requires pruning to enhance long features.

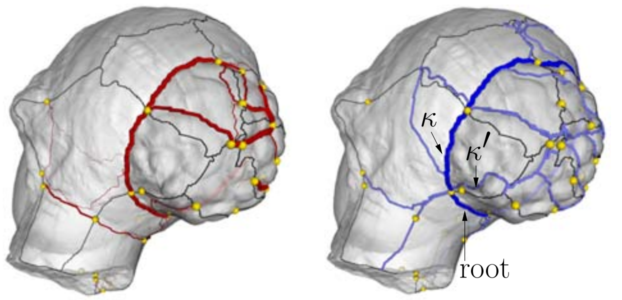


Figure 14: Left: An example of G_K (partially shown). Thicker pocket cuts have smaller weights. Right: An extracted tree from G_K. The boldest line is the best global cut for the root.

4.3 Resolving Concave Features

For convex volume decomposition, we define the cut plane of a global cut κ as the *best fit plane* of κ. For convex surface decomposition, we simply split the surface at the edges of κ.

A plane E fits κ best if E minimizes

$$\sum_{e\in\kappa} \text{concavity}(e)\cdot\mu_E(e)\ , \tag{5}$$

where $\mu_E(e)$ is the area between e and the projection of e to E. E can be approximated via a traditional principal component analysis using points sampled on κ.

Note that, sometimes, the intersection of E and the model P does not match the cut κ. An example of this problem is shown in Figure 15. This happens when the intersection traverses different pockets that κ does. It can be addressed by iteratively pushing E toward the vertices on the portion of κ that is misrepresented by the intersection, e.g., point a in Figure 15.

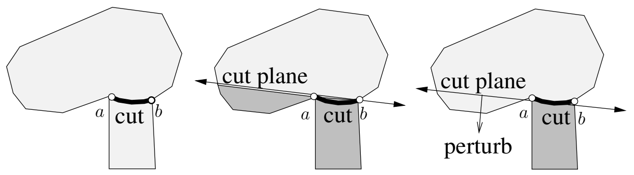


Figure 15: Left: A cut κ around the neck connecting points a and b. Mid: The best fit plane E of κ. In this case, E is slightly higher than a and E's intersection with the model does not match κ. Lighter and darker shades indicate different components after decomposition. Right: An improved cut plane by pushing E towards a.

4.4 Complexity Analysis

Theorem 4.3. *Let $\{C_i\}$, $i = 1,\ldots,m$, be the τ-approximate convex decomposition of a polyhedron P with n_e edges with zero genus. P can be decomposed into $\{C_i\}$ in $O(n_e^3 \log n_e)$ time.*

Proof. First, we show that ACD of a polyhedron P requires $O(n_v n_e \log n_v)$ time for each iteration in Algorithm 1, where n_v and n_e are the number of vertices and edges in P, resp. The dominant costs are the pocket cut computation, which extracts paths between knots on ∂P and can take $O(n_e \log n_v)$ time for each path extracted time using Dijkstra's algorithm. To resolve all r notches in P, Algorithm 1 will take $O(rn_v n_e \log n_v) = O(n_e^3 \log n_e)$. $\square$

Note that even though the time complexity of the proposed method is high, as seen in our experimental results, this is usually a very conservative estimate because the number of iterations required is usually small when the tolerance τ is not zero and the total number of pocket cuts is usually quite small.

5 ACD of Polyhedra with Arbitrary Genus

Because the convex hull of a polyhedron P is topologically a ball, multiple bridges may share one pocket for polyhedra with non-zero genus. For example, neither of the bridges α or β in Figure 16(a) can enclose any region by themselves. We address this problem by reducing the genus to zero.

Genus reduction is a process of finding sets of edges (called *handle cuts*) whose removal will reduce the number of *homological loops* on the surface of P. The problem of finding minimum length handle cuts is NP-hard [Erickson and Har-Peled 2002]. Several heuristics for genus reduction have been proposed (see a survey in [Zhang et al. 2003]). The identified handle cuts will then be used to prevent the paths of the bridge projections from crossing them. Figure 16(b) shows an example of a handle cut and the new bridge/pocket relation after genus reduction.

Although we can always use one of the existing heuristics, the bridge/pocket relationship can readily be used for genus reduction. Our approach is based on the intuition that the bridges that share the same pocket tell us approximate locations of the handles and the trajectory of how a hand "holds" a handle roughly traces out how we can cut the handle. For example, imagine holding the handle of the cup in Figure 16 with one hand: the hand must enter the hole though one of the bridges, e.g., β, and exit the hole from the other bridge, e.g., α. We call bridges that share a common pocket a set of "handle caps" of the enclosed handles. A model may have several sets of handle caps.

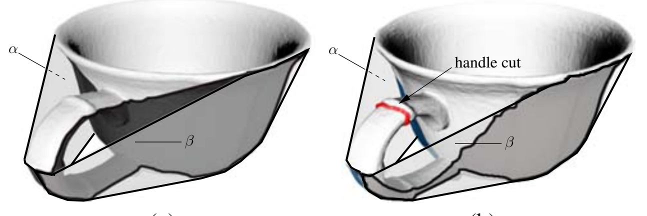


Figure 16: (a) The pocket (shaded area) is enclosed in the projected boundaries of two bridges β and α. (b) Pockets of β and α after genus reduction.

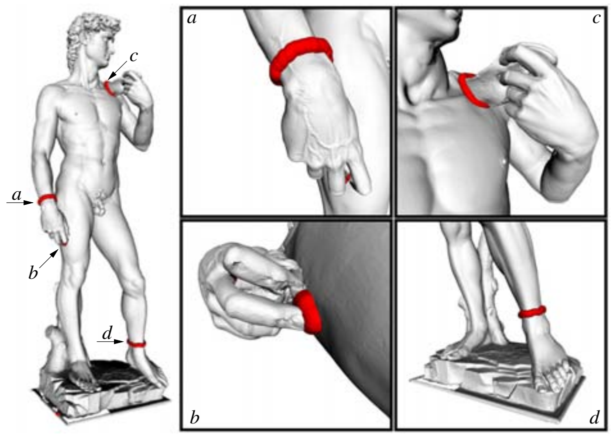


Figure 17: Four handle cuts found in the David model.

This intuition can be implemented by applying the following operations to identified handle cuts.

1. Flooding the polyhedral surface ∂P initiated from the projected boundaries of a set of handle caps. Vertices in a wavefront will propagate to neighboring unoccupied vertices.
2. Loops can be extracted by tracing in the backward direction of the propagation. For each pair of handle caps, we keep a shortest loop that connects their projected boundaries, if it exists.
3. Let G_h be a graph whose vertices are the handle caps and whose edges are the discovered handle cuts. Cycles in G_h indicate that the removal of all discovered handle cuts will separate P into multiple components. We can prevent P from being split by throwing away handle cuts so that no cycles are formed in G_h.
4. Check if the handle caps still share one pocket. If so, repeat the process described above until the remaining handle cuts are found.

Figure 17 shows a result of our approach. Note that we may not always reduce the genus of a model to zero because some handles are too small or can map to just one bridge, e.g., a handle completely inside a bowl. These "hidden" handles will eventually be unearthed as the decomposition process iterates if the concavity measurement of the handle is untolerable. For many applications, this behavior of ignoring insignificant handles can even represent the structure of the input model better [Wood et al. 2004].

6 Experimental Results

In this section, we compare exact (ECD) and approximate (ACD) convex decomposition. In addition, we consider four variants of ACD, i.e., solid or surface ACD, and ACD with or without feature grouping.

Implementation Details. There are three parameters, τ, ϵ, and δ, used in our proposed method. The first parameter is the concavity tolerance τ, which is used to control how convex the final components are and should be set according to the need of the application.

The second parameter is the bridge clustering threshold ϵ, which is the upper bound of the difference between the estimated concavity and the accurate concavity when the bridge clustering is not used. In our experiments, the value of ϵ does not significantly affect the final decomposition and is always set to be $\epsilon = \frac{\tau}{2}$.

The third parameter δ is used in the Douglas-Peucker (DP) algorithm, which is used to identify knots on the pocket boundaries for concave feature grouping. The value of δ is difficult to estimate and is set experimentally between $\frac{\tau}{10}$ and $\frac{\tau}{100}$.

Models. The models used in the experiments in this section are summarized in Table 1. In Table 1, for each model studied, we show the complexity of the model in terms of the number of edges, the ratio of notches with respect to the edges, and the physical file size in a simple BYU (Brigham Young University) format. In these 13 models, the David and the dragon models are not closed, i.e., with openings on their boundaries, and all the other models are closed.

6.1 Results

All experiments were performed on a Pentium 2.0 GHz CPU with 512 MB RAM. Our implementation of ACD of polyhedra is coded in C++. A summary of results for 13 models is shown in Table 1, which includes results from both solid and surface decomposition, and in Figures 18 and 19, which contain results of several approximation levels of ACD with and without feature grouping.

Result 1: *ACDs are orders of magnitude smaller than ECDs.* In Table 1, We show the size of the six decompositions, including solid $\text{ACD}_{0.2}$, solid $\text{ACD}_{0.02}$, solid ECD, surface $\text{ACD}_{0.2}$, surface $\text{ACD}_{0.02}$, and surface ECD, in terms of the number of final components and the physical file size in BYU format.

As seen in Table 1, the solid ACDs are orders of magnitude smaller than solid ECD. The solid $\text{ACDs}_{0.2}$ and solid $\text{ACDs}_{0.02}$ have 0.001% and 0.1% of the number of components that the solid ECDs have on average, resp. The physical file size of solid $\text{ACDs}_{0.2}$ and solid $\text{ACDs}_{0.02}$ are 0.08% and 0.16% of the size of the solid ECDs on average, resp. Note that the ECD process of the Armadillo model terminated early because it required more disk space than the available 20 GB. The results for ECD shown in Figure 18 are collected before termination, i.e., they are for an unfinished ECD, so all components are not yet convex. Figure 18 also shows that the solid ACD can be computed 72 times faster than the solid ECD. These times are representative of the savings offered by solid ACD over ECD.

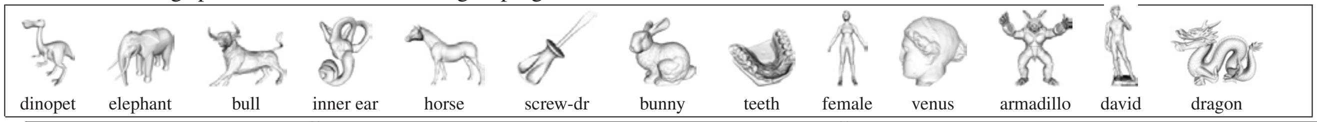
Although the file size of the surface ACDs is not significantly smaller than for the surface ECD, the surface $\text{ACDs}_{0.2}$ and surface $\text{ACDs}_{0.02}$ have 0.02% and 0.2% of the number of components that the ECD has on average. Figure 19 shows that ACDs only require a small constant factor increase in the computation time over the linear time surface ECD; this is representative of the relative cost of surface ACD and ECD. The table below summarizes these statistics.

	% solid ECD #components	% solid ECD file size	% surface ECD #components	% surface ECD file size
$\text{ACD}_{0.2}$	**0.001%**	**0.08%**	**0.02%**	**88.3%**
$\text{ACD}_{0.02}$	**0.1%**	**0.16%**	**0.2%**	**89.6%**

Result 2: *Solid ACDs are only slightly larger than surface ACDs.*

Table 1: Decompositions of 13 common models, where $|r|\%$ is the percentage of edges that are notches, $|e|$ is the number of edges, and S and $|P_i|$ are the physical (file) size and the number of components of the decomposition, resp. All models are normalized so that the radius of their minimum enclosing spheres is one unit. Feature grouping is used for ACDs.

Models



	Full model			Solid						Surface					
				ACD$_{0.2}$		ACD$_{0.02}$		ECD		ACD$_{0.2}$		ACD$_{0.02}$		ECD	
models	$\|r\|\%$	$\|e\|$	S	$\|P_i\|$	S	$\|P_i\|$	S	$\|P_i\|$	S	$\|P_i\|$	S	$\|P_i\|$	S	$\|P_i\|$	S
dinopet	34.9%	9,895	201 KB	13	252 KB	67	577 KB	5,607	38 MB	12	205 KB	62	226 KB	1,297	224 KB
elephant	30.4%	10,197	206 KB	13	338 KB	136	1.4 MB	5,349	50 MB	15	215 KB	123	250 KB	1,306	229 KB
bull	42.5%	18,594	379 KB	12	481 KB	211	2.3 MB	12,210	102 MB	12	388 KB	191	446 KB	3,486	444 KB
inner ear	34.0%	48,354	1.0 MB	31	1.4 MB	181	3.6 MB	14,591	171 MB	26	1.0 MB	89	1.1 MB	6,360	1.2 MB
horse	34.4%	59,541	1.3 MB	8	1.4 MB	77	2.4 MB	24,044	527 MB	8	1.3 MB	47	1.3 MB	8,095	1.4 MB
screw-dr	45.5%	81,450	1.8 MB	1	1.8 MB	44	3.0 MB	43,180	2.0 GB	1	1.8 MB	9	1.8 MB	15,052	2.1 MB
bunny	40.5%	104,496	2.3 MB	6	2.5 MB	178	6.6 MB	46,728	2.8 GB	6	2.3 MB	97	2.4 MB	16,549	2.7 MB
teeth	45.5%	349,806	7.9 MB	11	9.4 MB	307	18.8 MB	135,224	7.5 GB	29	8.0 MB	131	8.2 MB	67,059	9.4 MB
female	38.8%	365,163	8.5 MB	5	8.7 MB	67	10.9 MB	145,085	7.2 GB	5	8.5 MB	50	8.6 MB	51,580	9.3 MB
venus	43.8%	403,026	9.3 MB	3	9.5 MB	273	32.8 MB	166,555	18.2 GB	3	9.3 MB	164	9.6 MB	72,190	9.6 MB
armadillo	41.4%	518,916	12.1 MB	11	12.1 MB	98	14.2 MB	726,240	20+ GB	11	12.2 MB	85	12.4 MB	89,839	14.1 MB
david	38.7%	748,893	18.0 MB	models are						10	18.0 MB	170	18.3 MB	85,132	20.1 MB
dragon	42.8%	1,307,170	31.7 MB	not closed						12	31.8 MB	237	32.1 MB	246,053	37.3 MB

Table 1 also shows that the size of the solid ACDs are about 1.6 times larger than the surface ACDs due to the fact that the solid ACDs use cut planes to approximate (possibly non-planar) concave features.

Result 3: *ACDs with feature grouping are smaller than ACDs without feature grouping.* This experiment studies the effect of feature grouping on the ACDs of the Armadillo and the David models. We further investigate ACDs with different approximate levels. Figures 18 and 19 show results of solid and surface decomposition for a range of approximation value τ, respectively. Figures 18 and 19 show that feature grouping successfully reduces the size of both solid and surface decompositions. In particular, we see a slowly increasing size for ACDs with feature grouping as the value of τ decreases (i.e., as the convex approximation approaches an exact convex decomposition). In addition, with feature grouping, ACD produces structurally more meaningful components.

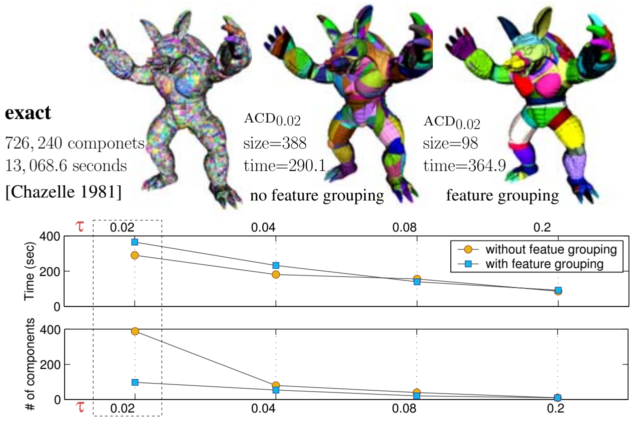


Figure 18: Convex solid decomposition. The size and time of ACD with and without feature grouping are shown for a range approximation values τ.

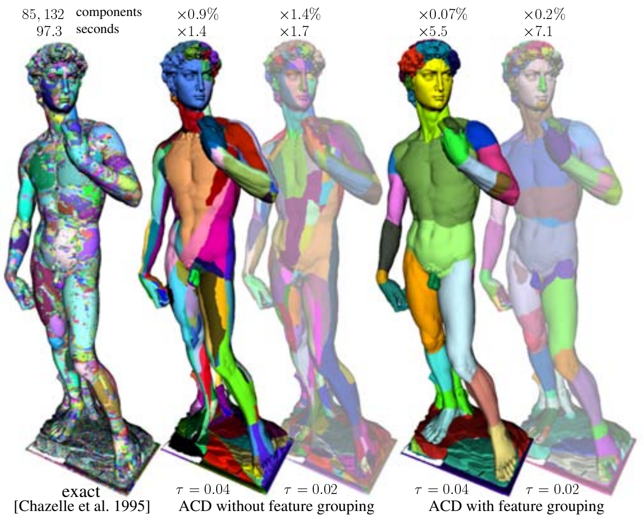


Figure 19: Convex surface decomposition. The leftmost figure shows a result of the exact decomposition using the "flood-and-retract" heuristic. The others are results of the approximate decomposition.

7 Applications of ACD

The convex hulls of the ACD components (and sometimes the components themselves) can be used by methods that usually operate on convex polyhedra, making them more efficient. This includes a large set of problems in computational geometry and graphics. Here, we demonstrate four applications including point location, shape representation, motion planning, and mesh generation.

Point location (solid ACD without feature grouping). Point location, which checks if a point x is in a polyhedron P, is a fundamental problem that can be found in ray tracing, simulation, and sampling. Point location can be solved more efficiently for convex polyhedra by checking if x is on the same side of all P's facets. Locating points for a non-convex model can benefit from ACD using the convex hulls of its ACD components if some errors can be tolerated, e.g., the particles in Figure 5. These errors are due to the difference between the component convex hulls of the ACD components and the original model.

In our experiments, point location of 10^8 random points is performed for the solid ECD and for the convex hulls of the $\text{ACD}_{0.02}$ components; point location in the ACD did not exploit the hierarchical structure of the ACD, but simply tested each component separately.

As seen in Figure 20, even using this naive strategy, point location in the ACD is about 23% faster than in the original teeth model. As seen with the elephant model, the advantage of the ACD over the ECD is even more pronounced. In both experiments, fewer than 1% errors were introduced using ACD.

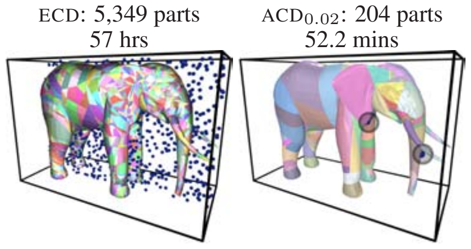


Figure 20: Point location of 10^8 points in the solid ECD and the convex hulls of the $\text{ACD}_{0.02}$ of the elephant model (6,798 triangles). Measured time includes time for decomposition and point location. Point location in $\text{ACD}_{0.02}$ has 0.99% errors. External points of 1000 samples in ECD are shown in the figure on the left and only the misclassified (as internal) points in ACDs are shown on the right.

Shape approximation (surface ACD with feature grouping). The components of an ACD can also be used for approximating shapes using the convex hulls of the ACD components. Figure 2 shows a simplified representation of the dragon model.

Although there are no well accepted criteria to compare decompositions, we can compare the skeletons extracted from the decompositions (see [Lien et al. 2006] for details), e.g., using graph edit distance [Bunke and Kandel 2000], which computes the cost of operations (i.e., inserting/removing vertices or edges) needed to convert one graph (skeleton) to another. Using this metric, Figure 21 shows that ACD still produces matching representations after deformations.

Motion planning (surface ACD with feature grouping). The ACD components can help to plan motion, e.g., for navigating in the human colon or removing a mechanical part from an airplane engine. Sampling-based motion planners have been shown to solve difficult motion planning problems; see a survey in [Barraquand et al. 1997]. These methods approximate the reachable configuration space (C-space) of a movable object by sampling and connecting random configurations to form a graph (or a tree). However, they also have several technical issues limiting their success on some important types of problems, such as the difficulty of finding paths that are required to pass through narrow passages.

ACD can address the so called "narrow passage" problem for some problems by sampling with a bias toward cuts between the ACD components of a workspace (for rigid or articulated robots). Figure 3 illustrates the advantage of this sampling strategy over uniform sampling [Kavraki et al. 1996]. Advantages of the ACD-based sampling are that more samples are placed in the narrower (difficult) regions and also the connections between the samples can be made more easily due to the nearly convex components.

Mesh generation (solid ACD with feature grouping). The ACD components can be used to generate tetrahedral meshes from the convex hulls of the ACD components using Delaunay triangulation. The convex hulls may further simplified, e.g., using triboxes [Crosnier and Rossignac 1999], to generate even coarser meshes. These meshes can later be used for, e.g., surface deformation. An illustration of this application is shown in Figure 4.

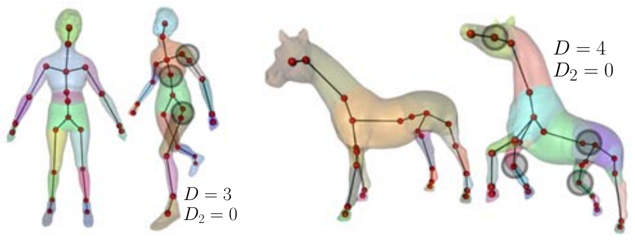


Figure 21: Skeletons extracted from the ACD components of two models and their deformations. D is the graph edit distance from the skeleton of the deformed model to that of the original model. D_2 is D without considering degree 2 vertices whose insertion and deletion do not change the topology of the graph.

8 Discussion and Future Work

We have presented a framework for decomposing a given polyhedron of arbitrary genus into *nearly convex* components. This provides a mechanism by which significant features are removed and insignificant features can be allowed to remain in the final approximate convex decomposition (ACD).

Despite our promising results, our current implementation has some limitations which we plan to address in future work, some of which can be solved without too much difficulty. For example, some uncommon types of open surfaces with non-zero genus, whose vertices on the convex hull are all convex, cannot be handled correctly by the proposed method. Also, splitting non-linearly separable features using a best fit cut plane can still generate a visually unpleasant decomposition. One possible way to address this problem is to use *curved* cut surfaces.

There are other issues that require further research. For example, our feature grouping method has difficulty in collecting long features that have relatively low concavity. One possible approach to address this issue is to adaptively select the knot identification threshold δ for each pocket. Another issue is the accuracy of the concavity measure. One possible efficient alternative to computing shortest paths, which as previously mentioned is NP-hard, is to use an adaptively sampled distance field [Frisken et al. 2000].

References

BAJAJ, C., AND DEY, T. K. 1992. Convex decomposition of polyhedra and robustness. *SIAM J. Comput. 21*, 339–364.

BARRAQUAND, J., KAVRAKI, L. E., LATOMBE, J.-C., LI, T.-Y., MOTWANI, R., AND RAGHAVAN, P. 1997. A random sampling scheme for path planning. *Int. J. of Rob. Res 16*, 6, 759–774.

BUNKE, H., AND KANDEL, A. 2000. Mean and maximum common subgraph of two graphs. *Pattern Recogn. Lett. 21*, 2, 163–168.

CHAZELLE, B., AND PALIOS, L. 1994. Decomposition algorithms in geometry. In *Algebraic Geometry and its Applications*, C. Bajaj, Ed. Springer-Verlag, ch. 27, 419–447.

CHAZELLE, B., DOBKIN, D. P., SHOURABOURA, N., AND TAL, A. 1995. Strategies for polyhedral surface decomposition: An experimental study. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 297–305.

CHAZELLE, B. 1981. Convex decompositions of polyhedra. In *Proc. 13th Annu. ACM Sympos. Theory Comput.*, 70–79.

CHOI, J., SELLEN, J., AND YAP, C. K. 1997. Approximate Euclidean shortest paths in 3-space. *Internat. J. Comput. Geom. Appl. 7*, 4 (Aug.), 271–295.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3, 905–914.

CROSNIER, A., AND ROSSIGNAC, J. 1999. Tribox-based simplification of three-dimensional objects. *Computers&Graphics 23*, 3, 429–438.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3, 848–855.

DEY, T. K., GIESEN, J., AND GOSWAMI, S. 2003. Shape segmentation and matching with flow discretization. In *Proc. Workshop on Algorithms and Data Structures*, 25–36.

ERICKSON, J., AND HAR-PELED, S. 2002. Optimally cutting a surface into a disk. In *Proceedings of the eighteenth annual symposium on Computational geometry*, ACM Press, 244–253.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proc. ACM SIGGRAPH*, 249–254.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graph. 23*, 3, 652–663.

GOSWAMI, S., DEY, T. K., AND BAJAJ, C. L. 2006. Identifying flat and tubular regions of a shape by unstable manifolds. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, ACM Press, New York, NY, USA, 27–37.

HERSHBERGER, J., AND SNOEYINK, J. 1992. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Internat. Sympos. Spatial Data Handling*, 134–143.

HUBELI, A., AND GROSS, M. 2001. Multiresolution feature extraction for unstructured meshes. In *Proceedings of the conference on Visualization '01*, 287–294.

JOE, B. 1994. Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *International Journal for Numerical Methods in Engineering 37*, 693–713.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3, 954–961.

KAVRAKI, L. E., SVESTKA, P., LATOMBE, J. C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat. 12*, 4 (August), 566–580.

KEIL, J. M. 2000. Polygon decomposition. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 491–518.

KENT, J. R., CARLSON, W. E., AND PARENT, R. E. 1992. Shape transformation for polyhedral objects. *SIGGRAPH Comput. Graph. 26*, 2, 47–54.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph. 22*, 3, 350–357.

LAI, Y.-K., ZHOU, Q.-Y., HU, S.-M., AND MARTIN, R. R. 2006. Feature sensitive mesh segmentation. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, ACM Press, New York, NY, USA, 17–25.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2005. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5, 444–465.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 362–371.

LI, X., TOON, T. W., AND HUANG, Z. 2001. Decomposing polygon meshes for interactive applications. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 35–42.

LIEN, J.-M., AND AMATO, N. M. 2004. Approximate convex decomposition of polygons. In *Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG)*, 17–26.

LIEN, J.-M., KEYSER, J., AND AMATO, N. M. 2006. Simultaneous shape decomposition and skeletonization. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, ACM Press, New York, NY, USA, 219–228.

LIU, S., MARTIN, R. R., LANGBEIN, F. C., AND ROSIN, P. L. 2006. Segmenting reliefs on triangle meshes. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, ACM Press, New York, NY, USA, 7–16.

MANGAN, A. P., AND WHITAKER, R. T. 1999. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4, 308–321.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3, 609–612.

PAULY, M., KEISER, R., AND GROSS, M. 2003. Multi-scale feature extraction on point-sampled surfaces. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 281–289.

PRAUN, E., AND HOPPE, H. 2003. Spherical parametrization and remeshing. *ACM Trans. Graph. 22*, 3, 340–349.

ROM, H., AND MEDIONI, G. 1994. Part decomposition and description of 3d shapes. In *Proc. International Conference of Pattern Recognition*, 629–632.

RUSINKIEWICZ, S. 2004. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*.

SHAPIRO, A., AND TAL, A. 1998. Polyhedron realization for shape transformation. *The Visual Computer 14*, 8/9, 429–444.

SHARIR, M., AND SCHORR, A. 1986. On shortest paths in polyhedral spaces. *SIAM J. Comput. 15*, 193–215.

SKLANSKY, J. 1972. Measuring concavity on rectangular mosaic. *IEEE Trans. Comput. C-21*, 1355–1364.

WHITE, E. R. 1985. Assessment of line-generalization algorithms using characteristic points. *The American Cartographer 12*, 1, 17–27.

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2004. Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2, 190–208.

WU, K., AND LEVINE, M. D. 1994. Recovering parametric geons from multiview range data. In *Proc. International Conference of Pattern Recognition*, 159–166.

WU, K., AND LEVINE, M. D. 1997. 3d part segmentation using simulated electrical charge distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 11, 1223–1235.

YAMAUCHI, H., LEE, S., LEE, Y., OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2005. Feature sensitive mesh segmentation with mean shift. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI' 05)*, IEEE Computer Society, Washington, DC, USA, 238—245.

YOSHIZAWA, S., BELYAEV, A., AND SEIDEL, H.-P. 2005. Fast and robust detection of crest lines on meshes. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, ACM Press, New York, NY, USA, 227–232.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2003. Feature-based surface parameterization and texture mapping. Git-gvu-03-29, Georgia Institute Technology.

ZUNIC, J., AND ROSIN, P. L. 2002. A convexity measurement for polygons. In *British Machine Vision Conference*, 173–182.

A Bridge Size Estimation

Algorithm 2 estimates the number of the required bridges so that the error of the approximated concavity is less than ϵ. Note that $C(\beta)$ in Algorithm 2 is a set of contiguous facets adjacent to β. The distance from the facets in $C(\beta)$ to the plane tangent to β is at most ϵ.

Algorithm 2 bridge_size_estimation(CH_P, ϵ)

```
Input. A convex hull $CH_P$ and a threshold $\epsilon$
Output. The number of bridges that can cover $\partial CH_P$
1: Let $B$ and $K$ be two empty sets
2: repeat
3:     Let $\beta$ be a facet of $\partial CH_P$ that is not in $K$
4:     $B = B \cup \beta$
5:     $K = K \cup C(\beta)$  ▷ $C(\beta)$ are facets that can be assigned to $\beta$
6: until $K = \partial CH_P$
7: return the size of $B$
```