Approximate Convex Decomposition of Polyhedra

Jyh-Ming Lien Nancy M. Amato
{neilien,amato}@cs.tamu.edu
Parasol Lab., Department of Computer Science
Texas A&M University

Technical Report TR06-002
Parasol Lab.
Department of Computer Science
Texas A&M University
January, 2006

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.

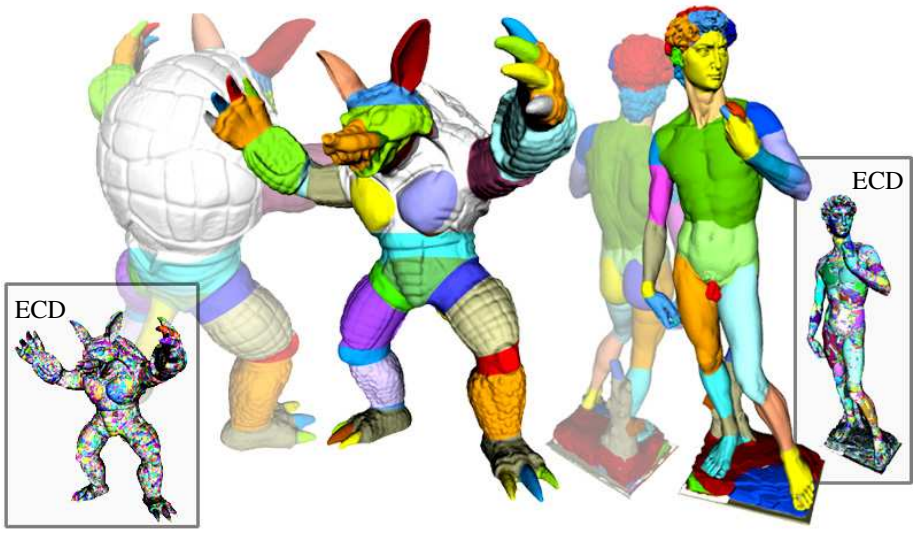


Figure 1: The approximate convex decompositions (ACD) of the Armadillo and the David models consist of a small number of nearly convex components that characterize the important features of the models better than the exact convex decompositions (ECD) that have orders of magnitude more components. The Armadillo (500K edges, 12.1MB) has a solid ACD with 98 components (14.2MB) that was computed in 232 seconds while the solid "ECD" has more than 726,240 components (20+ GB) and could not be completed because disk space was exhausted after nearly 4 hours of computation. The David (750K edges, 18MB) has a surface ACD with 66 components (18.1MB) while the surface ECD has 85,132 components (20.1MB).

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 [6] 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 [17] 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 [5].

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 and a surface ECD of the David model has 85,132 components (see Figure 1). Similar statistics for additional models are show in Table 1 in Section 7.

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 [21]. 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

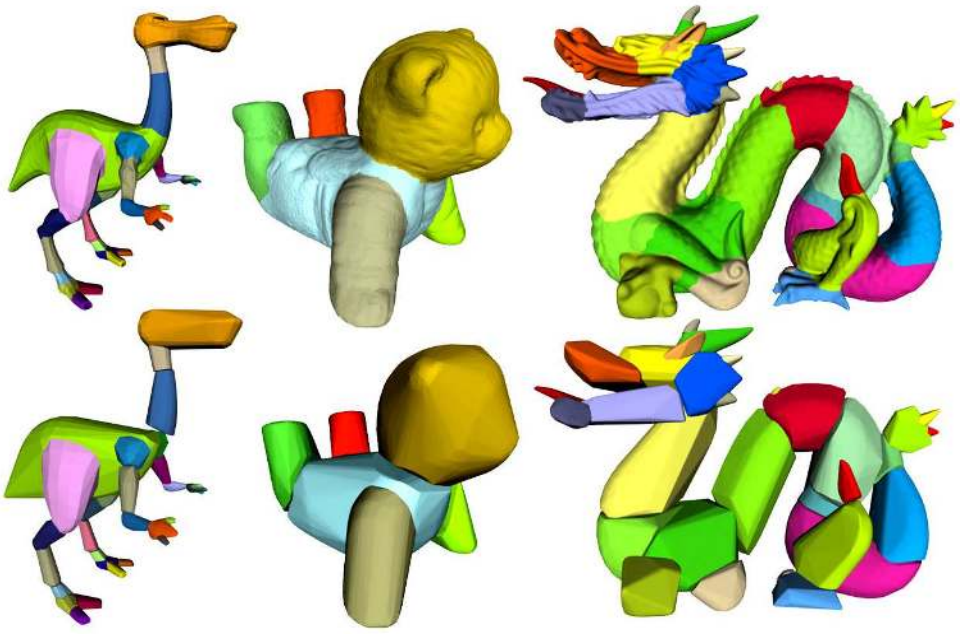


Figure 2: Examples of shape decomposition using ACD. The convex hulls of the components of the decomposition are also shown.

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 features effi ciently and resolving concave features to generate a 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 effi cient and produces better results.

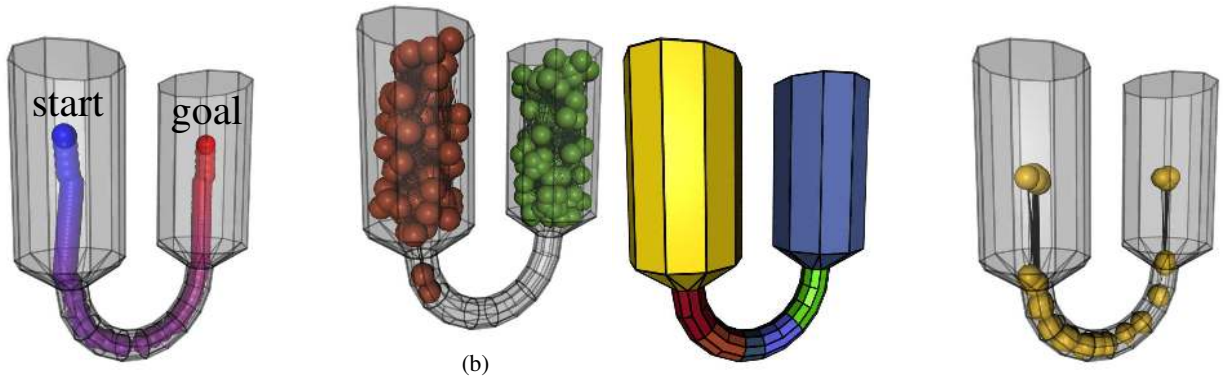
We demonstrate the feasibility of our approach by applying it to a number of complex models; see Figure 1 and Table 1. In general, even for very complex models, the ACDs have very few components, typically several orders of magnitude fewer than the ECDs. The size (memory) and computational time are also signifi cantly less, particularly for the solid ACDs.

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 ACDs of the Armadillo and the David models in Figure 1 identify anatomical features much better than the ECDs. Other applications of ACD include shape representation (Figure 2), motion planning (Figure 3), mesh generation (Figure 4), and point location (Figure 5).

Outline. We begin with preliminary defi nitions in Section 2. In Section 3, we give an overview of ACDand then describe several challenges of extending the polygonal approach to three-dimensions. We then describe ACD for genus zero polyhedra (Section 4) and for polyhedra of arbitrary genus (Section 5). We present applications and results in Sections 6 and 7, respectively.

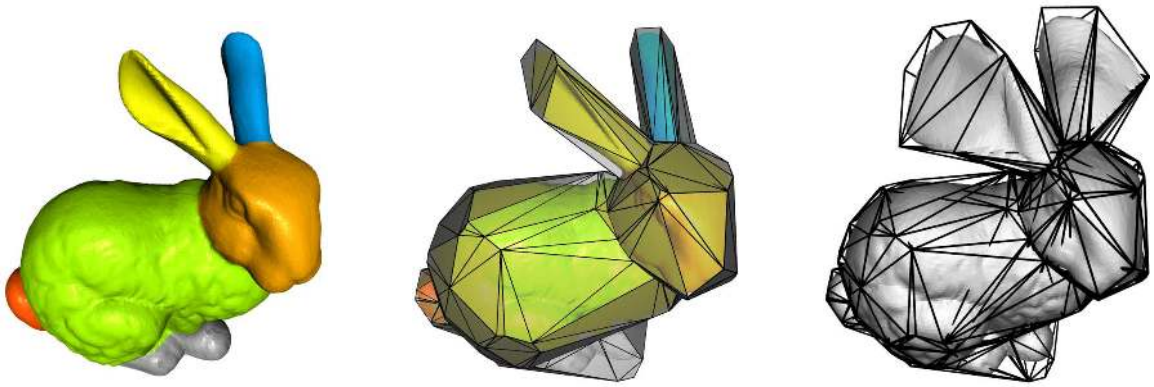
2 Preliminaries

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, H_P, is the smallest convex set enclosing P. P is said to be *convex* if $P = H_P$. Features of P (vertices in $\mathbb{R}^2$ and edges in $\mathbb{R}^3$) are *notches* (non-convex



(a) (b) (c) (d)

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 6 for detail.



(ACD) (tetrahedral mesh) (deformation)

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

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:

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

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 model's surface 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., $P_i \subset \partial H_{P_i}$ [5]. A *convex surface decomposition* of P is a decomposition of ∂P containing only convex surface components.

Concavity. In contrast to measures like area and volume, concavity does not have a well accepted defi nition. A few methods have been proposed that attempt to defi ne and measure the concavity of polygons [25, 21]. 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 defi ne the concavity of a model P as the maximum concavity of its boundary points, i.e.,

$$\mathrm{concave}(P) = \max_{x \in \partial P} \{\mathrm{concave}(x)\} \ ,$$

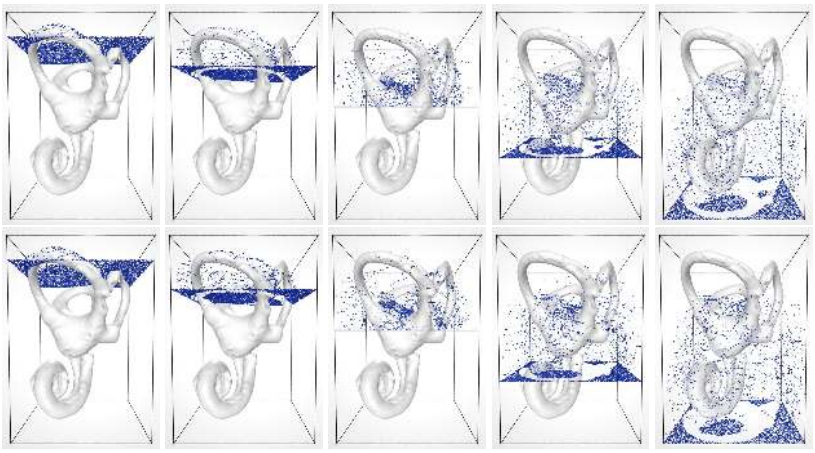


Figure 5: Snap shots of particle system with 10000 particles using the full model and convex hulls of ACD components. Which simulation is generated with ACD? Here, using the ACD instead of the full model is 2 times faster and does not introduce noticeable errors. See 'Point location' in Section 6 for details. (The lower row uses ACD.)

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 [28], where the convexity of a model P is defined as $\frac{\text{volume}(P)}{\text{volume}(H_P)}$.

Measuring Concavity. Intuitively, one can think of the concavity as the length of the path traveled by a point $x \in \partial P$ during the process of inflating a balloon starting from P's shape until it assumes the shape of H_P. Although a physically based simulation of this *balloon expansion* [18] 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., $\text{BRIDGES}(P) = \partial H_P \setminus \partial P$. Pockets are the portion of the boundary ∂P that is not on the convex hull boundary ∂H_P, i.e., $\text{POCKETS}(P) = \partial P \setminus \partial H_P$.

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., $\text{concave}(x) = \text{dist}(x, H_P) = \text{dist}(x, \beta_x)$.

For polygons, there is a natural one-to-one bridge/pocket matching that can be obtained easily. Also, in this case, Lien and Amato [21] 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.

Unfortunately, the techniques used for polygons do not extend easily to three-dimensions. In particular, there is no trivial one-to-one bridge/pocket matching and so we must define one and develop methods for computing it. In addition, while SL-concavity can still be computed efficiently, the best known methods for computing shortest paths on polyhedra require exponential time [24] and even methods [7] that approximate the shortest paths are too inefficient to be used in our approach.

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 $\text{concave}(P) < \tau$, where $\text{concave}(\rho)$ 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.,

$$\text{ACD}_\tau(P) = \{P_i \mid P_i \in \text{D}(P) \text{ and } \text{concave}(P_i) \leq \tau\}. \tag{2}$$

Thus, an ACD_0 is simply an exact convex decomposition.

Our general strategy for computing ACDs follows the approach for polygons. Briefly, 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)

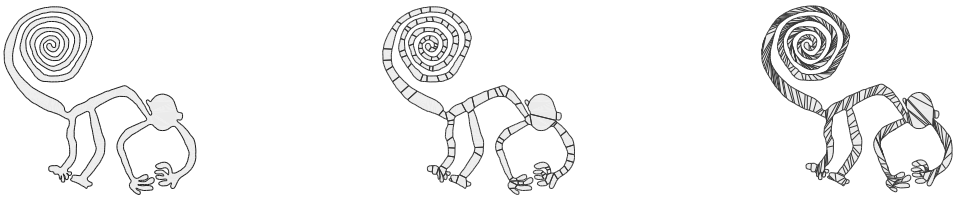
Input. A model, P, and tolerance, τ.
Output. A decomposition, $\{P_i\}$, such that $\max\{\text{concave}(P_i)\} \leq \tau$.

```
1: if concave(P) < τ then                          ▷ see Sections 4.1 and 5
2:     return P
3: else
4:     Let x be a feature (notch) realizing concave(P)
5:     {P_i} = resolve(P, x)                       ▷ see Sections 4.2 and 4.2
6:     for each component {P_i} do
7:         ACD(P_i, τ)
```

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

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

The approach outlined above is the same strategy applied to compute ACDs for polygons [21]. For a given polygon P, the concavity of notches x of the polygon P are computed using SL- or SP-concavity described in Section 2. Then, a notch x is resolved by adding a diagonal from x to ∂P such that the dihedral angle of x is less than 180°. Figure 6 shows an ACD of a polygon.



(a) (b) (c)

Figure 6: (a) The initial Nazca monkey has 1,204 vertices and 577 notches. (b) An ACD has 126 components with SL-concavity less than 0.5, and (c) A minimum convex decomposition has 340 convex components.

3.1 Challenges in Extending to Three Dimensions

In this section, we discuss the challenges of measuring and resolving concave features of polyhedra.

Measuring concave features. The relationship between pockets and bridges is no longer available directly for polyhedra. The problem of obtaining the bridge/pocket relationship is closely related to the problem of spherical [22] and simplical [19] parameterization. However, mesh parameterization is costly to compute. Polyhedron realization [23] 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.

Resolving concave features. The notch-cutting strategy [4] 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 [1]. Figures 7(a)(b) illustrate an ACD using cut planes that bisect dihedral angles.

A difficulty of this approach is selecting "good" cut planes. For example, in Figure 7(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 cut planes that produce *fewer* and/or *structurally more meaningful* components are not well known. Joe [14] proposed an approach to postpone processing notches whose resolution would produce small components, but it has problems with large models.

Our solution: feature grouping. Just as ACD provides an approximation that is more practical than ECD, we will address the challenges mentioned above using approximations that are more tractable, and in some cases, also provide more meaningful solutions. In particular, 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. Feature grouping is both more efficient and can improve solution quality.

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. As we will see in Section 4.1, bridge patches can be used to provide a conservative measure of concavity.

Resolution of concavity can also be improved by considering feature sets rather than individual features when determining cut planes to resolve notches. As discussed in Section 4.2. the quality of the decomposition can be greatly improved when the cut plane is defined with respect to a notch set.

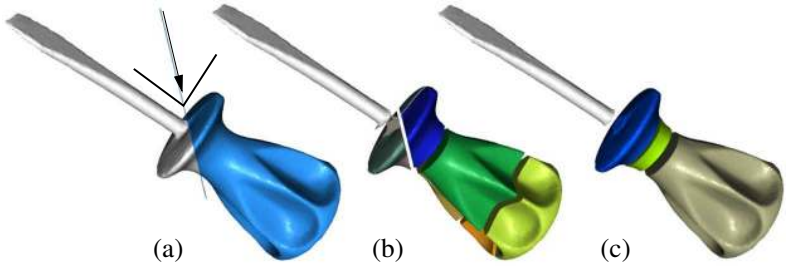


Figure 7: Resolving concavity (a) using a cut plane that bisects a dihedral angle results in (b) a decomposition with 10 components with concavity ≤ 0.1. In contrast, (c) carefully selected cut planes generate only 4 components with concavity ≤ 0.1.

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.

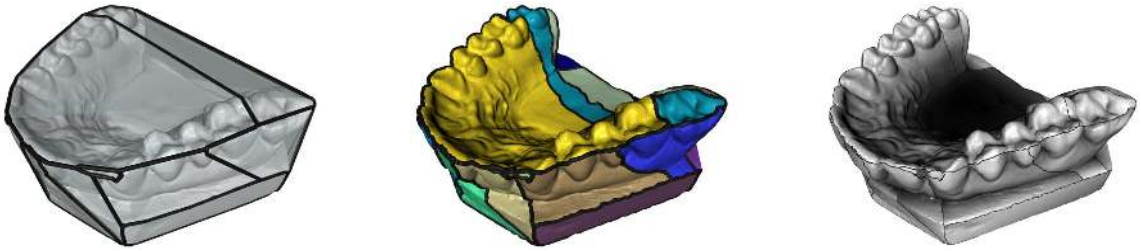
Our startegy to match bridges with pockets is to identify pockets by *projecting* convex hull edges to the polyhedron's surface. The projection of a convex hull edge e is a path on the polyhedron's surface ∂P connecting the end points of e; we compute the paths on ∂P using Dijkstra's algorithm. After 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.

Feature grouping: bridge patches – a conservative estimation. Finding pockets for all facets in ∂H_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 ∂H_P. We then designate a *"supporting" plane* that is tangent to ∂H_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. 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 $\varepsilon = 0.05$, only 20 bridges are identified for the model in Figure 8 which has 4,626 facets on its convex hull.

It is important to note that the estimated concavity measurement computed this way is always greater than (in an amount less than ε) 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.

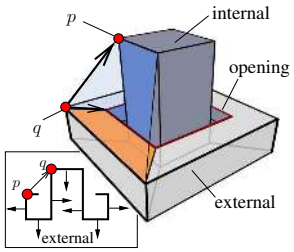
One way to compute bridge patches is from an outer approximation of a polyhedron. Here we use *Lloyd's* clustering algorithm adapted from [8] to identify bridges and to ensure that the maximum distance from the included facets to the supporting plane is less than ε. Details regarding this process are discussed in Appendix A. For all examples in this paper, we set $\varepsilon = \tau/2$.



(bridges) (pockets) (concavity)

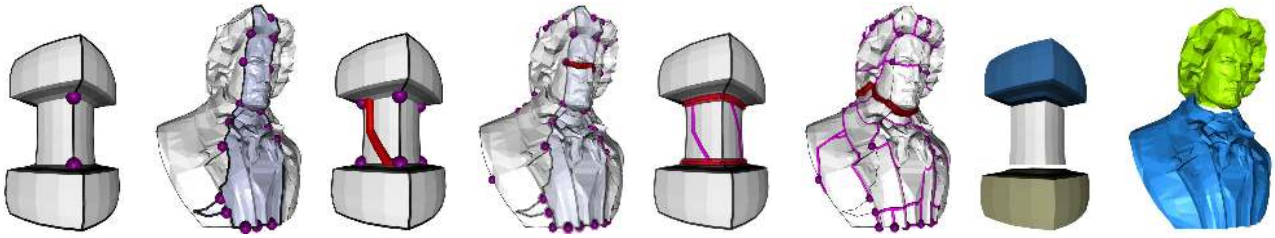
Figure 8: An identified bridge/pocket pair. The rightmost model is shaded so that darker areas indicate higher concavity.

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



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

Figure 9: 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 $G_{\mathscr{K}}$. (d) Solid (left) and surface (right) decompositions using the identified global cuts.

4.2 Resolving Concave Features

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.

Feature grouping: global cuts. Our strategy of grouping concave features is a 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*". This bottom-up approach attempts to (i) avoid high computational complexity, e.g., grouping features based on the solution of a maximum flow problem [15] on the full surface ∂P, (ii) avoid enhancing feature quality [20], and (iii) avoid using other processes, e.g., mesh simplification, to enhance features. Our approach is illustrated in Figure 9 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 [12] with simplification threshold δ, $0 \leq \delta \leq \tau$. A brief review of the DP algorithm is provided in Appendix B.
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 $\omega(\kappa) = |\kappa| / \sum_{v \in \kappa} \mathrm{concave}(v)$ 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* [13].
4. **Connecting pocket cuts into global cuts**. Our strategy is to organize the knots and pocket cuts in a graph $G_{\mathscr{K}}$ whose vertices are knots and edges are pocket cuts. The cycle with the minimum weight in $G_{\mathscr{K}}$ will be the global cut.

Next, we will provide more details and justify the choices of the steps mentioned above.

Pocket boundaries. First, it is natural to ask why the critical points on a projected bridge edge 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. The threshold δ controls the size of knots, i.e., a smaller δ implies more concave features will be identified; in this paper, we used $\delta = \tau/10$. We note that these pocket boundaries have similar functionality as the *exoskeleton* that connects critical points on ∂P coded with *average geodesic distance* [27].

Extracting cycles from graph $G_{\mathscr{K}}$. The process of extracting cycles is similar to that of constructing a minimum spanning tree (MST) $T_{\mathscr{K}}$ on $G_{\mathscr{K}}$ by greedily expanding the most promising branch into all its neighboring pockets in each iteration. A cycle is identified when two growing paths of $T_{\mathscr{K}}$ meet.

Resolving Concave Features. For convex volume decomposition, we define the cut plane of a (global) cut κ as the *best fit plane* of κ which can be approximated via a traditional principal component analysis using points sampled on κ. For convex surface decomposition, we simply split the surface at the edges of κ.

4.3 Complexity Analysis

ACD of a polyhedron P requires $O(n_v n_e \log n_v)$ time for each iteration, 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. However, as seen in our experimental results, this is usually a very conservative estimate. Also, the total number of pocket cuts is usually quite small (see Appendix C).

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 10(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 [10]. Several heuristics for genus reduction have been proposed (see a survey in [27]). The identified handle cuts will then be used to prevent the paths of the bridge projections from crossing them. Figure 10(b) shows an example of a handle cut and the new bridge/pocket relation after genus reduction.

(a) (b)

Figure 10: (a) The pocket (shaded area) is enclosed in the projected boundaries of two bridges β and α. (b) Pockets 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 10 with one hand: our 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. Details of genus reduction using handle caps are described in Appendix D.

Figure 11 shows a result of our approach. Note that we may not always reduce the genus of a model to zero because some handles 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 [26].

6 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 full model and for the convex hulls of the $\text{ACD}_{0.02}$ components; point location in the ACD did not utilize the hierarchical structure of the ACD, but simply tested each component separately. As seen in Figure 12, even using this naive strategy, point location in the ACD is about 23% faster

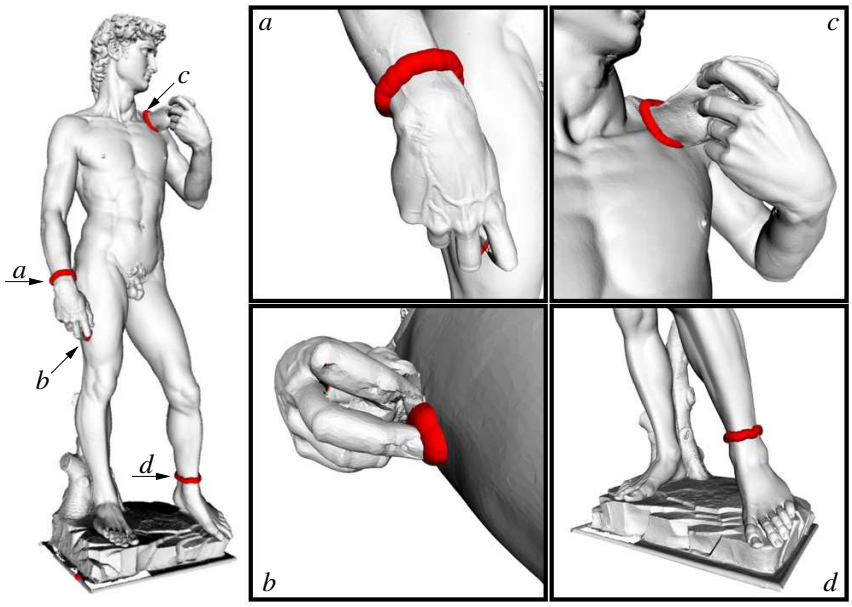


Figure 11: Four handle cuts found in the David model.

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, more than 99% of the queries were answered correctly using the ACD.

Shape representation (surface ACD with feature grouping). The components of an ACD can also be used for shape representation. In many cases the signifi cance of a feature depends on its volumetric proportion to its "base". For example, a 5 cm stick on a ball with 5 cm radius is a more signifi cant feature than a 5 cm stick on a ball with 5 km radius. This intuition can be captured by the concept of *convexity* defi ned as $\frac{\text{volume}(P)}{\text{volume}(H_P)}$. Figure 2 shows results from our approach that simply replaces the decomposition criterion, i.e., concavity (line 1 in Algorithm 1), with 0.7 convexity.

Although there are no well accepted criteria to compare decompositions, we can compare the skeletons extracted from the decompositions, e.g., using graph edit distance [3], 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 13 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 diffi cult motion planning problems; see a survey in [2]. These methods approximate the reachable confi guration space (C-space) of a movable object by sampling and connecting random confi gurations 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 diffi culty of fi nding 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 ACD components. Figure 3 illustrates the advantage of this sampling strategy over uniform sampling [16]. Advantages of the ACD-based sampling are that more samples are placed in the narrower (diffi cult) 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 simplifi ed, e.g., using triboxes [9], 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.

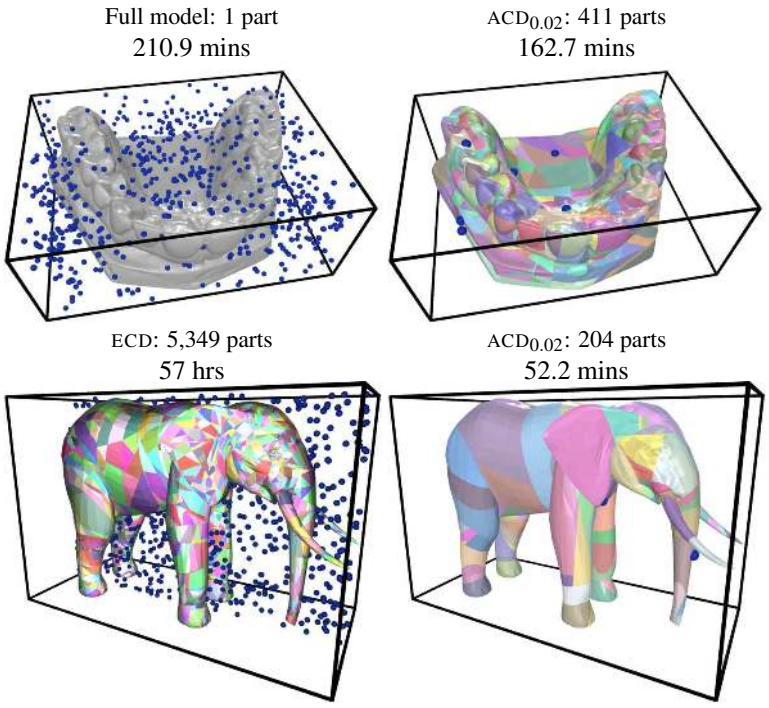


Figure 12: Point location of 10^8 points in the teeth model (233,204 triangles), in the elephant model (6,798 triangles), and in their solid ECD and the convex hulls of the $ACD_{0.02}$. Measured time includes time for decomposition and point location. Point location in $ACD_{0.02}$ of both models has 0.99% errors. External points of 1000 samples in full model and ECD are shown in the figures on the left and only the misclassified (as internal) points in ACDs are shown on the right.

7 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. All experiments were performed on a Pentium 2.0 GHz CPU with 512 MB RAM. A summary of results for 14 models is shown in Table 1 and in Figures 14 and 15.

As seen in Table 1, the solid ACDs are orders of magnitude smaller than solid ECD. The solid $ACDs_{0.2}$ and $ACDs_{0.02}$ 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 $ACDs_{0.2}$ and $ACDs_{0.02}$ 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 14 are collected before termination, i.e., they are for an unfinished ECD, so all components are not yet convex. Figure 14 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 $ACDs_{0.2}$ and $ACDs_{0.02}$ have 0.02% and 0.2% of the number of components that the ECD has on average. Figure 15 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 ACDs v.s. surface ACDs. 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.

ACDs with or without feature grouping. Figures 14 and 15 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

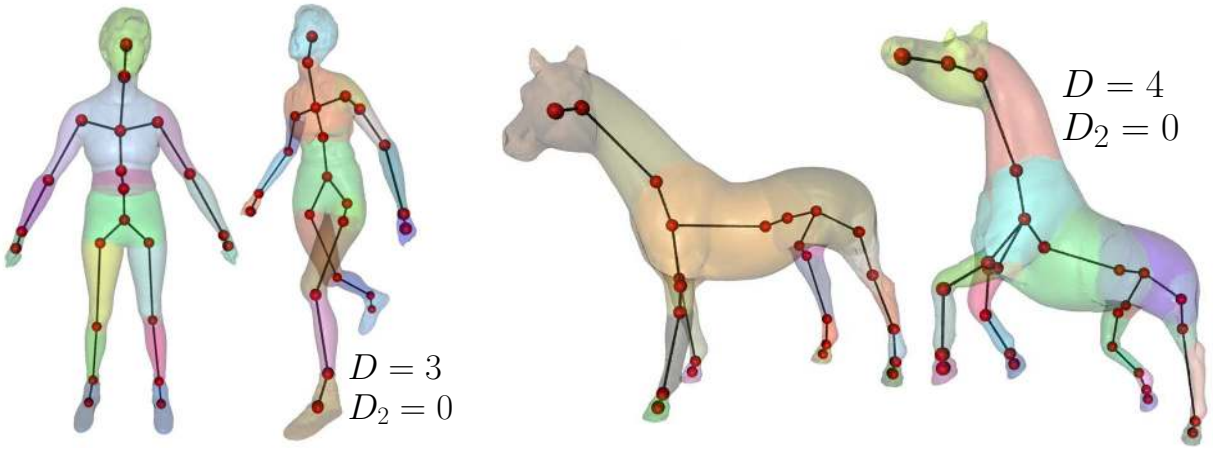


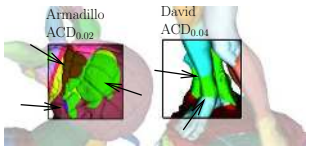
Figure 13: Skeletons extracted from the ACD components of two models and their deformations. D is the graph edit distance from the skeleton of the deformed model to that of the original model. D_2 is D without considering degree 2 vertices whose insertion and deletion do not change the topology of the graph.

feature grouping, ACDs produces structurally more meaningful components.

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). We have also demonstrated that the ACD framework is flexible – by simply changing the decomposition criterion from concavity to convexity, the ACD can be used as a shape descriptor of the input model.

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 "planes". We are also considering efficient alternatives to shortest paths for the concavity measure, such as using an adaptively sampled distance field [11]. Another issue that requires further research is that our feature grouping method has difficulty in collecting long features that have relatively low concavity. See the figure on the right.



Finally, several methods developed in this paper, such as the bridge/pocket identification, feature extraction, and genus reduction, may have application to other problems in computer graphics. How these tools can be applied to other areas requires more research.

References

[1] C. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21:339–364, 1992.

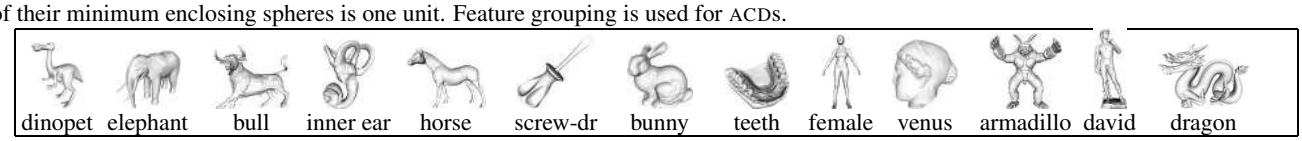
[2] J. Barraquand, L. K. J. Latombe, T. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Rob. Res*, 16(6):759–774, 1997.

[3] H. Bunke and A. Kandel. Mean and maximum common subgraph of two graphs. *Pattern Recogn. Lett.*, 21(2):163–168, 2000.

[4] B. Chazelle. Convex decompositions of polyhedra. In *Proc. 13th Annu. ACM Sympos. Theory Comput.*, pages 70–79, 1981.

[5] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 297–305, 1995.

Table 1: Decompositions of 14 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.



	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

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

[6] B. Chazelle and L. Palios. Decomposition algorithms in geometry. In C. Bajaj, editor, *Algebraic Geometry and its Applications*, chapter 27, pages 419–447. Springer-Verlag, 1994.

[7] J. Choi, J. Sellen, and C. K. Yap. Approximate Euclidean shortest paths in 3-space. *Internat. J. Comput. Geom. Appl.*, 7(4):271–295, Aug. 1997.

[8] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004.

[9] A. Crosnier and J. Rossignac. Tribox-based simplifi cation of three-dimensional objects. *Computers&Graphics*, 23(3):429–438, 1999.

[10] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 244–253. ACM Press, 2002.

[11] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fi elds: a general representation of shape for computer graphics. In *Proc. ACM SIGGRAPH*, pages 249–254, 2000.

[12] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplifi cation algorithm. In *Proc. 5th Internat. Sympos. Spatial Data Handling*, pages 134–143, 1992.

[13] A. Hubeli and M. Gross. Multiresolution feature extraction for unstructured meshes. In *Proceedings of the conference on Visualization '01*, pages 287–294, 2001.

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

[15] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.

[16] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional confi guration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[17] J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[18] J. R. Kent, W. E. Carlson, and R. E. Parent. Shape transformation for polyhedral objects. *SIGGRAPH Comput. Graph.*, 26(2):47–54, 1992.

[19] A. Khodakovsky, N. Litke, and P. Schröder. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.*, 22(3):350–357, 2003.

[20] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371, 2002.

[21] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polygons. In *Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG)*, pages 17–26, June 2004.

[22] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 22(3):340–349, 2003.

[23] A. Shapiro and A. Tal. Polyhedron realization for shape transformation. *The Visual Computer*, 14(8/9):429–444, 1998.

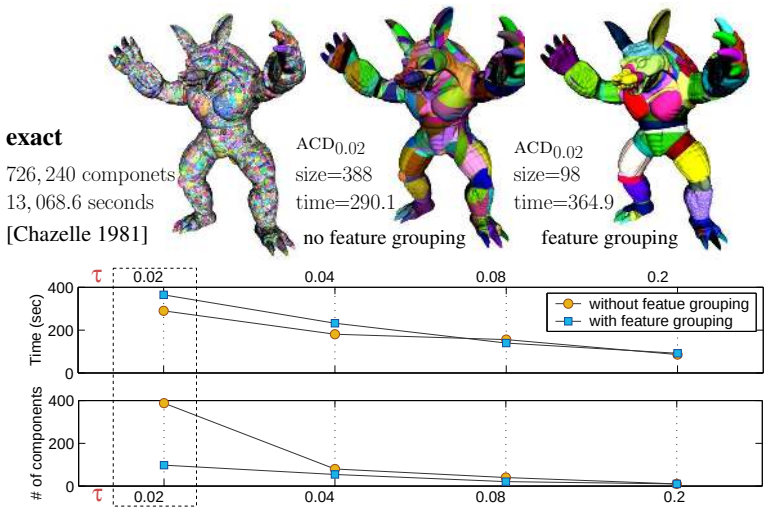


Figure 14: Convex solid decomposition. The size and time of ACD with and without feature grouping are shown for a range approximation values τ.

[24] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986.

[25] J. Sklansky. Measuring concavity on rectangular mosaic. *IEEE Trans. Comput.*, C-21:1355–1364, 1972.

[26] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Removing excess topology from isosurfaces. *ACM Trans. Graph.*, 23(2):190–208, 2004.

[27] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. Git-gvu-03-29, Georgia Institute Technology, 2003.

[28] J. Zunic and P. L. Rosin. A convexity measurement for polygons. In *British Machine Vision Conference*, pages 173–182, 2002.

A Convex Facet Clustering

As described in Section 4.1, we cluster 'nearly' coplanar and contiguous convex hull facets to form bridge patches. The bridge patches are 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. 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.

The second step can be solved using *Lloyd's* clustering algorithm introduced in [8].

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. Let $C(\beta)$ be a set of connected facets that can be assigned to the bridge β. Our estimation process is outlined in Algorithm 2.

B Douglas-Peucker Algorithm

The Douglas-Peucker (DP) line approximation algorithm is used to identify *knots* on a pocket boundary as mentioned in Section 4.2. Let L be a polygonal chain composed of n vertices $\{v_1, v_2, \cdots, v_n\}$. For a given threshold δ, the DP algorithm produces a simplification of L, called L'. Algorithm 3 outlines a simple version of the algorithm. A more efficient approach can be found in [12].

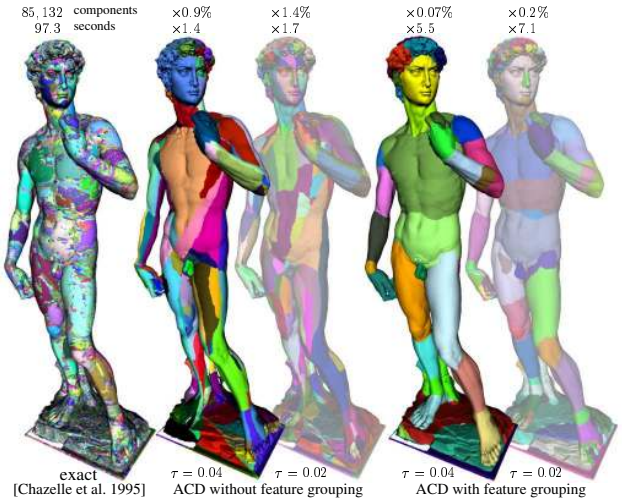


Figure 15: Convex surface decomposition. The leftmost figure shows a result of the exact decomposition. The others are results of the approximate decomposition.

Algorithm 2 H_P_clustering(H_P,ε)

Input. A convex hull H_P and a threshold ε
Output. The number of bridges that can cover ∂H_P

```
Let B and K be two empty sets
repeat
    Let β be a facet of ∂H_P that is not in K
    B = B ∪ β
    K = K ∪ C(β)                    ▷ C(β) are facets that can be assigned to β
until K = ∂H_P
return the size of B
```

C Pocket Cut Reduction

A pocket ρ with $|n_\rho|$ knots can have $O(|n_\rho|^2)$ pockets cuts but not all of them are interesting to us. In fact, we only need to consider $O(|n_\rho|)$ pockets cuts. This reduction is based on the following observation.

Observation C.1 *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. Therefore, we disallow a knot to connect to more than one knot from the same boundary but it is allowed to connect to knots from boundaries of different neighboring pockets. The result of this restriction is that the pocket cuts between two boundaries form a bipartite matching of their knots and only $O(|N_\rho|)$ pocket cuts need to be considered when connecting them into global cuts.

D Convex-hull-based Genus Reduction

We introduced a genus reduction method in Section 5, which computes handle cuts from a set of bridges (handle caps) that share a common pocket. This intuition can be implemented by applying the following to identifi ed 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.

Algorithm 3 DP(L,δ)

Input. A polygonal chain, $L = \{v_1, v_2, \cdots, v_n\}$, and threshold, δ.
Output. A simplified polygonal chain L'.

1: Let $v_k \in L$ be the vertex whose distance d_k to the line $\overline{v_1 v_n}$ is larger than all the other vertices in P
2: **if** $d_k > \delta$ **then**
3: return $L' = \{$ DP($\{v_1, \cdots, v_k\}$,δ), v_k, DP($\{v_k, \cdots, v_n\}$,δ) $\}$

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.