# APPROXIMATE CONVEX DECOMPOSITION AND ITS APPLICATIONS

A Dissertation

by

JYH-MING LIEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2006

Major Subject: Computer Science

APPROXIMATE CONVEX DECOMPOSITION AND ITS APPLICATIONS

A Dissertation

by

JYH-MING LIEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

| | |
|---|---|
| Chair of Committee, | Nancy M. Amato |
| Committee Members, | Ergun Akleman |
| | Ricardo Gutierrez-Osuna |
| | Donald H. House |
| | John C. Keyser |
| Head of Department, | Valerie E. Taylor |

December 2006

Major Subject: Computer Science

# ABSTRACT

Approximate Convex Decomposition and Its Applications. (December 2006)

Jyh-Ming Lien, B.S., National ChengChi University

Chair of Advisory Committee: Dr. Nancy M. Amato

Geometric computations are essential in many real-world problems. One important issue in geometric computations is that the geometric models in these problems can be so large that computations on them have infeasible storage or computation time requirements. Decomposition is a technique commonly used to partition complex models into simpler components. Whereas 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 work, we have developed an approximate technique, called Approximate Convex Decomposition (ACD), which decomposes a given polygon or polyhedron 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 ACD can represent the important structural features of the model more accurately by providing a mechanism for ignoring less significant features, such as wrinkles and surface texture. Our study of a wide range of applications shows that in addition to providing computational efficiency, ACD also provides natural multi-resolution or hierarchical representations. In this dissertation, we provide some examples of ACD's many potential applications, such as particle simulation, mesh generation, motion planning, and skeleton extraction.

To my parents

# ACKNOWLEDGMENTS

This work could not be accomplished without support from many people.

I would like to thank my advisor, Nancy Amato, for her guidance. Throughout my doctoral work she encouraged me to pursue my own research interests and to develop independent thinking and research skills.

I would like to thank my undergraduate advisor, Tsai-Yen Li, for teaching me about research.

I would like to thank my committee members, John Keyser, Donald House, Ergun Akleman, and Ricardo Gutierrez-Osuna, who supported me through this challenging journey.

I would like to thank everyone in the Algorithms & Applications Group in Parasol lab, in particular, Burchan Bayazit, Marco Morales Aguirre, Roger Pearce, Sam Rodriguez, Xinyu Tang, Shawna Thomas, Aimée Vargas, and Dawen Xie, for being my collaborators and my best friends for the past 6 years.

I would like to thank my wife, Shao-Jung Chang. Without her, I would not even know about Texas A&M and helped me to make up my mind to pursue my Ph.D. degree.

Finally, I would like to thank to my parents and my sisters for always being there.

I consider myself very lucky to have so many great people in my life.

# TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

# LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

# CHAPTER I

## INTRODUCTION

Shape is the essence of many geometric problems. One common strategy for dealing with large, complex shapes is to decompose them into components that are easier to process. Many different decomposition methods have been proposed – see, e.g., Chazelle and Palios [26] 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, mesh generation, pattern recognition [48], Minkowski sum computation [1], motion planning [57], skeletonization [89], and origami folding [44], perform more efficiently on convex objects.

Convex decomposition of polygons is a well studied problem and has optimal solutions under different criteria; see [70] 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 [33].

A major reason that convex decompositions 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. For example, while a minimum set of convex components can be computed efficiently for simple polygons without holes [31, 32, 71], the problem is NP-hard for polygons with holes [90]. This remains true in 3D 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

---

This dissertation follows the style of *IEEE Transactions on Automation Science and Engineering.*

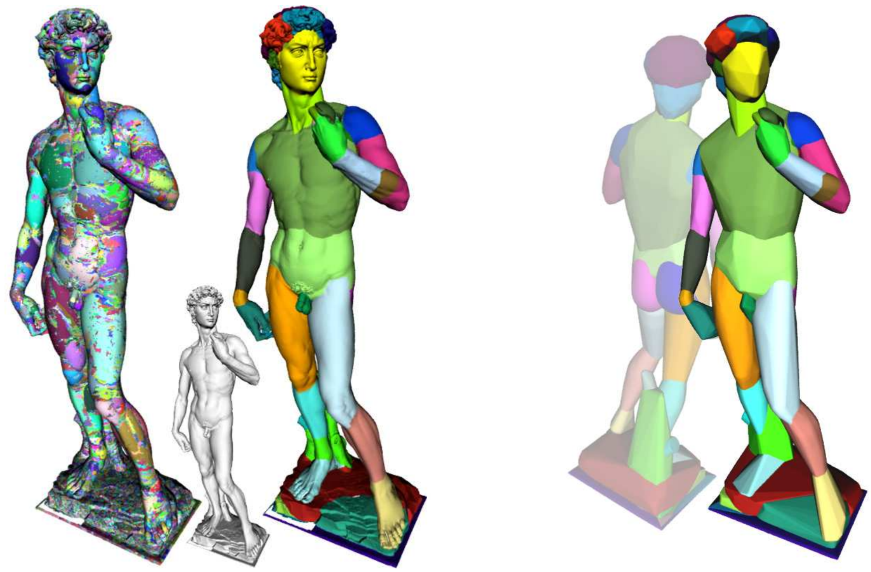
the polyhedron into a collection of convex surface patches. For example, a surface ECD of the David model has 85,132 components (see Figure 1) and a solid ECD of the Armadillo model has more than 726,000 components (see the figure on p. 63). Similar statistics for additional models are shown in the table on p. 87 in Chapter V.

In this research, we propose and 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, such as skeletonization, 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. ACD also simultaneously allows multi-resolution or hierarchical representations. The best way to illustrate ACD and its applications is through the graphics and animations that can be found at: *http://parasol.tamu.edu/∼neilien*

## A. Approximate Convex Decomposition (ACD)

Convex decomposition can be useful because many problems can be solved more efficiently for convex objects. However, generating convex decompositions can be time consuming (sometimes intractable) and can result in unmanageably large decompositions. To address these issues, we propose a partitioning strategy that decomposes a given 2D or 3D model into approximately convex components, resulting in an *approximate convex decomposition* (ACD) [85, 84, 88, 87]. We compute an ACD of a model recursively until all components in the decomposition have *concavity* less than some specified (tunable) parameter. Examples of ACD are shown in Figure 1.

For many applications, the approximately convex components of our ACD pro-



(a) (b)

Figure 1. (a) An exact convex decomposition (left) and an ACD (right) with convexity less than 0.04 of the David model have 85,132 and 66 components, resp. (b) The convex hulls of the ACD components represent David's shape.

vide similar benefits as convex components, while the resulting decomposition is both significantly smaller and can be computed more efficiently. We have shown both theoretically and experimentally that the ACD of polygons with zero or more holes and polyhedra with arbitrary genus can efficiently produce high quality decompositions. Applications that can benefit from this approach include collision detection [88], penetration depth estimation, mesh generation [106], and motion planning [88].

Another important aspect of an approximate convex decomposition is that it can more accurately represent important *structural features* of the model by providing a mechanism for ignoring less significant features, such surface texture; see Figure 1(b). We have shown that ACD can help applications such as skeletonization [89], perceptually meaningful decomposition [89], and shape deformation [106] to focus on the global shape of the model.

Our work in ACD has attracted a wide range of interest from the academic community and industry. In particular, we have received many requests to use ACD in robot grasping and navigation, Minkowski sum computation, rapid prototyping, and tele-immersion.

## B. Applications of ACD

Decomposition is usually used to provide efficiency for the applications. Convex decomposition provides even more efficiency because many algorithms work better with convex objects. In many applications of convex decomposition, the convex hulls of ACD components (and sometimes the components themselves) can be used by methods that usually operate on convex polygons or polyhedra, making them more efficient.

For example, point location, which is commonly used in particle simulation, checks if a given point is inside or outside of a model. This operation can be done more efficiently if the input model is convex. ACD can help improve the efficiently of point location for non-convex models by replacing each ACD component with its convex hull and then performing the point location using the convex hulls of the ACD components. Since each ACD component is contained in its convex hull, the point location may incorrectly identify some points as internal which they are in fact external to the model. Figure 2 illustrates a result of this ACD-based particle system. In this example, and indeed in many scenarios, the differences in the simulation using the full model and the approximated representation using ACD are barely noticeable.

Another important benefit of ACD is that ACD can capture key structural features. For example, the ACDs of the Armadillo and the David models in the figure on p. 63 identify anatomical features much better than the ECDs. Other applications

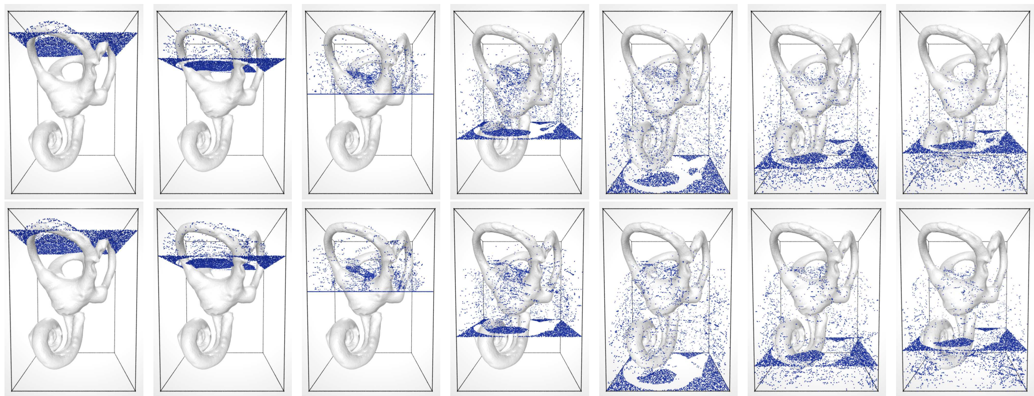


Figure 2. Snap shots of a particle system with 10000 particles using the full model and convex hulls of ACD components. Which simulation is generated with ACD? Here, using the ACD instead of the full model is two times faster and does not introduce noticeable errors. See 'Point location' in Chapter V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

that exploit this property of ACD include shape representation (Figure 3), motion planning (Figure 4), mesh generation (Figure 5).

In shape representation, we ensure that each component of ACD is within some volumetric ratio of its convex hull, e.g., the volumetric ratio between all the ACD components in Figure 3 and their convex hulls is larger than 70%.

In motion planning, we try to find a trajectory for a movable object to move from a start to a goal configuration in an environment without colliding with obstacles. ACD can help to identify narrow regions of the environment which are generally difficult scenarios for the sampling-based motion planners [13]. In Figure 4, we show that, with the same effort, the motion planning problem can be solved with ACD but cannot be solved using uniform sampling. See 'Motion planning' in Chapter VI for details.

ACD can also be used to generate tetrahedral meshes, which are commonly used in simulating physically based systems, e.g., deformation, by further decomposing

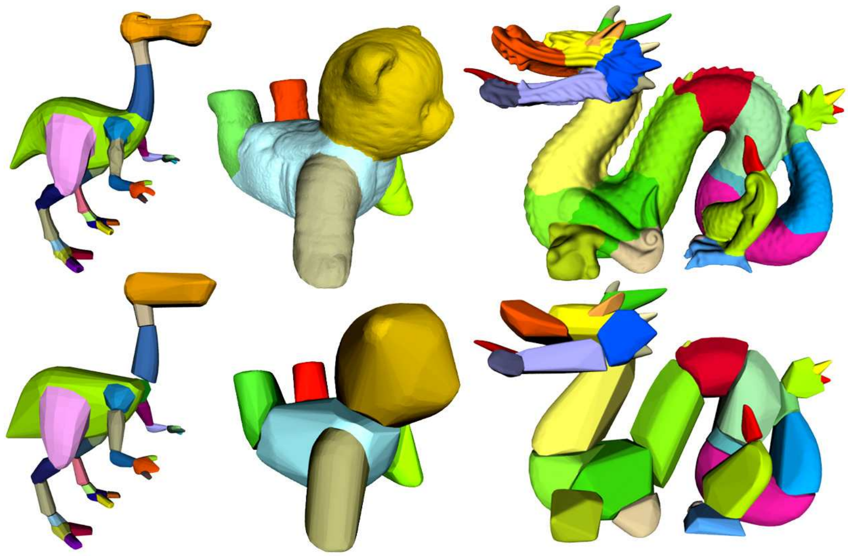


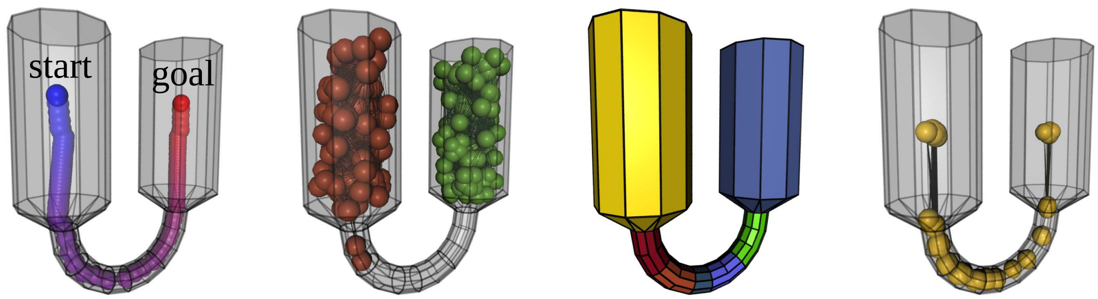
FIGURE 3. Examples of shape decomposition using ACD. The convex hulls of the components of the decomposition are also shown.

the convex hull of each ACD component into tetrahedra. Figure 5 shows a resulting tetrahedral mesh using ACD and a deformation generated using the tetrahedral mesh.

Detailed descriptions of these applications can be found in Chapter VI and Chapter VII.

## C. Outline of the Dissertation

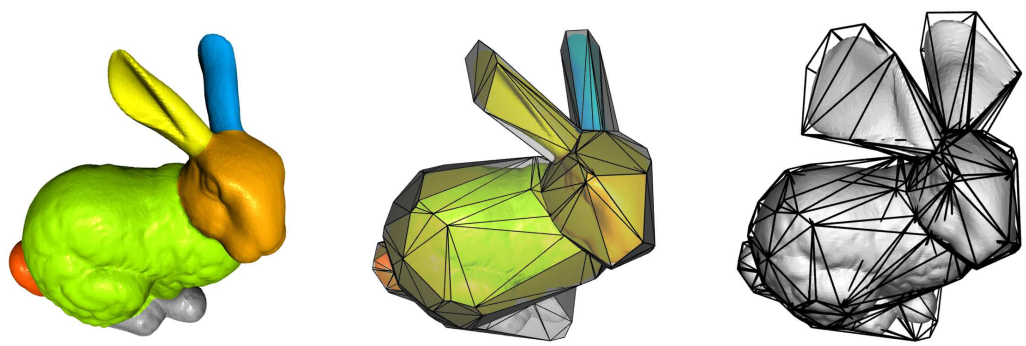
In this dissertation, we introduce a new approximate shape representation technique, Approximate Convex Decomposition (ACD). Definitions and notation used throughout the dissertation and related work on convex decomposition are discussed in Chapter II. A general framework of ACD with a high level discussion of the technique is presented in Chapter III. In Chapters IV and V, we describe techniques for computing ACDs of two-dimensional simple polygons with or without holes and three-dimensional polyhedral solids and surfaces of arbitrary genus, respectively. In both of these two chapters, we provide results illustrating that our approach results in high




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

FIGURE 4. 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).

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. Some representative applications of ACD are presented in Chapters VI and VII.



(ACD) (tetrahedral mesh) (deformation)

Figure 5. A tetrahedral mesh is generated from the (simplified) convex hulls of ACD components. The rightmost figure shows a deformation using this mesh.

# CHAPTER II

# PRELIMINARIES AND RELATED WORK

In this chapter, we first define notation that will be used throughout this dissertation and then we discuss related work on convex decomposition of polygons and polyhedra.

## A. Preliminaries

### 1. Polygons

A polygon $P$ is represented by a set of boundaries

$$\partial P = \{\partial P_0, \partial P_1, \ldots, \partial P_i\} \ ,$$

where $\partial P_0$ is the external boundary and $\partial P_{i>0}$ are boundaries of *holes* of $P$. Each boundary $\partial P_i$ consists of an ordered set of vertices $V_i$ which defines a set of edges $E_i$. Figure 6 shows an example of a simple polygon with nested holes. A polygon is *simple* if no nonadjacent edges intersect. Thus, a simple polygon $P$ with nested holes is the region enclosed in $\partial P_0$ minus the region enclosed in $\cup_{i>0} \partial P_i$. We note that nested polygons can be treated independently. For instance, in Figure 6, the region bounded by $\partial P_0$ and $\partial P_{1 \le i \le 4}$ and the region bounded by $\partial P_5$ can be processed separately.

The *convex hull* of a polygon $P$, $CH_P$, is the smallest convex set containing $P$. $P$ is said to be *convex* if $P = CH_P$. Vertices of $P$ are *notches* (non-convex features) if they have internal angles greater than 180°. A polygon $C$ is a component of $P$ if $C \subset P$. A set of components $\{C_i\}$ is a *decomposition* of $P$ if their union is $P$ and all $C_i$ are interior disjoint, i.e., $\{C_i\}$ must satisfy:

$$\mathrm{D}(P) = \{C_i \mid \cup_i C_i = P \text{ and } \forall_{i \ne j} C_i^\circ \cap C_j^\circ = \emptyset\} \ , \tag{2.1}$$

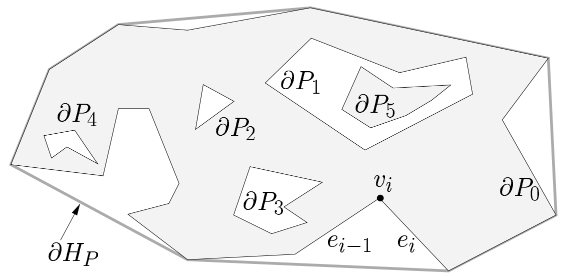



FIGURE 6. A simple polygon with nested holes.

where $C_i^\circ$ is the open set of $C_i$. A *convex decomposition* of $P$ is a decomposition of $P$ that contains only convex components, i.e.,

$$\mathrm{CD}(P) = \{C_i \mid C_i \in \mathrm{D}(P) \text{ and } C_i = CH_{C_i}\}. \tag{2.2}$$

A decomposition of $P$ is said to *resolve* a notch $v$ if $v$ was a notch in $P$ but is not a notch in the decomposition of $P$.

## 2. Polyhedra

Similarly, a polyhedron $P$ is also represented by a set of boundaries $\{\partial P_i\}$. The *convex hull* of a model $P$, $CH_P$, is the smallest convex set enclosing $P$. $P$ is said to be *convex* if $P = CH_P$. Edges of $P$ are *notches* (non-convex features) if they have internal angles greater than 180°. We say $C_i$ is a component of $P$ if $C_i \subset P$. A set of components $\{C_i\}$ is a *decomposition* of $P$ if their union is $P$ and all $C_i$ are interior disjoint, i.e., $\{C_i\}$ must satisfy:

$$\mathrm{D}(P) = \{C_i \mid \cup_i C_i = P \text{ and } \forall_{i \neq j} C_i^\circ \cap C_j^\circ = \emptyset\}, \tag{2.3}$$

where $C_i^\circ$ is the open set of $C_i$. A *convex decomposition* of $P$ is a decomposition of $P$ that contains only convex components; see Eqn. 2.2. Also, decomposition of $P$ is said to *resolve* a notch $e$ if $e$ was a notch in $P$ but is not a notch in the decomposition

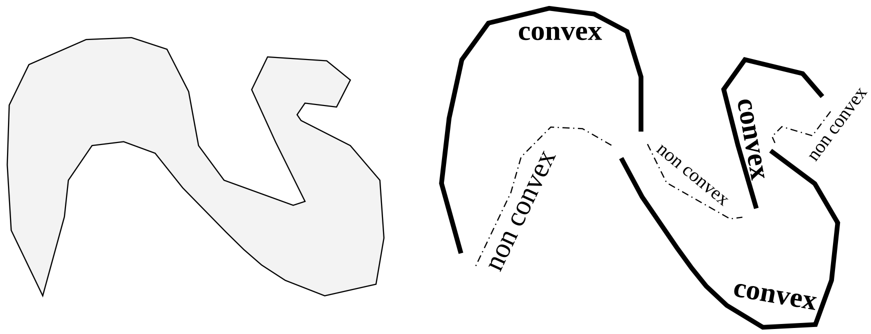



FIGURE 7. A surface patch is convex if it lies entirely on the surface of its convex hull. This figure shows a decomposition of a model into convex and non-convex surface patches.

of $P$.

## 3. Polyhedral Surface

For some applications, such as rendering [12], collision detection [12, 111], and penetration decomposition [74], the model's surface, rather than its solid components, is of most interest. For such applications, it is useful to decompose boundaries of a model into *surface patches*. We say $C$ is a surface patch of $P$ if $C \subset \partial P$. A set of surface patches $\{C_i\}$ is a *surface decomposition* of $P$ if their union is $\partial P$ and all $C_i$ are interior disjoint. A surface patch $C$ is *convex* if $C$ lies entirely on the surface of its convex hull $CH_C$, i.e., $C \subset \partial CH_C$ [33]. An illustration of this definition is shown in Figure 7. A *convex surface decomposition* of $P$ is a decomposition of $\partial P$ that contains only convex surface components.

## 4. Approximately ($\tau$) Convex

The success of our approach depends critically on the accuracy of the methods we use to prioritize the importance of the non-convex features. Intuitively, important features provide key structural information for *the application*. For instance, visu-

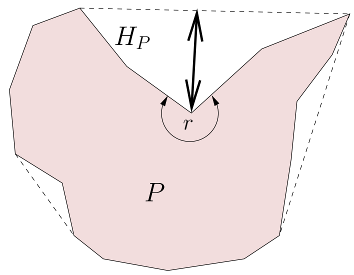



FIGURE 8. Vertex $r$ is a notch and its concavity is measured as the distance to the convex hull $CH_P$.

ally salient features are important for a visualization application, features that have significant impact on simulation results are important for scientific applications, and features representing anatomical structures are important for character animation tools. Although curvature has been one of the most popular tools used to extract visually salient features, it is highly unstable because it identifies features from local variations on the model's boundary. In contrast, the concavity measures we consider here identify features using global properties of the boundary. Figure 8 shows one possible way to measure the concavity of a polygon as the maximal distance from a vertex of $P$ ($r$ in this example) to the boundary of the convex hull of $P$. The intuition is that when the concavity (of a polygon or a polyhedron $P$) obtained using a certain concavity measure is "small enough" to be ignored, then $P$ can be considered to be convex or $P$ can be represented by its convex hull. We formalize this intuition with the following definition of $\tau$-convex, where the parameter $\tau$ is used to control how convex the components in the ACD will be.

**Definition A.1. concavity and $\tau$-convex.** *We say a polygon or a polyhedron $P$ has* concavity$(P) \leq \tau$, *or equivalently that $P$ is $\tau$-convex, if all vertices $v$ of $P$ have* concavity$(v) \leq \tau$, *where* concavity$(\rho)$ *denotes the concavity measurement of $\rho$.*

## B. Related Work on Convex Decomposition

Convex decomposition of polygons is a well studied problem and has optimal solutions under different criteria. In contrast, convex decomposition in three-dimensions is far less understood. In this section, we will review related work on convex decomposition of polygons and polyhedra.

Another set of related work is mesh generation which decomposes a polygon or a polyhedron into triangle, tetrahedral, quadrilateral or hexahedral meshes with an arbitrary number of additional (Steiner) points. Many strategies are proposed to generate meshes. A good survey of these strategies can be found in [101].

### 1. Convex Decomposition of Polygons

Many approaches have been proposed for decomposing polygons; see the survey by Keil [70]. The problem of convex decomposition of a polygon is normally subject to some optimization criteria to produce a minimum number of convex components or to minimize the sum of the length of the boundaries of these components (called minimum ink [70]). Convex decomposition methods can be classified according to the following criteria:

- Input polygon: simple, holes allowed or disallowed.
- Decomposition method: additional (Steiner) points allowed or disallowed.
- Output decomposition properties: minimum number of components, shortest internal length, etc.

For polygons with holes, the problem is NP-hard for both the minimum components criterion [90] and the shortest internal length criterion [69, 91].

When applying the minimum component criterion for polygons without holes, the situation varies depending on whether Steiner points (points in addition to the original vertices) are allowed. When Steiner points are not allowed, Chazelle [28] presents an $O(n \log n)$ time algorithm that produces fewer than $4\frac{1}{3}$ times the optimal number of components, where $n$ is the number of vertices. Later, Green [52] provided an $O(r^2n^2)$ algorithm to generate the minimum number of convex components, where $r$ is the number of notches. Keil [69] improved the running time to $O(r^2 n \log n)$, and more recently Keil and Snoeyink [71] improved the time bound to $O(n + r^2 \min(r^2, n))$. When Steiner points are allowed, Chazelle and Dobkin [32] propose an $O(n + r^3)$ time algorithm that uses a so-called $X_k$-pattern to remove $k$ notches at once without creating any new notches. An $X_k$-pattern is composed of $k$ segments with one common end point and $k$ notches on the other end points.

When applying the shortest internal length criterion for polygons without holes, Greene [52] and Keil [68] proposed $O(r^2n^2)$ and $O(r^2n^2 \log n)$ time algorithms, respectively, that do not use Steiner points. When Steiner points are allowed, there are no known optimal solutions. An approximation algorithm by Levcopoulos and Lingas [79] produces a solution of length $O(p \log r)$ with Steiner points, where p is the length of perimeter of the polygon, in time $O(n \log n)$.

Not all convex decomposition methods fall into the above classification. For example, instead of decomposing $P$ into convex components whose union is $P$, Tor and Middleditch [125] "decompose" a simple polygon $P$ into a set of convex components $\{C_i\}$ such that $P$ can be represented as $CH_P - \cup_i C_i$, where "$-$" is the set difference operator, and instead of decomposing a polygon, Fevens et al. [49] partition a constrained 2D point set $S$ into convex polygons whose vertices are points in $S$.

Recently, several methods have been proposed to partition a polygon at salient features. Siddiqi and Kimia [117] use curvature and region information to identify *limbs* and *necks* of a polygon and use them to perform decomposition. Simmons and Séquin [119] proposed a decomposition using an *axial shape graph*, a weighted medial axis. Tănase and Veltkamp [126] decompose a polygon based on the events that occur during the construction of a straight-line skeleton. These events indicate the annihilation or creation of certain features. Dey et al. [45] partition a polygon into *stable manifolds* which are collections of Delaunay triangles of sampled points on the polygon boundary. Since these methods focus on visually important features, their applications are more limited than our approximately convex decomposition. Moreover, most of these methods require pre-processing (e.g., model simplification [66]) or post-processing (e.g., merging over-partitioned components [45]) due to boundary noise.

## 2. Convex Decomposition of Polyhedra

Convex decomposition of three-dimensional polyhedra is not as well understood as the two-dimensional case. Although this topic has been studied for several decades, most of the work focuses on refining the complexity requirements of Chazelle's popular notch cutting approach. Indeed, Chazelle's notch-resolving approach has inspired many other researchers to find more robust and efficient implementations. To resolve a notch of a polyhedron $P$, a *cutting plane*, $CH_P$, passing through the notch separates the incident facets and results in a decomposition where the dihedral angles are both less than 180°.

Chazelle [27, 29] shows that at most $\frac{r^2+r+2}{2}$ convex components will be generated if only one cutting plane is used for each notch, $r_i$, and its sub-notches, $\{r_{ij}\}$. Here $r_{ij}$ is the j-th sub-notch generated by intersecting $r_i$ and the cutting planes for $r_j$, $\forall i \neq j$. His method works by cutting all notches with cutting planes in an arbitrary order. Therefore, the main issue of convex decomposition becomes how the polyhedron can be cut by a given plane. First, the intersection of the plane and the polyhedron, $W$, is a set of simple polygons with holes which may enclose other polygons. Since these polygons do not overlap, a tree structure of these polygons can be built in $O(k \log k)$ time with $k$ vertices in $W$. For a polygonal chain $p$, a polygonal chain $q$ is $p$'s ancestor if $q$ contains $p$ directly or indirectly, and a polygonal chain $r$ is a child (descendant) of $p$ if $r$ is contained in $p$ directly (indirectly). This is called the polygon nesting problem. This structure helps locate the polygon, $s$, in $W$ that contains the notch to be cut and all polygons inside $s$. The cutting process is then done by splitting the edges and faces that intersect the cutting plane and that contain the polygon $s$ and descendants of $s$. His method generates the worst case optimal $O(r^2)$ convex parts and uses $O(nr^3)$ time with $O(nr^2)$ space.

The notch cutting approach proposed by Bajaj and Dey [11] considered non-manifold models which may contain notches with isolated vertices and edges, or non-manifold vertices and edges and reflective edges with dihedral angles greater than 180°. Since their plane cutting approach will generate non-manifold polyhedra even if the initial model is manifold, each cutting procedure starts decomposing the model by removing non-manifold features and then resolves a reflective edge using its plane cutting. By using Bajaj and Dey's approach [10] to solve the polygon nesting problem and more careful analysis, they achieved a convex decomposition in $O(nr^2 + r^{\frac{7}{2}})$ time with $O(nr + r^{\frac{5}{2}})$ space. They also provide a similar but robust algorithm which operates under finite precision arithmetic computations in $O(nr^2 + nr \log n + r^4)$ time.

Hershberger and Snoeyink [56] obtained $O(nr + r^{\frac{7}{3}})$ worst-case time complexity by studying the complexity of the horizon of a segment in an incrementally constructed erased arrangement of $n$ lines.

As mentioned in [33], despite the practical motivation, little research on the convex decomposition of polyhedra has gone beyond the theoretical stage. Currently, decomposing the surface of polyhedra [33, 34] is a more active research area due to its simplicity in theory and implementation. A surface is called convex if it lies entirely on the boundary of its convex hull. Therefore, surface decomposition is a problem of generating a set of convex surfaces whose union is the surface the given model and intersection is an empty set. The applications of convex surface decomposition include rendering [12], collision detection [12, 111], and penetration depth [74]. Although generating a minimum number of convex surfaces is still NP-complete, Chazelle et al. [33] proposed several heuristics: space partition, space sweep, and flooding. They concluded that flood-and-retract will be the simplest and most efficient.

# CHAPTER III

## APPROXIMATE CONVEX DECOMPOSITION (ACD)

Research in Psychology has shown that humans recognize shapes by decomposing them into components [14, 95, 117, 120]. Therefore, one approach that may produce a natural visual decomposition is to partition at the most *visually noticeable features*, such as the most dented or bent area, or an area with branches. Our approach for approximate convex decomposition follows this strategy. Namely, we recursively remove (resolve) concave features in order of decreasing significance until all remaining components have concavity less than some desired bound. One of the key challenges of this strategy is to determine approximate measures of concavity. We consider this question in later chapters. In this chapter, we assume that such a measure exists.

More formally, our goal is to generate $\tau$-convex decompositions, where $\tau$ is a user tunable parameter denoting the concavity tolerance of the application. (See Definition A.1 on p. 12). $P$ is said to be $\tau$*-approximate* convex if concavity$(P) < \tau$, A $\tau$*-convex decomposition* of $P$, $\mathrm{CD}_\tau(P)$, is defined as a decomposition that contains only $\tau$-convex components; i.e.,

$$\mathrm{CD}_\tau(P) = \{C_i \mid C_i \in \mathrm{D}(P) \text{ and } \mathrm{concavity}(C_i) \le \tau\}. \tag{3.1}$$

Note that a 0-convex decomposition is simply an exact convex decomposition, i.e., $\mathrm{CD}_{\tau=0}(P) = \mathrm{CD}(P)$.

Algorithm 1 describes a *divide-and-conquer* strategy to decompose $P$ into a set of $\tau$-convex pieces. The algorithm first computes the concavity, and a point $x \in \partial P$ witnessing it, of the polygon or polyhedron $P$, i.e., $x$ is one of the most concave features in $P$. If the concavity of $P$ is within the specified tolerance $\tau$, $P$ is returned. Otherwise, if the concavity of $P$ is above the maximum tolerable value, then the

**Algorithm 1** Approx_CD(P, $\tau$)

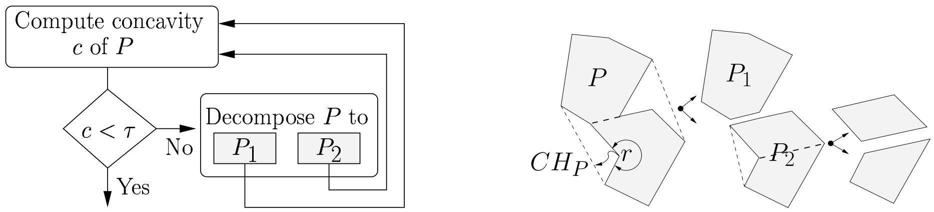
*Input.* A polygon or a polyhedron, $P$, and tolerance, $\tau$.
*Output.* A decomposition of $P$, $\{C_i\}$, such that $\max\{\text{concavity}(C_i)\} \leq \tau$.

```
c = concavity(P)
if c.value < τ then
    return P
else
    {C_i}=Resolve(P, c.witness).
    for Each component C ∈ {C_i} do
        Approx_CD(C,τ).
```

**Resolve**$(P, x)$ sub-routine will produce two components by resolving the concave feature at $x$, i.e., produce a decomposition of $P$ in which $x$ is a convex feature. In the next two chapters, we will discuss in detail about how concavity can be measured and how concave features can be resolved for polygons and polyhedra.

An overview of the decomposition process is shown in Figure 9(a). Due to the recursive application, the resulting decomposition has a natural hierarchy represented as a binary tree. An example is shown in Figure 9(b), where the original model $P$ is the root of the tree, and its two children are the components $P_1$ and $P_2$ resulting from the first decomposition. If the process is halted before convex components are obtained, then the leaves of the tree are approximate convex components. Thus, the hierarchical representation computed by our approach provides multiple Levels of Detail (LOD). A single decomposition is constructed based on the highest accuracy needed, but coarser, "less convex" components can be retrieved from higher levels in the decomposition hierarchy when the computation does not require that accuracy.

For some applications, the ability to consider only important features may not only be more efficient, but may also lead to improved results. In pattern recognition, for example, features are extracted from images and polygons to represent the shape of the objects. This process, e.g., skeleton extraction, is usually sensitive to small detail on the boundary, such as surface texture, which reduces the quality of the




(a) (b)

FIGURE 9. (a) Decomposition process. The tolerable concavity $\tau$ is user input. (b) A hierarchical representation of polygon $P$. Vertex $r$ is a notch and concavity is measured as the distance to the convex hull $CH_P$.

extracted features. By extracting a skeleton from the convex hulls of the components in an approximate decomposition, the sensitivity to small surface features can be removed, or at least decreased [83].

## A. Selection of Concavity Tolerance ($\tau$)

The main task that still needs to be specified in Algorithm 1 is how to measure the concavity of a polygon or a polyhedron. We use concavity measurement at a point as a primitive operation to decide whether a model $P$ should be decomposed and to identify concave features of $P$. In principle, our approach should be compatible with any reasonable measurement (the requirements for concavity measurement are discussed in the next section), and indeed the selection of the measure for the concavity tolerance $\tau$ should depend on the application. For example, for some applications, such as shape recognition, it may be desirable for the decomposition to be scale invariant, i.e., the decompositions of two different sized models with the same shape should be identical. Measuring the distance from $\partial P$ to $\partial CH_P$ is an example of measure that is not scale invariant because it would result in more components when decomposing a larger model. An example of a measure that could be scale invariant would be a unitless measure of the similarity of the model to its convex hull, or, one could simply normalize distances, e.g., by dividing by a scale parameter $s$, $d(\partial P, \partial CH_P)/s$.

## B. Concavity

In contrast to measures like radius, surface area, and volume, concavity does not have a well accepted definition. For our work, however, we need a quantitative way to measure the concavity of a polygon or polyhedron that can be computed in each iteration of Algorithm 1. A few methods have been proposed [121, 19, 39, 20, 9] that attempt to measure the concavity of an image (pixel) based polygon as the distance from the boundary of $P$ to the boundary of the pixel-based "convex hull" of $P$, called $CH'_P$, using Distance Transform methods. Since $P$ and $CH'_P$ are both represented by pixels, $CH'_P$ can only be nearly convex. *Convexity measurements* [123, 136] of polygons estimate the similarity of a polygon to its convex hull. For instance, the convexity of $P$ can be measured as the ratio of the area of $P$ to the area of the convex hull of $P$ [136] or as the probability that a fixed length line segment whose endpoints are randomly positioned in the convex hull of $P$ will lie entirely in $P$ [136]. To our knowledge, no concavity measure has been proposed for polyhedra.

Another complication with trying to use a global measure instead of a measure related to a feature of the polygon $P$, such as convexity, it that it is difficult to use such global measurements to efficiently identify *where* and *how* to decompose a polygon so as to increase the convexity measurements of the components. For example, Rosin [109] presents a shape partitioning approach that maximizes the convexity of the resulting components for a given number of cuts. His method takes $O(n^{2p})$ time to perform $p$ cuts. This exponential complexity forbids any practical use of this algorithm in our case.

Although ACD is not restricted to a particular measure, most of the measures


FIGURE 10. Although polygon $P_1$ is *visually* closer to being convex than polygon $P_2$, this is not identified by their convexity measurements, as defined in Eqn 7.2, which are equal, i.e., convexity$(P_1)$ = convexity$(P_2)$.

we consider in this work define the concavity of a model $P$ as the maximum concavity of its boundary points, i.e.,

$$\text{concavity}(P) = \max_{x \in \partial P}\{\text{concavity}(x)\} \ , \tag{3.2}$$

where $x$ are the vertices of $P$. We define the concavity of a point $x$, concavity$(x)$, as the distance from $x$ to the boundary of the convex hull $CH_P$. 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 if we used the *convexity* measurement in [123, 136], where the convexity of a model $P$ is defined as

$$\text{convexity}(P) = \frac{\text{volume}(P)}{\text{volume}(CH_P)} \ . \tag{3.3}$$

For example, the polygons, $P_1$ and $P_2$, shown in Figure 10 have the same convexity, but $P_1$ is visually closer to being convex than polygon $P_2$.

## 1. Retraction Function

In this work, we will define concavity using a *retraction function* that traces a path to the boundary of the convex hull. More formally, let $\text{retract}_x(t) : \partial P \to CH_P$ denote the function defining the trajectory of $x$ when $x$ is retracted from its original position to $\partial CH_P$. When $t = 0$, $\text{retract}_x(t)$ is $x$ itself. When $t = 1$, $\text{retract}_x(t)$ is a point on $\partial CH_P$. Assuming that this retraction exists for $x$, we define

$$\text{concavity}(x) = \int_{\text{retract}_x(0)}^{\text{retract}_x(1)} |d\ell| \ , \tag{3.4}$$

where $d\ell$ is a differential displacement vector along the curve $\text{retract}_x(t)$, i.e., $\text{concavity}(x)$ is the arc length of the function $\text{retract}_x(t)$ with $t$ from zero to one.

Intuitively, one can use the following analogy for the retraction function. Imagine that $P$ is a balloon placed in a mold with the shape of $CH_P$. As we pump air into the balloon $P$, it will gradually expand to assume the shape of $CH_P$. The trajectory for a point $x$ on $P$ is the path traveled by $x$ during the inflation from its position on the initial shape to its position on the the final shape of the balloon.

Unfortunately, although the intuition is simple, it is not easy to define or compute such a retraction path. For example, we can define this balloon expansion as a process of enlarging the inscribing balls of the points on the medial axis $\text{MA}(P)$ of $P$. The medial axis of $P$, $\text{MA}(P)$, is the set of points in $p \in P$ such that a maximal ball centered at $p$ and contained in $P$ is tangent to the boundary of $P$ in at least two points. Let $x$ be a point on $\partial P$ but not on $\partial CH_P$ and let $y$ be a point on $\text{MA}(P)$ whose maximum enclosing ball contacts $\partial P$ at $x$. See Figure 11(a). At time $t$, $x$ will be retracted away from $y$ in the direction of $\overrightarrow{y\,x}$, i.e.,

$$x_{t+\mathrm{d}t} = \text{retract}_x(t + \mathrm{d}t) = x_t + \overrightarrow{y_t\,x_t}\,\mathrm{d}t \ , \tag{3.5}$$

where $\mathrm{d}t$ is a unit time step. Another possible way of measuring concavity is to model $P$ using springs and then simulate inflation [72]. However, these methods are computationally expensive.

We next define a class of retraction functions that have proven suitable for use in


FIGURE 11. (a) Defining concavity retraction using the medial axis. (b) Straight line distance concavity (left) and shortest path distance concavity (right).

ACD. In particular, as shown later in this dissertation, the properties of the retraction functions in this class can be exploited to establish the correctness of our ACD approach.

**Definition B.1.** *Let $P = P^0$ be a polygon or polyhedron and let $P^{i+1}$ denote the decomposition of $P^i$ that results when one or more notches of $P^i$ is resolved.*

*We say that a retraction function $\gamma(x)$, or simply $\gamma$, is* simple *if:*

$$\text{concavity}_\gamma(P^i) \geq \text{concavity}_\gamma(P^j),\ \forall i < j, \tag{3.6}$$

*where* $\text{concavity}_\gamma(P^k) = \max_{x \in V^k}\{concave_\gamma(x)\}$, *and we say $\gamma$ is* stable *if:*

$$\gamma(x) \text{ in } P^i \ \geq \gamma(x) \text{ in } P^j\ \forall i < j \tag{3.7}$$

**Lemma B.2.** *If the retraction function $\gamma$ is simple and stable, then the point $x$ that maximizes $\gamma(x)$ must be a notch and resolving the concave feature at $x$ in $P^i$ will result in $P^{i+1}$ that has monotonically decreasing concavity.*

*Proof.* If the retraction function $\gamma(x)$ is stable, then resolving notches in $V^i$ cannot increase the concavity of the vertices in $V^{i+1}$. Therefore, if the vertex $x$ with maximum concavity in $P^i$ is resolved, then the concavity of $P^{i+1}$ cannot increase. Thus, $x$ must be in $V^i \setminus V^{i+1}$ and $x$ must be a notch. $\square$

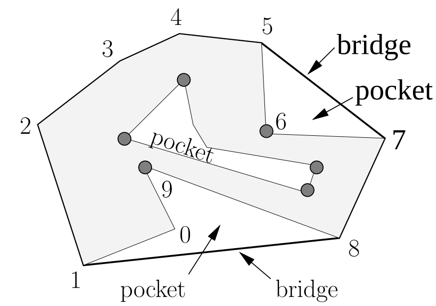



Figure 12. Vertices marked with dark circles are notches. Edge (5,7) is a bridge with an associated pocket $\{(5,6),(6,7)\}$. Edge (8,1) is a bridge with an associated pocket $\{(8,9),(9,0),(0,1)\}$.

The correctness arguments we make regarding ACD in Chapter IV only assume that the retraction function is simple and stable. That is, our framework is not dependent on the particular retraction methods studied in this work, and in particular, the same correctness guarantees will be provided by any retraction function that is simple and stable.

## 2. Bridges and Pockets

Our concavity measures use the concepts of *notches*, *bridges* and *pockets*; see Figure 12. Recall that vertices of a polygon and edges of a polyhedron, respectively, are *notches* if they have internal angles greater than 180°. For a given polygon $P$, bridges are convex hull edges that connect two non-adjacent vertices of $\partial P_0$, i.e., $\text{BRIDGES}(P) = \partial CH_P \setminus \partial P$. Pockets are maximal chains of non-convex-hull edges of $P$, i.e., $\text{POCKETS}(P) = \partial P \setminus \partial CH_P$. Note that the same definitions of bridge and pocket can also be applied to polyhedra.

Observation B.3 states the relationship between bridges, pockets, and notches for *polygons*.

**Observation B.3.** *Given a simple polygon $P$. Notches can only be found in pockets.*

*Each bridge has an associated pocket, the chain of $\partial P_0$ between the two bridge vertices. Hole boundaries are also pockets, but they have no associated bridge.*

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 $\beta_x$, i.e., concavity$(x) =$ dist$(x, \partial CH_P) =$ dist$(x, \beta_x)$.

For polygons, there is a natural one-to-one bridge/pocket matching that can be obtained easily. In Chapter IV, we propose two practical simple and stable retraction methods to compute concavity [85]: the straight-line distance to the bridge and the length of the shortest path to the bridge that does not intersect the polygon; see Figure 11(b).

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 Chapter V, we discuss how the bridge/pocket relationship can be computed. In addition, while SL-concavity can still be computed efficiently, the best known methods for computing shortest paths on polyhedra require exponential time [113] and even methods [36] that approximate the shortest paths are too inefficient to be used in our approach.

# CHAPTER IV

## APPROXIMATE CONVEX DECOMPOSITION OF POLYGONS

In this chapter, we describe our strategy for decomposing a polygon, containing zero or more holes, into *"approximately convex"* pieces. As we will see later in this chapter, for many applications, the approximately convex components of this decomposition provide similar benefits as convex components, while the resulting decomposition is both significantly smaller and can be computed more efficiently. Features of this approach are that it

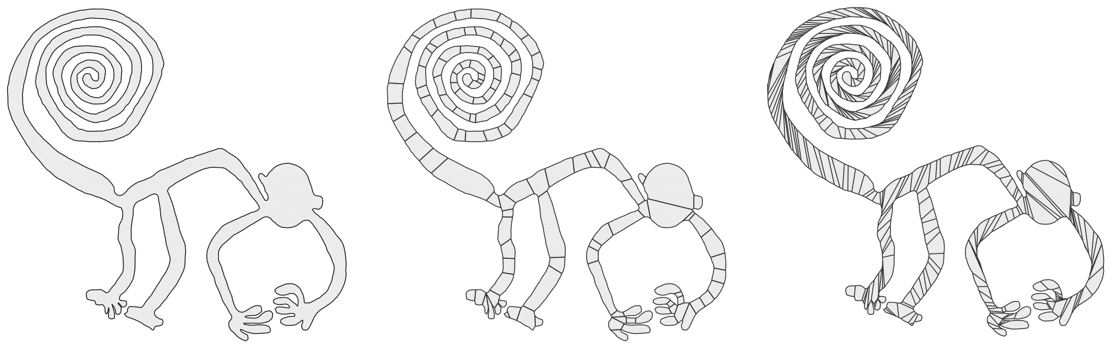
- applies to any simple polygon, with or without holes,
- provides a mechanism to focus on key features, and
- produces a hierarchical representation of convex decompositions of various levels of approximation.

Figure 13 shows an approximate convex decomposition with 128 components and a minimum convex decomposition with 340 components [71] of a Nazca line monkey.[†]

Our algorithm computes an ACD of a simple polygon with $n$ vertices and $r$ notches in $O(nr)$ time. In contrast, as described in Chapter II, exact convex decomposition is NP-hard [90, 69, 91] or, if the polygon has no holes, takes $O(nr^2)$ time [32, 71].

We follow the *divide-and-conquer* strategy, as described in Algorithm 1, to decompose a polygon $P$ into a set of $\tau$-convex pieces. Recall that the two main sub-routines required for this algorithm include sub-routines that measure and resolve concave

[†]Nazca lines [25] are mysterious drawings found in southwest Peru. They have lengths ranging from several meters to kilometers and can only be recognized by aerial viewing. Two drawings, monkey and heron, are used as examples in this chapter.



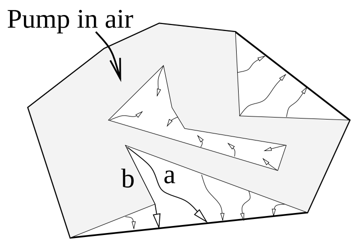
(a) (b) (c)

Figure 13. (a) The initial Nazca monkey has 1,204 vertices and 577 notches. The radius of the minimum bounding circle of this model is 81.7 units. Setting the concavity tolerance at 0.5 units, and not allowing Steiner points, (b) an approximate convex decomposition has 126 approximately convex components, and (c) a minimum convex decomposition has 340 convex components.

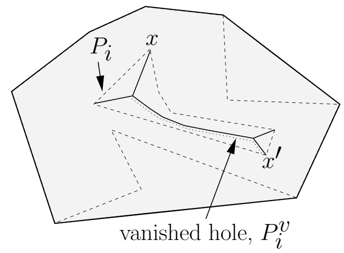
features. General issues and details regarding of our concavity measurements are presented in Section A. Next, in Section B, we discuss how a concave feature with unacceptable concavity can be resolved. In Section C, we analyze the complexity of the method and provide implementation details and experimental results in D.

## A. Measuring Concavity

Recall that the concavity of a boundary point $x$ of a polygon $P$ is the distance from $x$ to the boundary of $P$'s convex hull. In this section, we will discuss how the distance can be approximated for points that are on the external boundary and on hole boundaries.




(a)




(b)

FIGURE 14. (a) The initial shape of a non-convex balloon (shaded). The bold line is the convex hull of the balloon. When we inflate the balloon, points not on the convex hull will be pushed toward the convex hull. Path $a$ denotes the trajectory with air pumping and path $b$ is an approximation of $a$. (b) The hole vanishes to its medial axis and vertices on the hole boundary will never touch the convex hull.

### 1. Measuring Concavity for External Boundary ($\partial P_0$) Points

An intuitive way to define concavity for a point $x \in \partial P$, concavity($x$), is to consider the trajectory of $x$ when $x$ is retracted from its original position to $\partial CH_P$. Recall that we let $\text{retract}_x(t) : \partial P \to CH_P$ denote the function defining the trajectory of a point $x \in \partial P$ when $x$ is retracted from its original position to $\partial CH_P$. More details regarding the function $\text{retract}_x(t)$ can be found in Chapter III, where we also describe the properties that we require for the retraction function. An intuition of this retraction function is illustrated in Figure 14(a). Recall that we can think of $P$ as a balloon that is placed in a mold with the shape of $CH_P$. Although the initial shape of this balloon is not convex, the balloon will become so if we keep pumping air into it. Then the trajectory of a point on $P$ to $CH_P$ can be defined as the path traveled by a point from its position on the initial shape to the final shape of the balloon. Although the intuition is simple, a retraction path such as path $a$ in Figure 14(a) is not easy to define or compute.

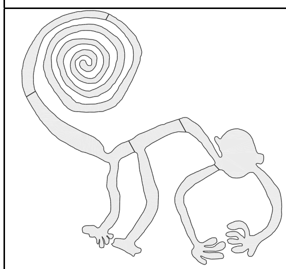
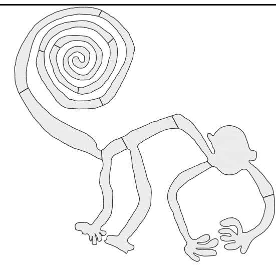
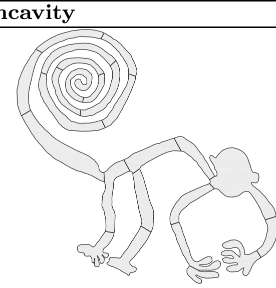
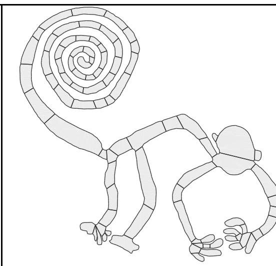
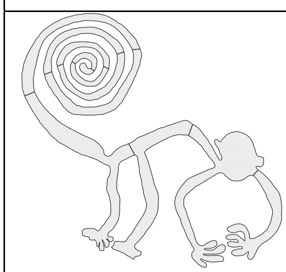
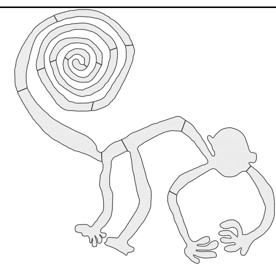
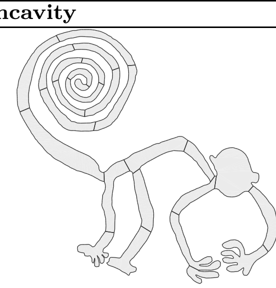
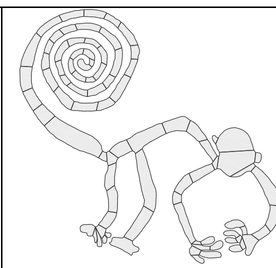
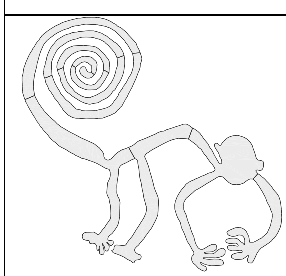
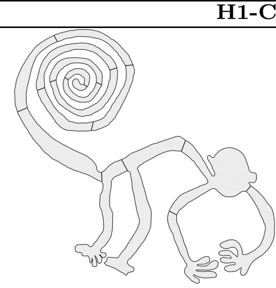
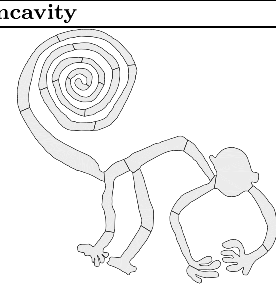
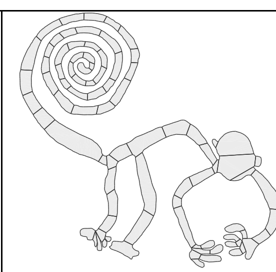
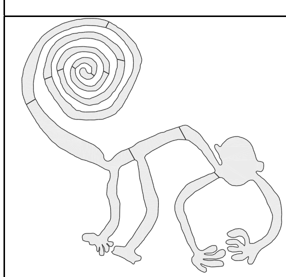
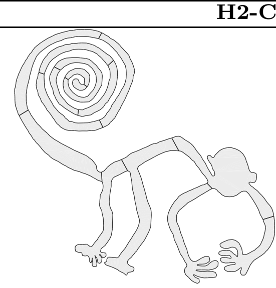
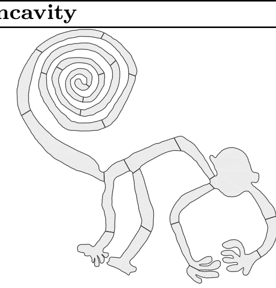
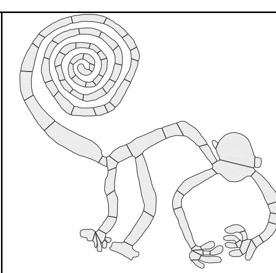
Below, we describe three methods for measuring an approximation of this retraction distance that can be used in Algorithm 1. Recall that each pocket $\rho$ on the external boundary $\partial P_0$ is associated with exactly one bridge $\beta$. In Section A.1.a, this retraction distance is measured by computing the straight-line distance from $x$ to the bridge. Although this distance is fairly easy to compute, as we will see in Section A.1.a, using it we cannot guarantee that the concavity of a point will decrease monotonically. A method that does not have this drawback is shown in Section A.1.b, where we extract a shortest path from $x$ to the bridge from a visibility tree contained in the pocket. Unfortunately, this distance is more expensive to compute. Hybrid approaches that seek the advantages of both methods are proposed in Section A.1.c.

a. Straight Line Concavity (SL-Concavity)

In this section, we approximate the concavity of a point $x$ on $\partial P_0$ by computing the straight-line distance from $x$ to its associated bridge $\beta$, if any. Note that this straight line may intersect $P$. Table 1 shows the decomposition of a Nazca monkey using SL-concavity.

Although computing the straight line distance is simple and efficient, this approach has the drawback of potentially leaving certain types of concave features in the final decomposition. As shown in Figure 15, the concavity of $s$ does not decrease monotonically during the decomposition. This results in the possibility of leaving important features, such as $s$, hidden in the resulting components. This deficiency is also shown in the first image of Table 1 ($\tau = 40$) when the spiral tail of the monkey is not well decomposed. These artifacts result because the straight line distance does not reflect our intuitive definition of concavity.

Table 1— Nazca monkey (Figure 13(a)) decomposition using SL-, SP-, H1-, and H2-Concavity with $\tau$ as 40, 20, 10, and 1 units. Recall that the radius of the minimum enclosing circle of the monkey is 81.7 units.

| $\tau = 40$ | $\tau = 20$ | $\tau = 10$ | $\tau = 1$ |
|---|---|---|---|
| **SL-Concavity** | | | |
|  (6 components) |  (13 components) |  (24 components) |  (90 components) |
| **SP-Concavity** | | | |
|  (12 components) |  (16 components) |  (26 components) |  (88 components) |
| **H1-Concavity** | | | |
|  (12 components) |  (16 components) |  (26 components) |  (88 components) |
| **H2-Concavity** | | | |
|  (12 components) |  (15 components) |  (25 components) |  (90 components) |

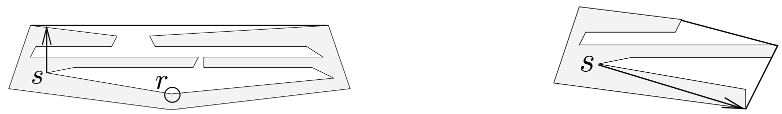



FIGURE 15. Let $r$ be the notch with maximum concavity measured using SL-concavity. After resolving $r$, the concavity of $s$ increases. If concavity$(r) < \tau$, then $s$ will never be resolved even if concavity$(s)$ would be larger than $\tau$ if the model were to be resolved at $r$.

b. Shortest Path Concavity (SP-Concavity)

In our second method, we find a shortest path from each vertex $x$ in a pocket $\rho$ to the bridge line segment $\beta = (\beta^-, \beta^+)$ such that the path lies entirely in the area enclosed by $\beta$ and $\rho$, which we refer to as the *pocket polygon* and denote by $P_\rho$. Note that $P_\rho$ must be a simple polygon. See Figure 16(a). In the following, we use $\pi(x, y)$ to denote the shortest path in $P_\rho$ from an object $x$ to an object $y$, where $x$ and $y$ can be edges or vertices. Two objects $x$ and $y$ are said to be *weakly visible* [8] to each other if one can draw at least one straight line from a point in $x$ to a point in $y$ without intersecting the boundary of $P_\rho$. A point $x$ is said to be *perpendicularly visible* from a line segment $\beta$ if $x$ is weakly visible from $\beta$ and one of the visible lines between $x$ and $\beta$ is perpendicular to $\beta$. For instance, points $a$ and $c$ in Figure 16(b) are perpendicularly visible from the bridge $\beta$ and $b$ and $d$ are not. We denote by $V_\beta^+$ the ordered set of vertices that are perpendicularly visible from $\beta$, where vertices in $V_\beta^+$ have the same order as those in $\partial P_0$.

We compute the shortest distance to $\beta$ for each vertex $x$ in $\rho$ according to the process sketched in Algorithm 2. First, we split $P_\rho$ into three regions, $P_{\rho\beta^-}$, $P_{\rho\beta}$, and $P_{\rho\beta^+}$ as shown in Figure 16(b). The boundaries between $P_{\rho\beta^-}$ and $P_{\rho\beta}$ and $P_{\rho\beta}$ and $P_{\rho\beta^+}$, i.e., $\overline{a\beta^-}$ and $\overline{c\beta^+}$, are perpendicular to $\beta$. As shown in Lemma A.2, the shortest paths for vertices $x$ in $P_{\rho\beta^-}$ or $P_{\rho\beta^+}$ to $\beta$ are the shortest paths to $\beta^-$ or $\beta^+$,

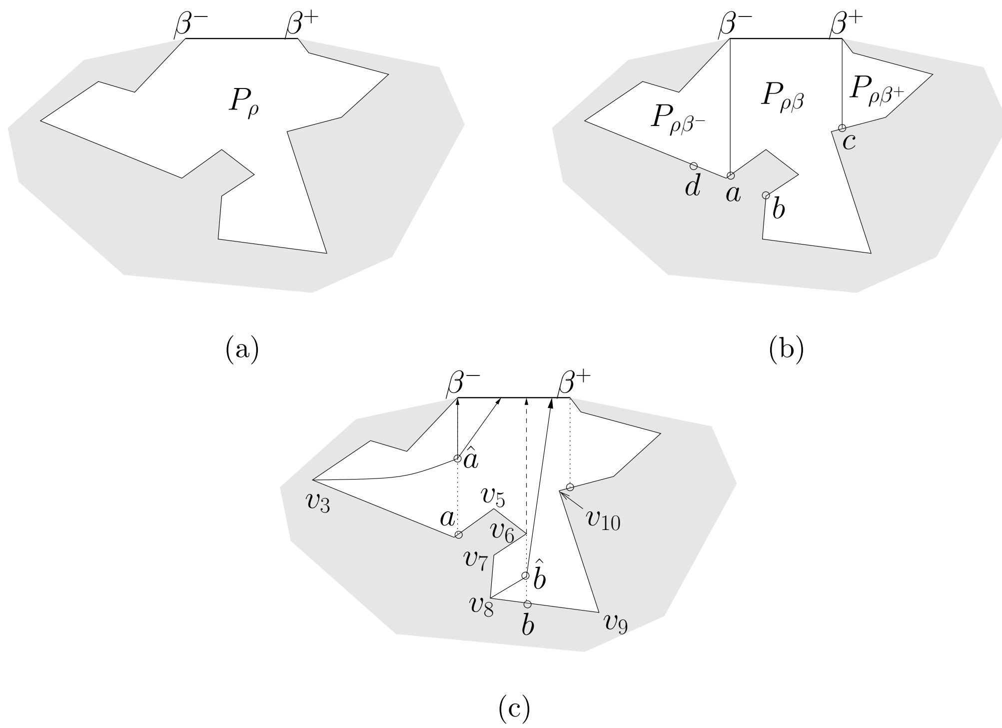



FIGURE 16. (a) $P_\rho$ is a simple polygon enclosed by a bridge $\beta$ and a pocket $\rho$. (b) Split $P_\rho$ into $P_{\rho\beta^-}$, $P_{\rho\beta}$, and $P_{\rho\beta^+}$. (c) $V_\beta^- = \{v_7, v_8, v_9\}$ and $V_\beta^+ = \{v_5, v_6, v_{10}\}$.

respectively. These paths can be found by constructing a *visibility tree* [53] rooted at $\beta^-$ ($\beta^+$) to all vertices in $P_{\rho\beta^-}$ ($P_{\rho\beta^+}$).

The shortest path for a vertex $x \in P_{\rho\beta}$ to $\beta$ is composed of two parts: the shortest path $\pi(x, y)$, from $x$ to some point $y$ perpendicularly visible to $\beta$, i.e., $y \in V_\beta^+$, and the straight line segment connecting $y$ to $\beta$, $\pi(y, \beta)$. Let $V_\beta^- = \{v \in \partial P_{\rho\beta}\} \setminus V_\beta^+$. Figure 16(c) illustrates an example of $V_\beta^+$ and $V_\beta^-$. For each $v \in V_\beta^+$, there exists a subset of vertices in $V_\beta^-$ that are closer to $v$ than to any other vertices in $V_\beta^+$. These vertices must have shortest paths passing through $v$. For instance, in Figure 16(c), $v_8$ and $v_7$ must pass through $v_6$. Moreover, these vertices can be found by traversing the vertices of $\partial P_{\rho\beta}$ in order. For example, vertices between $v_6$ and $v_{10}$ must have shortest paths passing through either $v_6$ or $v_{10}$.

We compute $V_\beta^+$ by first finding vertices in $P_{\rho\beta}$ that are weakly visible from $\beta$

**Algorithm 2** SP_Concavity($\beta$,$\rho$)

Split $P_\rho$ into polygons $P_{\rho\beta^-}$, $P_{\rho\beta}$, and $P_{\rho\beta^+}$ as shown in Figure 16(b).
Construct two visibility trees, $T^-$ and $T^+$, rooted in $\beta^-$ and $\beta^+$, respectively, to all vertices in $\rho$.
Compute $\pi(v, \beta)$, $\forall v \in P_{\rho\beta^-}$ (resp., $P_{\rho\beta^+}$) from $T^-$ (resp., $T^+$).
Compute an ordered set, $V_\beta^+$, in $P_{\rho\beta}$ from $T^-$ and $T^+$.
**for** each consecutive pair $(v_i, v_j) \in V_\beta^+$ **do**
    **for** $i < k < j$ **do**
        $\pi(v_k, \beta) = \min\big(\pi(v_k, v_i) + \pi(v_i, \beta), \pi(v_k, v_j) + \pi(v_j, \beta)\big)$.
Return $\{x, c\}$, where $x \in \rho$ is the farthest vertex from $\beta$ with distance $c$.

and then filtering out vertices that are not perpendicularly visible from $\beta$. If a vertex $v \in P_{\rho\beta}$ is weakly visible from $\beta$, both $\pi(v, \beta^-)$ and $\pi(v, \beta^+)$ must be *outward convex*. Following Guibas et al. [53], we say that $\pi(v, \beta^-)$ is outward convex if the convex angles formed by successive segments of this path keep increasing. Lemma A.1 [53] states the property of two weakly visible edges. Our problem is a degenerate case of Lemma A.1 as one of the edges collapses into a vertex, $v$. Therefore, finding weakly visible vertices of $\beta$ can be done by constructing two visibility trees rooted at $\beta^-$ and $\beta^+$.

**Lemma A.1.** [53] *If edge $\overline{ab}$ is weakly visible from edge $\overline{cd}$, the two paths $\pi(a, c)$ and $\pi(b, d)$ are outward convex.*

The following lemma shows that Algorithm 2 finds the shortest paths from all vertices in the pocket $\rho$ to its associated bridge line segment $\beta$.

**Lemma A.2.** *Algorithm 2 finds the shortest path from every vertex $v$ in pocket $\rho$ to the bridge $\beta$.*

*Proof.* First we show that, for vertices $v$ in region $P_{\rho\beta^-}$, $\pi(v, \beta)$ must pass through $\beta^-$ to reach $\beta$. If the shortest path $\pi(v, \beta)$ from some $v \in A$ does not pass through $\beta^-$ then it must intersect $\overline{\beta^- a}$ at some point which we denote $\hat{a}$. Vertex $v_3$ in Figure 16(c) is an example of such a vertex. However, the shortest path from $\hat{a}$ to $\beta$ is the line segment from $\hat{a}$ to $\beta^-$. This contradicts the assumption that $\pi(v, \beta)$ does not pass through $\beta^-$. Therefore, all points in $P_{\rho\beta^-}$ must have shortest paths passing through $\beta^-$. Also, it has been proved that the visibility tree contains the shortest paths [77] from one vertex to all others in a simple polygon. Therefore, Line 3 in Algorithm 2 must find shortest paths to $\beta$ for all vertices in $P_{\rho\beta^-}$. Similarly, it can be shown that $\pi(v, \beta)$ for all vertices in region $P_{\rho\beta^+}$ must pass through $\beta^+$.

For vertices $v$ in region $P_{\rho\beta}$, we show that $\pi(v, \beta)$ must pass through $V_\beta^+$ to reach $\beta$. If $v \in V_\beta^+$, then the condition is trivially satisfied. Hence we need only consider $v \in V_\beta^-$. Vertices $v_6 \in V_\beta^+$ and $v_8 \in V_\beta^-$ in Figure 16(c) are examples of such vertices. If the shortest path $\pi(v, \beta)$ for some $v \in V_\beta^-$ does not pass through $V_\beta^+$ then it must intersect the segment perpendicular to $\beta$ passing by some vertex in $V_\beta^+$. Let $v' \in V_\beta^+$ be such a vertex and denote the point where $\pi(v, \beta)$ intersects $\perp v'\beta$ as $\hat{b}$. Since the shortest path from $\hat{b}$ to $\beta$ is a straight line to $\beta$ and it passes through $v' \in V_\beta^+$, we have a contradiction to the assumption that $\pi(v, \beta)$ does not pass through some $v \in V_\beta^+$. Therefore, Algorithm 2 must find the shortest path to $\beta$ for all vertices in $P_{\rho\beta}$. □

The concavity of a vertex $v$ is the length of the shortest path from $v$ to its associated bridge $\beta$. To compute the SP-concavity of $\partial P_0$, we find all bridge/pocket pairs and apply Algorithm 2 to each pair. Examples of retraction trajectories using SP-concavity are shown in Figure 17.

Next, we show that concavity($P$) decreases monotonically in Algorithm 1 if we use the shortest path distance to measure concavity. The guarantee of monotonically decreasing concavity eliminates the problem of leaving important concave features untreated as may happen using SL-concavity (see Table 1).

**Lemma A.3.** *The concavity of $\partial P_0$ decreases monotonically during the decomposition in Algorithm 1 if we use SP-concavity.*

FIGURE 17. Shortest paths to the boundary of the convex hull.

*Proof.* We show that the concavity of a point $x$ in a pocket $\rho$ of $\partial P_0$ either decreases or remains the same after another point $x' \in \rho$ is resolved. Let $\beta$ be $\rho$'s bridge with $\beta^-$ and $\beta^+$ as end points. After $x'$ is resolved, $\rho$ breaks into two polygonal chains, from $\beta^-$ to $x'$ and from $x'$ to $\beta^+$. New pockets and bridges will be constructed for both polygonal chains. Since the shortest path from $x$ to the previous bridge $\beta$ must intersect the bridge for $x$'s new pocket, the new concavity of $x$ will decrease or remain the same. □

Finally, we show that Algorithm 2 takes $O(n)$ time to compute SP-concavity for all vertices on $\partial P_0$.

**Lemma A.4.** *Measuring the concavity of the vertices on the external boundary $\partial P_0$ using shortest paths takes $O(n)$ time, where $n$ is the size of $\partial P_0$.*

*Proof.* For each bridge/pocket, we show that the SP-concavity of all pocket vertices can be computed in linear time, which implies that we can measure the SP-concavity of $P$ in linear time. First, it takes $O(n)$ time to split $P$ into $P_{\rho\beta^-}$, $P_{\rho\beta}$, and $P_{\rho\beta^+}$ by computing the intersection between the pocket $\rho$ and two rays perpendicular to $\beta$ initiating from $\beta^-$ and $\beta^+$. Then, using a linear time triangulation algorithm [30, 2], we can build a visibility tree in $O(n)$ time. Finding $V^+(\beta)$ takes $O(n)$ time as shown in [53]. The loop in Lines 5 to 8 of Algorithm 2 takes $\sum |j - i| \leq n = O(n)$ time since the $(i, j)$ intervals do not overlap. Thus, Algorithm 2 takes $O(n)$ time and therefore we can measure the SP-concavity of $P$ in $O(n)$ time. □

c. Hybrid Concavity (H-Concavity)

We have considered two methods for measuring concavity: SL-concavity, which can be computed efficiently, and SP-concavity, which can guarantee that concavity decreases monotonically during the decomposition process. In this section, we describe a hybrid approach, called H-concavity, that has the advantages of both methods — SL-concavity is used as the default, but SP-concavity is used when SL-concavity would result in non-monotonically decreasing concavity of $P$.

SL-concavity can fail to report a significant feature $x$ when the straight-line path from $x$ to its bridge $\beta$ intersects $\partial P_0$. In this case, $x$'s concavity is under measured. Whether a pocket can contain such points can be detected by comparing the directions of the outward surface normals for the edges $e_i$ in the pocket and the outward normal direction $\vec{n}_\beta$ of the bridge $\beta$. The decision to use SL-concavity or SP-concavity is based on the following observation. Figure 18 illustrates this observation.

**Observation A.5.** *Let $\beta$ and $\rho$ be a bridge and pocket of $\partial P_0$, respectively. If* concavity$(\partial P)$ *does not decrease monotonically using the SL-concavity measure, there must be an edge $e \in \rho$ such that the normal vector of $e$, $\vec{n}_e$, and the normal vector of $\beta$, $\vec{n}_\beta$, point in opposite directions, i.e., $\vec{n}_e \cdot \vec{n}_\beta < 0$.*

This observation leads to Algorithm 3. We first use Observation A.5 to check if SL-concavity can be used. If so, the concavity of $P$ and its witness is computed using SL-concavity. Otherwise, SP-concavity is used. This approach improves the computation time and guarantees that the decomposition process has monotonically decreasing concavity.

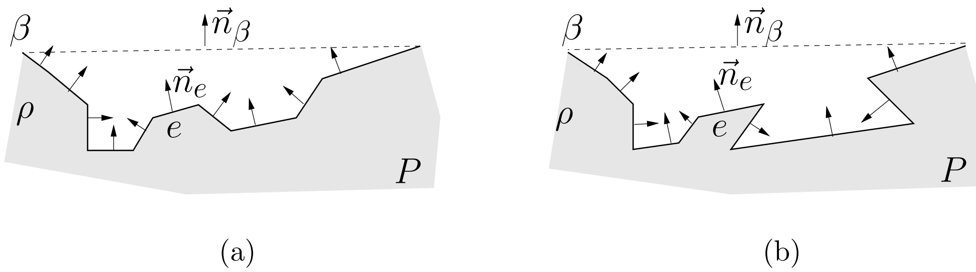



FIGURE 18. SL-concavity can handle the pocket in (a) correctly because none of the normal directions of the vertices in the pocket are opposite to the normal direction of the bridge. However, the pocket in (b) may result in non-monotonically decreasing concavity.

Another option is to use SL-concavity more aggressively to compute the decomposition even more efficiently. This approach is described in Algorithm 4. First, we use SL-concavity to measure the concavity of a given bridge-pocket pair. If the maximum concavity is larger than the tolerance value $\tau$, we split $P$. Otherwise, using Observation A.5, we check if there is a possibility that some feature with untolerable concavity is hidden inside the pocket. If we find a potential violation, then SP-concavity is used. This approach is more efficient because it only uses SP-concavity if SL-concavity does not identify any untolerable concave features. We refer to the concavities computed using Algorithm 3 and Algorithm 4 as H1-concavity and H2-concavity, respectively.

Unlike H1-concavity, decomposition using H2-concavity may not have monotonically decreasing concavity. Thus, the order in which the concave features are found for H1- and H2-concavity can be different. Table 1 shows the decomposition process using H1-concavity and H2-concavity, respectively. The decomposition using H1-concavity is identical to that using SP-concavity. The decomposition using H2-concavity is more similar to the decompositions that would result from using SP-concavity with a larger $\tau$ or from using SL-concavity with smaller $\tau$. We also observe

**Algorithm 3** H1-Concavity($\beta$, $\rho$)

```
if No potential hazard detected, i.e., ∄r ∈ ρ such that n⃗_r · n⃗_β < 0 then
    Return SL-concavity and its witness. (Section A.1.a)
else
    Return SP-concavity and its witness. (Section A.1.b)
```

**Algorithm 4** H2-Concavity($\beta$, $\rho$)

```
SL-concavity and its witness {x, c}. (Section A.1.a)
if c > τ then
    Return {x, c}.
if No potential hazard detected, i.e., ∄r ∈ ρ such that n⃗_r · n⃗_β < 0 then
    Return {x, c}.
Return SP-concavity and its witness. (Section A.1.b)
```

that the relative computation costs of the different measures are, from slowest to fastest: SP-concavity, H1-concavity, H2-concavity, and finally SL-concavity. Experiments comparing decompositions using these concavity measures are presented in Section D.

### 2. Measuring the Concavity for Hole Boundary ($\partial P_{i>0}$) Points

Note that in the balloon expansion analogy, points on hole boundaries will never touch the boundary $\partial CH_P$ of the convex hull $CH_P$. The concavity of points in holes is therefore defined to be infinity and so we need some other measure for them. We will estimate the concavity of a hole $P_i$ locally, i.e., without considering the external boundary $\partial P_0$ or the convex hull $\partial CH_P$. Using the balloon expansion analogy again, we observe the following.

**Observation A.6.** *$P_i$ will "vanish" into a set of connected curved segments forming the medial axis of the hole as it contracts when $\partial P_0$ transforms to $CH_P$. These curved segments will be the union of the trajectories of all points on $\partial P_i$ to $CH_P$ once $\partial P_i$ is merged with $\partial P_0$. Figure 14(b) shows an example of a vanished hole.*

a. Concavity for Holes

Recall that, from Observation B.3 in Chapter III, $\partial P_i$ can also be viewed as a pocket without a bridge. The bridge will become known when a point $x \in \partial P_i$ is resolved, i.e., when a diagonal between $x$ and $\partial P_0$ is added which will make $\partial P_i$ become a pocket of $\partial P_0$. If $x$ is resolved, the concavity of a point $y$ in $\partial P_i$ is concavity$(x)$ + dist$(x, y)$. We define the *concavity witness* of $x$, $cw(x)$, to be a point on $\partial P_i$ such that $\text{dist}(x, cw(x)) > \text{dist}(x, y),\ \forall y \neq cw(x) \in \partial P_i$. That is, if we resolve $x$, then $cw(x)$ will be the point with maximum concavity in the pocket $\partial P_i$. For associate distance measures (such as all those considered here), $x$ and $cw(x)$ are associative, i.e., $cw(cw(x)) = x$, so that if we resolve $cw(x)$, then $x$ will be the point with maximum concavity in the pocket $\partial P_i$. See Figure 19. Intuitively, the maximum $\text{dist}(p, cw(p))$, where $p \in \partial P_i$ represents the "diameter" of $P_i$. The *antipodal pair* $p$ and $cw(p)$ of the hole $P_i$ represent important features because $p$ (or $cw(p)$) will have the maximum concavity on $\partial P_i$ when $cw(p)$ (or $p$) is resolved. Our task is to find $p$ and $cw(p)$.

A naïve approach to find the antipodal pair $p$ and $cw(p)$ of $P_i$ is to exhaustively resolve all vertices in $\partial P_i$. Unfortunately, this approach requires $O(n^2)$ time, where $n$ is the number of vertices of $P$. Even if we attempt to measure the concavity of $P_i$ locally without considering $\partial P_0$ and $CH_P$, computing distances between all pairs of points in $\partial P_i$ has time complexity $O(n_i^2)$, where $n_i$ is the number of vertices of $P_i$.

b. Approximate Antipodal Pair, $p$ and $cw(p)$

Fortunately, there are some possibilities to approximate $p$ and $cw(p)$ more efficiently. As previously mentioned, in our balloon expansion analogy, a hole will contract to the medial axis which is a good candidate to find $p$ and $cw(p)$ because it connects all pairs of points in the hole $P_i$. Once $\partial P_i$ is merged to $\partial P_0$, concavity can be computed easily

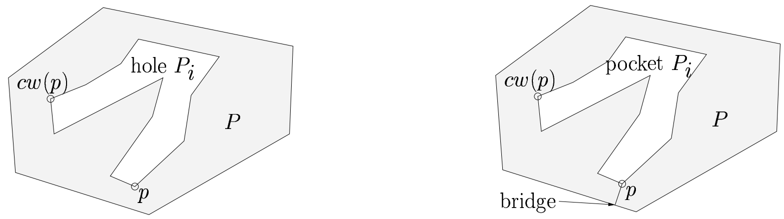



FIGURE 19. An example of a hole $P_i$ and its antipodal pair. The maximum distance between $p$ and $cw(p)$ represents the diameter of $P_i$. After resolving $p$, $P_i$ becomes a pocket and $cw(p)$ is the most concave point in the pocket.

from the trajectories on the medial axis. Since $P_i$ is a simple polygon, the medial axis of $P_i$ forms a tree and can be computed in linear time [35]. We can approximate $p$ and $cw(p)$ as the two points at maximum distance in the tree, which can be found in linear time.

Another way to approximate $p$ and $cw(p)$ is to use the Principal Axis (PA) of $P_i$. The PA for a given set of points $S$ is a line $\ell$ such that total distance from the points in $S$ to $\ell$ is minimized over all possible lines $\kappa \neq \ell$, i.e.,

$$\sum_{x \in S} \mathrm{dist}(x, \ell) < \sum_{x \in S} \mathrm{dist}(x, \kappa), \ \forall \kappa \neq \ell. \tag{4.1}$$

In our case, $S$ is the vertices of $P_i$. The PA can be computed as the Eigenvector with the largest Eigenvalue from the covariance matrix of the points in $S$. Once the PA is computed, we can find two vertices of $P_i$ in two extreme directions on PA, and select one as $p$ and the other as $cw(p)$. This approximation also takes $O(n)$ time.

Concavity measured using the PA resembles SL-concavity because in both cases concavity is measured as straight line distance and can be used when SL-concavity is desired. However, using the PA to measure SP-concavity can result in an arbitrarily large error; see Figure 20. Thus, when SP-concavity is desired, concavity should be measured using the medial axis.

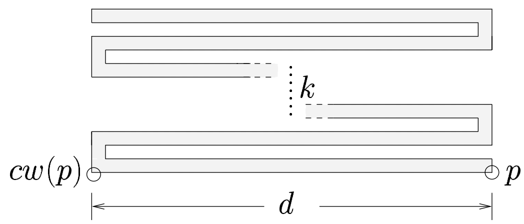



FIGURE 20. While the distance between the antipodal pair ($p$, $cw(p)$) computed using the principal axis is $d$, the diameter of the hole with $k$ turns is larger than $k \times d$. Note that $k$ can be arbitrarily large.

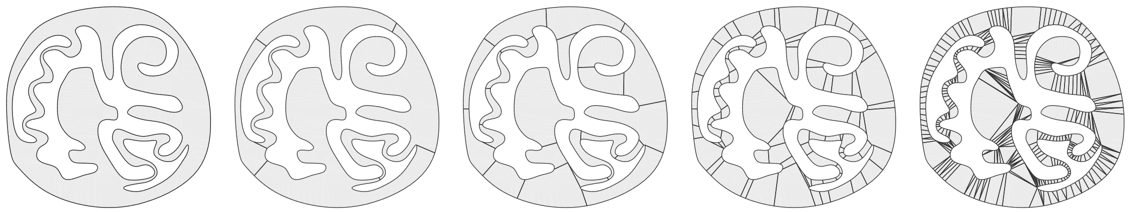
c. Measuring Hole Concavity

For a polygon with $k$ holes, we compute the antipodal pair, $p_i$ and $cw(p_i)$, for each hole $P_i$, $1 \leq i \leq k$. We use the antipodal pair of a hole to compute the concavity of the hole. The reason of using the antipodal pair is to reveal the largest possible concavity of the hole, thus revealing important features. A hole $P_i$ is resolved when a diagonal is added between $p_i$ and $\partial P_0$. Let $x$ be a vertex of $P$ closest to $p_i$ (or $cw(p_i)$) but not in $P_i$. Without loss of generality, assume $p_i$ is closer to $x$ than $cw(p_i)$. We define the concavity of a hole $P_i$ to be:

$$\text{concavity}(P_i) = \text{concavity}(x) + \text{dist}(x, p_i) + \text{dist}(p_i, cw(p_i)) + \delta \; . \qquad (4.2)$$

Since all vertices in a hole have infinite concavity, the term $\delta$ is defined as $concave(P_0)$ in Eqn. 4.2 to ensure that hole concavity is larger than the concavity of $P_0$, and $\text{concavity}(x) + \text{dist}(x, p_i)$ measures how "deep" the hole is from $\partial P_0$. If $x \in \partial P_0$, $\text{concavity}(x)$ is already known. Otherwise, $x$ is a vertex of a hole boundary $P_{j \neq i}$ and $\text{concavity}(x) = \text{concavity}(P_j)$.

Figure 21 shows an example of an ACD of a polygon with three holes.



(original) $(\tau = 5)$ $(\tau = 1)$ $(\tau = 0.1)$ $(\tau = 0)$

FIGURE 21. The original polygon has 816 vertices and 371 notches and three holes. The radius of the bounding circle is 8.14. When $\tau = 5$, 1, 0.1, and 0 units there are 4, 22, 88, and 320 components.

B. Resolving Concave Features

Given a polygon $P$, if the concavity of $P$ is above the maximum tolerable value, then the **Resolve**$(P, x)$ sub-routine in Algorithm 1 will resolve the concave feature at the vertex $x$ with the maximum concavity. A requirement of the **Resolve** subroutine is that if $x$ is on a hole boundary ($\partial P_i$, $i > 0$), then **Resolve** will merge the hole to the external boundary and if $x$ is on the external boundary ($\partial P_0$) then **Resolve** will split $P$ into exactly two components. See Algorithm 5 and Figure 22(a) and (b).

As described in Section A, the way we measure concavity and implement **Resolve** ensures that this is the case. For example, the concavity definition of the hole boundary in Eqn. 4.2 implies the order of resolution of the holes. An example is shown in Figure 23(b). Because $x$ is the closest vertex to $p_i$, the line segment $\overline{p_i x}$ will not intersect anything.

Our simple implementation of **Resolve** runs in $O(n)$ time. The process is applied recursively to all new components. The union of all components $\{C_i\}$ will be our final decomposition. The recursion terminates when the concavity of all components of $P$ is less than $\tau$. Note that the concavity of the features changes dynamically as the polygon is decomposed (see Figure 22(c)).

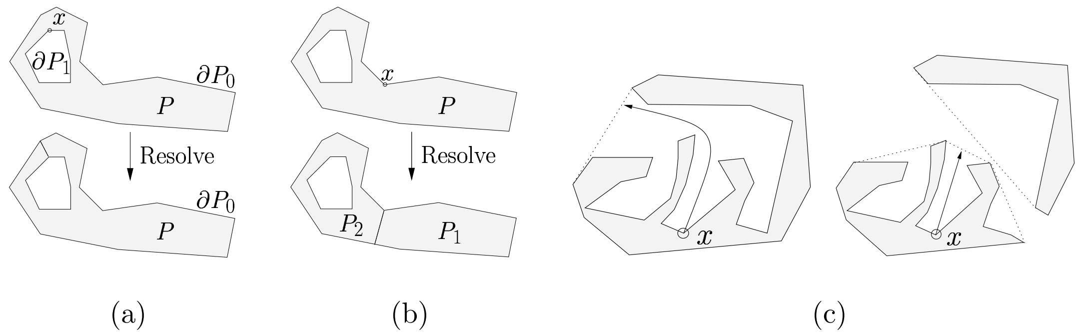



FIGURE 22. (a) If $x \in \partial P_{i>0}$, "Resolve" merges $\partial P_i$ into $P_0$. (b) If $x \in \partial P_0$, "Resolve" splits $P$ into $P_1$ and $P_2$. (c) The concavity of $x$ changes after the polygon is decomposed.

**Algorithm 5** Resolve($P$, $r$)

*Input.* A polygon, $P$, and a notch $r$ of $P$.
*Output.* $P$ with a diagonal added to $r$ so that $r$ is no longer a notch.

```
if r ∈ ∂P_0 then
    Add a diagonal \overline{rx} according to Eqn. 4.3, where x is a vertex in ∂P_0.
else
    Add a diagonal \overline{rx}, where x is the closest vertex to r in ∂P_0.
```

C. Correctness and Complexity Analysis

In this section, we will show that ACD will indeed produce 'more and more convex' components during the iterative decomposition process and will eventually produce an exact convex decomposition when the value of $\tau$ is set as zero. We will also show that ACD has $O(nr)$ time complexity, where $n$ and $r$ are the numbers of vertices and notches, respectively.

In Algorithm 1, we first find the most concave feature, i.e., the point $x \in \partial P$ with maximum concavity, and remove that feature $x$ from $P$. In this section, we show that $x$ must be a notch (Lemma C.2) and that if the tolerable concavity is zero then the result will be an exact convex decomposition, i.e., all notches must be removed (Lemma C.3). First, observe that if $x$ is a notch, then the concavity of $x$ must be larger than zero.

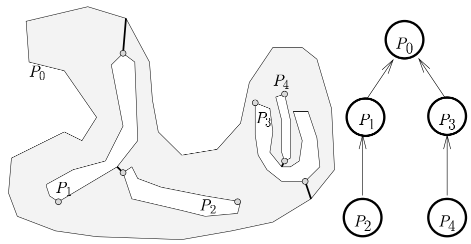



FIGURE 23. An example of hole resolution. Holes and the external boundary form a dependency graph which determines the order of resolution. In this case holes $P_1$ and $P_3$ will be resolved before $P_2$ and $P_4$. Dots on the hole boundaries are the antipodal pairs of the holes.

**Lemma C.1.** *If a point $r \in \partial P$ is a notch, then* concavity$(r)$ *is not zero.*

*Proof.* Each point $r$ on $\partial P$ is a (i) a point on the convex hull of $P$ (e.g., $r_1$ in Figure 24), (ii) a convex point, not on the convex hull of $P$ (e.g., $r_2$ in Figure 24), or (iii) a notch (e.g., $r_3$ in Figure 24). In case (i), then by definition concavity$(r) = 0$ and $r$ is not a notch. In all other cases, and in particular when $r$ is a notch, then concavity$(r) \neq 0$ ( since $r$ is not on $CH_P$, its distance to a bridge must be $> 0$).

□

**Lemma C.2.** *The concavity measures we have proposed (SL, SP, H1 or H2) are simple and stable. Hence, a point $x \in \partial P$ with maximum concavity, i.e., $\nexists y \in \partial P$ such that* $\text{dist}(y, CH_P) > \text{dist}(x, CH_P)$*, must be a notch.*

*Proof.* We first note that internal co-linear vertices do not contribute to the shape of $P$. Therefore, without loss of generality, all our algorithms and analysis assume such vertices do not exist (they can easily be removed in pre-processing), and hence we are guaranteed that no two consecutive vertices on $\partial P$ will have the same concavity.

We now show that SL-concavity and SP-concavity and our method for measuring the hole concavity are both simple and stable. We first consider SL-concavity. Assume $\beta$ is aligned along the $x$-axis. SL-concavity is stable because vertices are always

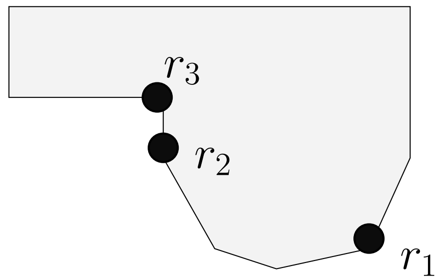



FIGURE 24. Point $r_1$ is on the boundary of the convex hull and points $r_2$ and $r_3$ are not. Point $r_3$ is a notch and points $r_1$ and $r_2$ are not.

retracted in the direction of the $y$-axis. Let $x$ be the lowest vertex on the $y$-axis. Since all vertices are above $x$, $x$ cannot have an internal angle less than 180°, i.e., $x$ must be a notch. Therefore, SL-concavity must also be simple. We next consider SP-concavity. Since all end points of the visibility tree are notches, resolving notches must reduce the concavity and will not affect the concavity of the remaining vertices. Thus, SP-concavity is simple and stable. For hole concavity, if we assume $\beta$ is perpendicular to the PA, then it is not difficult to see that hole concavity is similar to SL-concavity with the PA serving as the $y$-axis (i.e., the maximum concavity of a hole is the distance between the antipodal pair along the PA). Hence, hole concavity is also simple and stable. ☐

Although Algorithm 1 does not look for notches explicitly, Lemma C.2 establishes that Algorithm 1 indeed resolves notches and *only* notches.

In Lemma C.3, we show that Algorithm 1 resolves *all* notches when the tolerable concavity is zero. In this case, the approximate convex decomposition is an exact convex decomposition, i.e., $\mathrm{CD}_\tau(P)$ is equal to $\mathrm{CD}(P)$.

**Lemma C.3.** *Polygon $P$ is 0-convex if and only if $P$ is convex.*

*Proof.* If $P$ is convex, then $P$ has no notches. In this case, the concavity of $P$ is $\max_{x\in P}\{\text{concavity}(x)\} = \max_{x\in \partial P}\{\emptyset\} = 0$. Assume $P$ is not convex but that it has zero concavity. Since $P$ is not convex, $P$ has at least one notch. From Lemma C.1, we know that concavity($r$) $\neq 0$ and thus also concavity($P$) $\neq 0$. This contradiction establishes the lemma. □

Based on Lemma C.2 and Lemma C.3, we conclude our analysis of Algorithm 1 in Theorems C.4 and C.5.

**Theorem C.4.** *When $\tau = 0$, Algorithm 1 resolves* **all** *and* **only** *notches of polygon $P$ using the concavity measurements in Section A.*

*Proof.* By Lemma C.2, we know that ACD resolves only notches, and by Lemma C.3 that ACD resolves all notches when $\tau = 0$. □

**Theorem C.5.** *Let $\{C_i\}$, $i = 1, \ldots, m$, be a $\tau$-convex decomposition of a polygon $P$ with $n$ vertices, $r$ notches and $k$ holes. $P$ can be decomposed into $\{C_i\}$ in $O(nr)$ time.*

*Proof.* We first consider the case in which $P$ has no holes, i.e., $k = 0$. We will show that each iteration in Algorithm 1 takes $O(n)$ time. For each iteration, we compute the convex hull of $P$ and the concavity of $P$. The convex hull of $P$ can be constructed in linear time in the number vertices of $P$ [97]. To compute the concavity of $P$, we need to find bridges and pockets and compute the distance from the pockets to the bridges. Associating the bridges and pockets requires $O(n)$ time using a traversal of the vertices of $P$. When the shortest path distance is used, measuring concavity($P$) takes linear time as shown in Lemma A.4. When the straight line distance is used, each measurement of concavity($x$) takes constant time, where $x$ is a vertex of $P$. Therefore, the total time for measuring concavity($P$) takes $O(n)$ as well. Similarly, we can show that the hybrid approach takes $O(n)$ time. Moreover, **Resolve** splits $P$

into $C_1$ and $C_2$ in $O(n)$ time. Thus, each iteration takes $O(n)$ time for $P$ when $P$ does not have holes.

If the resulting decomposition has $m$ components, the total number of iterations of Algorithm 1 is $m-1$. Since each time we split $P$ into $C_1$ and $C_2$, at most three new vertices are created, the total time required for the $m-1$ cuts is $O(n+(n+3)+\ldots+(n+3*(m-2))) = O(nm+3\times\frac{(m-1)^2}{2}) = O(nm+m^2)$.

When $k>0$, we estimate the concavity of a hole locally using its principal axis ($O(n)$ time) and add a diagonal between the vertex with the maximum estimated concavity and its closest vertex of $\partial P$ ($O(n)$ time). For each hole that connects to $\partial P$, at most three new vertices are created. Therefore, resolving $k$ holes takes $O(nk+k^2)$ time.

Therefore, the total time required to decompose $P$ into $\{C_i\}$ is $O(nm+m^2)+O(nk+k^2) = O(n(m+k)+m^2+k^2)$ time. Since $m\leq r+1$ and $k<r$, $O(n(m+k)+m^2+k^2) = O(nr+r^2)$. Also, because $r<n$, $O(nr+r^2) = O(nr)$. Thus, decomposition takes $O(nr)$ time. $\square$

The number of components in the final decomposition, $m$, depends on the tolerance $\tau$ and the shape of the input polygon $P$. A small $\tau$ and an irregular boundary will increase $m$. However, $m$ must be less than $r+1$, the number of notches in $P$, which, in turn, is less than $\lfloor\frac{n-1}{2}\rfloor$. Detailed models, such as the Nazca line monkey and heron in Figures 13 and 27, respectively, generally have $r$ close to $\Theta(n)$. In this case, Chazelle and Dobkin's approach [32] has $O(n+r^3) = O(n^3)$ time complexity and Keil and Snoeyink's approach [71] has $O(n+r^2\min\{r^2,n\}) = O(n^3)$ time complexity. When $r=\Theta(n)$, Algorithm 1 has $O(n^2)$ time complexity.

## D. Experimental Results

In this section, we compare the final decomposition size and the execution time of the approximate convex decomposition (ACD) computed using different concavity measures and with the minimum component exact convex decomposition (MCD) [71]. We observe that ACD is significantly faster and produces fewer components when $\tau > 0$ and ACD remains significantly faster when $\tau = 0$. We also observe that, for models with the same shape but with different complexity, ACDs of these models remain very similar, i.e., ACD is not very sensitive to the complexity of the models with the same shape. We also compare the results and efficiency of ACDs computed with different types of concavity measures. We see that ACD with SL-concavity is the most efficient. We observe the same benefits (small size and high efficiency) for ACD of polygons with holes. Finally, we show that ACD can generate visually meaningful components.

### 1. Implementation Details

We implemented the proposed algorithm in C++, and used FIST [54] as the triangulation subroutine for finding the shortest paths in pockets. Instead of resolving a notch $r$ using a diagonal that bisects the dihedral angle of $r$, we use a heuristic approach intended to appeal to human perception. When selecting the diagonal for a particular notch $r$, we consider all possible diagonals $\overline{rx}$ from $r$ to a boundary point $x \in \partial P_0$. All diagonals are scored using the scoring function

$$f(r,x) = \begin{cases} 0 & : \ \overline{rx} \text{ does not resolve r} \\ \frac{(1+s_c\times \text{concavity}(x))}{(s_d\times \text{dist}(r,x))} & : \ \text{otherwise, where } \mathrm{s_c} \text{ and } \mathrm{s_d} \text{ are user defined scalars} \end{cases} \tag{4.3}$$

and the highest scoring one is selected as the diagonal for resolving $r$.

According to experimental studies [120], people prefer short diagonals to long diagonals. Thus, in addition to the concavity, we consider the distance as another criterion when selecting the diagonal to resolve $r$. Increasing $s_c$ favors concavity and increasing $s_d$ places more emphasis on the distance criterion. In our experiments, we found that by favoring shorter diagonal we can generate visually more meaningful components, therefore $s_c = 0.1$ and $s_d = 1$ are used. This scoring process adds $O(n)$ time to each iteration and therefore does not change the overall asymptotic bound.

## 2. Models

The polygons used in the experiments are shown in Figures 25–33. Summary information for these models is shown in Table 2. The model in Figure 29 has 18 holes and all the other models have no holes. The models in Figure 26 and 27 are referred to as $monkey_1$ and $heron_1$, respectively. Two additional polygons, with the same size and shape as $monkey_1$ and $heron_1$, are called $monkey_2$ and $heron_2$.

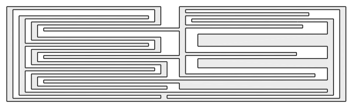
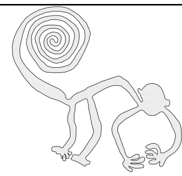
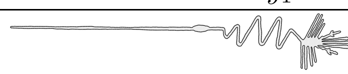
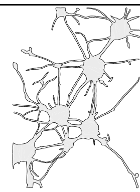
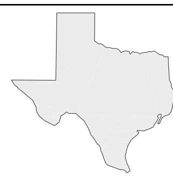
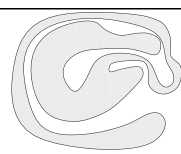
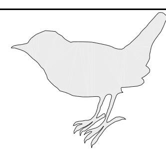
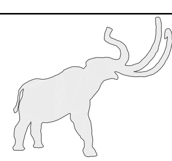
## 3. Results

All experiments were done on a Pentium 4 2.8 GHz CPU with 512 MB RAM. For a fair comparison, we re-coded the MCD implementation available at [122] from Java to C++. To provide an additional metric for comparison, we estimate the *quality* of the final decomposition $\{C_i\}$ by measuring its *convexity* [136]:

$$convex(\{C_i\}) = \frac{\sum_i \mathrm{area}(C_i)}{\sum_i \mathrm{area}(CH_{C_i})} \ , \tag{4.4}$$

where area$(x)$ is the area of an object $x$ and $CH_x$ is its convex hull. Eqn. 4.4 provides a *normalized* measure of the similarity of the $\{C_i\}$ to their convex hulls. Thus, unlike our concavity measurements, this convexity measurement is independent of the size, i.e., area, of polygons. For example, a set of convex objects will have convexity 1

Table 2— Summary information for models studied. In this table, $|v|$, $|r|$ and $|h|$ are the number of vertices, notches and holes, respectively, and $R$ is the radius of the minimum enclosing ball

| Name | Figure | $\|v\|$ | $\|r\|$ | $\|h\|$ | $R$ (units) |
|---|---|---|---|---|---|
| maze |  | 800 | 400 | 0 | 15.3 |
| $monkey_1$ |  | 1204 | 577 | 0 | 81.7 |
| $monkey_2$ | Same as $monkey_1$ | 9632 | 4787 | 0 | 81.7 |
| $heron_1$ |  | 1037 | 484 | 0 | 137.1 |
| $heron_2$ | Same as $heron_1$ | 8296 | 4122 | 0 | 137.1 |
| neuron |  | 1815 | 991 | 18 | 19.6 |
| texas |  | 139 | 62 | 0 | 17.4 |
| deep cave |  | 348 | 153 | 0 | 12.9 |
| bird |  | 275 | 133 | 0 | 15.4 |
| Mammoth |  | 403 | 185 | 0 | 16.5 |

regardless of their size.

### a. ACD Is Significantly Faster and Produces Fewer Components When $\tau > 0$

A general observation from our experiments is that when a little non-convexity can be tolerated, the ACD may have significantly fewer components and it may be computed significantly faster; see Table 3. For example, in Figure 25, by sacrificing 0.005 convexity, i.e., with $\tau = 0.1$, the ACD generates only 25% as many components as the MCD and it is almost 8 times faster. In Figure 26, by sacrificing 0.003 convexity, i.e., with $\tau = 0.1$, the ACD has 8/10 the components of the MCD and it is 6.3 times faster. By sacrificing 0.06 convexity, i.e., with $\tau = 1$, the ACD has 1/4 the components of the MCD and it is 10 times faster. In Figure 27, by sacrificing 0.02 convexity, i.e., with $\tau = 0.1$, the ACD has about 1/2 the components of the MCD and it is 7.6 times faster.

Similar observations can be found in the results for the larger monkey and heron models (Figures 26 and 27). For example, for the monkey, the radius of its bounding circle is about 82, and so 0.1 concavity means a one pixel dent in an $820 \times 820$ image, which is almost unnoticeable to the naked eye. Moreover, the convexity of 0.1-convex components of $monkey_1$ ($monkey_2$) is 0.997 (0.995) and the convexity of 0.1-convex components of $heron_1$ ($heron_2$) is 0.98 (0.976). No MCD data is collected for $monkey_2$ and $heron_2$ due to the difficulty of solving these large problems with the MCD code.

### b. ACD Is Always Faster When $\tau = 0$

We also observe that, when exact convex decomposition is needed ($\tau = 0$), our method does produce somewhat more components than the MCD (on average, 1.2 to 1.5 times more than ECD), but it is also always faster than ECD, especially when the size of the model is large. See Table 3.

TABLE 3— Comparing the decomposition size and time of the ACD and the MCD. Convexity and concavity in this table indicate the tolerance of the ACD. Note that $monkey_2$, $heron_2$ and neuron are not listed here because MCD does not work on these models.

| Name | concavity $\tau$ (units) | convexity (unitless) | size (ACD:MCD) | time (ACD:MCD) |
|---|---|---|---|---|
| maze | 0.1 | 99.5% | 1.0:4.0 | 1.0:8.0 |
| | 0.0 | 100.0% | 1.3:1.0 | 1.0:6.0 |
| $monkey_1$ | 0.1 | 99.7% | 8.0:10 | 1.0:6.3 |
| | 0.0 | 100.0% | 1.3:1.0 | 1.0:5.1 |
| $heron_1$ | 0.1 | 98.0% | 1.0:2.0 | 1.0:7.6 |
| | 0.0 | 100.0% | 1.4:1.0 | 1.0:5.9 |
| texas | 0.1 | 98.0% | 1.0:5.0 | 1.0:2.0 |
| | 0.0 | 100.0% | 1.5:1.0 | 1.0:2.0 |
| deep cave | 0.1 | 98.0% | 1.0:8.0 | 1.0:2.7 |
| | 0.0 | 100.0% | 1.2:1.0 | 1.0:1.3 |
| bird | 0.1 | 98.0% | 1.0:7.5 | 1.0:6.7 |
| | 0.0 | 100.0% | 1.4:1.0 | 1.0:1.6 |
| mammoth | 0.1 | 98.0% | 1.0:8.0 | 1.0:7.8 |
| | 0.0 | 100.0% | 1.4:1.0 | 1.0:2.7 |

### c. ACD of Models with the Same Shape but Different Complexity

This experiment, shown in Figure 28, reveals another interesting property of the ACD: regardless of the complexity of the input, the ACD generates almost identical decompositions for models with the same shape when $\tau$ is above a certain value. For example when $\tau > 0.01$, ACD generates the same number of components for both $monkey_1$ and $monkey_2$ and for $heron_1$ and $heron_2$.

### d. Differences among the Concavity Measures

The maze-like model (Figure 25) illustrates differences among the concavity measures. When $\tau \geq 10$, the convexity measurements in Figure 25(d) show that SL-concavity misses some important features that are found by SP-concavity (and thus also by H1-

concavity and H2-concavity). When $\tau$ is less than 5, the SL-concavity measurement has similar output as SP-concavity and hybrid measurements. In Figure 25(c), we also see that SP-concavity is more expensive to compute and that H2-concavity is "shape" sensitive, i.e., H2-concavity requires more (less) time if the input shape is complex (simple). Computing H2-concavity is also faster than computing H1-concavity.

e. ACD of Holes

We also observe that the ACD of polygons with holes can be generated efficiently as ACD of polygons without holes. A polygonal model of planar neuron contours is shown in Figure 29. It has 18 holes and roughly 45% of the vertices are on hole boundaries. Figure 29(b) shows the decomposition using the proposed hole concavity and SP-concavity measures. The dashed line (at $Y = 0.06$) in Figure 29(c) is the total time for resolving the 18 holes. Once all holes are resolved, the ACD produces similar results as before. No MCD was computed because the algorithm cannot handle holes.

f. ACD Generates Visually Meaningful Components

The ACD also generates visually meaningful components, such as legs and fingers of the monkey in Figure 13 and wings and tails of the heron in Figure 27. More results that demonstrate this property are shown in Figures 30 to 33. The main reason for generating visually meaningful components is that ACD decomposes the models at high concavity areas, which is usually the most dented or bent area, or an area with branches. Experimental evidence indicates that these areas are the places that humans decompose shapes into components [14, 95, 117, 120] for shape recognition.

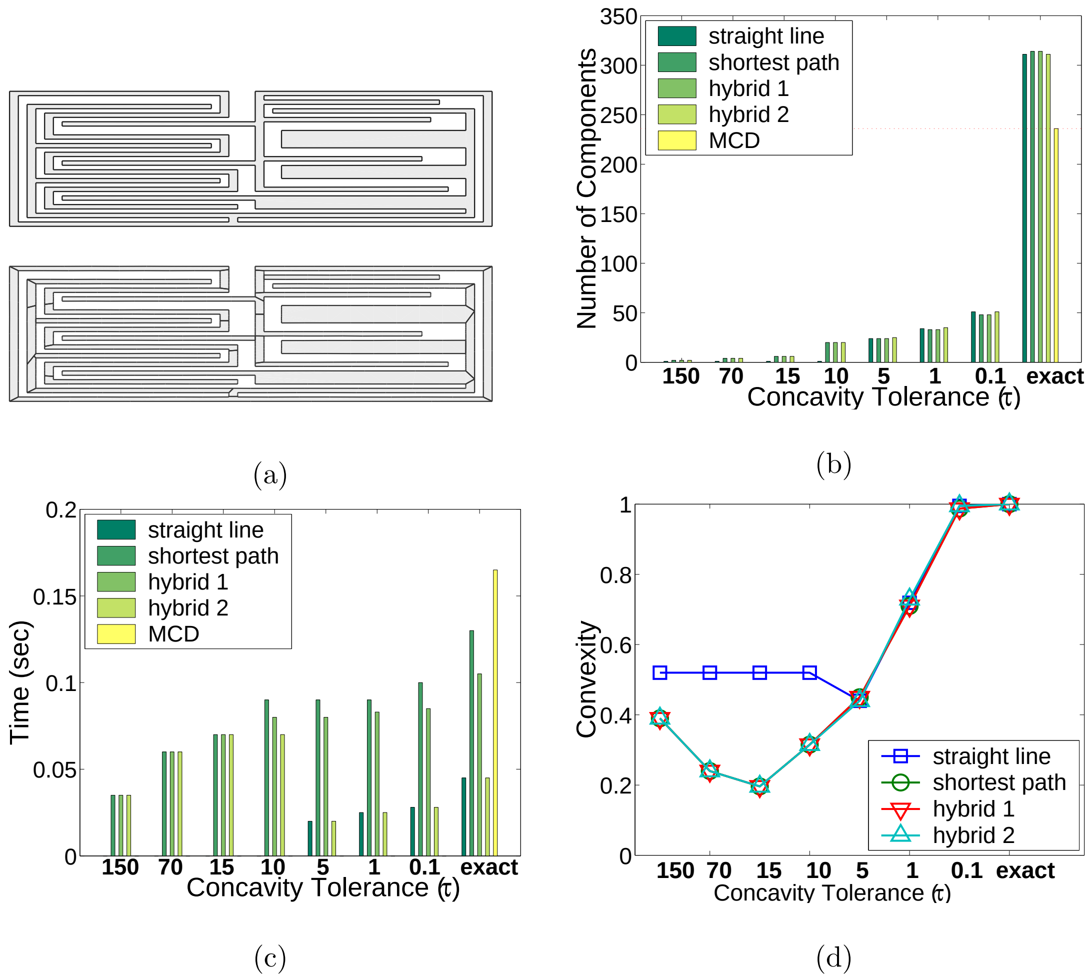



FIGURE 25. (a) Initial (top) and approximately (bottom) decomposed Maze models. The initial Maze model has 800 vertices and 400 notches. (b) Number of components in final decomposition. (c) Decomposition time. (d) Convexity measurements.

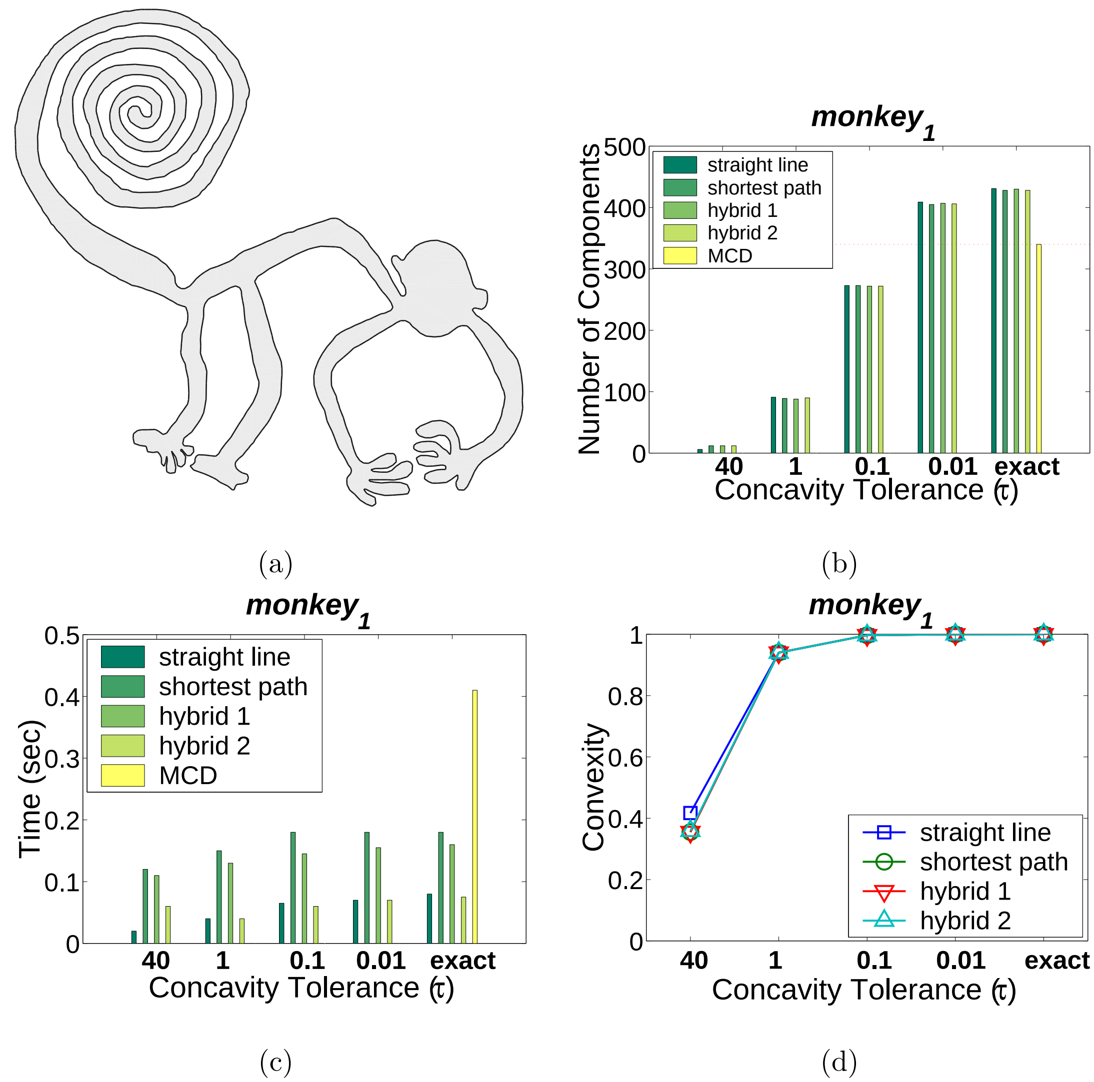
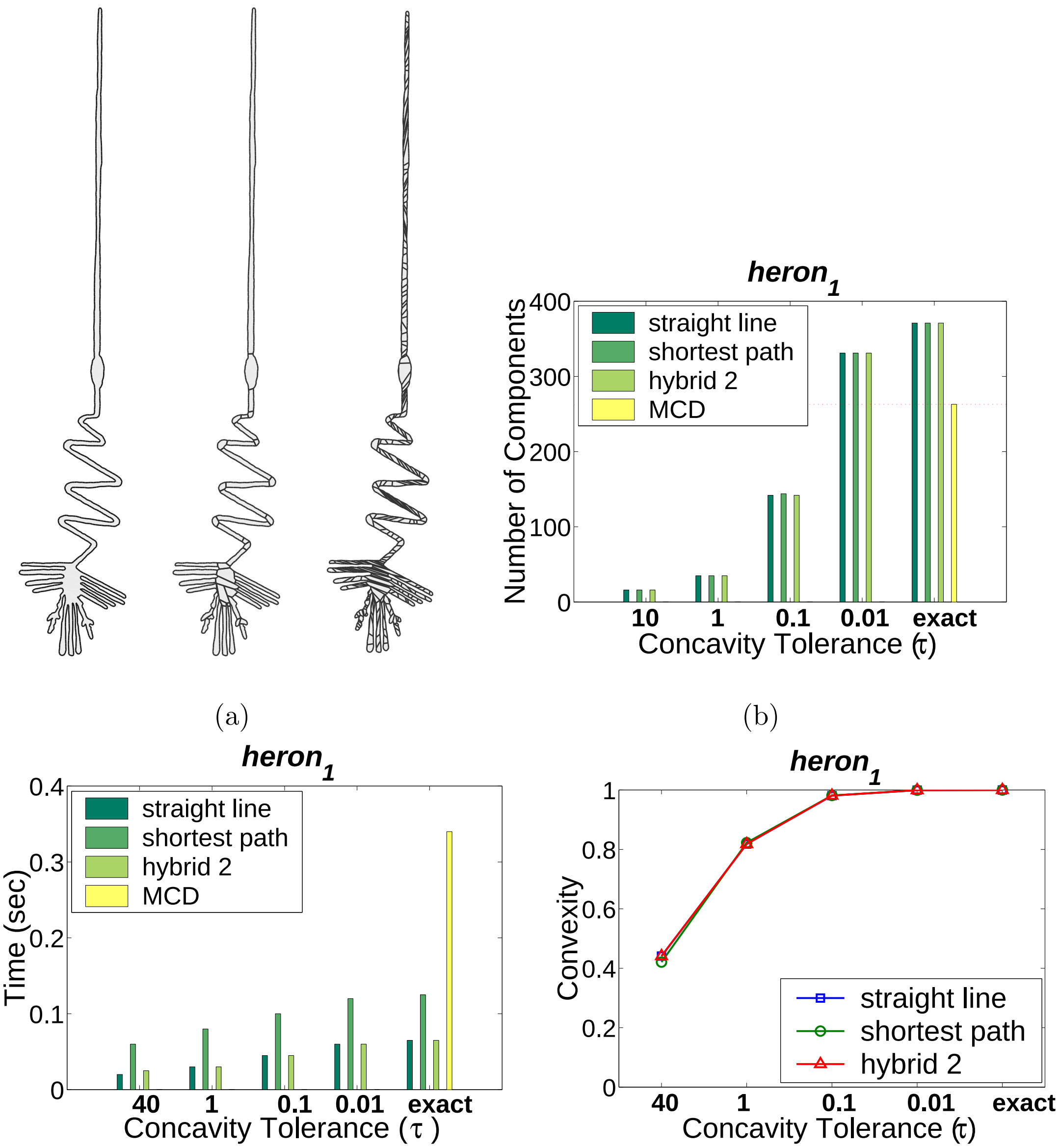
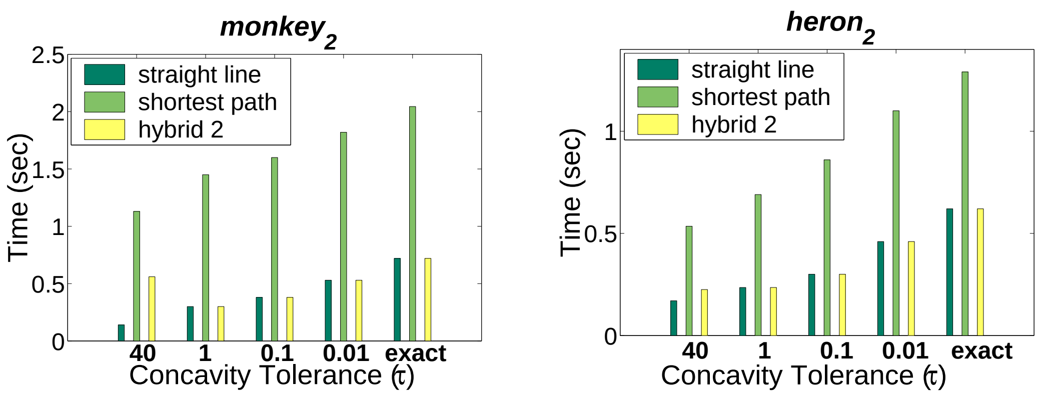



FIGURE 26. (a) Initial model of Nazca Monkey; see Figure 13. (b) Number of components in final decomposition. (c) Decomposition Time. (d) Convexity measurements.

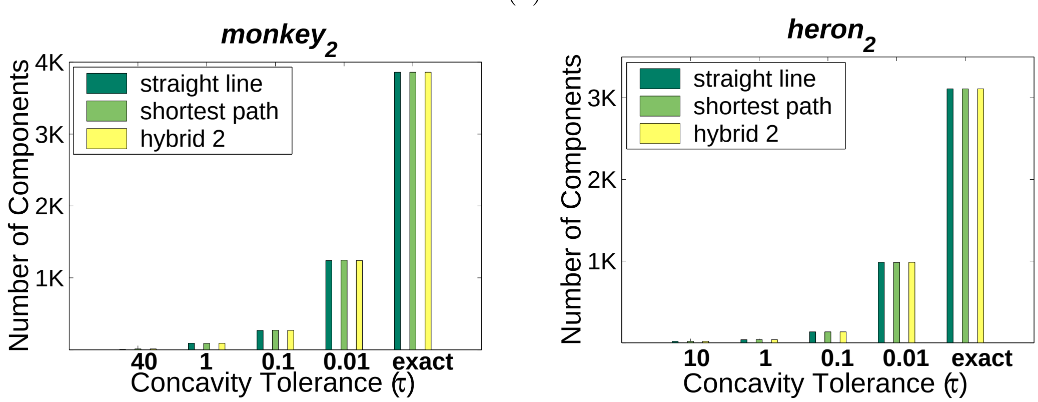



(a) (b) (c) (d)

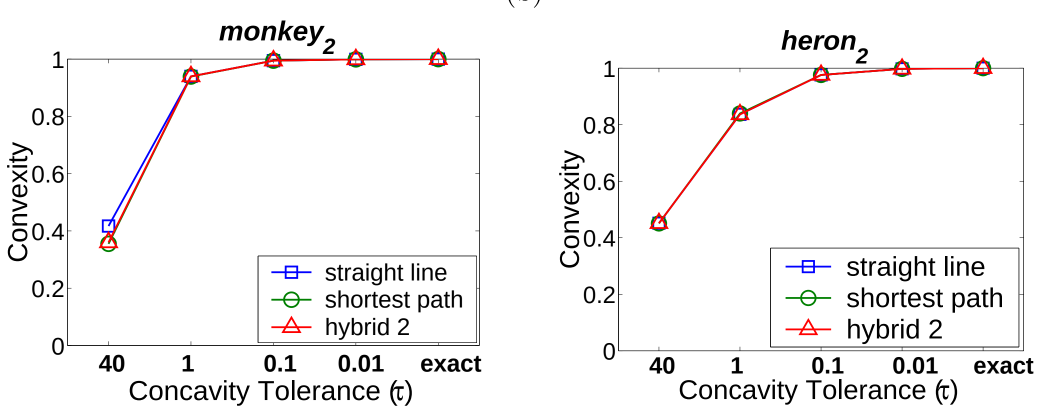
FIGURE 27. (a) Top: The initial Nazca Heron model bounding circle is 137.1 units. Middle: Decomposition using approximate convex decomposition. 49 components with concavity less than 0.5 units are generated. Bottom: Decomposition using optimal convex decomposition. 263 components are generated. (b) Number of components in final decomposition. (c) Decomposition time. (d) Convexity measurements.




(a)




(b)




(c)

FIGURE 28. Left: *monkey*$_2$. Right: *heron*$_2$. (b) Number of components in final decomposition. (c) Decomposition time. (d) Convexity measurements.

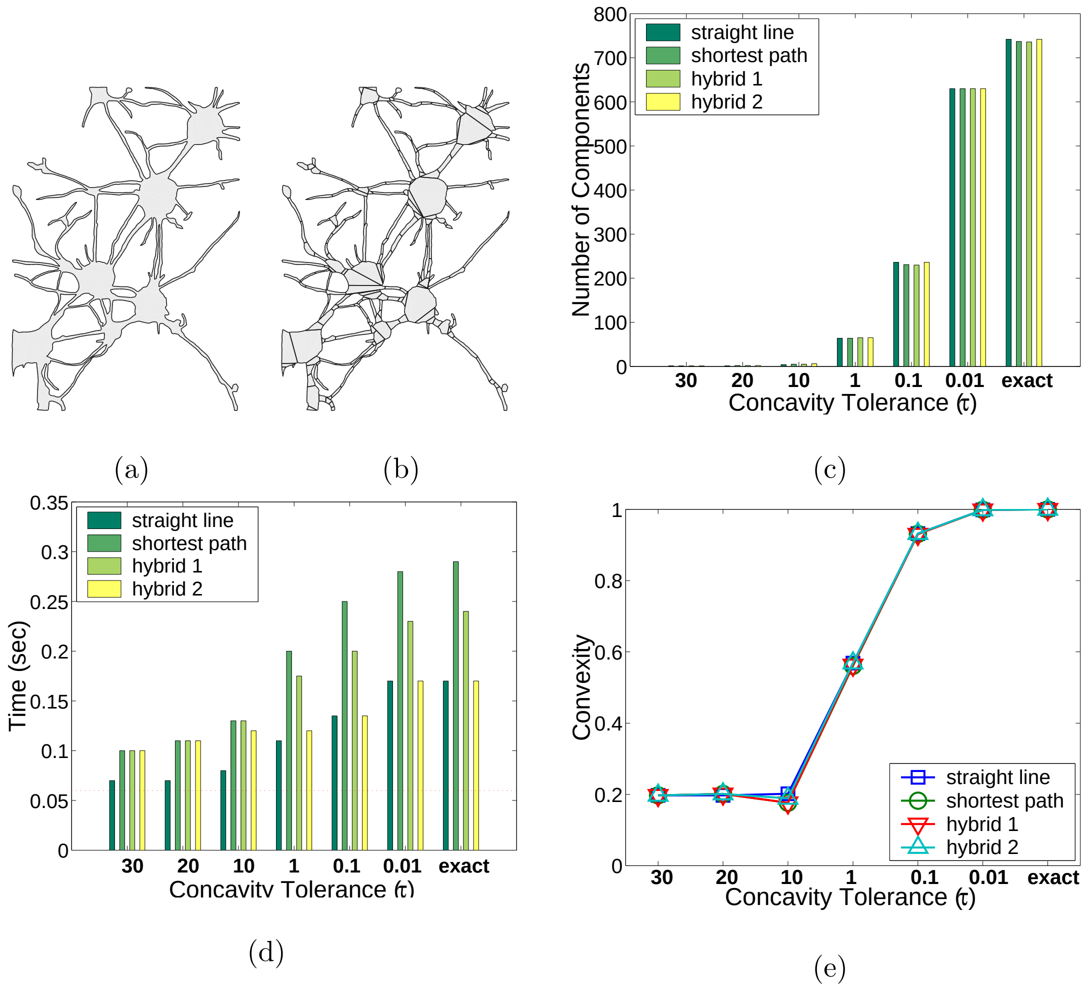
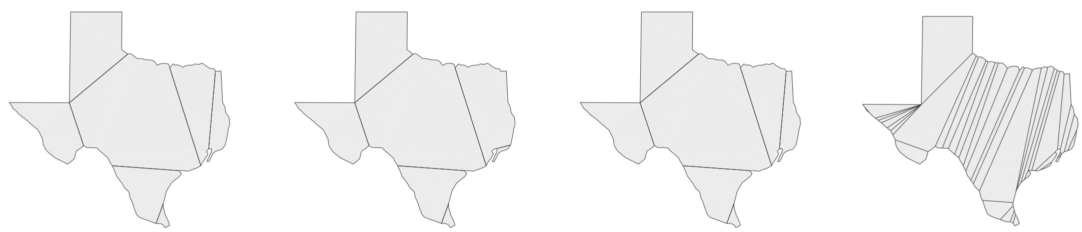
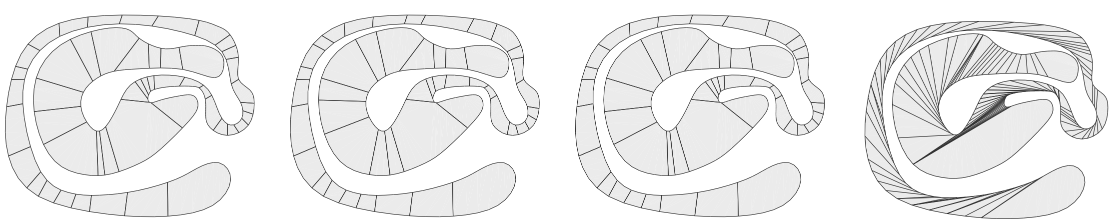



FIGURE 29. (a) The initial model of neurons has 1,815 vertices and 991 notches and 18 holes. The radius of the enclosing circle is 19.6 units. (b) Decomposition using approximate convex decomposition. Final decomposition has 236 components with concavity less than 0.1 units. (c) Number of components in final decomposition. (d) Decomposition Time. The dashed line indicates the time for resolving all holes. (e) Convexity measurements.



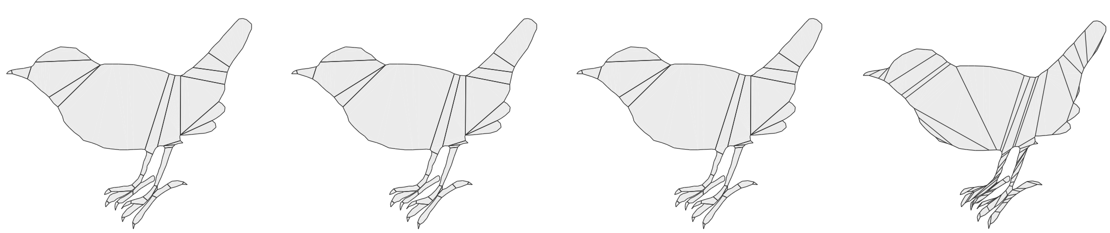
(SL, size=7) (SP, size=7) (H2, size=7) (MCD, size=38)

Figure 30. Texas. Approximate components are 1-convex.



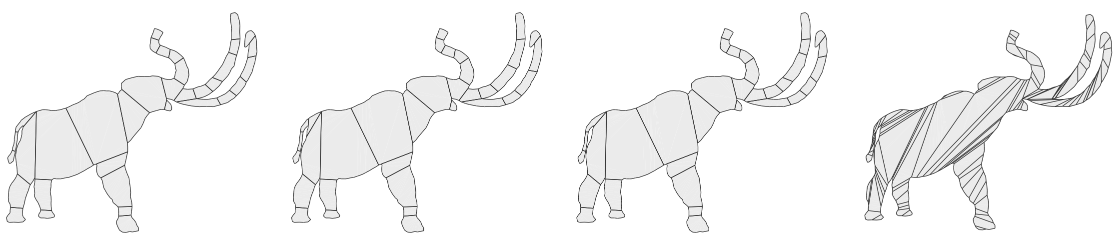
(SL, size=49) (SP, size=48) (H2, size=49) (MCD, size=126)

Figure 31. Deep cave. Approximate components are 0.1-convex.



(SL, size=37) (SP, size=37) (H2, size=37) (MCD, size=75)

Figure 32. Bird. Approximate components are 0.1-convex.



(SL, size=35) (SP, size=34) (H2, size=34) (MCD, size=105)

FIGURE 33. Mammoth. Approximate components are 0.2-convex.

# CHAPTER V

## APPROXIMATE CONVEX DECOMPOSITION OF POLYHEDRA

In this chapter, we describe practical methods for computing a *solid* ACD of a polyhedron of arbitrary genus, which consists of a collection of nearly convex volumes whose union equals the original polyhedron, and a *surface* ACD of a polyhedral surface, which partitions the surface of the polyhedron into a collection of nearly convex surface patches. Solid and surface ACD of polyhedra have many potential applications including shape representation (Figure 3), motion planning (Figure 4), mesh generation (Figure 5), and point location (Figure 2).

Similar to 2D ACD, 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 until they are convex 'enough.' While this follows the general approach used successfully for polygons, there are several operations that were relatively straightforward for polygons but which become nontrivial for polyhedra. The main challenges include computing the concavity of features efficiently 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*, which enables sets of features to be processed together, which is both more efficient and produces better results.

As mentioned in Chapter II, convex decomposition of polyhedra is not as well understood as polygons and little research on the convex decomposition of polyhedra has gone beyond the theoretical stage. Using the simple notch-cutting strategy, Chazelle [29] shows that this strategy can generate the worst case optimal $O(r^2)$ convex parts and uses $O(nr^3)$ time with $O(nr^2)$ space, where $n$ and $r$ are the number of edges and notches, respectively. In contrast, even for very complex models, ACDs have very few

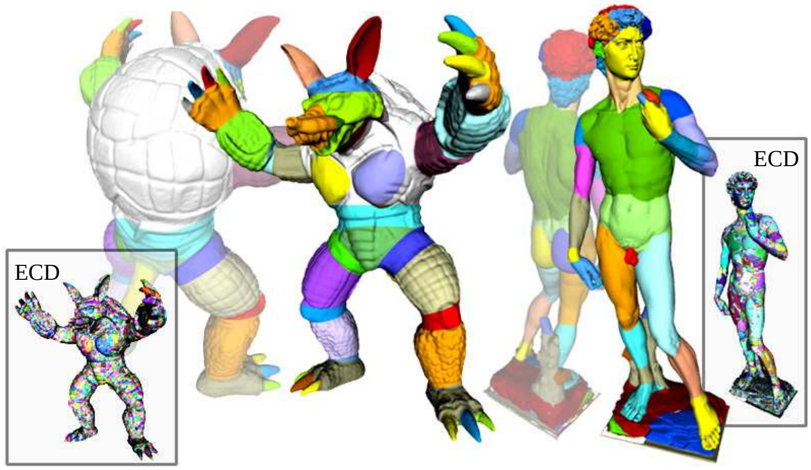



FIGURE 34. The approximate convex decompositions (ACD) of the Armadillo and the David models consist of a small number of nearly convex components that characterize the important features of the models better than the exact convex decompositions (ECD) that have orders of magnitude more components. The 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).

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. In this chapter, we demonstrate the feasibility of our approach by applying it to a number of complex models; see Figure 34 and the table on p. 88.

ACD of polyhedra follows the same framework described in Algorithm 1 to decompose a polyhedron $P$ into a set of $\tau$-convex components. As in Chapter IV, we will discuss two main sub-routines required by Algorithm 1, i.e., measuring and resolving of concave features of polyhedra. In Section A, we describe several challenges of extending the concavity measures and resolution proposed for polygons to three-dimensions. We then describe ACD for genus zero polyhedra (Section B) and then for polyhedra of arbitrary genus (Section C). Finally, we present results in Section D.

## A. Challenges in Extending to Three Dimensions

Recall that, for a given polygon, ACD computes the concavity of the polygon using SL-, SP-, or H-concavity. Then, ACD resolves the polygon by adding a diagonal at the notch with the maximum concavity. While these operations were straightforward for polygons, they become nontrivial for polyhedra. In this section, we discuss the challenges of measuring and resolving concave features of polyhedra.

### 1. Measuring Concave Features

The bridges and pockets of a polygon have a unique one to one map. Therefore, the concavity of the vertices of a pocket can be measured as the distances to the uniquely associated bridge. The unique mapping 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 [105] and simplical [73] parameterization. However, mesh parameterization is costly to compute. Polyhedron realization [112] 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.

### 2. Resolving Concave Features

A polygon with untolerable concavity is resolved by adding a diagonal at the most concave feature (notch). This strategy is called *notch-cutting*, and can be easily extended to 3D. The notch-cutting strategy [27] that splits a polyhedron with a cut

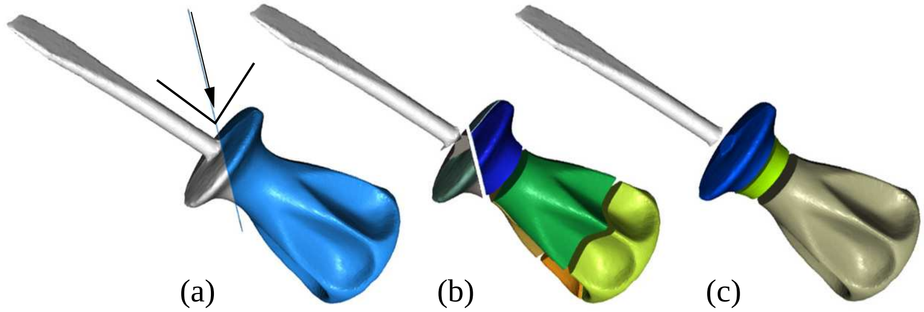



FIGURE 35. Resolving concavity (a) using a cut plane that bisects a dihedral angle results in (b) a decomposition with 10 components with concavity $\leq 0.1$. In contrast, (c) carefully selected cut planes generate only 4 components with concavity $\leq 0.1$.

plane can be used to resolve notches in Algorithm 1. The details of this notch-cutting strategy are discussed in [11]. Figures 35(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 35(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 [65] 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. 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. Figure 36 shows an example of the bridge/pocket relationship with and without grouping. As we will see in Section 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 2. the quality of the decomposition can be greatly improved when the cut plane is defined with respect to a notch set.

## B. 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.

### 1. Measuring Concave Features

Recall that we define the concavity of a vertex $x$ as the distance from $\partial P$ 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 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 $\partial P$ connecting the end points of $e$; we compute the paths on $\partial P$ using Dijkstra's algorithm. After the convex hull edges are

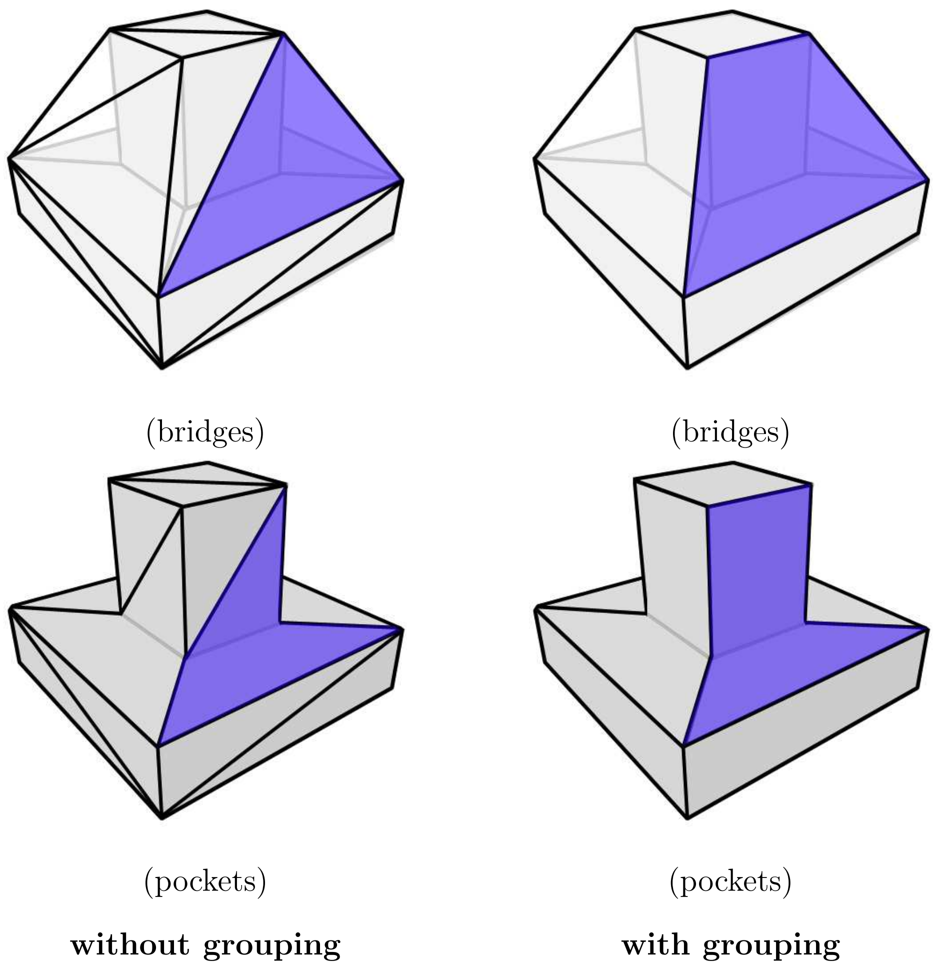



FIGURE 36. The bridges and the pockets with and without bridge grouping (clustering).

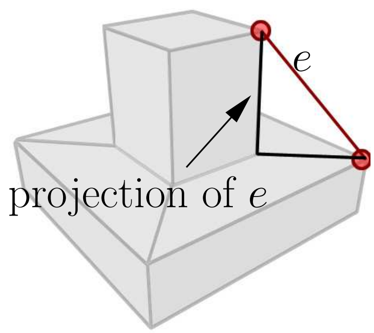
projected, the set of all (connected) polyhedral facets bounded by the projected edges forms a pocket. See Figure 37. After matching bridges with pockets, we measure the concavity of $x$ in pocket $\rho$ as the straight line distance to the tangent plane of $\rho$'s associated bridge $\beta$.

**Feature grouping: bridge patches – a conservative estimation**. Finding pockets for all facets in $\partial CH_P$ 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 $\partial CH_P$. We then designate a *"supporting" plane* that is tangent to $\partial CH_P$ 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 38. 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 $\epsilon$. For example, when $\epsilon = 0.05$, only 20 bridges are identified for the model in Figure 37 which has 4,626 facets on its convex hull.

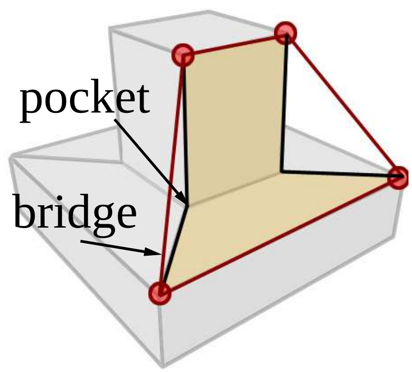
One way to compute bridge patches is from an outer approximation of a polyhedron. Here we use *Lloyd's* clustering algorithm adapted from [38] to identify bridges and to ensure that the maximum distance from the included facets to the supporting plane is less than $\epsilon$. Our clustering process is 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.

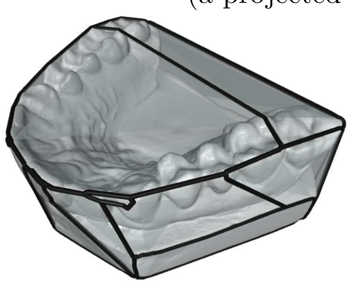
In the first step, we estimate the required bridge size for a given threshold $\epsilon$ by incrementally creating bridges and assigning convex hull facets to the bridges until all the convex hull facets are assigned. We say that a facet can be assigned to a bridge if



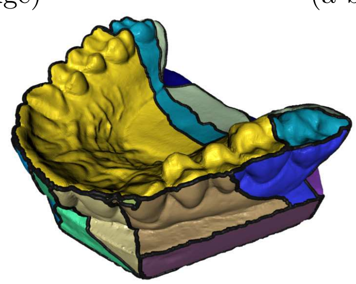

(a projected edge)



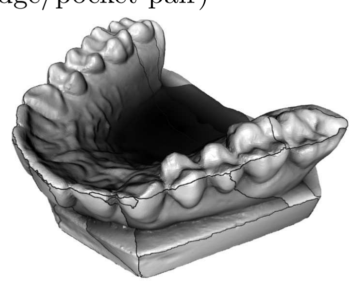

(a bridge/pocket pair)



(bridges)



(pockets)



(concavity)

FIGURE 37. 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.

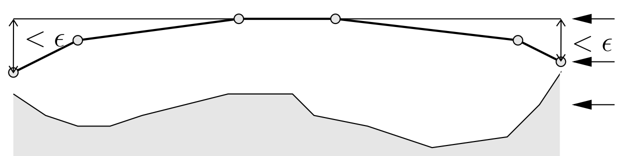



FIGURE 38. A bridge patch and its supporting plane.

---

**Algorithm 6** CH_cluster_size_estimation($CH_P$,$\epsilon$)

---

*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 $B$

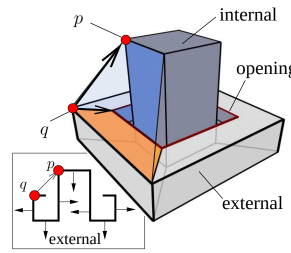
---

the distance between them is less than $\epsilon$. Let $C(\beta)$ be a set of connected facets that can be assigned to the bridge $\beta$. Our estimation process is outlined in Algorithm 6.

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 [38], 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 B.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 B.1.** *The estimated concavity measurement is always greater than, in an amount less than $\epsilon$, or equal to the concavity measured as convex hull facets are projected individually.*




**Polygonal surface 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 $\beta$ from the projection of $\beta$'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.

## 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.

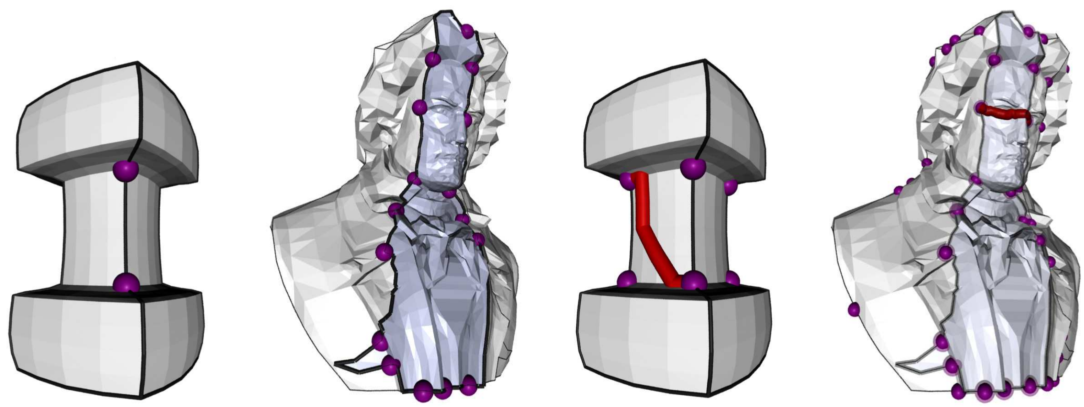
Our strategy of grouping concave features is a bottom-up approach in which critical points, called "*knots*", on the boundary of each pocket are connected into local feature sets, called "*pocket cuts*", which are then grouped to form global feature sets, called "*global cuts*". This bottom-up approach attempts to (i) avoid high computational complexity, e.g., grouping features based on the solution of a maximum flow problem [66] on the full surface $\partial P$, (ii) avoid enhancing feature quality [80], and (iii) avoid using other processes, e.g., mesh simplification, to enhance features. Our approach is illustrated in Figure 39 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 [55] with simplification threshold $\delta$, $0 \leq \delta \leq \tau$.
2. *Computing pocket cuts.* A pocket cut is a chain of consecutive edges in a pocket $\rho$ whose removal will bisect $\rho$. Here, pocket cuts are paths connecting pairs of knots, and we consider all knot pairs for $\rho$.
3. *Weighting cuts.* The weight of a cut determines the quality of the cut. We compute the weight of each pocket cut $\kappa$ 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 $\kappa$ and $\gamma(\kappa)$ is the accumulated curvature of the edges in $\kappa$. The curvature of an edge $e$ is measured using the *best fit polynomial* [63].
4. *Connecting pocket cuts into global cuts.* Our strategy is to organize the knots and pocket cuts in a graph $G_{\mathcal{K}}$ whose vertices are knots and edges are pocket cuts. The cycle with the minimum weight in $G_{\mathcal{K}}$ will be the global cut.

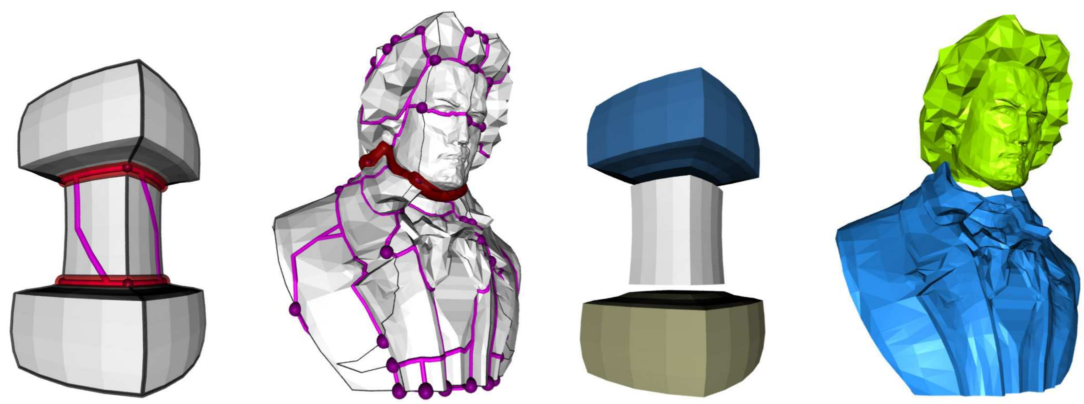
Next, we will provide more details and justify the choices of the steps mentioned above.

### a. 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 $\pi_e$, we also consider a projected bridge edge as a critical representation of a polyhedral boundary. Note



(a) identifying knots (b) computing pocket cuts



(c) extracting global cuts (d) splitting the model

FIGURE 39. 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 $G_{\mathcal{K}}$. (d) Solid (left) and surface (right) decompositions using the identified global cuts.

**Algorithm 7** DP($L$,$\delta$)

*Input.* A polygonal chain, $L = \{v_1, v_2, \cdots, v_n\}$, and threshold, $\delta$.
*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($\{v_1, \cdots, v_k\}$,$\delta$), $v_k$, DP($\{v_k, \cdots, v_n\}$,$\delta$) $\}$

that the end points of $\pi_e$ are both vertices of the convex hull. Intuitively, the vertices of $\pi_e$ are *samples* of $\partial P$ and therefore encode important geometric features related to concavity over the traversal from one *peak* to another *peak* i.e., $\pi_e$ is an evidence that shows how the convex hull vertices are connected on $\partial P$.

b. Identifying Knots

The Douglas-Peucker (DP) line approximation algorithm is shown to be good at revealing critical points [128] and is used to identify knots. Let $L$ be a polygonal chain composed of $n$ vertices $\{v_1, v_2, \cdots, v_n\}$. For a given threshold $\delta$, the DP algorithm produces a simplification of $L$, called $L'$. Algorithm 7 outlines a simple version of the algorithm. A more efficient approach can be found in [55].

Using DP simplification to identify knots is natural for our purposes because it resembles the concept of ACD. A critical point (resp., a knot) of a polyline $\pi$ is a farthest point from the line segment (resp., the bridge) connecting the end points of $\pi$. This provides an explanation of why we can extract important concave features by simplifying $\pi_e^*(i)$. See Figure 40.

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:

$$\pi_e^*(i) = \big(d_i, \text{concavity}(\pi_e(i))\big),\ 0 \le i \le 1, \tag{5.1}$$

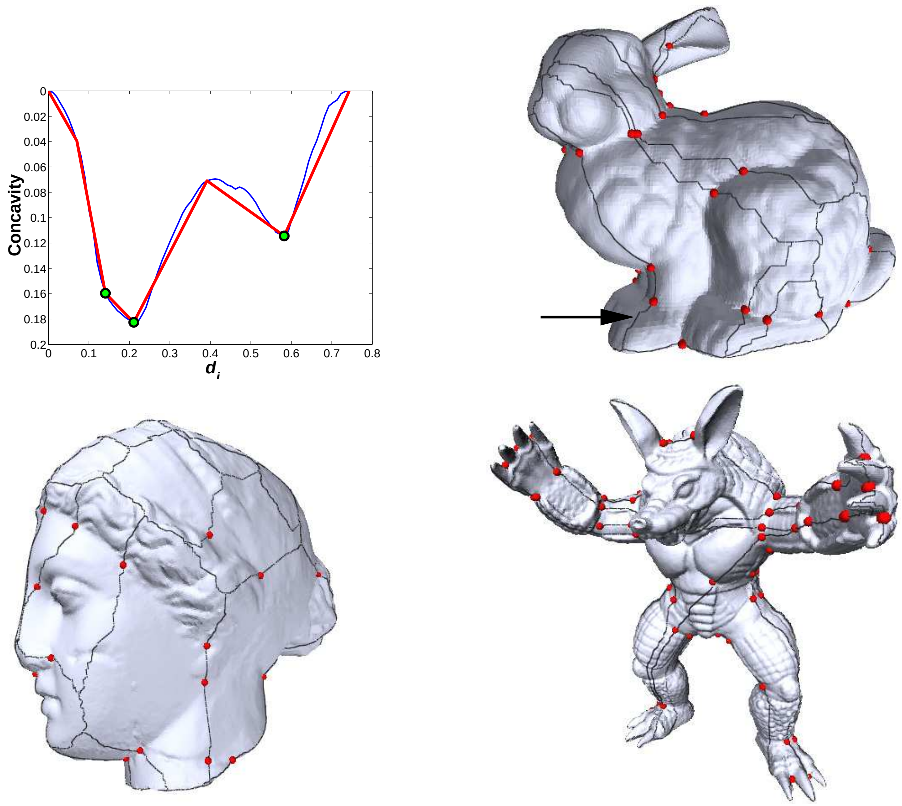



FIGURE 40. The thin line in the plot is a pocket boundary of the Stanford Bunny (indicated by an arrow) in concavity domain. Its simplification is shown in a thicker line and identified knots are marked as dots. The points on the boundaries of pockets of the Bunny, Venus, and Armadillo models are knots.

where $d_i = i \cdot |e|$ and $|e|$ is the length of $e$. Then $\pi_e^*(i)$ is simplified using the DP algorithm [55]. We call a vertex a "knot" if it is a *notch* in $\pi_e(i)$ with concavity larger than $\delta$, $0 \leq \delta \leq \tau$.

The threshold $\delta$ controls the size of knots, i.e., a smaller $\delta$ implies more concave features will be identified; in this chapter, we used $\delta = \tau/10$. We note that these pocket boundaries have similar functionality as the *exoskeleton* that connects critical points on $\partial P$ coded with *average geodesic distance* [134].

c. Computing Pocket Cuts

A pocket cut is a chain of consecutive edges in a pocket $\rho$ whose removal will bisect $\rho$. For a given pair of knots, we can form a pocket cut by computing a path using Dijkstra's algorithm that maximizes the total concavity along the path connecting the knots. Let $N_{\rho_i}$ be a set of knots on the boundary between $\rho$ and one of its neighboring pockets $\rho_i$. Any path in $\rho$ that connects any two knots between $N_{\rho_i}$ and $N_{\rho_j}$, $i \neq j$, is a pocket cut of $\rho$. Thus, a pocket with $|N_\rho|$ knots has $O(|N_\rho|^2)$ pocket cuts. Figure 41(a) and (b) shows a pocket with its knots on the boundary and all of its pocket cuts, respectively.

Not all of these $O(|N_\rho|^2)$ pocket cuts, denoted by $K_\rho$, in $\rho$ are interesting to us. In fact, we only need to consider $O(|n_\rho|)$ pocket cuts. This reduction is based on the following observation.

**Observation B.2.** *Let $n_{\rho_i}$ be a set of knots on the boundary between $\rho$ and one of its neighboring pockets $\rho_i$. Pocket cuts between each pair $n_{\rho_i}$ and $n_{\rho_j}$ in $\rho$ form a non-crossing minimum (weight) bipartite matching.*

We say two pocket cuts $\kappa_\rho$ and $\kappa'_\rho$ cross each other if $\kappa'_\rho$ will become disconnected after $\rho$ is separated by $\kappa_\rho$. 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; see Figure 41(c). 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; see Figure 41(d).

Let $\mathcal{K}_\rho \subset K_\rho$ be a subset of pocket cuts of $\rho$ that satisfy these criteria. It is easy to see that the size of $\mathcal{K}_\rho$ is $O(|N_\rho|)$. $\mathcal{K}_\rho$ can be extracted from $K_\rho$ using the minimum weight bipartite matching (w.r.t. a weight function W) followed by an iterative deletion of cross cuts.

**Cup-shape pocket**. Because knots are identified on the boundary of a pocket $\rho$, we cannot find any pocket cut if the boundary of $\rho$ is near its bridge $\beta$, e.g., a cup shape pocket. Indeed, decomposing a cup shaped model into visually meaningful components is known to be difficult. In our case, this problem can be easily identified by checking if the vertex with the maximum concavity of $\rho$ is a knot and, if not, as illustrated in Figure 42, it can be solved by simply subdividing $\beta$ and $\rho$ into smaller bridges and pockets and forcing the new pocket boundary to pass the maximum concavity of $\rho$.

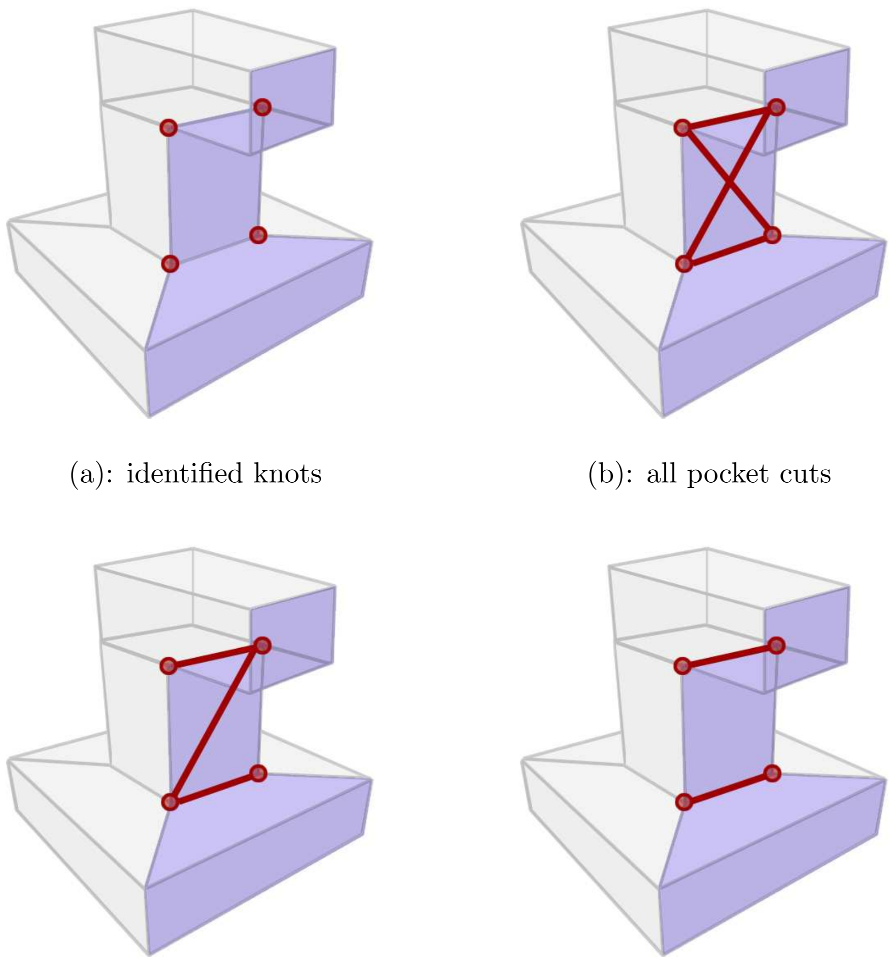
d. Weighting a Cut

The weight of a cut determines the quality of the cut. Curvature is known to be the most popular tool to evaluate extracted features, e.g., for non-photorealistic rendering [43], texture mapping [80], and shape segmentation [51]. However, estimating curvature of an entire model is difficult. Expensive preprocessing, such as mesh smoothing, simplification [66] and function approximation [100], or postprocessing, such as Hysteresis thresholding [63], are generally required. All these operations require input from users.

Despite its ability to identify *surface* features, we argue that curvature, by itself, is not sufficient to identify *structural* features. Thus, we define the weight of a cut as:

$$\mathrm{W}(\kappa) = \omega(\kappa)\gamma(\kappa), \tag{5.2}$$

where $\omega(\kappa) = |\,\kappa\,|/\mathrm{concavity}(\kappa)$ is the reciprocal of the *mean concavity* of a cut $\kappa$ and $\gamma(\kappa)$ is the accumulated curvature of the edges in $\kappa$. The curvature of an edge $e$ is measured using the *best fit polynomial* [63] of the intersection of the model and the plane bisecting $e$. Since curvature is only measured on cuts, instead of on the entire



(a): identified knots

(b): all pocket cuts

(c): non-crossing pocket cuts

(d): bipartite matching pocket cuts

FIGURE 41. (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.

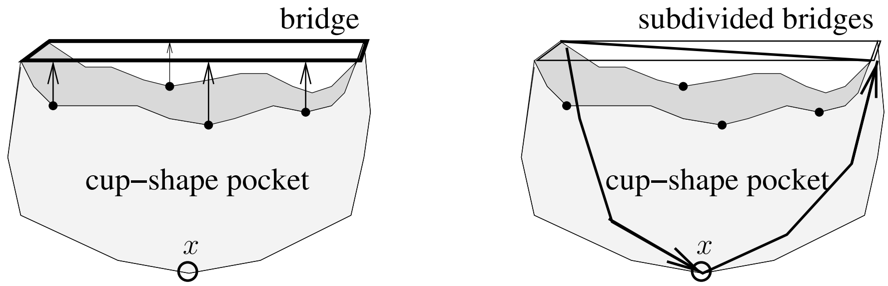



FIGURE 42. Left: A cup-shape pocket and its bridge. The black dots on the boundary of the pocket are knots, which are very close to the bridge. We know that this is a cup-shape pocket because its most concave feature, $x$, is not a knot. Right: The bridge is subdivided and the new pocket boundary is forced to pass $x$.

model, the computation is less expensive.

### e. Extracting Cycles from Graph $G_{\mathcal{K}}$

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

Once pocket cuts from all pockets of a model $P$ are computed, they can be connected into cuts. Our strategy is to organize the knots and pocket cuts in a graph and then to extract cuts from it. We define a graph $G_{\mathcal{K}} = \{V, E\}$, where $V = \cup_{\rho \in P} N_\rho$ and $E = \cup_{\rho \in P} \mathcal{K}_\rho$, i.e., knots and selected pocket cuts in $P$. We call such a graph $G_{\mathcal{K}}$ a *cut graph*. An example of $G_{\mathcal{K}}$ is shown in Figure 43.

Let $\kappa_\rho$ be a pocket cut to be resolved, e.g., the pocket cut that contains the most

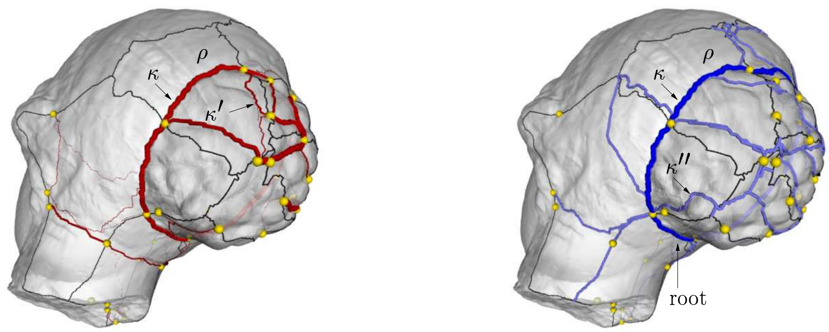



FIGURE 43. Left: An example of $G_{\mathcal{K}}$ (partially shown). Thicker pocket cuts have smaller weights. Right: An extracted tree from $G_{\mathcal{K}}$. The bold line is the best cut for the root.

concave vertex. To find cuts that include $\kappa_\rho$, we extract $T_{\mathcal{K}}$ rooted at $\kappa_\rho$ from $G_{\mathcal{K}}$. $T_{\mathcal{K}}$ is constructed so that a path from the root $\kappa_\rho$ to a leaf will consist of concave features that can be resolved together.

The process of building a tree $T_{\mathcal{K}}$ from $G_{\mathcal{K}}$ is similar to that of constructing a minimum spanning tree on $G_{\mathcal{K}}$. An exception is that we do not allow a node $x$ of the tree to grow into a pocket if the pocket is visited by an ancestor of $x$ because a cut can only visit a pocket once. For example, in Figure 43, the tree cannot expand from $\kappa$ to $\kappa'$. In addition, we allow new pocket cuts to be added during the tree construction to explore low concavity areas, e.g., $\kappa''$ in Figure 43. These new pocket cuts are computed as the shortest paths measured in geodesic distance. A MST that is built directly on vertices and edges of a polyhedron has been used for feature extraction, e.g., [104]. However, unlike $T_{\mathcal{K}}$ which is built on knots and pocket cuts, their MST requires pruning to enhance long features.

A valid cut in $T_{\mathcal{K}}$ consists of two paths from the root $\kappa_\rho$ to two leaves which end at the same pocket and are from two different sub-trees of the root. The minimum weighted cut in $T_{\mathcal{K}}$ is the final cut for $\kappa_\rho$.

## 3. Resolving Concave Features

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

A plane $E$ fits $\kappa$ best if $E$ minimizes

$$\sum_{e \in \kappa} \text{concavity}(e) \times \mu_E(e) \; , \tag{5.3}$$

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

Note that, sometimes, the intersection of $E$ and the model $P$ does not match the target cut $\kappa$. This happens when the intersection traverses different pockets than $\kappa$ does. It can be addressed by iteratively pushing $E$ toward the vertices on the portion of $\kappa$ that is misrepresented by the intersection. An example of $E$ and its improvement is shown in Figure 44.

## 4. Complexity Analysis

**Theorem B.3.** *Let $\{C_i\}$, $i = 1, \dots, m$, be the $\tau$-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 $\partial P$ and can take $O(n_e \log n_v)$ time for each path extracted

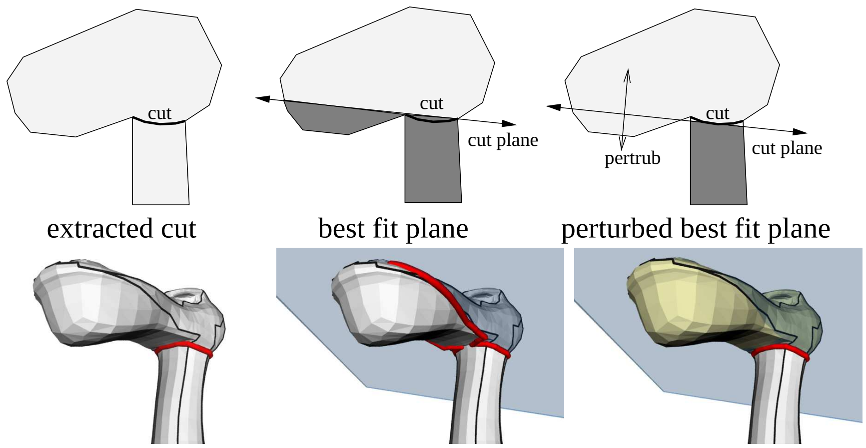



FIGURE 44. Left: A cut $\kappa$ around the neck. Mid: The best fit plane of $\kappa$. Its intersection with the model does not match $\kappa$. Lighter and darker shades shown in the figures indicate different components after decomposition. Right: An improved cut plane.

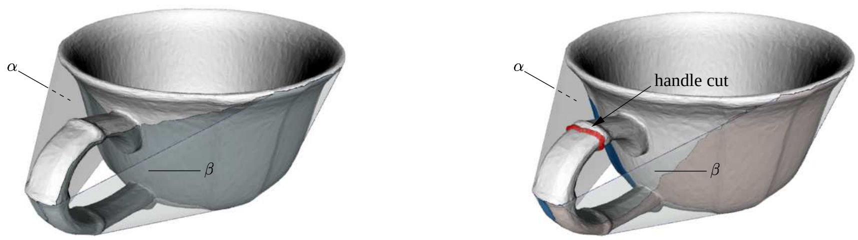
time using Dijkstra's algorithm. To resolve all $r$ notches in $P$, Algorithm 1 will take $O(rn_vn_e \log n_v) = O(n_e^3 \log n_e)$. □

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 $\tau$ is not zero and the total number of pocket cuts is usually quite small.

## C. 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 $\alpha$ or $\beta$ in Figure 45(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




(a) (b)

FIGURE 45. (a) The pocket (shaded area) is enclosed in the projected boundaries of two bridges $\beta$ and $\alpha$. (b) Pockets after genus reduction.

of finding minimum length handle cuts is NP-hard [47]. Several heuristics for genus reduction have been proposed (see a survey in [134]). The identified handle cuts will then be used to prevent the paths of the bridge projections from crossing them. Figure 45(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 45 with one hand: the hand must enter the hole though one of the bridges, e.g., $\beta$, and exit the hole from the other bridge, e.g., $\alpha$. 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.

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

1. Flooding the polyhedral surface $\partial P$ 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 46 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 [129].

## D. 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.

### 1. Implementation Details

There are three parameters, $\tau$, $\epsilon$, and $\delta$, used in our proposed method. The first parameter is the concavity tolerance $\tau$, which is used to control how convex the final

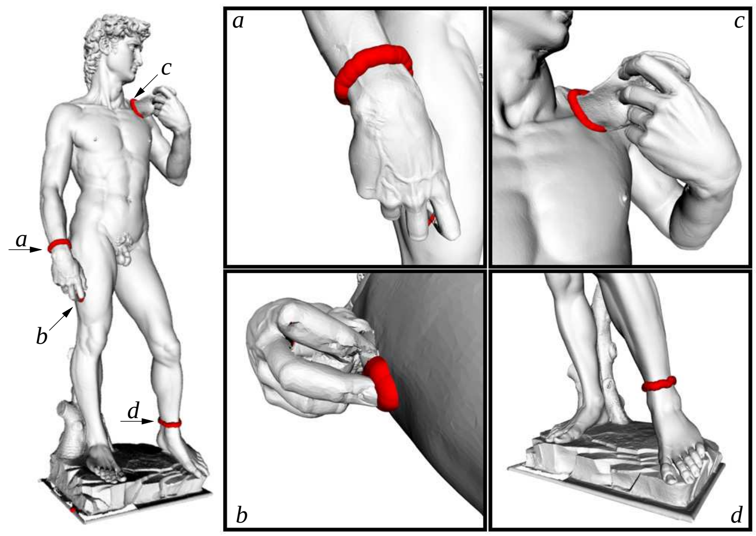



FIGURE 46. Four handle cuts found in the David model.

components are and should be set according to the need of the application.

The second parameter is the bridge clustering threshold $\epsilon$, 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 $\epsilon$ does not significantly affect the final decomposition and is always set to be $\epsilon = \frac{\tau}{2}$.

The third parameter $\delta$ is used in the Douglas-Peucker (DP) algorithm [55], which is used to identify knots on the pocket boundaries for concave feature grouping. The value of $\delta$ is difficult to estimate and is set experimentally between $\frac{\tau}{10}$ and $\frac{\tau}{100}$.

## 2. Models

The models used in the experiments in this section are summarized in Table 4. In Table 4, 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, which first defines all the vertices of a model and then defines how these vertices are connected into facets. 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.

### 3. 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 5, which includes results from both solid and surface decomposition, and in Figures 47 and 48, which contain results of several approximation levels of ACD with and without feature grouping.

#### a. ACDs Are Orders of Magnitude Smaller Than ECDs

In Table 5, 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 5, 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 47 are collected before termination, i.e., they are for an unfinished ECD, so all components are not yet convex. Figure 47 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

TABLE 4— Decompositions of 13 common models, where $|r|\%$ is the percentage of edges that are notches, $|e|$ is the number of edges, and $S$ is the physical (file) size. All models are normalized so that the radius of their minimum enclosing spheres is one unit.

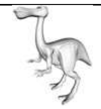
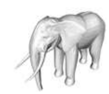
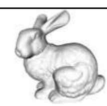
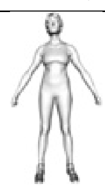
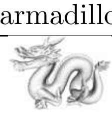
| *models* | $\lvert r\rvert\%$ | $\lvert e\rvert$ | $S$ | *models* | $\lvert r\rvert\%$ | $\lvert e\rvert$ | $S$ |
|---|---|---|---|---|---|---|---|
|  dinopet | 34.9% | 9,895 | 201 KB |  elephant | 30.4% | 10,197 | 206 KB |
|  elephant | 42.5% | 18,594 | 379 KB |  inner ear | 34.0% | 48,354 | 1.0 MB |
|  horse | 34.4% | 59,541 | 1.3 MB |  screw driver | 45.5% | 81,450 | 1.8 MB |
|  bunny | 40.5% | 104,496 | 2.3 MB |  teeth | 45.5% | 349,806 | 7.9 MB |
|  female | 38.8% | 365,163 | 8.5 MB |  venus | 43.8% | 403,026 | 9.3 MB |
|  armadillo | 41.4% | 518,916 | 12.1 MB |  david | 38.7% | 748,893 | 18.0 MB |
|  dragon | 42.8% | 1,307,170 | 31.7 MB | | | | |

Table 5— Decompositions of 13 common models, where $S$ and $|P_i|$ are the physical (file) size and the number of components of the decomposition, resp. Feature grouping is used for ACDs. Note that the David and the dragon models are not closed, thus they do not have results for solid decomposition.

| | **Solid** | | | | | |
|---|---|---|---|---|---|---|
| | $ACD_{0.2}$ | | $ACD_{0.02}$ | | ECD | |
| *models* | $\|P_i\|$ | $S$ | $\|P_i\|$ | $S$ | $\|P_i\|$ | $S$ |
| dinopet | 13 | 252 KB | 67 | 577 KB | 5,607 | 38 MB |
| elephant | 13 | 338 KB | 136 | 1.4 MB | 5,349 | 50 MB |
| bull | 12 | 481 KB | 211 | 2.3 MB | 12,210 | 102 MB |
| inner ear | 31 | 1.4 MB | 181 | 3.6 MB | 14,591 | 171 MB |
| horse | 8 | 1.4 MB | 77 | 2.4 MB | 24,044 | 527 MB |
| screw-dr | 1 | 1.8 MB | 44 | 3.0 MB | 43,180 | 2.0 GB |
| bunny | 6 | 2.5 MB | 178 | 6.6 MB | 46,728 | 2.8 GB |
| teeth | 11 | 9.4 MB | 307 | 18.8 MB | 135,224 | 7.5 GB |
| female | 5 | 8.7 MB | 67 | 10.9 MB | 145,085 | 7.2 GB |
| venus | 3 | 9.5 MB | 273 | 32.8 MB | 166,555 | 18.2 GB |
| armadillo | 11 | 12.1 MB | 98 | 14.2 MB | 726,240 | 20+ GB |

| | **Surface** | | | | | |
|---|---|---|---|---|---|---|
| | $ACD_{0.2}$ | | $ACD_{0.02}$ | | ECD | |
| *models* | $\|P_i\|$ | $S$ | $\|P_i\|$ | $S$ | $\|P_i\|$ | $S$ |
| dinopet | 12 | 205 KB | 62 | 226 KB | 1,297 | 224 KB |
| elephant | 15 | 215 KB | 123 | 250 KB | 1,306 | 229 KB |
| bull | 12 | 388 KB | 191 | 446 KB | 3,486 | 444 KB |
| inner ear | 26 | 1.0 MB | 89 | 1.1 MB | 6,360 | 1.2 MB |
| horse | 8 | 1.3 MB | 47 | 1.3 MB | 8,095 | 1.4 MB |
| screw-dr | 1 | 1.8 MB | 9 | 1.8 MB | 15,052 | 2.1 MB |
| bunny | 6 | 2.3 MB | 97 | 2.4 MB | 16,549 | 2.7 MB |
| teeth | 29 | 8.0 MB | 131 | 8.2 MB | 67,059 | 9.4 MB |
| female | 5 | 8.5 MB | 50 | 8.6 MB | 51,580 | 9.3 MB |
| venus | 3 | 9.3 MB | 164 | 9.6 MB | 72,190 | 9.6 MB |
| armadillo | 11 | 12.2 MB | 85 | 12.4 MB | 89,839 | 14.1 MB |
| david | 10 | 18.0 MB | 170 | 18.3 MB | 85,132 | 20.1 MB |
| dragon | 12 | 31.8 MB | 237 | 32.1 MB | 246,053 | 37.3 MB |

the number of components that the ECD has on average. Figure 48 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. Table 6 summarizes these statistics.

TABLE 6— ACD v.s. ECD.

| | % 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%** |

b. Solid ACDs Are Only Slightly Larger Than Surface ACDs

Table 5 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.

c. 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 47 and 48 show results of solid and surface decomposition for a range of approximation value $\tau$, respectively. Figures 47 and 48 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 $\tau$ decreases (i.e., as the convex approximation approaches an exact convex decomposition). In addition, with feature grouping, ACD produces structurally more meaningful components.

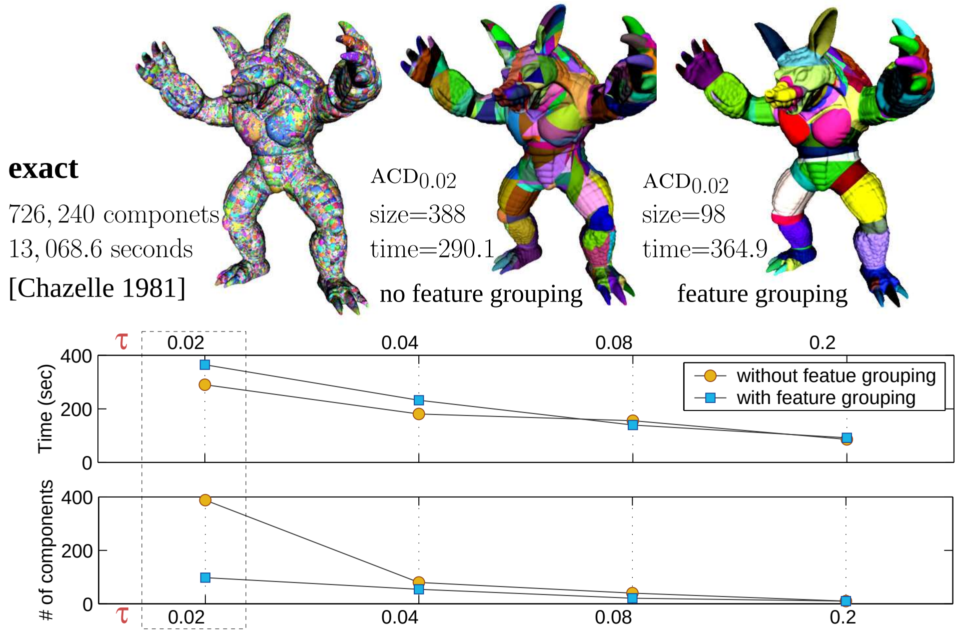
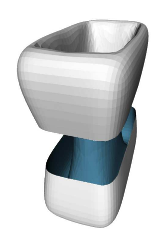



FIGURE 47. Convex solid decomposition. The size and time of ACD with and without feature grouping are shown for a range approximation values $\tau$.

## E. Discussion of Limitations

Despite our promising results, our current implementation for polyhedra has some limitations which we plan to address in future work, some of which can be solved without too much difficulty.

First, some uncommon types of open surfaces with "non-zero genus", see an example shown on the right, whose vertices on the convex hull are all convex, cannot be handled correctly by the proposed method.



Second, 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" whose concavity should also be acceptable to ensure no new untolerable features are introduced by the decomposition.

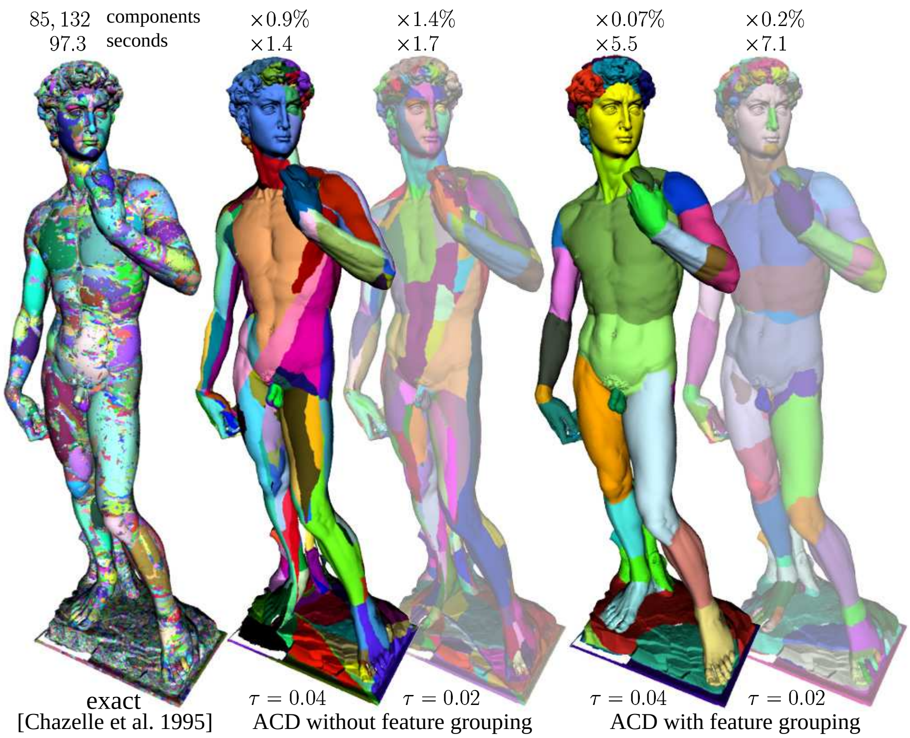



FIGURE 48. Convex surface decomposition. The leftmost figure shows a result of the exact decomposition. The others are results of the approximate decomposition.

Third, our feature grouping method has difficulty in collecting long features that have relatively low concavity as demonstrated in Figure 49.

Finally, we would like to consider efficient alternatives to shortest paths for the concavity measure, which is known to be a problem in NP hard [113] and has high time complexity even if computed approximately [36], such as by using an adaptively sampled distance field [50].

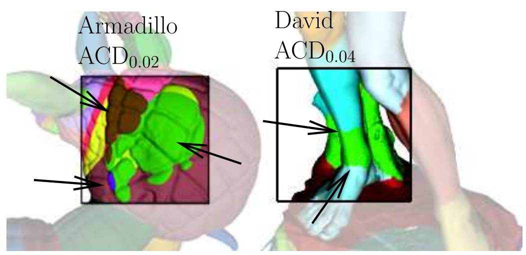



FIGURE 49. Problems of finding meaningful cuts in the low concavity areas.

# CHAPTER VI

## APPLICATIONS OF APPROXIMATE CONVEX DECOMPOSITION

Many problems, such as checking if a point is inside or outside of a polygon, can be solved more efficiently if they operate on convex objects. ACD components can also provide similar functionality. In this chapter, we will present some of the many potential applications of ACD. In most of these examples, a major gain in efficiency is obtained by using the convex hulls of the ACD components (and sometimes the components themselves) instead of using exact convex components. Sometimes using the convex hulls of the ACD components might introduce errors into the resulting computations, but in many cases these errors are small and can be tolerated. This includes a large set of problems in computational geometry and graphics, such as collision detection, mesh generation, pattern recognition, skeletonization, and origami folding. In this chapter, we consider four applications including point location, shape representation, motion planning, and mesh generation in a high level. Table 7 summarizes the studied applications and type of ACD used in this applications chapter. In Chapter VII, we will show in detail how ACD can be used to extract skeleton and shape decomposition simultaneously.

TABLE 7— Studied applications and type of ACD used.

| **Application** | **Solid/Surface** | **Feature Grouping** |
|---|---|---|
| Point location | Solid | No |
| Shape decomposition | Surface | Yes |
| Motion planning | Surface | Yes |
| Mesh generation | Solid | Yes |

**Algorithm 8** PointLocation_ACD

*Input.* A polygon or a polyhedron $P$, tolerance $\tau$, and a set of points $S$.
*Output.* Report points that are inside $P$ and those that are outside $P$.

```
Generate ACD_τ of P
for each point s ∈ S do
    for each component C ∈ ACD_τ do
        if s ∈ CH_C then
            Mark s as inside
    if s is not marked as inside then
        Mark s as outside
```

A. Point Location

**ACD type: solid ACD without feature grouping**. Point location, which checks if a point $x$ is in a polygon or 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 objects than for non-convex objects. Point location for a convex object $P$ can be done by checking if a point is on the same side of all $P$'s boundary.

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. Algorithm 8 outlines a naïve ACD-based point location by iteratively locating each point against each convex hull of the ACD component. If a point is inside one of the convex hulls, then the point is reported as inside; otherwise outside. Algorithm 8 may mis-classify points, which should be classified as external points but are classified as internal points using ACD. This is due to the difference between the convex hulls of the ACD components and the original model. Note that these misclassified points are usually close to the boundary. The distance between the misclassified points and the model depends on how convex the components are. This feature is very useful for some applications. For example, in a particle system, shown in Figure 2, the motion of the particles can be computed more efficiently using the ACD-based point location while the small errors, introduced by ACD, in a system with thousands of particles are hardly noticeable.

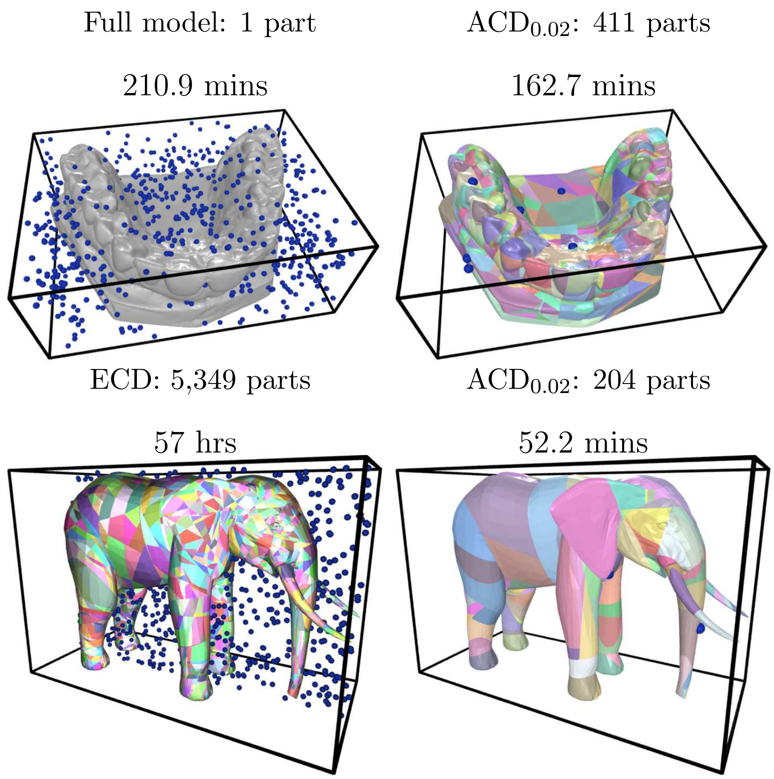



FIGURE 50. Point location of $10^8$ points in the teeth model (233,204 triangles), in the elephant model (6,798 triangles), and in their solid ECD and the convex hulls of the $ACD_{0.02}$. Measured time includes time for decomposition and point location. Point location in $ACD_{0.02}$ of both models 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.

In our experiments, point location of $10^8$ random points is performed for the full model and for the convex hulls of the $ACD_{0.02}$ 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 50, even using this naïve strategy, point location in the ACD is about 23% faster than in the original teeth model.

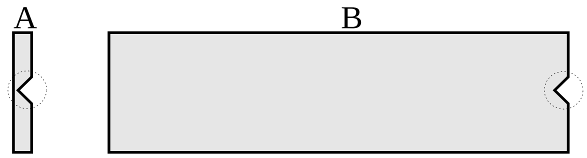



Figure 51. The features (circled) in polygons A and B have the same concavity but have different effects on the shapes of A and B. For polygon B, its concave feature has almost no effect on its overall shape.

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.

## B. Shape Representation

**ACD type: surface ACD with feature grouping**. The components of an ACD can also be used for shape representation. In this section, we present a strategy to generate shape decomposition using ACD. In many cases the significance 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 significant feature than a 5 cm stick on a ball with 5 km radius. Another example illustrating this idea is shown in Figure 51. This intuition can be captured by the concept of *convexity* defined as $\frac{\text{volume}(P)}{\text{volume}(CH_P)}$.

Algorithm 9 describes the ACD-based shape decomposition using convexity. First, instead of using concavity, we use convexity to check if a component is acceptable. Next, if the component has untolerable convexity, then we decompose the component. Figure 3 in Chapter I shows results from our approach that simply replaces the decomposition criterion, i.e., concavity, with 0.7 convexity.

**Algorithm 9** ShapeDecomp_ACD

*Input.* A polygon or a polyhedron, $P$, and minimum convexity, $\xi$.
*Output.* A decomposition of $P$, $\{C_i\}$, such that $\min\{convexity(C_i)\} \geq \xi$.

1: **if** $convexity(P) \geq \xi$ **then**
2:     return $P$
3: **else**
4:     $c = \text{concavity}(P)$
5:     $\{C_i\}$=**Resolve**($P$, $c$.witness).
6:     **for** each component $C \in \{C_i\}$ **do**
7:         ShapeDecomp_ACD($C$, $\xi$).

## C. Motion Planning

**ACD type: surface ACD with feature grouping**. Motion planning provides a tool to generate and control an object's motion by allowing the user to set initial and final arrangements of the objects and to specify constraints on the motion [75]. Motion planning has many applications, e.g., for navigating in the human colon or removing a mechanical part from an airplane engine. The ACD components can help to plan motion more efficiently. Since the motion planning problem has been shown to be intractable [23], researchers have focused on sampling-based motion planning strategies. The idea behind these strategies is to approximate the topology of the free configuration space (C-space) of a robot by sampling and connecting random configurations to form a graph [67, 132, 18, 86] (or a tree [76, 61, 96]) without explicitly computing the C-space.

Sampling-based motion planners have been shown to solve difficult motion planning problems; see a survey in [13]. 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 motion planning problems by sampling with a bias toward cuts between ACD components and

**Algorithm 10** PRM_ACD

*Input.* A robot $A$ and a polygon or a polyhedron, $P$, that describes the workspace.
*Output.* A roadmap $R$ that encodes the free C-space of $A$.

```
1: Generate ACD_τ of P
2: for each component C ∈ ACD_τ do                ▷ sample configurations of A
3:     for each centroid o of C and C's openings do
4:         repeat
5:             randomly place A around o with a random orientation (and joint angles)
6:             keep the configuration of A if collision free
7:         until k samples are generated around o
8: Connect each sample to its nearby samples to form a roadmap R using simple local
   planners.
```

the centroids of each component. Algorithm 10 shows an outline of this approach. First the model used to represent the workspace is decomposed using ACD. Then the robot is randomly placed near the centroids of the components and the cuts (openings) between components. These randomly generated configurations then form a network, called *roadmap*, by connecting each of them to its nearby configurations.

This sampling strategy is useful because narrow corridors usually have high concavity and are identified during the decomposition process. Our strategy samples configurations in these difficult areas and helps reveal the connectivity of the free C-space.

Figure 4 in Chapter I illustrates the advantage of this sampling strategy over uniform sampling [67]. In this example, we can see that the graph constructed using ACD represents the free C-space better than using the uniform sampling [67] with the same number (200) of collision-free samples.

Note that advantages of the ACD-based sampling are not only that more samples are placed in the narrower (difficult) regions but also the connections between the samples can be made more easily due to the nearly convex components.

**Algorithm 11** Meshing_ACD

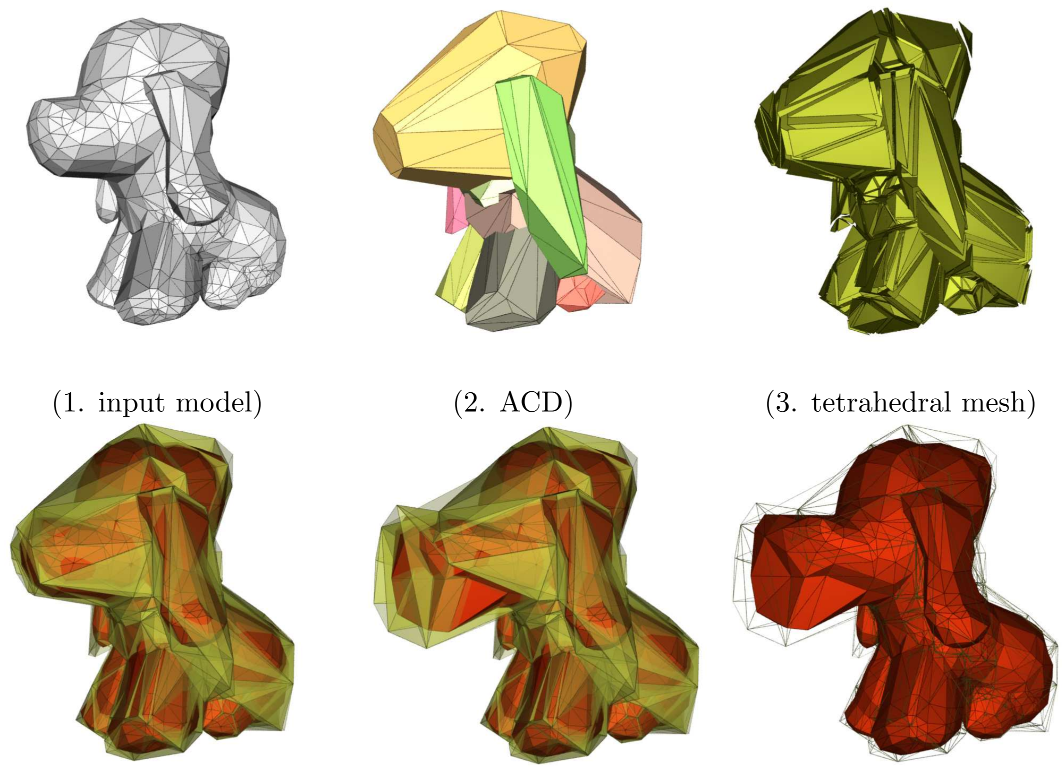
*Input.* A polygon or a polyhedron, $P$, and tolerance $\tau$.
*Output.* A tetrahedral mesh that approximates the shape of $P$
1: Generate $\text{ACD}_\tau$ of $P$
2: Let $M$ be an empty mesh
3: **for** each component $C \in \text{ACD}_\tau$ **do**
4: Generate a tetrahedral mesh $M_C$ by triangulating (a subset of) the vertices of $C$.
5: $M = M \cup M_C$

D. Mesh Generation

**ACD type: solid ACD with feature grouping**. Mesh generation is a process of decomposing a model into a set of tetrahedra or hexahedra. The resulting tetrahedral or hexahedral meshes can then be used in many applications, such as for modeling physically based deformation using Finite Element Method; see, e.g., [98].

The ACD components can be used to generate tetrahedral meshes from the ACD components using Delaunay triangulation [17, 64]. Algorithm 11 outlines this ACD-based mesh generation. This approach is favorable because it is known that generating tetrahedral or hexahedral meshes is easier and faster for convex objects, e.g., by connecting the centroid of the component to each vertex of the component or using Delaunay triangulation.

Note that sometimes the convex hulls of ACD components can still contain many triangles, thus the convex hulls may further simplified, e.g., using triboxes [40], 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 52 and Figure 5 in Chapter I.



(1. input model) (2. ACD) (3. tetrahedral mesh)

(4. bind input to the mesh) (5. deform mesh) (6. deformed input)

FIGURE 52. Hierarchical deformation. First, ACD is built from the input model. Next, a tetrahedral mesh is built from the components of ACD. Then, the input model is bound to the tetrahedral mesh. Finally, deformations that are applied to the tetrahedral mesh can be indirectly applied to the input model.

# CHAPTER VII

## SHAPE DECOMPOSITION AND SKELETONIZATION USING ACD

Shape decomposition partitions a model into (visually) meaningful components. Recently shape decomposition has been applied to texture mapping [110], shape manipulation [66], shape matching [94, 45, 51], and collision detection [82]. Early work on shape decomposition can be found in pattern recognition and computer vision; see surveys in [108, 131].

A *skeleton* is a lower dimensional object that essentially represents the shape of its target object. Because a skeleton is simpler than the original object, many operations, e.g., shape recognition and deformation, can be performed more efficiently on the skeleton than on the full object. The process of generating such a skeleton is called skeleton extraction or *skeletonization*. Examples of *automatic* skeleton extraction include the Medial Axis Transform (MAT) [16] and skeletonization into a *one dimensional poly-line skeleton* (or simply *1D skeleton*) [24, 88, 66].

Skeletons have been extracted from different sources, such as voxel (image) based data [135, 103, 15], boundary represented models [37, 4, 130], and scattered points [127], and for different purposes, such as shape description [114, 115], shape approximation [5, 133], similarity estimation [58], collision detection [21, 62], biological applications [3], navigation in virtual environments [81], and animation [124, 66].

Although it has been noted before that a good shape decomposition can be used to extract a high quality skeleton [88, 66] and that a high quality skeleton can be used to produce a good decomposition [82], this relationship between shape decomposition and skeleton extraction is a relatively unexplored concept, especially in 3D. Instead, when a relationship is noted, the skeletons are usually treated as an intermediate result or a by-product of the shape decomposition.

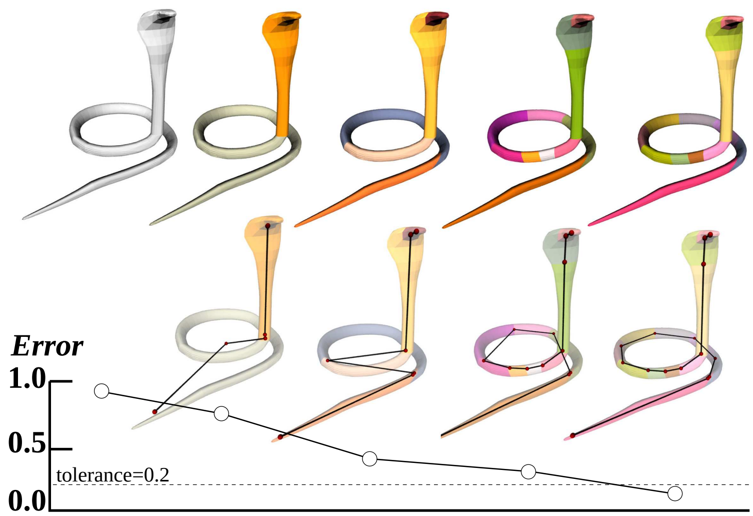



FIGURE 53. The skeleton (shown in the lower row) evolves with the shape decomposition (shown in the upper row).

In this chapter, we propose an integrated framework for simultaneous shape decomposition and skeleton extraction that not only acknowledges, but actually exploits the interdependence between these two operations. First, a simple skeleton is extracted from the components of the current decomposition. Then, this extracted skeleton is used to evaluate the quality of the decomposition. If the skeleton is satisfactory under some user defined criteria, we report the skeleton and the decomposition as our final results. Otherwise, the components are further decomposed into finer parts using approximate convex decomposition (ACD) [88, 85], which decomposes a given component by 'cutting' its most concave features. Figure 54 illustrates this proposed framework and Figure 53 shows an example of the co-evolution process of the shape decomposition and skeleton extraction.

As we will show, our proposed approach has several advantages and makes contributions as listed below.

- This recursive refinement strategy generates multi-resolution skeletons, from coarse to fine levels of detail, which are useful for some applications.

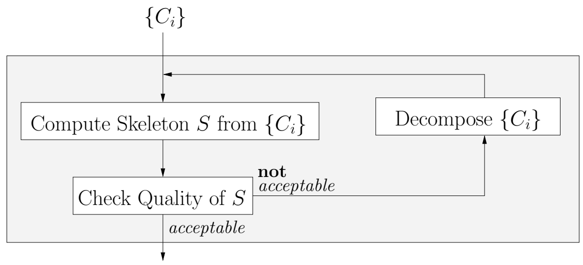



FIGURE 54. Simultaneous shape decomposition and skeleton extraction. The set $\{C_i\}$ is a decomposition of the input model $P$ and initially $\{C_i\} = \{P\}$.

- *Divide-and-conquer* algorithms that operate on the decompositions or skeletons can be more efficient because refinement is applied to the more complex regions but not to areas with less variation.

- The extracted skeleton is invariant under translation, rotation, and uniform scale, and is not very sensitive to boundary noise and skeletal deformations.

- Our approach does not require any pre-processing, e.g., model simplification, or any post-processing, e.g., skeleton pruning, which are required by many of the existing methods, e.g., [82, 66, 130].

- Our framework is general enough to work for both 2D polygons and 3D polyhedra.

## A. Related Work

Both shape decomposition and skeleton extraction have been studied for decades and there is a large amount of previous work. In this review, we concentrate on recent developments most relevant to our work.

**Shape decomposition**. Inspired by psychological studies, such as recognition by components [14] and the minima rule [59, 60], methods have been proposed to partition models at salient features to produce visually meaningful components. In pattern recognition, Rom and Medioni [108] partition a model into a set of tubular (generalized cylinder) shapes according to their curvature properties. As a preprocessing step for mesh generation, Sonthi et al. [93] identify closed sets (loops) of edges with required convexity and use them to decompose a model into solid parts. However, these methods work best with simple models with sharp internal angles, such as mechanical parts.

Methods that are applicable to models with general shapes also exist. Wu and Levine [131] propose a partitioning method based on a simulated electrical charge distribution on the surface of a model. Mangan and Whitaker [94] and Page et al. [102] decompose polygonal meshes by applying watershed segmentation with curvature computation. Li et al. [82] decompose polygonal meshes at critical points along skeletons obtained via model simplification. Dey et al. [45] segment a model, in $\mathbb{R}^2$ or $\mathbb{R}^3$, into *stable manifolds*, which are collections of Delaunay complexes of sampled points on the boundary. Katz and Tal [66] cluster mesh facets into fuzzy regions, carefully partition facets in those regions, and successfully produce perceptually clean cuts between decomposed components. A similar approach using a different clustering technique can also be found in [92]. Interactive methods [78, 51] that identify features via human assistance have also been shown to produce high quality and clean decompositions.

**Skeletonization**. The Medial Axis (MA), Voronoi diagram, Shock graph and Reeb graph are common skeleton representations. Although the MA can represent a lossless shape descriptor [16], it is difficult and expensive to compute accurately in high ($> 2$) dimensional space [41]. Several ideas for approximating the MA have been proposed, e.g., using Voronoi diagram, and its dual Delaunay triangulation [4, 6, 46], of densely sampled points from the object boundary. Shock graphs [118, 42], another representation of the MA, encode the formation order and, therefore, the importance of each part of the MA. Reeb graphs, a type of 1D skeleton, extracted from various Morse functions, are a powerful tool for shape matching [127, 116, 7, 58]. Since Morse functions are defined on mesh vertices, re-meshing [58, 7] is usually needed to generate a good (accurate) skeleton.

Several methods have been proposed to extract a skeleton from the components of a decomposition [88, 66]. Skeletons can also be constructed by simplifying (contracting) a polygonal mesh to line segments [82].

**Multi-scale and multi-resolution skeletons**. Multi-scale skeletons [107, 99] consist of a set of skeletons, $S_0, \ldots, S_N$, whose union represents a complete skeleton of the model. $S_0$ is the most important part of the skeleton, representing global topology, while $S_N$ encodes local features and is sensitive to local changes. Multi-resolution skeletons [58] consist of a set of skeletons, $S_0, \ldots, S_N$, that encode topology at different levels of detail. $S_0$ will have the coarsest skeleton and $S_N$ will contain the most detailed information. This representation is desired for some applications. For instance, to extract similar items from a 3D database, a rough skeleton can be used to reject many unlikely models and incrementally refine the skeleton to get better matches. As previously mentioned, one of the features of our framework is that its recursive nature results in the construction of multi-resolution skeletons.

## B. Framework

We propose a framework that simultaneously performs shape decomposition and skeleton extraction. For a given polyhedron $P$, **S**imultaneous **S**hape decomposition and **S**keleton extraction (SSS) (see Algorithm 12) constructs a skeleton for the model from (local) skeletons extracted from each component of a decomposition, evaluates

**Algorithm 12** SSS($P$)

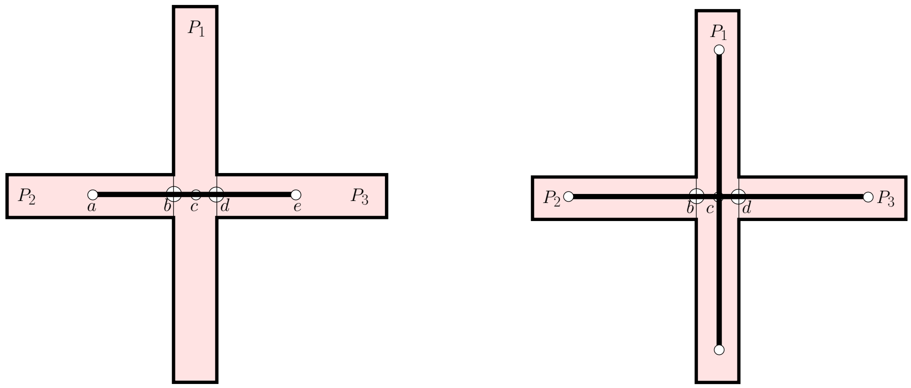
$S = Ext\_Skeleton(P)$
**if** $Error(P, S) \leq \tau$ **then**
    Report $S$ as $P$'s skeleton and report $P$ as a component
**else**
    $\{C_i\} = Decompose(P)$
    **For** each $C \in \{C_i\}$ **do** return SSS($C$)

the extracted skeleton components, and continues *refining* the decomposition and the associated skeleton components until the quality of the skeleton for each component is satisfactory, e.g., the error estimation of the skeleton for the respective component is smaller than a tunable threshold $\tau$.

There are three important sub-routines in Algorithm 12: $Ext\_Skeleton(P)$, which extracts a skeleton from a component $P$; $Error(P, S)$, which estimates the quality of the extracted skeleton; and $Decompose(P)$, which separates $P$ into sub-components when the extracted skeleton is not acceptable. We discuss methods for skeleton extraction $Ext\_Skeleton(P)$ in Section 1, and methods for quality measurement $Error(P, S)$ in Section 2. Recall that our choice for the $Decompose(P)$ sub-routine is approximate convex decomposition.

## 1. Extracting Skeletons

In this section, we discuss two simple methods to extract a (*local*) skeleton from a component of a decomposition. These local skeletons can be connected to form a global skeleton of the input model. The centroid method is a simple approach that can result in skeletons that do not represent the shape of the object. The second method, based on the principal axis (defined below) of a component, is slightly more expensive to compute, but leads to improved skeletons in some cases.




(using centroids) (using the principal axis)

FIGURE 55. This example shows a problem that arises when skeletonization is based only on the centroids. Points $b$ and $d$ are the centers of the openings and $a$, $c$ and $e$ are the centers of the components $P_1$, $P_2$ and $P_3$, respectively. This problem can be addressed using the principal axis.

**Using Centroids**. One of the easiest ways to construct a skeleton for a component $C$ (in a decomposition) is to connect the centroids of the openings, called *opening centroids*, on $\partial C$ to the centroid of $C$. These openings are generated when a component is split into sub-components during the decomposition process,

Several similar methods for extracting skeletons have been proposed [88, 66]. Although this approach is simple and generates fairly good results one of the major drawbacks of this type of skeleton is its inability to represent some types of shapes. For example, the skeleton of a cross-like model in Figure 55 extracted using its centroids is only a line segment instead of two crossing line segments. The method described next attempts to address this problem.

**Using the Principal Axis**. In this method, we extract a skeleton from a component $C$ (in a decomposition) using the principal axis of the convex hull $CH_C$ of $C$. The principal of a set of points is defined in Eqn. 4.1 in Chapter IV. Instead of connecting the centroids of $C$'s openings to the center of mass of $C$, we connect these centroids to the principal axis enclosed in $CH_C$. Figure 56 shows an example of skeletons constructed in this manner.

Let $\mathrm{PA}(CH_C)$ be a line through the center of mass of $CH_C$ parallel to the principal axis of $CH_C$. Our method connects an opening centroid to one of the $k$ points on $\mathrm{PA}(CH_C) \cap CH_C$. These $k$ points, denoted by $\mathcal{P}$, evenly subdivide $\mathrm{PA}(CH_C) \cap CH_C$ into $k + 1$ line segments. The selection of the value of $k$ is based on the desired minimum skeleton link length. Let $\mathcal{P}' \subset \mathcal{P}$ be a set of points to which the opening centroids connect. Figure 56 illustrates $\mathcal{P}$ and $\mathcal{P}'$ with circles along $\mathrm{PA}(CH_C)$. Then, the final skeleton $S$ of $C$ contains line segments that connect the opening centroids to $\mathcal{P}'$ and line segments that connect the $\mathcal{P}'$.

To minimize the chance of getting a long skeleton with many joints, we match the opening centroids to $\mathcal{P}$ so that the cardinality of $\mathcal{P}'$ and the distances from the opening centroids to $\mathcal{P}'$ are minimized. We solve this optimization matching problem using dynamic programming. Details of how we find the optimal solution are discussed in Section D.

In cases where all the points in $\mathcal{P}'$ lie only on one side of the center of mass $c$ of $CH_C$, e.g., $\mathcal{P}'$ in Figure 56(b), line segments that connect to the points in $\mathcal{P}'$ are not enough to represent the entire component. In such cases, the skeleton will connect $\mathcal{P}'$ with the end point of $\mathcal{P}$ on the other side of the center of mass $c$. Similarly, when $\mathcal{P}'$ contains only $c$, the skeleton will connect $c$ with the end points of $\mathcal{P}$ on both sides of $c$, e.g., the skeleton of the component $P_1$ in Figure 55 (using the principal axis).

Figure 57 shows three skeletons: two extracted skeletons using the centroid and the principal axis methods, and one skeleton manually generated by a professional animator. One can see that the skeleton extracted using the principal axis is topologically more similar to the animator generated skeleton than the skeleton generated using the centroid method. In Section D, we analyze the similarity of these skeletons

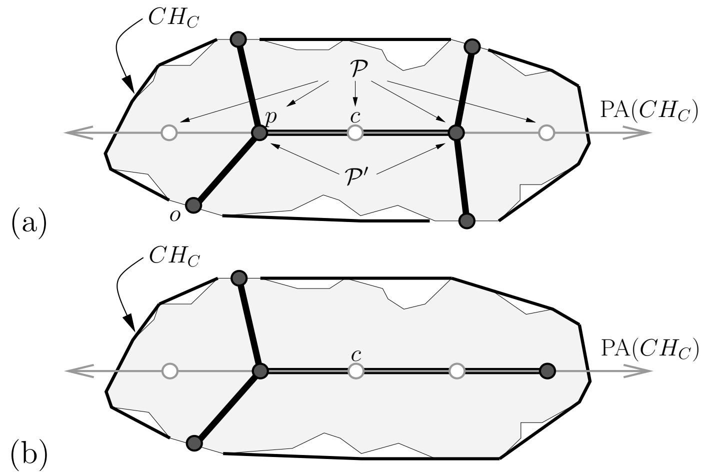


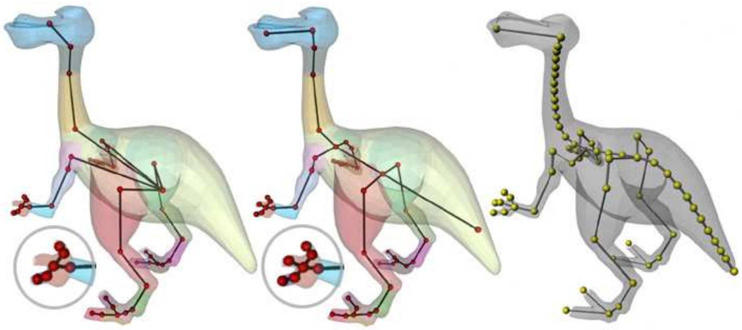

FIGURE 56. Using the principal axis of the convex hull $CH_C$ to extract a skeleton from a component. Skeletons are shown in dark thick lines and skeletal joints are shown in dark circles and $c$ denotes the center of mass of $CH_C$. (a) Opening centroids are connected to both sides of $c$. (b) Opening centroids are connected to only one side of $c$.

using graph edit distance.

## 2. Measuring Skeleton Quality

Although several criteria exist for measuring the quality of a skeleton, the general principles we adopt are that the skeleton should reside in the interior of the model and it should encode the "topology" of the model's shape. Thus, using these general criteria, our strategy to compute the quality of a skeleton $S$ is to compare $S$ with its associated component $C$. In this section, we propose three methods for measuring quality. This first method checks whether $S$ intersects $\partial C$ and the second method checks the topological representation of $S$ w.r.t. $C$. In the third method, we propose an adaptive measurement based on the volume of the component.

An important property of these three methods is that the error of the skeleton becomes smaller as the decomposition becomes finer. This property is justified at the



(centroids) (principal axis) (manually)

FIGURE 57. Notice the differences of these skeletons at the torso, the head, and the fingers.

end of this section. Figure 59 shows extracted skeletons based on these three quality measurements.

**Checking penetration**. Our first method measures the quality of $S$ by checking whether $S$ intersects the component boundary $\partial C$. If so, the function $Error(C, S)$ returns a large number (larger than the tolerable value $\tau$). Otherwise, zero will be returned. The consequence is that $C$ will be decomposed if $\partial C \cap S \neq \emptyset$.

As seen in Figure 59, skeletonization using penetration detection stops evolving after a few iterations and does not produce skeletons that represent the dragon or the bird.

**Measuring centeredness**. In the second method, we measure the offsets of $S$ from the level sets of the geodesic distance map on $\partial C$. The value for each point in this map is the shortest distance to its closest opening of $C$. Ideally, a skeleton should pass through all connected components in all level sets. Therefore, this measurement method simply checks the number of times that $S$ does not do so. An example of this measurement is shown in Figure 58.

Let $L_C$ be all the connected components in the level sets of $C$. We define the

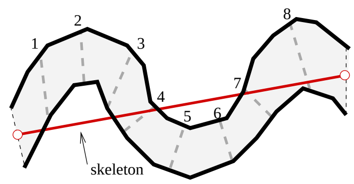



FIGURE 58. The error measurement for this skeleton, which intersects level sets 4, 7 and 8, is $\frac{5}{8}$.

error of a skeleton $S$ as:

$$Err(C,S) = \frac{\sum_{l_c \in L_C} f(l_c,S)}{|L_C|} \ , \tag{7.1}$$

where $f(l_c, S)$ returns 0 if $S$ intersects component $l_c$, and 1 otherwise, and $|L_C|$ is the total number of the connected components in $C$. Details of how we compute the level sets and $f(l_c, S)$ are discussed in Section D.

As seen in Figure 59, skeletonization using the centeredness measurement captures the shape of the dragon and the bird better then simply using penetration detection, but it over segments the tail of the bird and does not produce accurate skeletons in the feet of the dragon or the bird.

**Measuring convexity**. Our idea for the last quality measurement comes from the observation that in many cases the significance of a feature depends on its volumetric proportion to its "base". This concept can be captured by using convexity. Recall that we define the *convexity* of a component $C$ defined as $\text{convexity}(C) = \frac{\text{volume}(C)}{\text{volume}(CH_C)}$, where $\text{volume}(X)$ is the volume of a set $X$. Thus, we can define the error measurement as:

$$Err(C,S) = 1 - \text{convexity}(C) \ . \tag{7.2}$$

Assume that the skeleton $S$ is a good representation of the convex hull $CH_C$. Then, a smaller difference between $CH_C$ and $C$ means that $S$ is a better representation of $C$. Thus, although the skeleton $S$ is not included in Equation 7.2, $S$ is implicitly

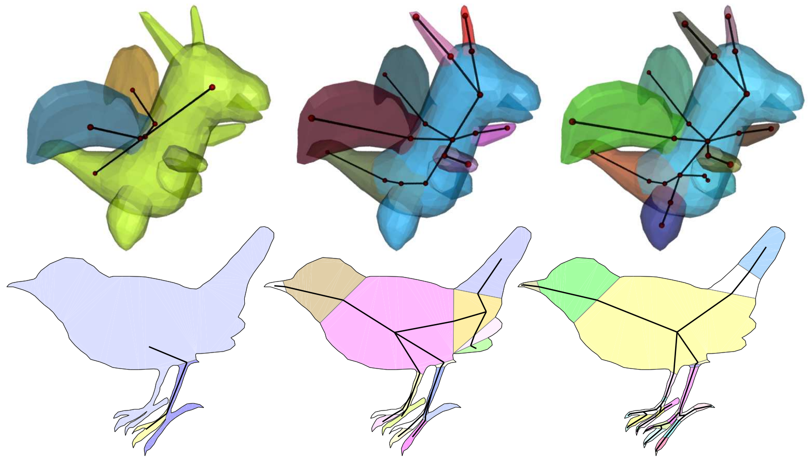



FIGURE 59. Final skeletons of a dragon polyhedron and a bird polygon extracted using different quality estimation functions: checking penetration, measuring centeredness, and measuring convexity. The maximum tolerable errors for centeredness and convexity are 0.2 and 0.3, respectively.

considered in terms of $CH_C$.

As seen in Figure 59, using convexity produces the most realistic skeleton that captures the overall shape of the dragon and the bird and also identifies the detailed features of their feet.

**Skeleton quality vs. ACD**. Here, we show that the error measurements of a skeleton, i.e., penetration, centeredness, and convexity, decrease as the input model is decomposed. This is a critical property, which allows the SSS framework to terminate.

**Lemma B.1.** *Let $S$ be the skeleton of a polyhedron $P$ and let $S'$ be the skeleton of the components of the ACD of $P$. The error estimation of $S'$ must be smaller than the error estimation of $S$ measured using penetration, centeredness, and convexity defined in Section 2.*

*Proof.* We show that all error measurements become zero if the input model is convex. For penetration, because the segments connecting any two points inside the convex

**Algorithm 13** $SSS_{ACD}(P)$

1: Compute a skeleton $S$ from $P$ using the *Principal Axis* of $CH_C$.
2: Estimate the quality of $S$ using *convexity*.
3: **if** $S$ is acceptable **then**
4: Report $S$ as $P$'s skeleton and report $P$ as a component.
5: **else**
6: $\{C_i\} = ACD(P)$.
7: **For** each $C \in \{C_i\}$ **do** return $SSS_{ACD}(C)$

object must not intersect its boundary, a skeleton will never penetrate the object. For the same reason, the skeleton must not be 'outside' of any level set of a convex component. Finally, because the convexity of a convex object is one, its error must be zero. □

## C. Putting It All Together

Algorithm 13 shows a fleshed-out version of the proposed simultaneous shape decomposition and skeletonization framework. Here we suggest using the principal axis, convexity and approximate convex decomposition for local skeleton extraction, quality measurement and partitioning, respectively. Algorithm 13 is used for all the experiments in Section D. We would like to emphasize that the choice of these methods is made based on our own experience. The framework is not restricted to these selected sub-routines, which can be replaced by other methods to fit particular needs of an application.

## D. Implementation and Results

### 1. Implementation Details

**From a Principal Axis to a Skeleton**. Here, we show how a local skeleton can be computed using the principal axis. Our goal is to find a mapping $M : O \rightarrow \mathcal{P}$ from the opening centroids $O$ to the points $\mathcal{P}$ on the principal axis so that the total length of the mapping and the number of the mapped points (joints) in $\mathcal{P}$ is minimized. We let the score function $F$ of a mapping $M$ be defined as

$$F(M) = s_1 \cdot |M| + s_2 \cdot J(M) \ , \tag{7.3}$$

where $|M|$ and $J(M)$ are the length and the number of joint of mapping $M$, and $s_1$ and $s_2$ are user specified scalars. $s_1$ and $s_2$ are constants set to ten and one, resp. A brute force approach to find an optimal solution will take $O(|\mathcal{P}|^{|O|})$ time, where $|\mathcal{P}|$ and $|O|$ are the number of vertices in $\mathcal{P}$ and $O$, respectively. This exponential time complexity is in general impractical for most applications.

The main idea of finding the optimal mapping is to group opening centroids $O$ and connect each group to a point in $\mathcal{P}$. After knowing how $O$ is grouped, it takes $O(|\mathcal{P}||O|)$ time to find a solution.

Grouping $O$ can be done using dynamic programming. An observation that enables us to group $O$ is that two centroids are likely to be grouped when their closest points in $\mathcal{P}$ are close. Thus, we first sort $O$ with respect to the closest points in $\mathcal{P}$ and then group the sorted elements of $O$. A dynamic programming approach for grouping $O$ is shown in Algorithm 14. In Algorithm 14, we use $G[i, j]$ to denote the optimal solution for the sub-problem $\{O_i, \cdots, O_j\}$. We use $G_iG_j$ to denote two joints without merging two groups $G_i$ and $G_j$. We use $< G_iG_j >$ to denote the joint that merges two groups $G_i$ and $G_j$ to one group.

**Compute level sets and centeredness**. A level set of a component $C$ in a decomposition is a set of points on the surface $\partial C$ of the component with the same geodesic distance to the closest opening of $C$. A connected component in a level set is a list of connected edges, which usually forms a loop on $\partial C$. A level set can have one or multiple connected component(s). These level sets can be computed, similar

**Algorithm 14** Optimal Matching($O$, $\mathcal{P}$)

**for** $i \in \{1, \cdots, |O|\}$ **do**
  $G[i, i] = O_i$
**for** $l \in \{2, \cdots, |O|\}$ **do**
  **for** $i \in \{1, \cdots, |O| - l + 1\}$ **do**
    $j = i + l - 1$
    $G[i, j] =< O_i \cdots O_j >$
    $score = F(G[i, j], \mathcal{P})$ ▷ $F$ is defined in Eqn. 7.3
    **for** $k \in \{i, \cdots, j - 1\}$ **do**
      $s = F(G[i, k]G[k + 1, j], \mathcal{P})$
      **if** $s_1 < score$ **then**
        $G[i, j] = G[i, k]G[k + 1, j]$
        $score = s_1$

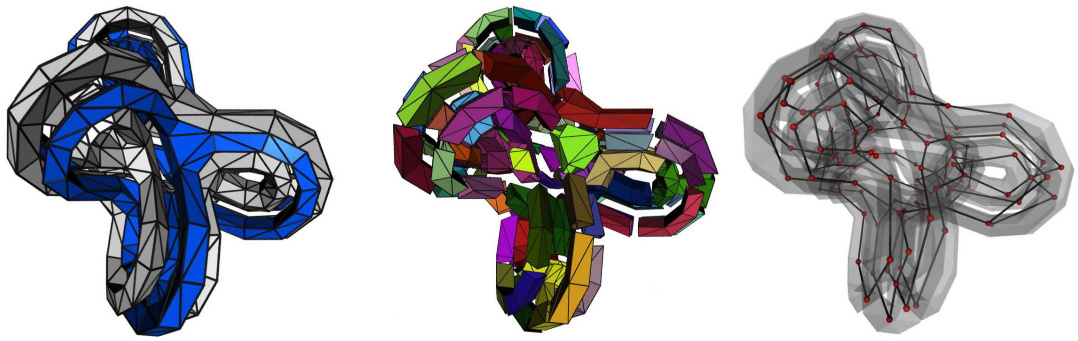
to the construction process of a Reeb graph [115], by flooding the entire $\partial C$ from the boundaries of the openings of $C$. In each iteration of this flooding process, the wavefronts will propagate from the visited vertices to unvisited vertices via incident edges.

To compute centeredness, we need to know how a skeleton $S$ intersects the level sets of $C$, i.e., we need the function $f(l_c, S)$ used in Eqn 7.1, which returns zero if $S$ intersects the level set $l_c$. The function $f(l_c, S)$ can be implemented by simply checking the intersection between each line segment of $S$ and the triangulation of $l_c$.

## 2. Experimental Results

The experiments in this section are used to demonstrate the *efficiency*, the *robustness*, and several *applications* of the proposed method. The method was implemented in C++ and all these experiments are performed on a Pentium 2.0 GHz CPU with 512 Mb RAM. Seventeen decompositions and their associated skeletons are shown in Figures 59 to 63 and in Tables 8 and 9.

**Efficiency**. A summary of the studied models, which include several game characters, a high genus model, and two scanned models, and the skeletonization



(input) (decomposition) (skeleton)

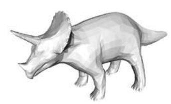
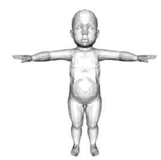
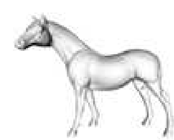
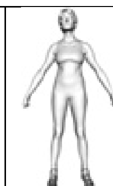
FIGURE 60. This figure shows the decomposition and the skeleton of a model with 18 handles.

and decomposition time of these models is shown in Table 8. Table 8 shows that the processing time of SSS depends on both the size of the model and on the complexity of the shape. For example, even though the model in Figure 60 has the fewest triangles, its large genus (18) increases the processing time. In general, our proposed SSS method can handle models with thousands of triangles in less than a half a minute and scales well for models with tens or hundreds of thousands of triangles.

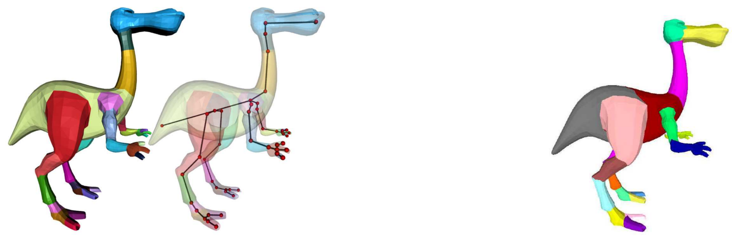
We further show that SSS is efficient by comparing our results to two recently proposed shape decomposition and skeletonization methods that have been shown to produce very promising results; see Figures 61 and 62, respectively. In both experiments, SSS generates results similar to those results reported previously but SSS can produce the shape decomposition and the skeletons about 30 times and 5 times faster than those methods reported in [66] and [130], respectively. We note that there are no well-accepted criteria to compare the quality of these decompositions and skeletons quantitatively, and therefore we do not intend to claim that our results are necessarily better.

**Robustness**. In this set of experiments, we show that SSS is robust under perturbation and deformation, meaning that the shape decompositions and skeletons

Table 8— Experimental results of SSS

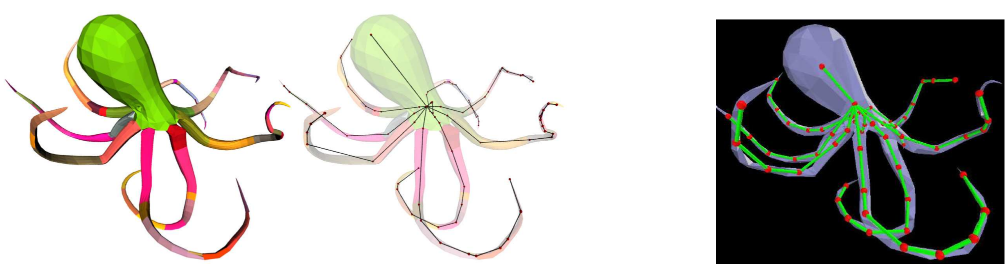
| Model |  Figure 60 |  Figure 63 |  Table 9 |  Figure 57 |  Figure 62 |  Figure 63 |  Table 9 |  Figure 53 |  Table 9 |
|---|---|---|---|---|---|---|---|---|---|
| Size | 1,984 | 3,392 | 5,660 | 6,564 | 8,276 | 11,180 | 39,694 | 48,312 | 243,442 |
| Time | 15.6 | 2.6 | 1.7 | 1.5 | 8.8 | 3.4 | 19.4 | 30.1 | 73.3 |

Note: Size is measured as the number of the triangles of each model and the processing time is measured in seconds.



(SSS) (Katz and Tal [66])

FIGURE 61. The decomposition with 0.7 convexity and the associated skeleton of the dino-pet model (with 6,564 triangles) are computed in 1.5 seconds whereas Katz and Tal's approach takes 57 seconds (on a P4 1.5 GHz CPU with 512 Mb RAM).



(SSS) (Wu et al. [130])

FIGURE 62. The decomposition with 0.7 convexity and the associated skeleton of the octopus model (with 8,276 triangles) are computed in 8.8 seconds whereas Wu et al.'s approach takes 53 seconds (on a P4 1.5 GHz CPU with 512 Mb RAM) using a simplified version of this model (with 2,000 triangles).

remain approximately the same after the input models are perturbed and deformed. The results are shown in Table 9.

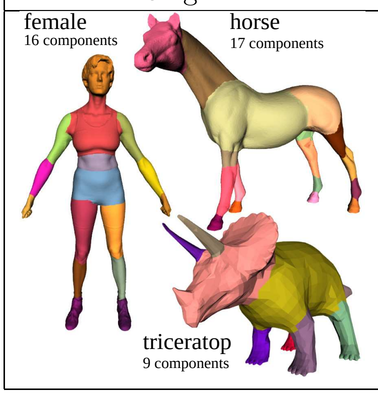
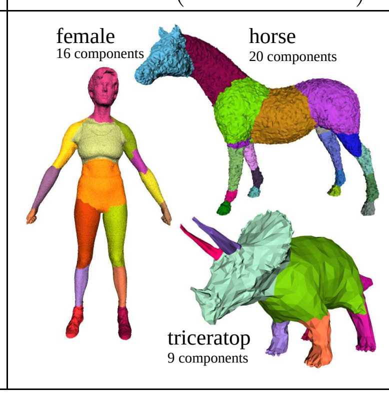
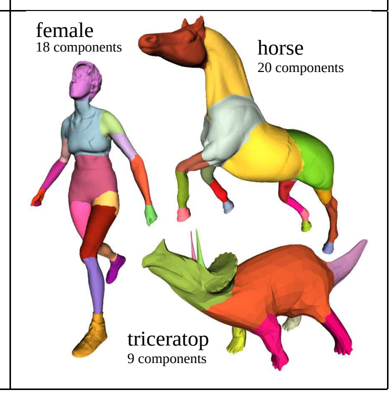
Although there are no well accepted criteria to measure the differences among decompositions, we can measure the similarity of these skeletons, e.g., using graph edit distance [22] which computes the cost of operations (i.e., inserting/removing vertices or edges) needed to convert one graph to another. In this section, we simply associate one unit of cost with each operation.

We measure two types of distances, denoted as $D_O$ and $D_O^2$. $D_O$ is the graph edit distance from a skeleton to the skeleton extracted from the original mesh. Because removing or inserting a degree-two node does not change the topology of a graph, we are also interested in the distance, denoted as $D_O^2$, that does not count operations that create and remove degree-two nodes. Table 9 shows that $D_O$ remains small for both perturbed and deformed models and $D_O^2$ is zero in all cases.

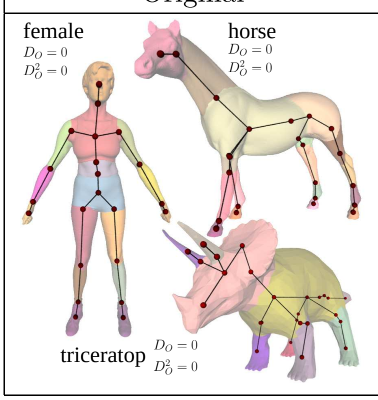
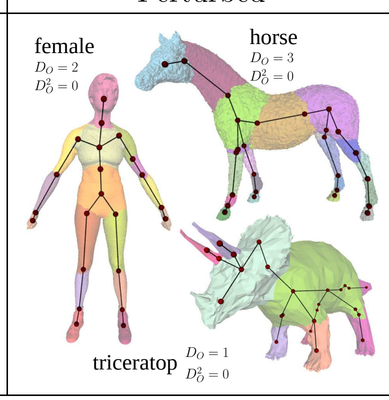
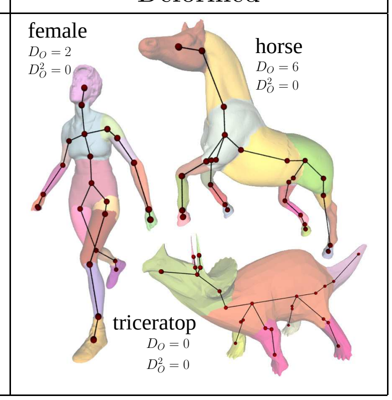
The extracted skeleton can be readily used to create animations. We demonstrate this advantage by re-targeting motion captured data to the skeletons extracted using our method. In Figure 63, we show a sequence of images obtained from a skeleton-based boxing animation of a baby and a robot using motion data captured from an adult male. Note that the baby and the robot models have different body proportions and rest poses. Other animations, including walking and pushing a box, are provided on our webpages. We use motion captured data instead of a hand-made animation to show that the extracted skeletons are robust enough to be used by arbitrarily selected motions and not only carefully designed motion. The motions, i.e., joint angles, are manually copied from the captured data to the skeletal joints.

Table 9— Robustness tests using perturbed and skeletal deformed meshes. $D_O$ is the graph edit distance between a skeleton extracted from a perturbed or deformed mesh and a skeleton extracted from the original mesh. $D_O^2$ is $D_O$ without counting operations on degree-2 nodes (which do not change the topology of the skeleton).

**Shape Decomposition.** 70% convexity

| Original | Perturbed (random noise) | Deformed |
|---|---|---|
|  |  |  |

**Extracted Skeletons.** 70% convexity

| Original | Perturbed | Deformed |
|---|---|---|
|  |  |  |

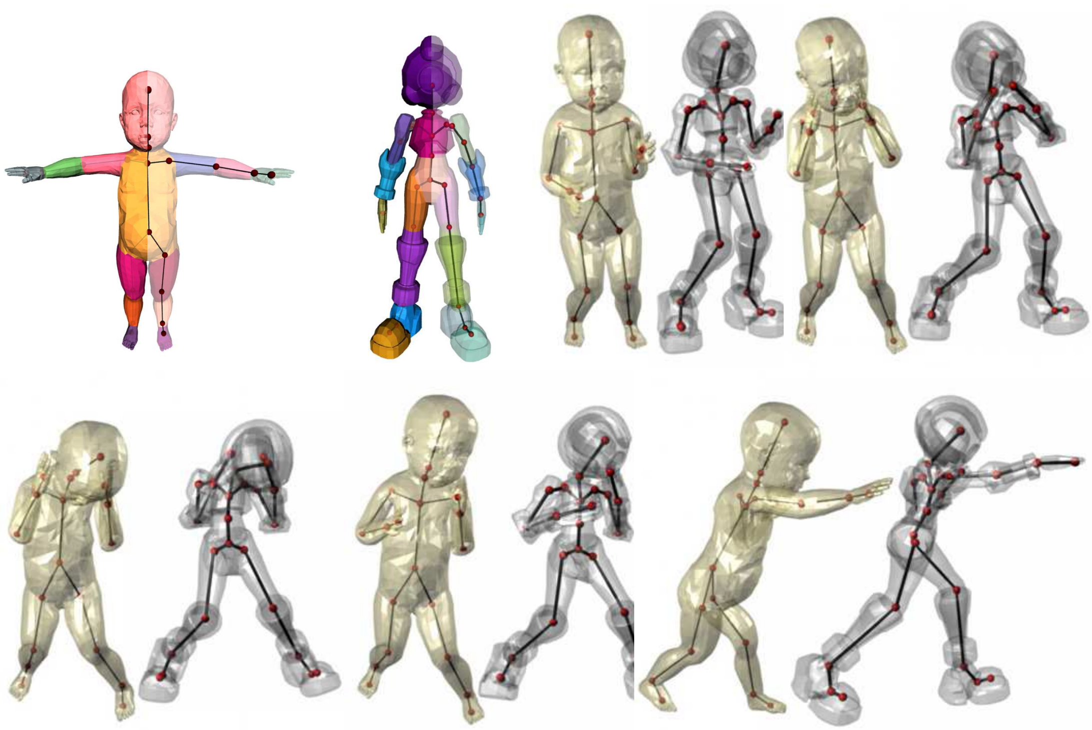


FIGURE 63. An animation sequence obtained from applying the boxing motion capture data to the extracted skeletons from a baby model and a robot model. The motion capture data (action number 13_17) are downloaded from the Carnegie Mellon University Graphics Lab motion capture database. The first two figures in the sequence are the shape decompositions and the skeletons of the baby and the robot. Note that not all joint motions from the data are used because the extracted skeletons have fewer joints.

## E. Discussion

In this section, we propose a framework that simultaneously generates shape decompositions and skeletons. This framework is inspired by the observation that both operations share many common properties and applications but are generally considered as independent processes. This framework extracts the skeleton from the components in a decomposition and evaluates the skeleton by comparing it to the components. The process of simultaneous shape decomposition and skeletonization iterates until the quality of the skeleton becomes satisfactory.

We studied two simple skeleton extraction methods, using the centroids and the principal axis, and three quality evaluation measurements, that compute penetration, centeredness and convexity, respectively. In the experiments, we demonstrate that the proposed framework is efficient, robust under perturbation and deformation, and can readily be used, e.g., to generate animations and plan motion.

There are several ways to extend the current work. First, there is a need to establish a systematic framework for comparing qualities of shape decomposition and skeletons using more quantitative measuring methods and benchmarks. Although the proposed quality measurements are based on a general idea of what a good skeleton should be, more studies are needed to investigate application-specific measurement criteria that should produce better and more "comparable" results. Second, not all models, such as a bowl, can have reasonable 1D skeletons. We are interested in using the same framework to extract the approximated medial axis from the components in a decomposition based on the idea that it is easier to extract the medial axis from a convex object than from a non-convex object. Finally, because the extracted skeletons and shape decompositions in our method co-evolve, we can provide more meaningful shape decompositions by using information from the extracted skeletons, e.g., merging components if the skeletons extracted from those components do not change the global skeleton made from the entire decomposition.

# CHAPTER VIII

# CONCLUSION AND FUTURE WORK

## A. Conclusion

In this dissertation, we proposed a method for decomposing a polygon or a polyhedron into approximately convex components that are within a user-specified tolerance of convex.

In Chapter IV, we presented ACD of simple polygons. For simple polygons, when the tolerance is set to zero, our method produces an exact convex decomposition in $O(nr)$ time which is faster than existing $O(nr^2)$ methods that produce a minimum number of components, where $n$ and $r$ are the number of vertices and notches, respectively, in the polygon. We proposed some heuristic measures to approximate our intuitive concept of concavity: a fast and less accurate straight line (SL) concavity, a slower and more precise shortest path (SP) concavity, and hybrid (H1 and H2) concavity methods with some of the advantages of both. We illustrated that our approximate method can generate substantially fewer components than an exact method in less time, and in all cases, producing components that are $\tau$-approximately convex. Our approach was seen to generate visually meaningful components, such as the legs and fingers of the Nazca monkey and the wings and tail of the Nazca heron.

An important feature of our approach is that it also applies to polygons with holes, which are not handled by previous methods. Our method estimates the concavities for points in a hole locally by computing the "diameter" of the hole before the hole boundary is merged into the external boundary.

In Chapter V, we extended the framework to decompose 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.

In Chapters VI and VII, we presented several applications of ACD including point location, shape representation, motion planning, mesh generation, and skeleton extraction. In most of these applications, the convex hulls of the ACD components are used to approximately represent the shapes of the objects.

## B. Future Work

Shape computations play fundamental and critical roles in many fields. ACD is just a starting point for approximating shapes and there is still a lot of work remaining to be done. We believe that the concept of approximate convex decomposition can be applied to problems involving collision detection, shape rendering, shape simplification, mesh compression, and shape identification. The study of these fundamental problems can be applied to more specific problems in the domains of robotics, computer graphics, computational neuroscience and computational chemistry/biology.

Several methods developed in this dissertation, 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. For example, studying the resemblance between the vertices on the convex hull and the critical points on an average geodesic distance coded mesh may speed up many applications that require geodesic distance computation.

Finally, one criterion of the decomposition is to minimize the concavity of its

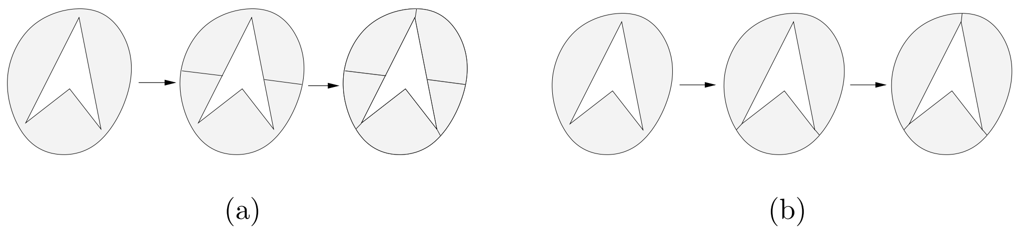



FIGURE 64. (a) Decomposition that minimizes concavity. (b) Decomposition using the proposed method.

components. Our decomposition method does not try to find a cut that splits a given model $P$ into two components with minimum concavity. There are two reasons that we do not do so. First, greedily minimizing concavity does not necessarily produce fewer components. Second, the decomposed components with minimum concavity may not represent significant features. For instance, in order to minimize the convexity of $P$ in Figure 64(a), $P$ will be decomposed into $P_1$ and $P_2$ so that $\max(\text{concavity}(P_1), \text{concavity}(P_2))$ is minimized. However, doing so splits the model at unnatural places and will ultimately generate more components. Therefore, we are interested in investigating whether a non-greedy method can reduce the size of the decomposition and can still represent significant features.

# REFERENCES

[1] P. K. Agarwal, E. Flato, and D. Halperin, "Polygon decomposition for efficient construction of minkowski sums," in *European Symposium on Algorithms*, January, 2000, pp. 20–31. [Online]. Available: citeseer.nj.nec.com/agarwal00polygon.html

[2] N. M. Amato, M. T. Goodrich, and E. A. Ramos, "A randomized algorithm for triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, vol. 26, pp. 245–265, 2001, special issue for the 16th ACM Symposium on Computational Geometry (SoCG 2000).

[3] N. Amenta, S. Choi, M. E. Jump, R. K. Kolluri, and T. Wahl, "Finding alpha-helices in skeletons," Dept. of Computer Science, The University of Texas at Austin, Tech. Rep., 2002.

[4] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Computational Geometry*, vol. 19, no. 2-3, pp. 127–153, 2001. [Online]. Available: citeseer.nj.nec.com/amenta01power.html

[5] D. Attali, P. Bertolino, and A. Montanvert, "Using polyballs to approximate shapes and skeletons," in *Proceedings of International Conference on Pattern Recognition (ICPR'94)*, 1994, pp. 626–628.

[6] D. Attali and J.-O. Lachaud, "Delaunay conforming iso-surface; skeleton extraction and noise removal," *Computational Geometry: Theory and Applications*, vol. 19, no. 2-3, pp. 175–189, 2001. [Online]. Available: http://dept-info.labri.u-bordeaux.fr/ lachaud/pub.html

[7] M. Attene, S. Biasotti, and M. Spagnuolo, “Re-meshing techniques for topological analysis,” in *Proc. of the Shape Modeling International (SMI’01)*, May 2001, pp. 142–151.

[8] D. Avis and G. T. Toussaint, “An optimal algorithm for determining the visibility of a polygon from an edge,” *IEEE Trans. Comput.*, vol. C-30, no. 12, pp. 910–1014, 1981.

[9] O. E. Badawy and M. Kamel, “Shape representation using concavity graphs,” *ICPR*, vol. 3, pp. 461–464, 2002.

[10] C. Bajaj and T. K. Dey, “Polygon nesting and robustness,” *Inform. Process. Lett.*, vol. 35, pp. 23–32, 1990.

[11] ——, “Convex decomposition of polyhedra and robustness,” *SIAM J. Comput.*, vol. 21, pp. 339–364, 1992.

[12] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal, “BOXTREE: A hierarchical representation for surfaces in 3D,” *Comput. Graph. Forum*, vol. 15, no. 3, pp. C387–C396, C484, Sept. 1996, proc. Eurographics’96.

[13] J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, “A random sampling scheme for path planning,” *Int. J. of Rob. Res*, vol. 16, no. 6, pp. 759–774, 1997.

[14] I. Biederman, “Recognition-by-components: A theory of human image understanding,” *Psychological Review*, vol. 94, pp. 115–147, 1987.

[15] I. Bitter, A. E. Kaufman, and M. Sato, “Penalized-distance volumetric skeleton algorithm,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 3, pp. 195–206, 2001.

[16] H. Blum, "A transformation for extracting new descriptors of shape," in *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed. Cambridge, MA: MIT Press, 1967, pp. 362–380.

[17] J.-D. Boissonnat, "Automatic solid modeler for robotics applications," in *Robotics Research: Third International Symposium. MIT Press Series in Artificial Intelligence.* Cambridge, MA: MIT Press, 1986, pp. 65–72.

[18] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, May 1999, pp. 1018–1023.

[19] G. Borgefors and G. S. di Baja, "Methods for hierarchical analysis of concavities," in *Proceedings of the Conference on Pattern Recognition (ICPR)*, vol. 3, 1992, pp. 171–175.

[20] ——, "Analyzing nonconvex 2d and 3d patterns," *Computer Vision and Image Understanding*, vol. 63, no. 1, pp. 145–157, 1996.

[21] G. Bradshaw and C. O'Sullivan, "Sphere-tree construction using dynamic medial axis approximation," in *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation.* ACM Press, 2002, pp. 33–40.

[22] H. Bunke and A. Kandel, "Mean and maximum common subgraph of two graphs," *Pattern Recogn. Lett.*, vol. 21, no. 2, pp. 163–168, 2000.

[23] J. F. Canny, *The Complexity of Robot Motion Planning.* Cambridge, MA: MIT Press, 1988.

[24] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic, "Interactive skeleton-driven dynamic deformations," *ACM Transactions on Graphics*,

vol. 21, no. 3, pp. 586–593, 2002.

[25] G. Castillero, “Ancient, giant images found carved into Peru desert,” October 2002, National Geographic News. [Online]. Available: http://news.nationalgeographic.com

[26] B. Chazelle and L. Palios, “Decomposition algorithms in geometry,” in *Algebraic Geometry and Its Applications*, C. Bajaj, Ed. New York, NY: Springer-Verlag, 1994, ch. 27, pp. 419–447.

[27] B. Chazelle, “Convex decompositions of polyhedra,” in *Proc. 13th Annu. ACM Sympos. Theory Comput.*, 1981, pp. 70–79.

[28] ——, “A theorem on polygon cutting with applications,” in *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, 1982, pp. 339–349.

[29] ——, “Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm,” *SIAM J. Comput.*, vol. 13, pp. 488–507, 1984.

[30] ——, “Triangulating a simple polygon in linear time,” *Discrete Comput. Geom.*, vol. 6, no. 5, pp. 485–524, 1991.

[31] B. Chazelle and D. P. Dobkin, “Decomposing a polygon into its convex parts,” in *Proc. 11th Annu. ACM Sympos. Theory Comput.*, 1979, pp. 38–48.

[32] ——, “Optimal convex decompositions,” in *Computational Geometry*, G. T. Toussaint, Ed. Amsterdam, Netherlands: North-Holland, 1985, pp. 63–133.

[33] 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.*, 1995, pp. 297–305.

[34] B. Chazelle and L. Palios, "Decomposing the boundary of a nonconvex polyhedron," in *Proc. 3rd Scand. Workshop Algorithm Theory*, ser. Lecture Notes Comput. Sci., vol. 621. Springer-Verlag, 1992, pp. 364–375.

[35] F. Chin, J. Snoeyink, and C. A. Wang, "Finding the medial axis of a simple polygon in linear time," *Discrete and Computational Geometry*, vol. 21, no. 3, pp. 405–420, 1999.

[36] J. Choi, J. Sellen, and C. K. Yap, "Approximate Euclidean shortest paths in 3-space," *Internat. J. Comput. Geom. Appl.*, vol. 7, no. 4, pp. 271–295, Aug. 1997.

[37] J.-H. Chuang, C.-H. Tsai, and M.-C. Ko, "Skeletonization of three-dimensional object using generalized potential field," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1241–1251, 2000.

[38] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational shape approximation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 905–914, 2004.

[39] A. G. Cohn, "A hierarchical representation of qualitative shape based on connection and convexity," in *International Conference on Spatial Information Theory*, 1995, pp. 311–326.

[40] A. Crosnier and J. Rossignac, "Tribox-based simplification of three-dimensional objects," *Computers & Graphics*, vol. 23, no. 3, pp. 429–438, 1999.

[41] T. Culver, J. Keyser, and D. Manocha, "Exact computation of the medial axis of a polyhedron," *Comput. Aided Geom. Des.*, vol. 21, no. 1, pp. 65–98, 2004.

[42] C. M. Cyr and B. B. Kimia, "3d object recognition using shape similarity-based aspect graph," in *ICCV'01*, 2001.

[43] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 848–855, 2003.

[44] E. D. Demaine, M. L. Demaine, and J. S. B. Mitchell, "Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami," in *Symposium on Computational Geometry*, 1999, pp. 105–114. [Online]. Available: citeseer.nj.nec.com/demaine99folding.html

[45] T. K. Dey, J. Giesen, and S. Goswami, "Shape segmentation and matching with flow discretization," in *Proc. Workshop on Algorithms and Data Structures*, 2003, pp. 25–36.

[46] T. K. Dey and W. Zhao, "Approximate medial axis as a voronoi subcomplex," in *ACM Symposium on Solid Modeling and Applications*, 2002, pp. 356–366.

[47] J. Erickson and S. Har-Peled, "Optimally cutting a surface into a disk," in *Proceedings of the Eighteenth Annual Symposium on Computational Geometry.* 2002, pp. 244–253.

[48] H. Y. F. Feng and T. Pavlidis, "Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition," *IEEE Trans. Comput.*, vol. C-24, pp. 636–650, 1975.

[49] T. Fevens, H. Meijer, and D. Rappaport, "Minimum convex partition of a constrained point set," in *Abstracts 14th European Workshop Comput. Geom.* Universitat Polytènica de Catalunya, Barcelona, 1998, pp. 79–81.

[50] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "Adaptively sampled distance fields: A general representation of shape for computer graphics,"

in *Proc. ACM SIGGRAPH*, 2000, pp. 249–254.

[51] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by example," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 652–663, 2004.

[52] D. H. Greene, "The decomposition of polygons into convex parts," in *Computational Geometry*, ser. Adv. Comput. Res., F. P. Preparata, Ed. Greenwich, CT: JAI Press, 1983, vol. 1, pp. 235–259.

[53] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, "Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, vol. 2, pp. 209–233, 1987.

[54] M. Held, "FIST: Fast industrial-strength triangulation of polygons," University at Stony Brook, Tech. Rep., 1998.

[55] J. Hershberger and J. Snoeyink, "Speeding up the Douglas-Peucker line simplification algorithm," in *Proc. 5th Internat. Sympos. Spatial Data Handling*, 1992, pp. 134–143.

[56] J. E. Hershberger and J. S. Snoeyink, "Erased arrangements of lines and convex decompositions of polyhedra," *Comput. Geom. Theory Appl.*, vol. 9, pp. 129–143, 1998.

[57] S. Hert and V. J. Lumelsky, "Polygon area decomposition for multiple-robot workspace division," *International Journal of Computational Geometry and Applications*, vol. 8, no. 4, pp. 437–466, 1998. [Online]. Available: citeseer.nj.nec.com/hert98polygon.html

[58] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3d shapes," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 203–212.

[59] D. Hoffman and W. Richards, "Parts of recognition," *Cognition*, vol. 18, pp. 65–96, 1984.

[60] D. Hoffman and M. Singh, "Salience of visual parts," *Cognition*, vol. 63, pp. 29–78, 1997.

[61] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 1997, pp. 2719–2726.

[62] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics (TOG)*, vol. 15, no. 3, pp. 179–210, 1996.

[63] A. Hubeli and M. Gross, "Multiresolution feature extraction for unstructured meshes," in *Proceedings of the Conference on Visualization '01*, 2001, pp. 287–294.

[64] B. Joe, "Geompack. A software package for the generation of meshes using geometric algorithms," *Advances in Engineering Software and Workstations*, vol. 13, no. 5–6, pp. 325–331, Sept. 1991.

[65] ——, "Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions," *International Journal for Numerical Methods in Engineering*, vol. 37, pp. 693–713, 1994.

[66] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 954–961, 2003.

[67] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, August 1996.

[68] J. M. Keil, "Decomposing polygons into simpler components," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1983.

[69] ——, "Decomposing a polygon into simpler components," *SIAM J. Comput.*, vol. 14, pp. 799–817, 1985.

[70] ——, "Polygon decomposition," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 2000, pp. 491–518.

[71] M. Keil and J. Snoeyink, "On the time bound for convex decomposition of simple polygons," in *Proceedings of the 10th Canadian Conference on Computational Geometry*, M. Soss, Ed. Montréal, Québec, Canada: School of Computer Science, McGill University, 1998, pp. 54–55. [Online]. Available: citeseer.nj.nec.com/keil98time.html

[72] J. R. Kent, W. E. Carlson, and R. E. Parent, "Shape transformation for polyhedral objects," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 47–54, 1992.

[73] A. Khodakovsky, N. Litke, and P. Schröder, "Globally smooth parameterizations with low distortion," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 350–357, 2003.

[74] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha, "Fast penetration depth computation for physically-based animation," in *ACM Symposium on Computer Animation*, 2002.

[75] J. C. Latombe, *Robot Motion Planning.* Boston, MA: Kluwer Academic Publishers, 1991.

[76] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2000, pp. SA45–SA59.

[77] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, vol. 14, pp. 393–410, 1984.

[78] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel, "Intelligent mesh scissoring using 3d snakes," in *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications (PG'04)*, October 2004, pp. 279–287.

[79] C. Levcopoulos and A. Lingas, "Bounds on the length of convex partitions of polygons," in *Proc. 4th Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, ser. Lecture Notes Comput. Sci., vol. 181 , 1984, pp. 279–295.

[80] 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*, 2002, pp. 362–371.

[81] T.-Y. Li, J.-M. Lien, S.-Y. Chiu, and T.-H. Yu, "Automatically generating virtual guided tours," in *Computer Animation.* IEEE Computer Society, May 1999, pp. 99–106.

[82] X. Li, T. W. Toon, and Z. Huang, “Decomposing polygon meshes for interactive applications,” in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 2001, pp. 35–42.

[83] J.-M. Lien and N. M. Amato, “Approximate convex decomposition,” Parasol Lab, Dept. of Computer Science, Texas A&M University, Tech. Rep. TR03-001, Jan 2003.

[84] ——, “Approximate convex decomposition,” in *Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG)*, June 2004, pp. 457–458, video Abstract.

[85] ——, “Approximate convex decomposition of polygons,” in *Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG)*, June 2004, pp. 17–26.

[86] J.-M. Lien, S. L. Thomas, and N. M. Amato, “A general framework for sampling on the medial axis of the free space,” in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, September 2003, pp. 4439–4444.

[87] J.-M. Lien and N. M. Amato, “Approximate convex decomposition for polygons,” *Computational Geometry: Theory & Applications*, vol. 35, no.1-2, pp. 100–123, 2006.

[88] ——, “Approximate convex decomposition of polyhedra,” Parasol Lab, Dept. of Computer Science, Texas A&M University, Tech. Rep. TR06-002, Jan 2006.

[89] ——, “Simultaneous shape decomposition and skeletonization,” in *Proceedings of ACM Solid and Physical Modeling Symposium (SPM’06)*, June 2006.

[90] A. Lingas, “The power of non-rectilinear holes,” in *Proc. 9th Internat. Colloq. Automata Lang. Program.*, ser. Lecture Notes Comput. Sci., vol. 140, 1982, pp. 369–383.

[91] A. Lingas, R. Pinter, R. Rivest, and A. Shamir, "Minimum edge length partitioning of rectilinear polygons," in *Proc. 20th Allerton Conf. Commun. Control Comput.*, 1982, pp. 53–63.

[92] R. Liu and H. Zhang, "Segmentation of 3d meshes through spectral clustering," in *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications (PG'04)*, October 2004, pp. 298–305.

[93] Y. Lu, R. Gadh, and T. J. Tautges, "Volume decomposition and feature recognition for hexahedral mesh generation," in *Proc. 8th International Meshing Roundtable*, October 1999, pp. 269–280.

[94] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 308–321, 1999.

[95] D. Marr, "Analysis of occluding contour," in *Proc. Roy. Soc. London*, 1977, pp. 441–475.

[96] E. Mazer, J. M. Ahuactzin, and P. Bessiere, "The Ariadne's clew algorithm," in *Journal of Artificial Robotics Research (JAIR)*, vol. 9, 1998, pp. 295–316.

[97] D. McCallum and D. Avis, "A linear algorithm for finding the convex hull of a simple polygon," *Inform. Process. Lett.*, vol. 9, pp. 201–206, 1979.

[98] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," 2002, pp. 49–54.

[99] R. Ogniewicz and O. Kubler, "Hierarchic voronoi skeletons," *Pattern Recognition*, vol. 28, no. 3, pp. 343–359, 1995.

[100] Y. Ohtake, A. Belyaev, and H.-P. Seidel, "Ridge-valley lines on meshes via implicit surface fitting," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 609–612, 2004.

[101] S. J. Owen, "A survey of unstructured mesh generation technology," in *7th International Meshing Roundtable*, 1998, pp. 239–267.

[102] D. L. Page, A. F. Koschan, and M. A. Abidi, "Perception-based 3d triangle mesh segmentation using fast marching watersheds," in *Proceedings of the 2003 Conference on Computer Vision and Pattern Recognition (CVPR '03)*, 2003, pp. 27–32.

[103] R. M. Palenichka, M. B. Zaremba, and U. du Quebec, "Multi-scale model-based skeletonization of object shapes using self-organizing maps," in *16th International Conference on Pattern Recognition (ICPR'02)*, 2002, pp. 10 143–10 147.

[104] M. Pauly, R. Keiser, and M. Gross, "Multi-scale feature extraction on point-sampled surfaces," in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 2003, pp. 281–289.

[105] E. Praun and H. Hoppe, "Spherical parametrization and remeshing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 340–349, 2003.

[106] S. Rodriguez, J.-M. Lien, and N. M. Amato, "Planning motion in completely deformable environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2006, pp. 2466–2471.

[107] H. Rom and G. Medioni, "Hierarchical decomposition and axial shape description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 10, pp. 973–981, 1993.

[108] ——, “Part decomposition and description of 3d shapes,” in *Proceedings of International Conference on Pattern Recognition (ICPR'94)*, 1994, pp. 629–632.

[109] P. L. Rosin, “Shape partitioning by convexity,” *IEEE Transactions on System, Man, and Cybernetics - Part A : System and Humans*, vol. 30, no. 2, pp. 202–210, March 2000.

[110] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe, “Multi-chart geometry images,” in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 2003, pp. 146–155.

[111] P. Schorn, “Accurate and efficient algorithms for proximity problems,” in *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 24–27.

[112] A. Shapiro and A. Tal, “Polyhedron realization for shape transformation,” *The Visual Computer*, vol. 14, no. 8/9, pp. 429–444, 1998.

[113] M. Sharir and A. Schorr, “On shortest paths in polyhedral spaces,” *SIAM J. Comput.*, vol. 15, pp. 193–215, 1986.

[114] D. J. Sheehy, C. G. Armstrong, and D. J. Robinson, “Shape description by medial surface construction,” *IEEE Trans. Visualizat. Comput. Graph.*, vol. 2, no. 1, pp. 62–72, Mar. 1996.

[115] Y. Shinagawa and T. L. Kunii, “Constructing a reeb graph automatically from cross sections,” *IEEE Computer Graphics and Applications*, vol. 11, no. 6, pp. 44–51, 1991.

[116] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien, “Surface coding based on morse theory,” *IEEE Computer Graphics and Applications*, vol. 11, no. 5, pp.

66–78, 1991.

[117] K. Siddiqi and B. B. Kimia, "Parts of visual form: Computational aspects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 3, pp. 239–251, 1995.

[118] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker, "Shock graphs and shape matching," in *ICCV*, 1998, pp. 222–229. [Online]. Available: citeseer.nj.nec.com/siddiqi98shock.html

[119] M. Simmons and C. H. Séquin, "2d shape decomposition and the automatic generation of hierarchical representations," *International Journal of Shape Modeling*, vol. 4, pp. 63–78, 1998.

[120] M. Singh, G. Seyranian, and D. Hoffma, "Parsing silhouettes: The short-cut rule," *Perception & Psychophysics*, vol. 61, pp. 636–660, 1999.

[121] J. Sklansky, "Measuring concavity on rectangular mosaic," *IEEE Trans. Comput.*, vol. C-21, pp. 1355–1364, 1972.

[122] J. Snoeyink, "Minimum convex decomposition," software. [Online]. Available: http://www.cs.unc.edu/∼snoeyink/

[123] H. I. Stern, "Polygonal entropy: A convexity measure," *Pattern Recognition Letters*, vol. 10, pp. 229–235, 1989.

[124] M. Teichmann and S. Teller, "Assisted articulation of closed polygonal models," in *Proceedings of the Eurographics Workshop*, 1998, pp. 254–254. [Online]. Available: citeseer.nj.nec.com/teichmann98assisted.html

[125] S. Tor and A. Middleditch, "Convex decomposition of simple polygons," *ACM Transactions on Graphics*, vol. 3, no. 4, pp. 244–265, 1984.

[126] M. Tănase and R. C. Veltkamp, "Polygon decomposition based on the straight line skeleton," in *Proceedings of the Nineteenth Conference on Computational Geometry (SoCG).* ACM Press, 2003, pp. 58–67.

[127] A. Verroust and F. Lazarus, "Extracting skeletal curves from 3d scattered data," in *International Conference on Shape Modeling and Applications.* IEEE Computer Society, 1999, pp. 194–201.

[128] E. R. White, "Assessment of line-generalization algorithms using characteristic points," *The American Cartographer*, vol. 12, no. 1, pp. 17–27, 1985.

[129] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder, "Removing excess topology from isosurfaces," *ACM Trans. Graph.*, vol. 23, no. 2, pp. 190–208, 2004.

[130] F.-C. Wu, W.-C. Ma, P.-C. Liou, R.-H. Laing, and M. Ouhyoung, "Skeleton extraction of 3d objects with visible repulsive force," in *Computer Graphics Workshop 2003, Hua-Lien, Taiwan*, 2003.

[131] K. Wu and M. D. Levine, "3d part segmentation using simulated electrical charge distributions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 11, pp. 1223–1235, 1997.

[132] Y. Wu, "An obstacle-based probabilistic roadmap method for path planning," Master's thesis, Department of Computer Science, Texas A&M University, 1996.

[133] G. Wyvill and C. Handley, "The 'thermodynamics' of shape," in *Proc. of the Shape Modeling International (SMI'01)*, May 2001, pp. 2–8.

[134] E. Zhang, K. Mischaikow, and G. Turk, "Feature-based surface parameterization and texture mapping," Georgia Institute Technology, GIT-GVU-03-29, 2003.

[135] Y. Zhou and A. W. Toga, "Efficient skeletonization of volumetric objects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 3, pp. 196–209, 1999.

[136] J. Zunic and P. L. Rosin, "A convexity measurement for polygons," in *British Machine Vision Conference*, 2002, pp. 173–182.

# VITA

Jyh-Ming Lien was born on January 27, 1977, in Taipei, Taiwan. He graduated from Taipei Municipal Chien Kuo High School in 1995 and received his B.S. in Computer Science at National ChengChi University, Taipei, Taiwan, in 1999. From 1999, until now he has been a student and graduate assistant in the Department of Computer Science at Texas A&M University.

Mr. Lien's permanent address is: 3F No.11, Lane 1, Sec. 2, Chenggong Rd., Yonghe, Taipei, 234, Taiwan.