Approximate Convex Decomposition *

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

Technical Report TR03-001
PARASOL LAB
Department of Computer Science
Texas A&M University
January 21, 2003

Abstract

One common strategy for dealing with large, complex models is to partition them into pieces that are easier to handle. While decomposition into convex components results in pieces that are easy to process, such decompositions can be costly to construct and often result in representations with an unmanageable number of components. In this paper, we propose an alternative partitioning strategy that decomposes a given polyhedron into *"approximately convex"* pieces. 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. Indeed, for many models, an approximate convex decomposition can more accurately represent the important structural features of the model by providing a mechanism for ignoring insignificant features, such as wrinkles and other surface texture. We propose a simple algorithm to compute *approximate convex decompositions* of polyhedra of arbitrary genus to within a user specified tolerance. This algorithm measures the significance of the model's features and resolves them in order of priority. As a by product, it also produces an elegant hierarchical representation of the model. We illustrate its utility in constructing an approximate skeleton of the model that results in significant performance gains over skeletons based on an exact convex decomposition.

*This research supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACI-0113971, CCR-0113974, EIA-9810937, EIA-0079874, and by the Texas Higher Education Coordinating Board grant ARP-036327-017.

Figure 1: Each component is approximately convex (concavity less than 10 by our measure). There are a total of 17 components.

1 Introduction

Decomposition is a technique commonly used to simplify complex models into smaller sub-models that are easier to handle. Convex decomposition divides polyhedra into convex components. Due to the important properties of convex objects, many algorithms perform more efficiently on convex objects than on non-convex objects. For example, mesh generation [24] and penetration depth computations [21] only have efficient solutions for convex models. In general, convex decomposition has application in a diverse range of areas such as collision detection [12], constructive solid geometry modeling [17], feature extraction [13], and mesh generation [24], to name just a few.

In many applications, however, the detailed features of the model are not crucial and in fact considering them only serves to obscure the important structural features and adds to the processing cost. In such cases, an approximate representation of the model, such as our proposed approximate convex decomposition, that captures the key structural features would be preferable. One important example is *skeleton extraction*. The skeleton is a low dimensional object which essentially represents the "shape" of the higher-dimensional target object. The process of generating such a skeleton is called skeleton extraction. One-dimensional skeletons extracted from three dimensional objects have application in shape recognition [30], virtual world navigation [4], and deformation [7]. One of the most important issues for extraction methods is their sensitivity to noise/turbulence of the boundary [27, 31, 4, 30]. A well known skeleton extraction method, the Medial Axis Transform, suffers this problem. In this paper, we demonstrate that significant profit gains result from using our approximate decomposition for skeleton extraction.

Convex decomposition of three-dimensional polyhedra is traditionally done by iteratively removing the polyhedron's *"non-convex features"* (called *notches*) in an arbitrary order. This operation, called *notch resolution* or *notch cutting*, continues until all components are convex. Finding the minimum number of disjoint convex components for a non-convex polyhedron is known to be NP-hard [26]. Using notch-cutting with only one cutting plane for each notch, Chazelle [8, 9] shows that at most $\frac{r^2+r+2}{2}$ convex components will be generated, where r is the total number of notches of the polyhedron. This method generates the worst case optimal $O(r^2)$ convex

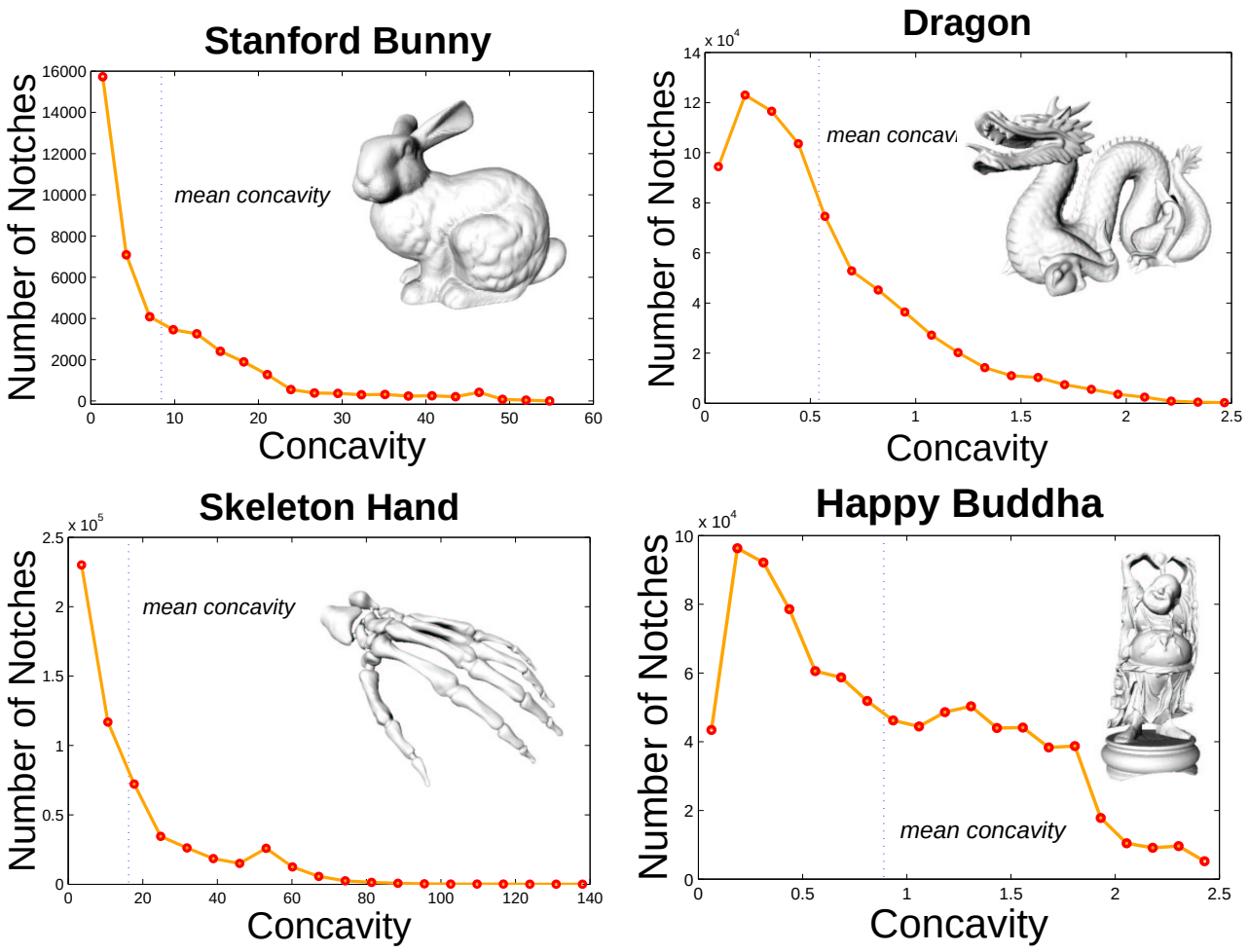


Figure 2: Large models from the Large Geometric Models Archive, Georgia Tech. Stanford Bunny. 35,947 vertices and 69,451 triangles. Dragon. 437,645 vertices and 871,414 triangles. Skeleton Hand. 327,323 vertices and 654,666 triangles. Happy Buddha. 543,652 vertices and 1,087,716 triangles.

parts and uses $O(nr^3)$ time and $O(nr^2)$ space.

In this work, we are interested in the decomposition of large and complicated polyhedra containing many notches. Due to hardware advances, such as faster CPUs and larger memories, the complexity of polyhedra used in CAD, computer graphics, game development, and simulation is increasing [25, 6]. Moreover, models that are created from range scanners and surface reconstruction normally consist of a huge number of triangles [15, 28]. For these detailed models, we observed that the proportion of edges that are notches becomes very large. For example, in Figure 2, 40.6% of the 104,288 edges of the Stanford Bunny are notches, 57.3% of the 1,309,256 edges of the dragon are notches, 57.3% of the 981,999 edges of the hand model are notches, and 54.5% of the 1,631,574 edges of the Happy Buddha are notches. Thus, in these models, roughly half the edges are notches. Assuming there are $\frac{n}{2}$ notches, the standard notch cutting approach requires $O((\frac{n}{2})^4)$ time and produces $O((\frac{n}{4})^2)$ convex components. Therefore, for such models, resolving all notches to obtain a convex decomposition requires not only an extremely expensive decomposition procedure, but also dramatically decreases the performance of the applications which use the decomposition due to the huge number of tiny components.

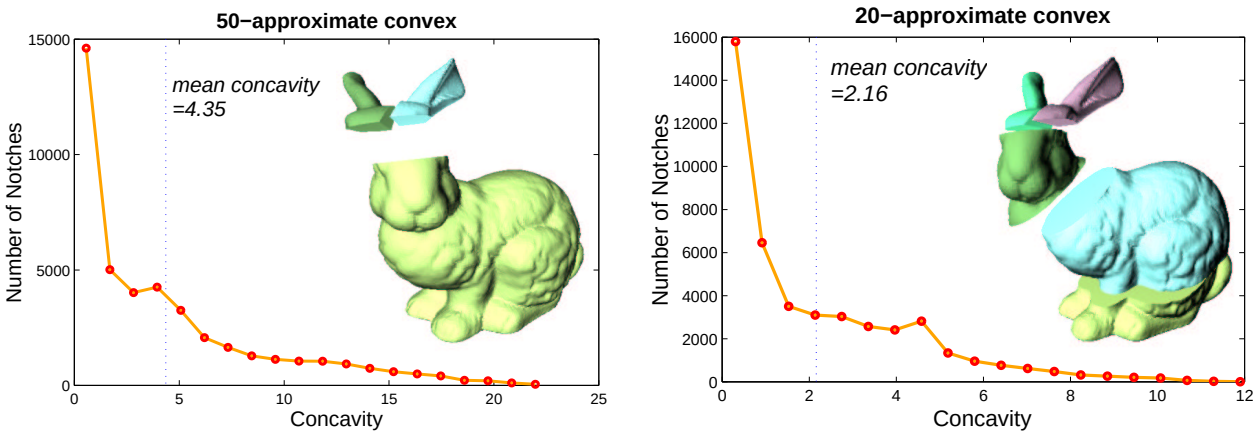


Figure 3: **Upper**: Each component is 50-*approximate* convex. There are 3 components. **Lower**: Each component is 20-*approximate* convex. There are 5 components.

1.1 Our Approach

In this paper, we propose a new approach to the decomposition problem. Our motivation is that for some models and applications, some of the notches can be considered *insignificant*, and allowed to remain in the final decomposition, while others are more important, and must be resolved. Intuitively, important notches provide key structural information while insignificant notches have little effect on the application and, perhaps, on visualization. Thus, our strategy is to identify and resolve the important notches. We develop a measure of the importance of a notch that is related to the *concavity* of the polyhedron due to that notch. In particular, we define a τ-*approximate* convex component to be a polyhedron whose *concavity* is at most τ (See Figure 4(a)); details regarding the measurement of concavity are described in Section 4. Figure 2 shows the notch concavity distribution for the models studied. As is easily seen, initially there are only a few notches that have large concavity. In Figure 3, the bunny is decomposed into 50-*approximate* and 20-*approximate* convex components. This example illustrates that resolving significant notches reduces the concavity of the remaining notches dramatically.

Thus, instead of looking for an optimal and exact solution in terms of complexity and convexity, respectively, our method decomposes the given polyhedron into approximately convex components. With this approximation approach, notches produced by wrinkles or boundary turbulence of detailed models will be ignored and only important notches will be separated. As we will demonstrate, the number of resulting components can be significantly less than with traditional exact decomposition approaches. The time complexity of the proposed algorithms is a function of the tolerance and complexity/smoothness of the model. Smaller tolerances and more jagged models result in higher time complexity. Furthermore, the cutting order of the notches in our method produces a useful hierarchical representation of the polyhedron, where the approximation factor decreases monotonically with the level in the hierarchy. Other benefits of and applications for this approximation method are discussed in Section 4.

2 Related Work

Two-dimensional polygons. Many approaches have been proposed for decomposing two-dimensional polygons, which is significantly easier than decomposing three-dimensional polyhedra. The problem of convex decomposition is normally subject to some optimization criteria to produce a minimum number of convex components or to minimize the internal length of the components. Convex decomposition methods can be classified according to whether (1) the input polygon is simple (has no holes), and (2) Steiner points (additional vertices) can be introduced during the decomposition.

For polygons *with holes*, under both the *minimum components* and the *shortest internal length* criterion, the problem is NP hard when either allowing or disallowing Steiner points [22, 23, 20]. For polygons without holes,

when Steiner points are not allowed, there exist $O(r^2n^2)$ [16] and $O(r^2n\log n)$ [20, 19] algorithms for generating polygons with a minimum number of convex components. When Steiner points are allowed, Chazelle and Dobkin [10, 11] use an X_k-pattern to remove k notches at once while creating no new notches. Their algorithm generates a minimum number of convex components in $O(n + r^3)$ time. Using the shortest internal length criterion and without generating Steiner points, Greene [16] developed an $O(r^2n^2)$ time algorithm and Keil [19] presented an $O(r^2n^2\log n)$ time approach. When Steiner points are allowed, there are no known optimal solutions.

Three-dimensional polyhedra. Convex decomposition of three-dimensional polyhedra is not as well understood as the two-dimensional case. Although this topic has been researched for a few decades, most of the work focuses on refining the complexity requirements of Chazelle's popular notch cutting approach. Indeed, Chazelle's approach has inspired many other researchers to find more robust and efficient implementations.

A *notch* of a manifold polyhedron is an edge with *dihedral angle* of at least 180°, where the dihedral angle of an edge is defined as the internal angle between its two incident facets. To resolve a notch, a *cutting plane*, HP, passing through the notch separates the incident facets, f_1 and f_2, and results in a decomposition where the dihedral angles of (HP, f_1) and (HP, f_2) are both less than 180° (See Figure 4(b).)

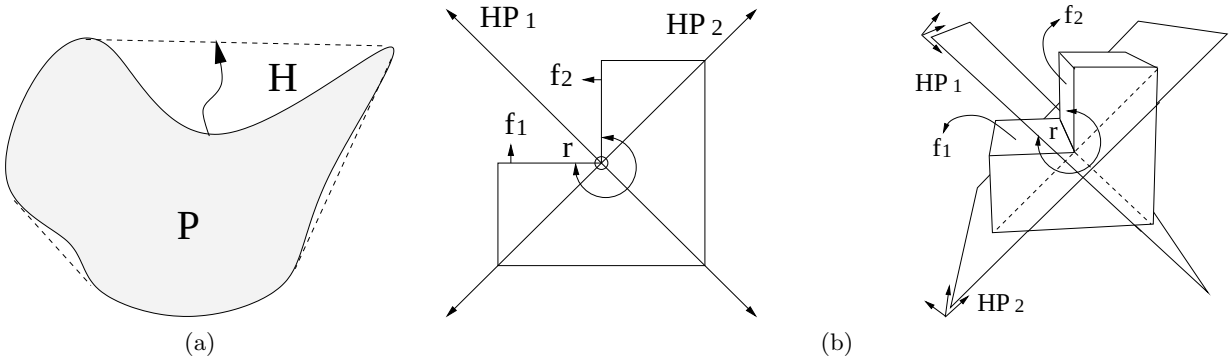


Figure 4: (a) This polygon is a τ-*approximate* convex component if the measurement of its concavity is less than τ. (b) The dihedral angle of r is the internal angle of f_1 and f_2. Note that this angle is larger than the 180°, r a notch. Both HP_1 and HP_2 contain r but only HP_1 cuts the dihedral angle of r into two less than 180° angles. HP_1 is the cutting of r and HP_2 is not.

Chazelle [8, 9] 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 [3] 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 [1] 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 [17] recently 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.

3 Preliminaries

In this section, we define the notation used in this paper.

A manifold polyhedron P represented by its boundaries ∂P consists of a set of vertices, a set of ridges, and a set of facets, where $\partial P = \{\partial P_0, \partial P_1, \ldots, \partial P_t\}$ is a set of disjoint *2-manifold* boundaries and ∂P_0 is the outer-most boundary and $\partial P_{i\geq 1}$ are hole boundaries of P. For simplicity, and without loss of generality, we will assume that the space enclosed by $\partial P_{i\geq 1}$ is empty.

The convex hull of P, $\mathcal{H}$, is the smallest convex set of vertices of P. Let H be the boundary of $\mathcal{H}$, i.e. $H = \partial\mathcal{H}$. P is said to be convex if $\mathcal{H} = P$. A set of components is said to be a *partition* of P if $\text{PART}(P) = \{C_i \mid \bigcup_i C_i = P \text{ and } \bigcap_i C_i = \emptyset\}$. Then a *convex decomposition* of P is a partition of P that contains only convex components, i.e., $\text{CD}(P) = \{C_i \in \text{PART}(P) \mid C_i \text{ is convex}, \forall i\}$.

A traditional way to generate convex decomposition is to cut notches iteratively in arbitrary order. However, to cut P, it is more intuitive to start the cut from the most *visually noticeable features*, such as the most dented or bent area or the area with branches. We denote this property as *concavity* and use $concave(r)$ to denote some measure of the concavity of the notch r. The concavity of a polyhedron is the maximum concavity of its notches (i.e. $concave(P) = \max_{r\in P}(concave(r))$.) The significance of a notch is provided by the measurement of its concavity. That is a notch r is visually important if $concave(r)$ is large.

P is a *τ-approximate* convex if $concave(P)$ is less than τ (See Figure 4(a).) Note that if P is *0-approximate* convex, then P is convex. The *τ-approximate* convex decomposition of P is defined as a partition that contains only *τ-approximate* convex components; i.e., $\text{CD}_\tau(P) = \{C_i \in \text{PART}(P) \mid concave(C_i) \leq \tau, \forall i\}$. When $\tau = 0$, $\text{CD}_\tau(P)$ is equal to $\text{CD}(P)$. The goal of this paper is to generate *τ-approximate* convex decompositions.

To measure concavity, we are interested in the relationship between *bridges* and *pockets* of P. Let f_H be a facet of H. f_H is a member of a bridge of P iff $f_H \setminus \partial P$ is not empty, i.e., f_H does not entirely lay on the boundary of P. The bridge of P is then defined as : $bridge(P) = \{f_H \mid f_H \in H \text{ and } f_H \setminus \partial P \neq \emptyset\}$. Let f_P be a face of ∂P. f_P is an element of the pocket of P iff $f_P \setminus H$ is not empty. Therefore the pocket of P is defined as : $pocket(P) = \{f_P \mid f_P \in \partial P \text{ and } f_P \setminus H \neq \emptyset\}$. For convenience, we indicate a member of $bridge(P)$ ($pocket(P)$) as a bridge (pocket). Note that the incident faces of a notch of P must be in $pocket(P)$. Examples of bridges and pockets for a two-dimensional case are shown in Figure 7(a) and for a three-dimensional case are shown in Figure 9(a).

Now, we define notation used for separating P into two components for a given cutting plane, HP. Let $\mathcal{F}_i$ be a set of facets in ∂P_i that intersect HP and $\mathcal{I}_i$ be the intersection of $\mathcal{F}_i$ and HP. $\mathcal{I}_i$ contains a set of points in two dimensions and a set of polygons with holes and islands in three dimensions. Since facets in ∂P do not intersect each other, for any given pair, $\mathcal{F}_i$ and $\mathcal{F}_j$, we must be able to determine if $\mathcal{F}_i$ is completely in $\mathcal{F}_j$ or not. Let $C(\mathcal{I}_i)$ denote a set of points on HP enclosed by $\mathcal{I}_i$. $\mathcal{F}_i$ is inside $\mathcal{F}_j$ if and only if $\mathcal{I}_i \in C(\mathcal{I}_j)$. This relationship helps fill holes after separating P. Let $\widehat{\mathcal{F}}_i$ be a sub-set of facets in $\mathcal{F}_i$ that separate ∂P_i into exactly two sub-boundaries if facets in $\widehat{\mathcal{F}}_i$ are all split. Notice that $\widehat{\mathcal{F}}_i$ is not unique for a given $\mathcal{F}_i$. Let the intersection of $\widehat{\mathcal{F}}_i$ and HP be $\widehat{\mathcal{I}}_i$. A polygonal example is provided in Figure 6(b).

4 Approximate Convex Decomposition

In this section, we propose possible approaches for measuring polyhedral concavity and sketch a framework algorithm (Algorithm 4.1) for approximating convex decomposition. The algorithm itself is very simple. Details for 2D polygons and 3D polyhedra will be provided in later sections.

4.1 Measurement of Concavity

In contrast to radius, surface area, and volume, concavity itself does not have a well defined meaning. Hence, we need a quantitative way to measure the concavity of a notch, and therefore of a polyhedron. In this paper, we use the distance from the notch to the convex hull of the model to represent the concavity (i.e. $concave(r) =$

$dist(r, H)$) since convexity properties are more well defined. This distance tells us how far the notch is away from the convex hull.

Let $retraction(r, H, t) : \mathcal{H} \rightarrow \mathcal{H}$ denote the function that defines the trajectory of the notch r when r is retracted from its original position to H. When $t = 0$, $retraction(r, H, 0)$ is r itself. When $t = 1$, $retraction(r, H, 1)$ is the final position of the notch on H. Thus, the $dist(r, H)$ can be computed by integrating the function $retraction(r, H, t)$ from $t = 0$ to 1. An intuition of this distance definition is illustrated in Figure 5(a). Think of P as a balloon which is placed in a mold with the shape of H. 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 H can be defined as the path traveled for a point on the balloon from initial shape to its final shape. Although the intuition is simple, a retraction path such as path a in Figure 5(a) is not easy to compute. It may be approximated as *path* b.

Note that, by pumping air, points on the hole boundary will not touch H, the concavity of notches in holes, will be infinity. To resolve these notches, denoted by r_{hole}, we use the fact that holes will "vanish" to curves or a curved surfaces in the final shape, and this vanished hole can be a guide to find the cutting plane to resolve r_{hole}. Since the cutting plane will be part of the convex hull of the new components, the concavity of r_{hole} in new components can be estimated as the distance from r_{hole} to the cutting plane. To minimize this concavity, we try to align the cutting plane with the vanished hole. Figure 5(b) shows a vanished hole and two possible cutting lines to resolve this hole. The concavity of the notches in the new components created using HP_1 will be less than that using HP_2.

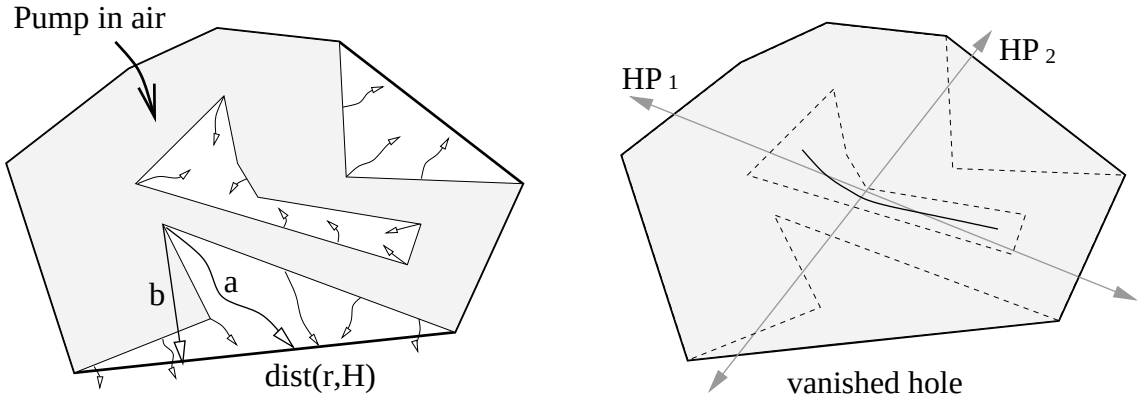


Figure 5: (a) The initial shape of a non-convex balloon (shaded). The bold line is the convex hull of the balloon. When we inflate the balloon, points not on the convex hull will be pushed toward the convex hull. Path a denotes the trajectory with air pumping and path b is an approximation of a. (b) The hole vanishes to a curve and vertices on the hole boundary will not touch the convex hull.

4.2 Framework Algorithm

Algorithm 4.1 Approx_CD(P, τ)

Input. A polyhedron, P, and tolerance, τ.
Output. $max(concave(C_i)) \leq \tau$, $P = \bigcup_{i=1}^{N} C_i$ and $\bigcap_{i=1}^{N} C_i = \emptyset$.

```
1: Build the convex hull, H, for P.
2: Let r be the notch with maximum dist(r, H).
3: if dist(r, H) < τ then
4:    return P.
5: end if
6: Let HP be the cutting plane of r. HP bisects P into P1 and P2.
7: return Approx_CD(P1,τ) and Approx_CD(P2,τ).
```

Algorithm 4.1 describes a *divide-and-conquer* strategy to decompose P into a list of τ-*approximate* convex pieces. The algorithm first computes the convex hull of P and finds the notch, r, with the maximum distance away from the convex hull. A hyperplane is then defined to resolve r and separate P into two components P_1 and P_2. The combination of the results of the two recursive calls for P_1 and P_2 will be our final answer. The recursion terminates when the distance from r to the convex hull is less than τ.

Note that this recursive approach is different from cutting notches with large concavity to small concavity iteratively because the notch concavity (but not the notch) will change after every cut. Moreover, this algorithm requires the cut to separate P into two pieces instead of only resolving a notch which may or may not separate P as in [8, 9, 3].

For a given notch r and its cutting plane HP, Algorithm 4.2 separates P into P_1 and P_2. The first step of Algorithm 4.2 computes the facets of ∂P_0 that intersect HP and have r in its enclosing space $C(\widehat{\mathcal{I}}_0)$. Next, for each hole boundary ∂P_i, $\forall i \geq 1$, if ∂P_i intersects HP and its intersection is inside $\widehat{\mathcal{F}}_0$, we will compute and split facets in $\widehat{\mathcal{F}}_i$. The last step in this separation process fills holes and group boundaries in P_1 and P_2.

Note that it is necessary to maintain the manifold property for decomposed components. Even if the input polyhedron is manifold, it is possible to generate non-manifold components [3]. In Figure 6(a), the vertex s is *2-manifold* before the polyhedron is decomposed by resolving notch r but may not be so after the cutting. Disturbing s to above or below HP can prevent this from happening.

Algorithm 4.2 Separate(P, HP, r)

Input. A polyhedron, P, a cutting plane HP, and a notch r.
Output. HP bisects P into P_1 and P_2.

```
1: Compute F̂_0 from ∂P_0 so that r ∈ C(Î_0).
2: Split all facets in F̂_0.
3: for each hole boundary ∂P_i, ∀i ≥ 1 do
4:    if F̂_i of ∂P_i is not empty and Î_i ∈ C(Î_0) then
5:       Split all facets in F̂_i.
6:    end if
7: end for
8: Fill holes from splitting and classify boundaries into P_1 and P_2.
```

This framework provides an $O(n\hat{r}(\tau)^3 + \hat{r}(\tau) * n \log n)$ time algorithm, where $\hat{r}(\tau)$ is number of cuts required to generate τ-approximate convex components; note there will be $\hat{r}(\tau) + 1$ such components. This value depends on the complexity (smoothness) of the model and the tolerance τ. The time for measuring the concavity is $\hat{r}(\tau) * n \log n$. Each of the $\hat{r}(\tau)$ concavity measurements involves an $O(n \log n)$ convex hull calculation.

4.3 Benefits of the Framework

Algorithm 4.1 has several advantages. First, approximate decomposition will reduce the number of unnecessary cuts which generate numerous insignificant pieces and degrade the performance of the decomposition itself and its applications. Decomposed components also convey this important property. In addition, level of detail (LOD) which is normally used in terms of visualization can also be applied to simulate the use of different tolerance rates. Usually, simulations like physically-based animation can tolerate less accurate results when the simulated objects are far away or not in the region of interest. In this case, a rough decomposition will be sufficient. Therefore, this LOD for simulation can be adaptively achieved by refining the decomposition approximation using a smaller τ when higher accuracy is required and retrieving coarser level in decomposition hierarchy. The value for τ can be specified according to the image size rendered on the screen.

Second, due to the recursive nature, the resulting decomposition is a hierarchical binary tree. The convex hull of the original model P is the root of the tree which has two children as convex hulls of P_1 and P_2. Leaves of the tree are approximate convex components. This kind of representation has useful properties and is used in many areas like virtual reality for scenes graph and constructive solid geometry (CSG) for set operations and collision detection for fast rejection.

Third, as an example in pattern recognition, features are extracted from images, polygons, and polyhedra to represent the shape of the objects. This process (skeleton extraction) is usually affected by the turbulence of the

boundary which reduces the quality of the extracted features. The proposed approximate method can improve the quality by extracting a skeleton from the convex hull of the decomposition components. The same idea can be applied to other problems to avoid boundary turbulences.

An implementation of Algorithm 4.1 and 4.2 for two and three dimensional models is described in Section 5 and 6, respectively.

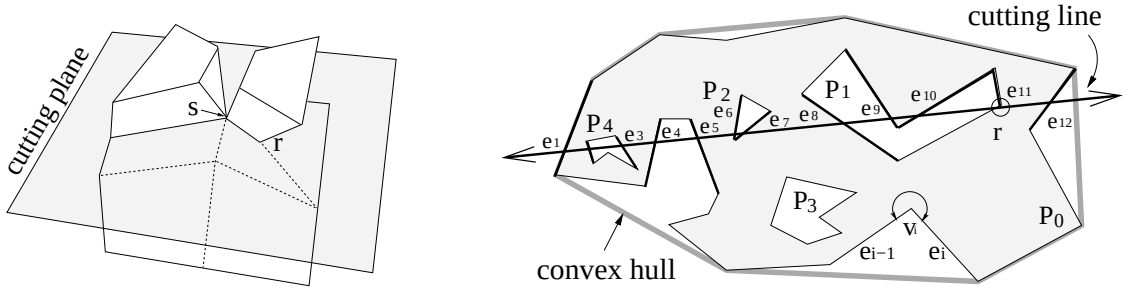


Figure 6: (a) Vertex s will be non-manifold after r is resolved. It is necessary to maintain the manifold property for decomposition components. (b) An example of polygon with holes. P_0 is the polygonal chain and P_1 to P_4 are boundary of holes. $e_1, \ldots, e_{12}$ are intersecting edges. $\mathcal{F}_0 = (e_1, e_4, e_5, e_{12})$ and $\widehat{\mathcal{F}}_0 = (e_5, e_{12})$. $\widehat{\mathcal{F}}_1 = (e_8, e_{11})$.

5 Convex Decomposition of 2D Polygon

Although there are optimal solutions for two-dimensional polygon decomposition under certain criteria, our algorithm is easier and more easily demonstrated in two-dimensional examples. The ideas are useful for two-dimensional problems also help us understand three-dimensional problems.

A simple polygon, P, with an arbitrary number of holes can be represented by a list of polygonal chains.

5.1 Measurement of Concavity

In this section, we will discuss how the $dist(r, H)$ is defined in Section 4 and Figure 5 can be achieved for polygons. Recall that each pocket can be associated with exactly one bridge and notches will only show up in pockets. We approximate the notch r on ∂P_0 by computing the straight-line distance from r to the bridge of hosting pocket of r. Let the vertices in the outer most polygonal chain of P be numbered from 0 to $n-1$ as their id and if vertex v_i and vertex v_j are connected then $|id(v_i) - id(v_j)|$ must be one, here $id(v_i)$ is the numerical id of v_i.

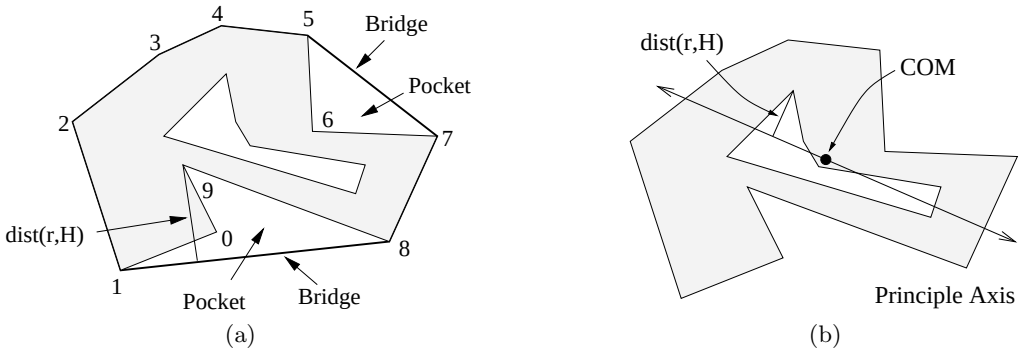


Figure 7: (a) Edges (8,1) and (5,7) are bridges, and (5,6) and (8,9), (9,0), (0,1) are both pockets. dist(9,H) is the distance from vertex 9 to the line defined by (8,1). (b) COM is the center of mass of all vertices on the hole boundary. The principal axis is the cutting line of this hole.

To find bridges and pockets, each vertex on the convex hull is examined iteratively. If the difference between the id of the current vertex v_i and the id of the previously examined vertex v_j is larger than one, edge (v_j, v_i) is a bridge and the edges containing vertices with ids from $id(v_j) + 1$ to $id(v_i) - 1$ are in the pocket. Figure 7(a) illustrates this process. Algorithm 5.1 shows how to compute distances to the convex hull for notches on the outer most boundary. After the most concave notch r is identified, the cutting line, HP, is selected to align with the vector $\vec{n_{f1}} + \vec{n_{f2}}$ and passing through r, i.e., $\vec{n_{f1}}$ and $\vec{n_{f2}}$ are normals of incident edges of r.

Algorithm 5.1 Dist2Hull(∂P_0,H)

Input. The out-most boundary of P and its convex hull H.

```
1: Find all bridges on H and pockets on ∂P_0.
2: for each bridge, b, and pocket, p do
3:    Let a line l align with b.
4:    for each notch r in p do
5:       Compute distance from r to l.
6:    end for
7: end for
```

The notches on hole boundaries have infinite concavity. To resolve these notches, we need to find a cutting line that reduces the concavity of these notches to be finite. In fact, any line penetrating the hole and splitting it into two parts can reduce the concavity to finite. Our goal to this problem is to find a cutting line that minimize the resulting concavity.

As mentioned in Section 4.1, the best candidate is the cutting line that aligns with the vanished hole. Nevertheless, computing a vanished hole is expensive. To avoid computing the vanished hole, we approximate this cutting line by computing the principal axis of the hole (See Figure 7(b).) The Principal axis for a given set of points can be computed as the Eigen vector with the largest Eigen value from the covariance matrix of these points. Algorithm 5.2 computes a cutting line for notches in the hole. After a hole is resolved, these notches will become part of the outer most boundary of the new polygon and their concavity can be measured using Algorithm 5.1.

Algorithm 5.2 CuttingLine_For_Hole(∂P_i)

Input. The hole boundary ∂P_i of P, $i > 0$.

```
1: Compute principal axis, pa, from ∂P_i
2: Report pa as the cutting line.
```

5.2 Split Polygon

To separate P into two components using this HP, we need to (1) compute $\widehat{\mathcal{F}}_0$ with $r \in C(\widehat{\mathcal{I}}_0)$, (2) find ∂P_i enclosed by $C(\widehat{\mathcal{I}}_0)$ for $i > 0$, and (3) split the intersecting edges, fill the hole, and classify the boundaries into two components.

To compute $\widehat{\mathcal{F}}_0$ that contains r in its enclosing area, we first find all the edges on ∂P_0 that intersect HP. Then, from this list, we find the edge e_0^l that is closest to the left side of r and the edge e_0^r that is closest to the right side of r along HP. Finally, let $\widehat{\mathcal{F}}_0$ be $\{e_0^l, e_0^r\}$. $\widehat{\mathcal{F}}_0$ in Figure 6(b) is $\{e_5, e_{12}\}$.

For a given hole boundary ∂P_i, we can check if the intersecting edges of ∂P_i are enclosed in $C(\widehat{\mathcal{I}}_0)$. If so, $\widehat{\mathcal{F}}_i$ then is identified by the left most edge and e_i^r and the right most edge e_i^r along HP. In the Figure 6(b), $\widehat{\mathcal{F}}_1$ is e_8, e_{11} and $\widehat{\mathcal{F}}_2$ is e_5, e_6 for P_1 and P_2, respectively.

In the last step, all edges in $\widehat{\mathcal{F}}_0$ and $\widehat{\mathcal{F}}_i$ are ordered and every consecutive pair of edges, e_i and e_j, is split and connected. In the Figure 6(b), (e_5, e_6), (e_7, e_8), (e_{11}, e_{12}) are pairs that will be split and connected. To classify boundaries into two groups, P_1 and P_2, a sweep line approach [2] can solve this polygon nesting problem. The two dimensional implementation of Algorithm 4.2 is sketched in Algorithm 5.3. Figure 8 shows two examples of decomposition with different thresholds. It is notable that the number of decomposed components is much less than the number of notches.

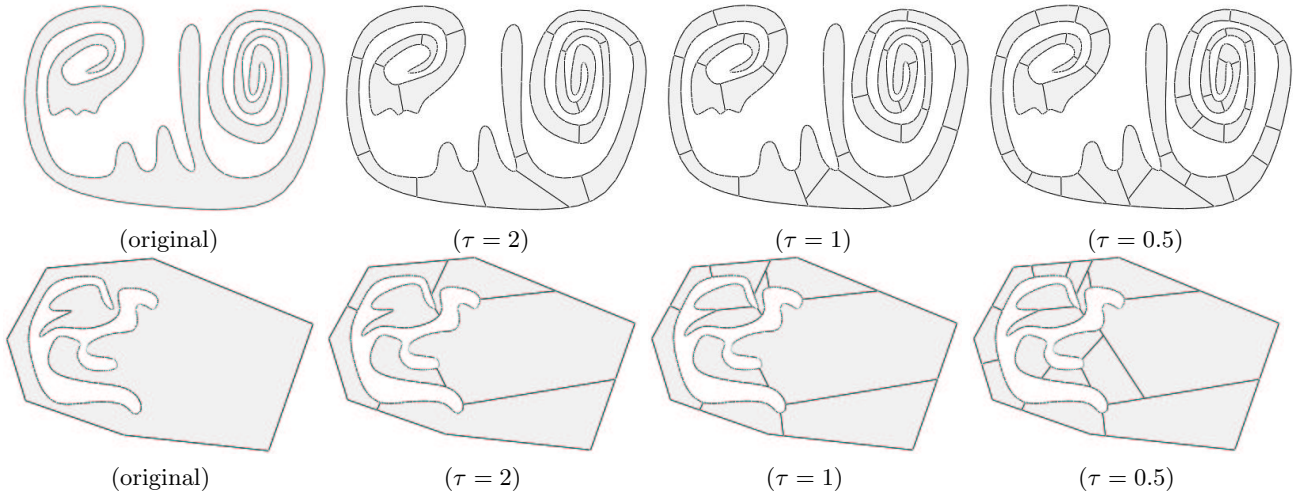


Figure 8: **Up** : The original polygon has 452 vertices and 211 notches. When $\tau = 2$, there are 18 convex components. When $\tau = 1$, there are 26 convex components. When $\tau = 0.5$, there are 36 convex components. **Down** : The original polygon has 487 vertices and 269 notches and one hole. When $\tau = 2$, there are 7 convex components. When $\tau = 1$, there are 11 convex components. When $\tau = 0.5$, there are 16 convex components.

Algorithm 5.3 Separate2D(P, HP, r)

1: $\widehat{\mathcal{F}}_0 \leftarrow e_0^l$ and $\widehat{\mathcal{F}}_0 \leftarrow e_0^r$.
2: **for** each hole boundary ∂P_i, $\forall i \geq 1$ **do**
3: **if** e_i^l and e_i^r are enclosed by e_0^l and e_0^r **then**
4: $\widehat{\mathcal{F}}_i \leftarrow e_i^l$ and $\widehat{\mathcal{F}}_i \leftarrow e_i^r$.
5: **end if**
6: **end for**
7: Sort edges in $\widehat{\mathcal{F}}_0$ and $\widehat{\mathcal{F}}_i$ and split each consecutive pair of edges.
8: Classify boundaries into P_1 and P_2.

6 Convex Decomposition of 3D Polyhedron

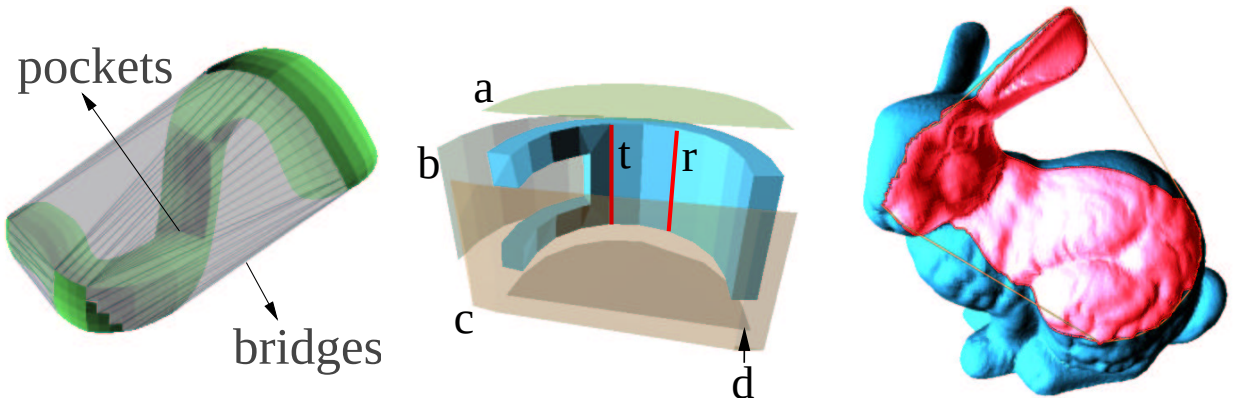
The abstract topological map [29] is used to represent the boundaries of a polyhedron P.

6.1 Measurement of Concavity

Similar to the polygonal case, the concavity of the notch r on ∂P_0 is measured by computing the distance from r to the bridge of the hosting pocket of r. However, unlike the polygonal case, pockets and bridges do not have a one-to-one correspondence in the polyhedral case (See Figure 9.) A notch is directly or indirectly connected to multiple bridges. This means that there is more than one way for a notch to retract to H, but only one of the retractions conveys the most meaningful concavity measurement. Simply determing the maximum or the minimum distance will not obtain the information we want. For example, in Figure 9(b), notch r can be retracted in four possible directions to four different bridges (i.e., bridge a, b, c, and d.)

In fact, retraction is a global operation. This means that the retraction direction of a notch is affected by the retraction directions of other notches. For instance, in Figure 9(b), if notch r is retracted in direction b, the notch t is more likely to be retracted in the similar direction, otherwise the trajectories of t and r will intersect. However, pushing t in the direction of b actually increases the concavity of t. Only the direction that retracts to the bridge c is more natural.

We propose a simple heuristic (Algorithm 6.1) to associate a notch with one single bridge, and then the concavity is the straight-line distance from the notch to that bridge. Let $b \in bridge(P)$ and $e_b = (p_b, q_b)$ be an edge of b defined by end points. Then we compute a shortest path on ∂P_0 from p_b to q_b and do so for all edges



(a) (b) (c)

Figure 9: (a) An example of pockets and bridges. (b) Notch r can be retracted to H in four possible directions, (c) Partition ∂P based on bridges.

of b. Finally, the notches enclosed in these paths of b are now associated with b only. After applying this to all bridges of P, ∂P_0 is partitioned into patches. Figure 9(c) illustrates the result of this process.

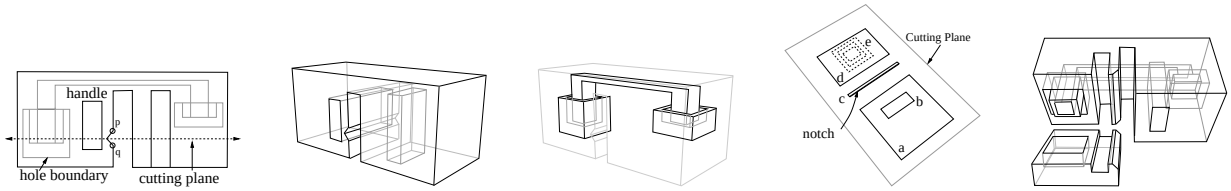
Algorithm 6.1 Dist2Hull(∂P_0,H)

Input. The out-most boundary of P and its convex hull H.

```
Find all bridges on H and pockets on ∂P_0.
for each bridge b do
    for each edge e = (p, q) of b do
        Find the shortest path on ∂P_0 from p to q.
    end for
end for
for each notch r do
    Find which patch r belongs to and compute distance to its bridge.
end for
```

The concavity for the notch r on ∂P_i, $i > 0$, is infinite. As in Section 5.1, the cutting plane is selected as the principal plane of the vertices on ∂P_i.

6.2 Split Polyhedron



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

Figure 10: (a) The side view of the polyhedron. (b) The out-most boundary of the polyhedron. (c) The hole boundary of the polyhedron. (d) The intersection between cutting plane and the polyhedron. (e) The polyhedron is separated into two sub-components.

After the cutting plane is selected, we are ready to cut P. Our goal is to (1) split P into exactly two sub-components, and (2) maintain the manifold property. We use the same technique used in Section 5.2 to satisfy both requirements.

To bisect P, similar to the polygonal case, we need to (1) compute $\widehat{\mathcal{F}}_0$ with $r \in C(\widehat{\mathcal{I}}_0)$, (2) find ∂P_i enclosed by $C(\widehat{\mathcal{I}}_0)$ for $i > 0$, and (3) split the intersecting facets, fill the hole, and classify boundaries to sub-components. The polyhedron in Figure 10 will be used to illustrate this splitting process.

$\widehat{\mathcal{F}}_0$ is a set of facets of ∂P_0 that contains r in its enclosing area $C(\widehat{\mathcal{I}}_0)$. An example of $\mathcal{I}_0$ is shown by the solid-line boundaries in Figure 10(d). Algorithm 6.2 sketches the process to compute $\widehat{\mathcal{F}}_0$ from $\mathcal{F}_0$. First we find the smallest polygon in $\mathcal{I}_0$ that encloses r or has r on its boundary and split the facets along this polygon. Then, all the other polygons are tested iteratively until ∂P is bisected. In our example, $\widehat{\mathcal{F}}_0$ will contain two polygons (c) and (d) and polygon (a, b) will be ignored.

Algorithm 6.2 Find$\widehat{\mathcal{F}}_0(\mathcal{F}_0, r)$

```
Let s be the smallest polygon in I_0 that encloses r.
Split facets in F_0 along s and put these facets into F̂_0.
Let p and q be vertices on different side of s.
while p and q are connected do
    Let t be an unhandled polygon and split facets in F_0 along t.
    if (p and q are disconnected) or (∂P_0 is not split) then
        Put split facets into F̂_0.
    end if
end while
```

To split the hole boundary ∂P_i, $i > 0$, that is enclosed in $\widehat{\mathcal{F}}_0$, $\widehat{\mathcal{I}}_i$ only contain boundaries that are not enclosed in any other boundaries in ∂I_i. In Figure 10(d), boundaries in $\mathcal{I}_i$ are shown as dashed lines and e is the only member of $\widehat{\mathcal{I}}_i$ and all other boundaries contained in e are ignored. $\widehat{\mathcal{F}}_i$ contains facets that is split along boundaries in $\widehat{\mathcal{I}}_i$.

In the last step, all facets in $\widehat{\mathcal{F}}_0$ and $\widehat{\mathcal{F}}_i$ are ordered into a nesting hierarchy. From this hierarchical tree, we fill holes enclosed by boundaries in level $2i$ and level $2i + 1$. For instance, we need to fill the hole enclosed by d and e which are from level 0 and 1 in the hierarchy. Remaining hole boundaries will then be classified into P_1 and P_2 using sweep plane technique to solve the polyhedral nesting problem.

Algorithm 6.3 Separate3D(P, HP, r)

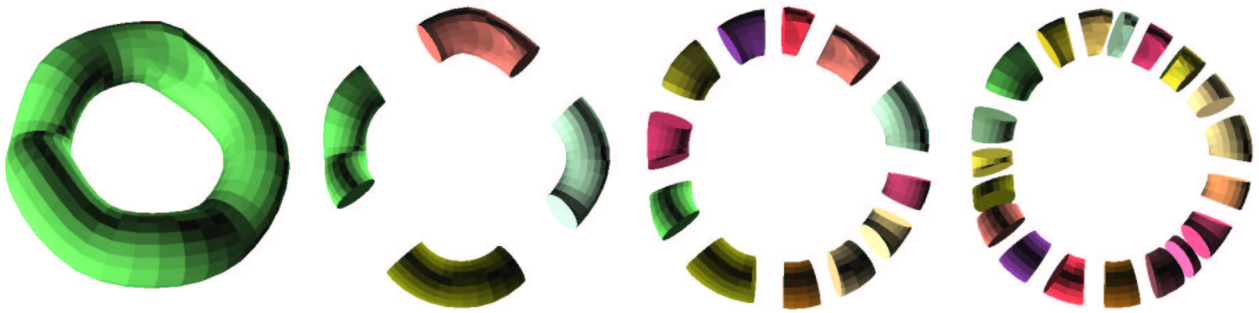
```
Compute F̂_0 by invoking Algorithm 6.2.
for each hole boundary ∂P_i, ∀i ≥ 1 do
    if F_i is enclosed in F̂_0 then
        Split facets along all out-most boundaries and put these facets into F̂_i.
    end if
end for
Build nesting hierarchy from all facets in F̂_0 and F̂_i.
Fill holes and classify boundaries into P_1 and P_2.
```

Decomposition examples are shown in Figure 11, 12, and 13. It is clear that number of decomposition components is far less than the number of notches, and these components have strong visual effect, such as legs, the tail, and the head that can be easily identified in Figure 12.

7 Skeleton Extraction

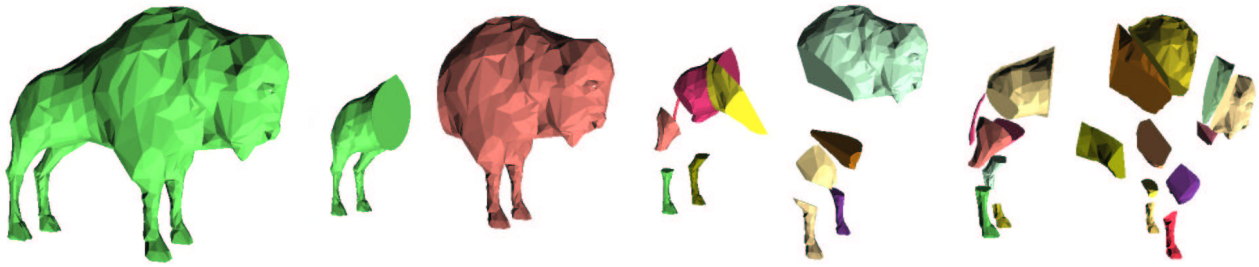
We present a novel skeleton extraction method to demonstrate the power of our approximate convex decomposition. The Medial Axis Transform is the most well studied skeleton extraction method [5]. However, due to its high complexity in computation and structure, and sensitivity to noise, the Medial Axis is generally not a good shape descriptor for three-dimensional objects.

Recently, researchers focus on extracting one-dimensional skeletons from three-dimensional models. Nevertheless, most proposed methods are extended from two dimensional approaches. These methods either explicitly or implicitly use image (voxel) based modeling during skeleton extraction. [27, 4] explicitly using models obtained



(original) ($\tau = 4$) ($\tau = 1$) ($\tau = 0.5$)

Figure 11: The original polygon has 2400 edges and 555 notches and one hole. When $\tau = 4$, there are 4 convex components. When $\tau = 1$, there are 12 convex components. When $\tau = 0.5$, there are 19 convex components.



(original) ($\tau = 4$) ($\tau = 1$) ($\tau = 0.5$)

Figure 12: The original polygon has 3157 edges and 1077 notches and one hole. When $\tau = 4$, there are 2 convex components. When $\tau = 1$, there are 11 convex components. When $\tau = 0.5$, there are 18 convex components.

from MRI and CT data sets, and [31, 30] use boundary represented polyhedral models but implicitly use grid points enclosed to trace out skeletons. All image based approaches suffer from same problem: How fine should the voxel be? Coarse voxels tend to produce disconnected skeletons which lose the topology information of the original model. Fine voxels require more processing time. Determining the size of the voxel is not a trivial task. Without voxelizing the model, Lazarus and Verroust [14] extract the skeleton from polyhedral surfaces using *Level Set Diagram*. However, their method cannot deal with models with handles and the skeleton generated might penetrate surface.

Figure 14 shows our skeleton extraction process. For a given polyhedron P, the skeleton, S, is the *Principal Axis* (PA) of the convex hull of P. This is based on the fact that the PA of the convex object C must be inside C and represent a fairly good skeleton of C. We expect the same result for P if P is a τ-approximate convex and τ is small. Next, the quality of S is measured, i.e., S should not intersect P and should stay in the center [27, 4, 30]. If S is good enough, S is reported as the skeleton of P. Otherwise, P is decomposed into P_1 and P_2 and the skeleton of P is the combination of the skeletons of P_1 and P_2. Two skeletons are connected through the center of the intersection of P and the cutting plane that split P (See Figure 14.)

Figure 15, 16, and 17 illustrate the evolution of skeletons. Note that, unlike image-based methods which uniformly process all grids, our method saves time in the easy area and concentrates on refining the difficult area, i.e., the highly twisted area. Moreover, our method is robust in terms of shape description. We added noise to the boundary of the testing models. Although not quantitatively measured, the quality of the resulting skeletons is equally good as those without noise.

(original) ($\tau = 6$) ($\tau = 2$) ($\tau = 0.5$)

Figure 13: The original polygon has 3360 edges and 1055 notches and one hole. When $\tau = 6$, there are 8 convex components. When $\tau = 2$, there are 22 convex components. When $\tau = 0.5$, there are 55 convex components.

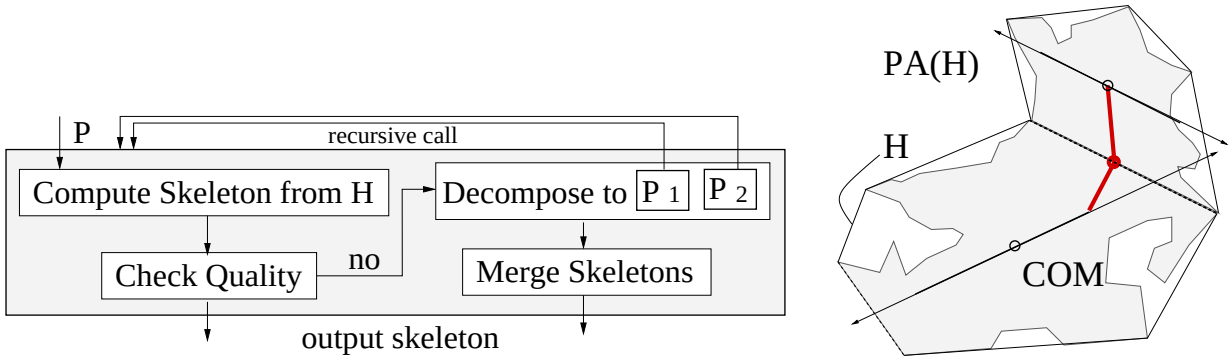


Figure 14: **Left** : Flow chart of skeleton extraction. **Right**: Merging two skeletons.

8 Conclusion

An approximate convex decomposition is presented. We show that most notches of complex models are insignificant in terms of visualization and simulation. Therefore, if some accuracy can be sacrificed, we can handle more practical problems. The significance of a notch is identified by calculating its *concavity*. We defined the concavity as the distance from the notch to the convex hull of the polyhedron. The decomposition starts by finding the notch with maximum concavity. If the maximum concavity is not tolerable, we resolve the notch and split the polyhedron into two components. Otherwise, this polyhedron is approximately convex.

In addition to efficiency, approximate convex decomposition has other interesting advantages, such as level of detail, a hierarchical structure, and insensitivity to noise. We believe that not only many applications, such as collision detection, physically-based simulation, and mesh generation, can benefit from these properties, but that we also need new applications inspired by this new approach, e.g. skeleton extraction and model simplification. In particular, skeleton extraction is used to demonstrate the power of our proposed approach.

Our future work will focus on refining the skeleton extraction process. By selecting the cutting plane carefully, we can further reduce the number of cuts required and therefore generate better skeletons. Little work has been proposed for this problem [18]. Another problem we will study is model simplification. The convex hull is known as one type of simplification. However, it has not been applied to practical problems. Figure 18 shows that model simplification can be performed by taking the union of the convex hulls of approximately convex components.

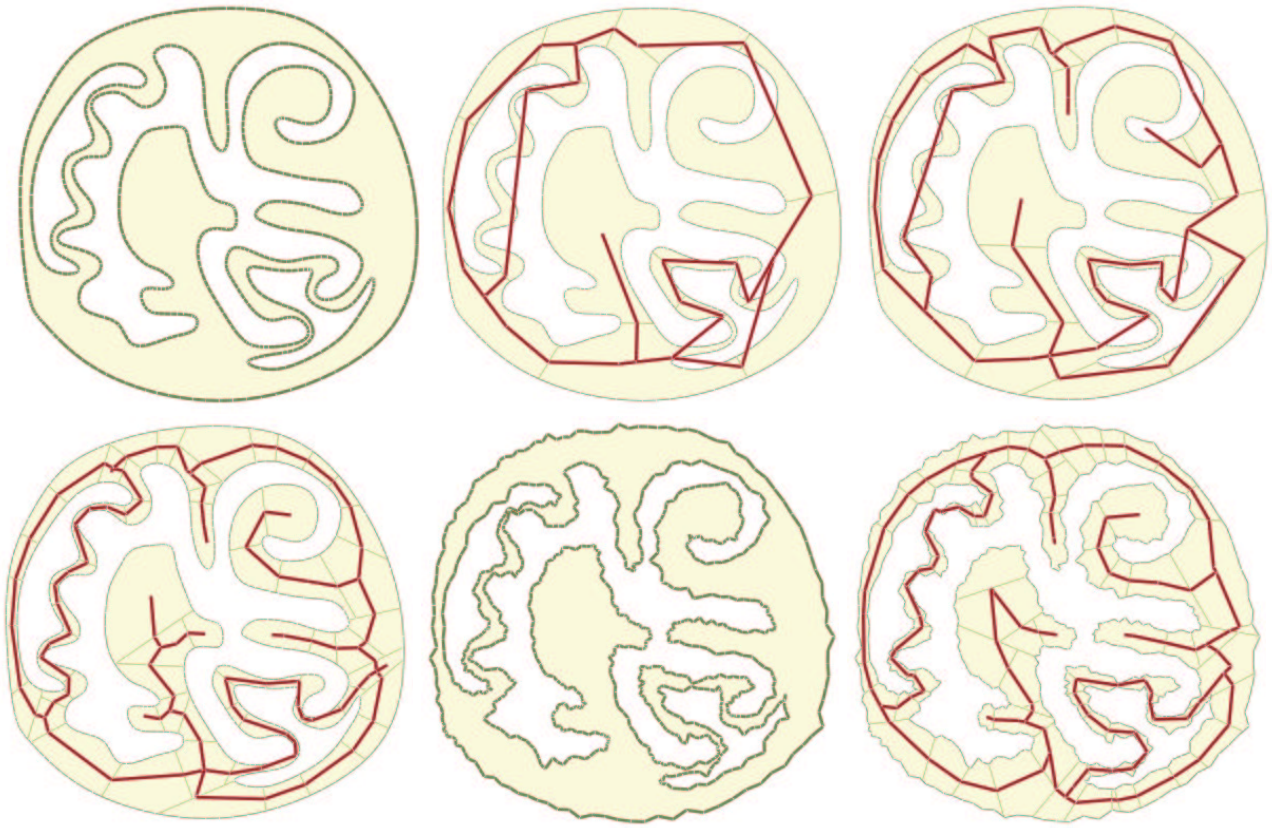


Figure 15: **Up**: The skeleton evolving during the decomposition. There are three holes in the polygon. **Down**: Vertices position is disturbed by Gaussian noise.

References

[1] C. Bajaj and T. K. Dey. Polygon nesting and robustness. In *Proceedings of the International Workshop on Discrete Algorithms and Complexity*, pages 33–40, Fukuoka, Japan, 1989. Institute of Electronics, Information and Communication Engineers (IEICE), Tokyo.

[2] C. Bajaj and T. K. Dey. Polygon nesting and robustness. *Inform. Process. Lett.*, 35:23–32, 1990.

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

[4] Ingmar Bitter, Arie E. Kaufman, and Mie Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):195–206, 2001.

[5] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[6] David Brickhill. Incredibly dense meshes. In *Proceedings of Game Developers Conference*, 2002.

[7] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popovic. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics*, 21(3):586–593, 2002.

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

[9] Bernard Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984.

[10] Bernard Chazelle and D. P. Dobkin. Decomposing a polygon into its convex parts. In *Proc. 11th Annu. ACM Sympos. Theory Comput.*, pages 38–48, 1979.

[11] Bernard Chazelle and D. P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pages 63–133. North-Holland, Amsterdam, Netherlands, 1985.

[12] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. In *International Conference on Intelligent Robots and Systems*. IEEE Computer Society, 2000.

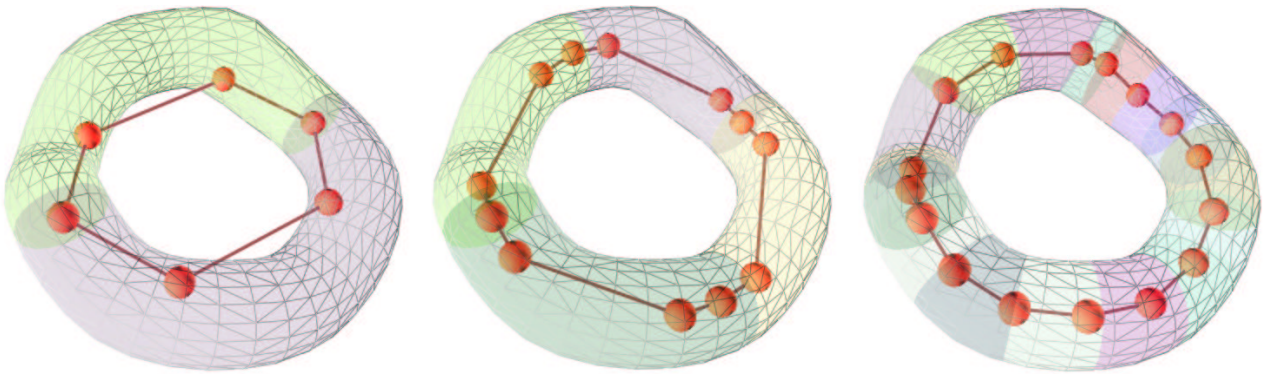


Figure 16: The skeleton evolving during the decomposition.

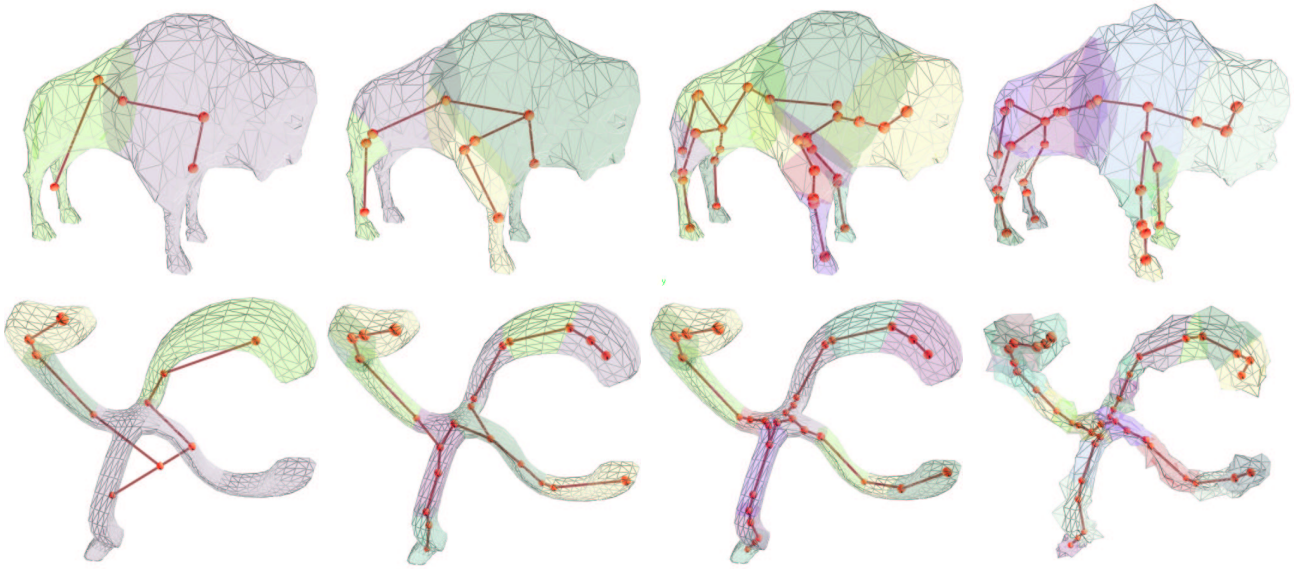


Figure 17: The first three columns show the evolution of the skeletons. The last column shows the skeletons of models that were disturbed by Gaussian noise.

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

[14] Lazarus Francis and Verroust Anne. Level set diagrams of polyhedral objects. Technical report rr-3546, The French National Institute for Reasearch in Computer Science and Ccontrol (INRIA), 1998.

[15] GATech. Large geometric models archive. http://www.cc.gatech.edu/projects/large_models.

[16] D. H. Greene. The decomposition of polygons into convex parts. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 235–259. JAI Press, London, England, 1983.

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

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

[19] J. M. Keil. *Decomposing Polygons into Simpler Components.* PhD thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1983.

[20] J. M. Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14:799–817, 1985.

[21] Young J. Kim, Miguel A. Otaduy, Ming C. Lin, and Dinesh Manocha. Fast penetration depth computation for physically-based animation. In *ACM Symposium on Computer Animation*, 2002.

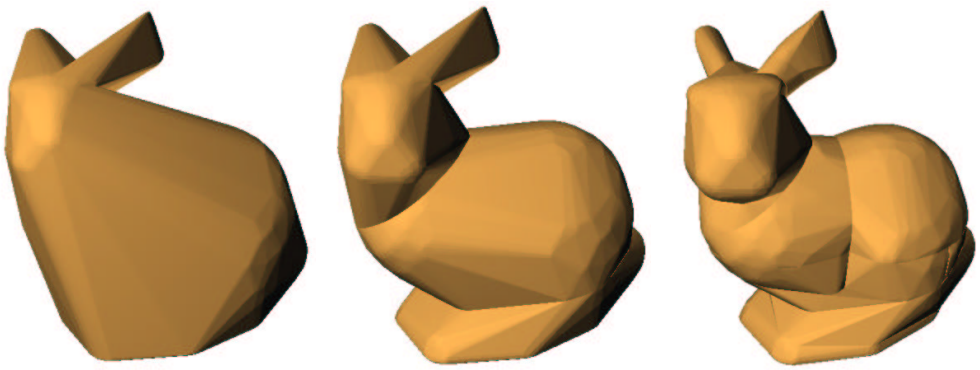


Figure 18: Model simplification using the union of convex hulls of approximately convex components.

[22] A. Lingas. The power of non-rectilinear holes. In *Proc. 9th Internat. Colloq. Automata Lang. Program.*, volume 140 of *Lecture Notes Comput. Sci.*, pages 369–383. Springer-Verlag, 1982.

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

[24] Yong Lu, Rajit Gadh, and Timothy J. Tautges. Volume decomposition and feature recognition for hexahedral mesh generation. In *8th International Meshing Roundtable*, pages 269–280, 1999.

[25] Dinesh Manocha, Mark J. Kilgard, Michael Doggett, Shankar Krishnan, Ming C. Lin, and Ned Greene. Interactive geometric computations with graphics hardware. In *SIGGRAPH2002 Course Notes*, 2002.

[26] J. O'Rourke and K. J. Supowit. Some NP-hard polygon decomposition problems. *IEEE Trans. Inform. Theory*, IT-30:181–190, 1983.

[27] H. Rom and G. Medioni. Hierarchical decomposition and axial shape description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):973–981, 1993.

[28] Stanford. The stanford 3d scanning repository. http://www-graphics.stanford.edu/data/3Dscanrep.

[29] Remco C. Veltkamp. Generic programming in CGAL, the Computational Geometry Algorithms Library. In F. Arbab and Ph. Slusallek, editors, *Proceedings of the 6th Eurographics Workshop on Programming Paradigms in Graphics, Budapest, Hungary, 8 September 1997*, pages 127–138, 1997.

[30] Fu-Che Wu, Wan-Chun Ma, and Ming Ouhyoung. Skeleton extraction of 3d objects with radial basis function. In *ACM Multimedia*, 2002.

[31] Yong Zhou and Arthur W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196–209, 1999.