Decomposing Polygon Meshes for Interactive Applications

Xuetao Li, Tong Wing Woon, Tiow Seng Tan and Zhiyong Huang

School of Computing
National University of Singapore

{ lixuetao | woontw | tants | huangzy }@comp.nus.edu.sg

Abstract

This paper discusses an efficient and effective framework to decompose polygon meshes into components. This is useful in various interactive graphics applications, such as, mesh editing, establishing correspondence between objects for morphing, computation of bounding volume hierarchy for collision detection and ray tracing. In this paper, we formalize the notion of a component as a sub-volume of an object with homogeneous geometric and topological features. Next, we describe the proposed framework, which adapts the idea of edge contraction and space sweeping to decompose an object automatically. Finally, we demonstrate an application of this framework to improve bounding volume hierarchies constructed by state-of-the-art collision detection systems such as RAPID and QuickCD.

Keywords

Geometric Modeling, Shape, Components, Collision Detection, Morphing, Ray Tracing, Spatial Data Structures.

1 Introduction

The polygon mesh is among the most common data structures used for representing objects in computer graphics. Geometrically, it is a piecewise linear surface consisting of a collection of polygons pasted along the edges. Its popularity stems from: (1) simplicity for fast rendering; (2) efficiency in interactive applications such as collision detection; (3) versatility in easy transformation to other representations; and (4) providing a good approximation of real world objects. Recent advances in 3D scanning and acquisition technology have also ascertained the importance of polygon meshes in representing complex objects. Thus, the polygon mesh is often used as a common denominator for digital models.

NC, March 19—21, 2001

ACM Symp. on Interactive 3D Graphics, pp35—42, pp. 243

Unfortunately, a polygon mesh does not capture high-level structures. In general, high-level abstractions are useful for managing data in applications. However, it remains a challenge to define useful high-level structures for arbitrary meshes, not to mention computing them automatically.

In searching for good high-level structures, psychological studies have provided some useful guidelines. They have shown that human shape perception is partly based on decomposition, and any complex object can be regarded as an arrangement of simple primitives, or components [BIED87]. Representation of an arbitrary object in terms of a few components has long been a tradition in interactive graphics, CAD, and virtual simulation. Hence, we aim to decompose a polygon mesh into components that accord to human shape perception, and more importantly, are beneficial to interactive graphics applications.

To decompose a polygon mesh where only the connectivity of polygons is known, we address three key issues: (1) What features can be used to delineate components reliably? (2) How should these features be computed? and (3) How should components be extracted automatically based on these features? The contribution of this paper is a framework for decomposing polygon meshes into components by treating them as surfaces of a volume. The results are components that are homogenous in geometric and topological features. This decomposition can benefit many applications, such as level of detail control, collision detection, ray tracing, mesh editing, shape interpolation and morphing. To demonstrate its efficiency and effectiveness, we have integrated our framework into several collision detection engines. The results of our findings have been encouraging.

The remaining paper is organized as follows. Section 2 reviews related previous work. Section 3 gives a definition of a component. Sections 4 to 6 describe details of the major steps in the proposed framework. Section 7 demonstrates one possible use of the framework in improving collision detection results. Finally, section 8 concludes the paper.

2 Related Previous Work

Decomposition of shape is studied extensively in the computer vision community. Approaches, such as generalized cylinders [BINF71], [NB77], geons [BIED87], superquadric [PENT86] and their extensions were used in 2D images and range data. Though these approaches also focus on acquiring components with homogeneous characteristics as in our approach, there are no trivial extensions to work on 3D polygon meshes. Decomposition of 3D (volume) digital shapes based on a hierarchical

decomposition method was discussed by [BBS99], and may be extended to work on polygon meshes. Their work used the intuitive notion of local thickness to guide the decomposition and addressed the issue of robustness due to the translation of a model. Another work on volumetric objects was presented recently by Gagvani and Silver [GS00]. Our approach focuses on a more formal treatment of decomposition applied to polygon meshes.

In [FS92], Falcidieno and Spagnuolo proposed an algorithm to classify polygon meshes into curvature regions. Chazelle *et al.* [CDST95] addressed the problem of dividing a polyhedron surfaces into a collection of convex pieces. Recently, Mangan and Whitaker extended the 2D watershed method to 3D polygon meshes [MW99]. Gregory *et al.* proposed decomposing the boundary of a polyhedron into the same number of morphing patches from the interactively defined feature nets [GSLM99]. Our approach differs from these, in that we consider volume as a part of the homogeneous characteristics in decomposition.

Tan *et al*. [TCL99] recently achieved good results in decomposing objects through the use of vertex-based simplification. The approach works well for the geometric or inorganic models such as helicopters. However, it is not robust in that the results are sensitive to user specified parameters. In particular, it is not suitable for organic models that have no clear boundaries among their parts or components, such as that between the limbs and torso of a human. In addition, a homogeneous part of the object may be undesirably partitioned into different components, as there is no good control provided by simplification models in identifying boundaries among components. Our paper presents an approach that solves the mentioned problems, in particular when applied to organic models.

3 Preliminaries

By decomposition, we refer to the process of breaking down an object to arrive at a collection of meaningful components. By meaningful, we mean that a component can be distinguished perceptually from the rest of the object. From the cow model of Figure 1, we can easily see that it consists of horns, ears, legs, head, body, tail etc., that make up the object. There are clearly some characteristics that distinguish one part from the others. From observation and experiment, we found that two important distinguishing characteristics are geometry and topology. The notion of component is thus formulated using these two measures.

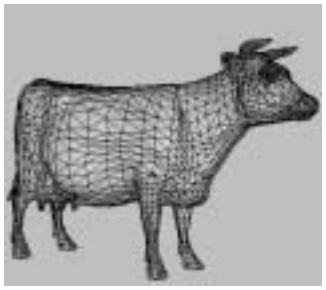


Figure 1: Polygon Mesh

Geometric and Topological Functions. Among the many ways of defining a 3D volume, one way is through space sweeping. Some examples of space sweeping are, (1) rotating a constant 2D region (e.g. $(x\text{-}5)^2+(y\text{-}5)^2< 2$) around an axis (e.g. y), or more generally, (2) sweeping a parameterized 2D region along a 3D space curve. The notion of a component in our framework is based on the latter. For each component of a 3D object, we can imagine it to be the result of some sweep of a varying planar region (*cross-section*) along some path.

An object (or a component) can be represented as a sweep of some parameterized function of cross-sections along some path as follows:

$$\bigcup_{t=t_{start}}^{t_{end}} G(t)$$

where $t \in [t_{start}, t_{end}]$ denotes a particular point on the path, and $G(t)$ is the cross-section at t. Next, we define two functions $F(t)$ and $T(t)$:

(1) The geometric function $F(t)$ is given by

$$F(t) = measure(\ G(t)\ ),$$

where *measure* is the measurement, such as the area or perimeter of cross-section $G(t)$.

(2) The topological function $T(t)$ is given by

$$T(t) = \begin{cases} 0, & \text{topology of } G(t-\varepsilon) \text{ is different from } G(t) \\ 1, & \text{otherwise} \end{cases}$$

where ε is an arbitrarily small positive number.

In order to obtain components that conform to the idea of a simple primitive, some restrictions are placed on the values of functions $F(t)$ and $T(t)$ by introducing the concept of a *critical point.*

Critical Points and Components. A critical point in $[t_{start}, t_{end}]$ is a boundary point between components that captures change in either geometry, topology or both.

Figure 2 shows some sample profiles of $F(t)$. Other possible profiles such as symmetric ones or a combination of the given examples are considered in a similar way. At t_c as indicated by dashed lines, each of the profiles is divided into two segments. Naturally, it is desirable to allocate the intervals $[t_1, t_c]$ and $[t_c, t_2]$ to two distinct components. By computing a few derivatives of the profiles, we note that the common characteristic of all profiles at t_c is that $F^{(n)}(\ t_c) = 0$, where $F^{(n)}$ is the first vanishing derivative of $F(t)$ for some n, and $F^{(n)}$ crosses zero at t_c.

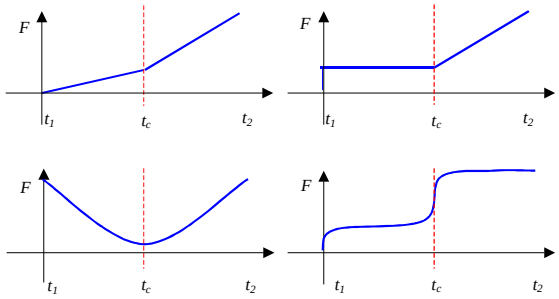


Figure 2: Sample Profiles of F(t)

In order to restrict components to simple shapes, there is an additional constraint that topology must be constant, i.e. $T(t)$ must be non-zero in each component. From these conditions, we define a critical point at t as:

$$(F^{(n)}(t) = 0 \text{ and } F^{(n)}(t-\varepsilon) \cdot F^{(n)}(t+\varepsilon) < 0) \text{ or } T(t) = 0$$

Thus we can define a *component* as follows:

$$C = \bigcup_{t=t_a}^{t_b} G(t), \text{ such that there is no critical point in } (t_a, t_b)$$

where $t_a, t_b \in [t_{start}, t_{end}]$.

In practice, we would like the interval (t_a, t_b) to be as large as possible. With this definition, the problem of decomposing an object into components is thus transformed to one that locates critical points.

Proposed Framework. We adapt the space-sweep technique commonly used in computational geometry [HMMN84] to sweep the given polygon mesh with a plane, called the *sweep plane*, along some *sweep path*. The sweep path is approximated through the *skeleton* of the mesh, which we define in the next section. During sweeping, the geometric and topological functions are computed. Whenever a consecutive pair of critical points $\{t_a, t_b\}$ are found, the part of the polygon mesh between t_a and t_b is extracted as a component.

The following sections address technical issues of this approach. Section 4 describes the derivation of skeleton from edge contraction based simplification. Section 5 describes how to augment this skeleton to form a tree for deriving a sweep path. Finally, Section 6 describes space sweeping for component extraction. For the purpose of discussion, the subsequent sections assume that triangular meshes are used instead of the more general polygon meshes. This does not affect the discussion in any way since it is straightforward to convert from one to the others [RR94]. We also assume that suitable preprocessing – like welding of very close vertices – is performed prior to applying the method.

4 Skeletonization

As described in section 3, a volume may be obtained through space sweeping of a cross-section function over some path. To solve the inverse problem, we must first determine the sweep path of a given object. This section justifies the use of skeletal edges from simplification as the basis for deriving the sweep path and then describes how they are computed.

Related Structures. A suitable sweep path should be one that maintains the constancy or gradual change of the sizes of the cross-sections. The notions of generalized cylinder [BINF71], [NB77], [NCS94] and medial axis [CKM99] describe very well the structural features of a 3D object. But these features are non-trivial and expensive to compute. In the case of the medial axis, complicated structures (involving quadric surfaces among others) are produced. In many cases they are not suitable to be used in sweeping. Similarly, other (generalized) Voronoi diagrams [BOIS84], [VO95] are not appropriate for our purposes. We seek some one-dimensional structure to guide our sweep path as discussed next.

Verroust and Lazarus [VL00] proposed a method of extracting skeletal curves from polygon meshes. Their method computes skeletal curves through sample points on the surface of the given mesh. The success of the method relies on the sampling rate and the locations of source points (that are origins of skeletal curves) among other user specifications. Though this may also be applicable to our framework, it is unnecessarily complex and requires user input. Our method, based on [HOPP96], is automated and simpler to implement.

Skeletal Edge and Skeleton. In edge contraction, edges of polygons in a mesh are ranked according to some cost function, which usually measures an error based on the effect of the removal of an edge. Unlike [HOPP96], we do not require an elaborate and expensive cost function, as our objective is not to preserve appearance. Instead we can make use of the Euclidean length of the edge as the cost. In each step, the edge with the minimal cost is selected to collapse into a vertex at the average location of its endpoints, and triangles incident to the edge are removed. Each edge (u, v) is associated with a list of triangles called the *associated triangle list*, $ATL(u, v)$ that collapsed to this edge or its endpoints u and v. During the process, whenever an edge (u, v) is not incident to any triangle, it is designated as a *skeletal edge* and vertices u and v maintain their positions until the end of the process. The process iterates until all triangles have been collapsed to edges or vertices. In other words, we are left with only skeletal edges, whose union is the *skeleton* of the mesh. Figure 3 is a skeleton for the cow model, and Appendix A illustrates a detailed 2D example of the process.

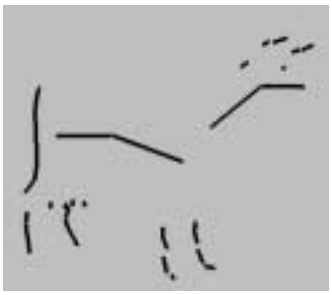


Figure 3: Skeleton

Besides avoiding the complexity of other methods, the proposed method is able to work with uniformly subdivided as well as non-uniformly subdivided surfaces. For convex simple shapes, the skeleton converges toward the medial axis. As the algorithm approximates simplification, smaller features are extracted first. For meshes with a bounded number of edges incident to each of its n vertices, the algorithm has an efficient runtime of $O(n \log n)$ where each iteration requires $O(\log n)$ time in searching for the shortest edge to collapse and $O(1)$ time in updating all the relevant data structures.

5 Sweep Order

Skeletal edges provide a good basis for sweeping, but before that can be done we need to organize the set of skeletal edges (that are

disjoint in general) into a linear order for sweeping. To do this, we define a *skeletal tree* from the skeletal edges by adding *virtual edges* to connect disjoint skeletal edges. Then we define a traversal order of the edges of the skeletal tree by grouping them into *branches*. The ordering of space sweeping and the sweeping paths are then determined by the ordering and position of the branches.

Virtual Edge and Skeletal Tree. During skeletonization, a triangle is either contracted to a skeletal edge or an endpoint on a skeletal edge. Consider triangles Δ_1 and Δ_2 that are incident to a same edge. Suppose Δ_1 is contracted to b and Δ_2 is contracted to d where (a, b) and (c, d) are two disjoint skeletal edges. To connect them, we add a *virtual edge* (b, d) to the collection of skeletal edges. For the 2D example in the appendix, $\Delta v_5 v_6 v_8$ and $\Delta v_5 v_2 v_6$ sharing (v_5, v_6) are contracted to v_{12} and v_9 respectively; we thus have the virtual edge (v_{12}, v_9).

On the other hand, suppose Δ_1 is contracted to b and Δ_2 is contracted to (c, d), where (a, b) and (c, d) are two disjoint skeletal edges. We have the issue as to whether (b, c) or (b, d) should be a virtual edge. This is resolved because the computation of virtual edges is in fact an integral part of the skeletonization process mentioned in the previous section. The skeletonization process creates a virtual edge whenever two disjoint parts are the result of an edge contraction (rather than after the whole process). That is (b, c) should be a virtual edge if b was a result of contracting some edge incident to c, and likewise for (b, d).

For a connected mesh, the result of adding virtual edges is a connected graph. Such a graph is called a *skeletal tree* if it has no cycles. Otherwise, cycles are removed by applying a standard minimal spanning tree algorithm [CLR90]. The cost between each pair of nodes u and v is defined as the inverse of the size of $ATL(u, v)$. Figure 4 shows a skeletal tree where the skeletal edges are drawn in bold strokes and virtual edges in thin strokes.

Figure 4: Skeletal Tree

Branch. From a leaf vertex of the skeletal tree, we can define a *branch* – starting with the edge incident to the leaf vertex – as the maximal chain of edges whose vertices (excluding the first and the last) are incident to exactly two edges in the skeletal tree. Each branch corresponds to one continuous part of the whole sweep path $[t_{start}, t_{end}]$, with the leaf vertex as the starting point, and the last vertex of the chain as the ending point. In general, there are, as many branches as the number of leaves.

Ordering of Branch. The union of the *ATL* over the edges of a branch, with first vertex u and last vertex v, corresponds to a part of the object. Its surface area can be approximated as:

$$A = \int_u^v f(t)\, dt$$

where $f(t)$ measures the perimeter of the cross-section at t. Assuming $f(t)$ varies linearly over the branch, we can simplify the computation to the following:

$$A = (\text{length of branch}) * \frac{f(u) + f(v)}{2}$$

Note that this approximation is used, as skeletonization results in fairly uniform branches.

Branches are then ordered for sweeping based on surface area of their *ATL*. Those branches having a smaller area are favored for sweeping first. This ordering allows small but significant components to be extracted first so that they are not absorbed into larger components. During sweeping, edges of the skeletal tree that have been swept are removed.

Note that the collection of triangles in an *ATL* seem to approximate a component. Likewise, the union of associated triangles of a branch also approximate a component. However, they are only able to capture coarse features and may not have a clear boundary between different parts. Hence space sweeping, clarified in the next section, is necessary to ensure that each component conforms to the definition in section 3.

6 Space Sweeping

In the course of sweeping, the geometric and topological functions are computed and analyzed. When a consecutive pair of critical points is found, the part of the polygon mesh that is swept is extracted as a component.

To begin a space sweep, we have a branch and its corresponding set of triangles, which are obtained from the union of the *ATL's* of edges from the branch. The sweep plane, given by $P(t) = P(v, n)$, is determined by a reference point v and a normal vector n as shown in Figure 5. For any given vertex u from the mesh, we can compute its direction from v. If $(v - u) \cdot n < 0$, we say that u is below the sweep plane. If it is equal to zero, u lies on the plane. Otherwise u is above the sweep plane.

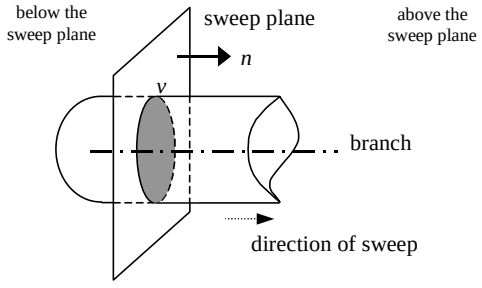


Figure 5: Sweep Plane

Sweep Plane Movement. The reference point v is either a vertex of the mesh or a point on an edge of the mesh. Specifically, v is on

the boundary of the cross-section $G(t)$. Initially, n is the normal vector along the first skeletal edge of a given branch B. v is then an arbitrary point along the intersection of the mesh and the sweep plane passing through the first endpoint of B. Since a polygon mesh is a discrete representation, we need only to approximate the sweep path by advancing in discrete steps. To advance $P(v, n)$ to a new v, one way is to move it to the next nearest vertex above the sweep plane. In this way, the total number of cross-sections will be proportional to the number of vertices of the mesh. For dense meshes, this is an expensive operation. For such cases, a more pragmatic approach is to advance in fixed steps. In order not to over- or underestimate the step size, we can compute the amount as a function of the edge length distribution.

Cross-section $G(t)$. An edge in the polygon mesh is an *intersecting edge* of $P(t)$ if one of its endpoints is either on or above $P(t)$, and the other endpoint is on or below $P(t)$. An *intersecting triangle* of $P(t)$ is a triangle incident to one or more intersecting edges of $P(t)$. The set of intersecting triangles of $P(t)$ is denoted as $\Delta = \{ \Delta_i \mid i = 1, 2, \ldots, k\}$. The intersection of $P(t)$ with Δ is a collection of line segments forming the boundary of $G(t)$. This boundary is either a single simple polygon (see t_1 of Figure 6) or a collection of simple polygons (see t_2 of Figure 6).

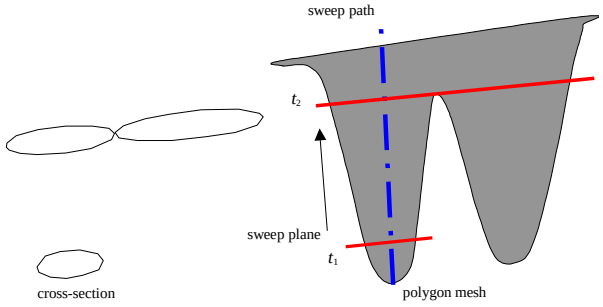


Figure 6: Cross-section

Sweep Plane Orientation. Each branch comprises a set of skeletal edges which provide a general direction for the sweep plane movement. However, the orientation of the skeletal edge is not necessarily correct, as it is dependent on the result of edge contraction. Figure 7 shows an example where the object is straight but the branch is crooked. To account for this error, the orientation of the sweep plane is not entirely based on the skeletal edges but computed adaptively.

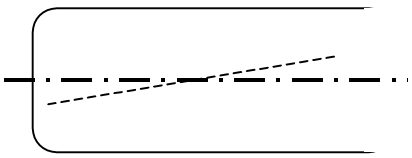


Figure 7: Skeletal Edges Orientation

Hence, when the sweep plane advances from $P(u, n)$ to $P(v, n')$, its orientation is subjected to change. To compute n', we first set it to n. It is then adjusted so that the sweep path will follow the natural orientation (n_s) of the volume, and the cross-sections can approximate those obtainable through sweeping along the medial axis. Figure 8 illustrates this idea where (b) is a better orientation than (a).

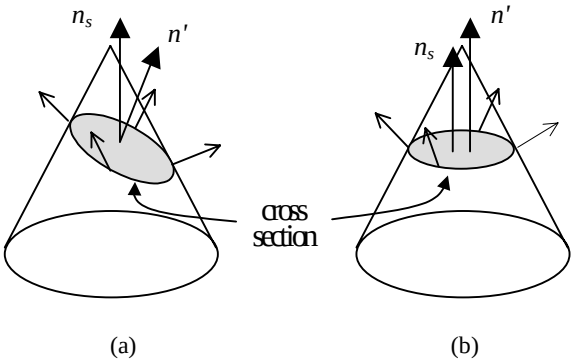


Figure 8: Sweep Plane Orientations

Specifically, we would like to have n' oriented based on the normal vectors of triangles Δ_i in Δ. The simple approach of summing the normals does not work as it gives a meaningless direction for a cylindrical object. We present a more robust formula for n_s as follows:

$$n_s = \sum_{i=1}^{k} N(\Delta_i) \times N(\Delta_{i+1})$$

where $N(\Delta_i)$ and $N(\Delta_{i+1})$ are normal vectors of consecutive triangles in Δ (with $\Delta_{k+1} = \Delta_1$), ordered in an anticlockwise manner about n'. Intuitively, the cross product of $N(\Delta_i)$ and $N(\Delta_{i+1})$ captures the essence of a local optimal direction for the sweep plane with respect to the shared edges of Δ_i and Δ_{i+1}. Thus, a sum over all such cross products points to an aggregated direction.

$F(t)$ and $T(t)$ Computation. With intersecting triangles of $P(t)$ in Δ, it is easy to compute the geometric function $F(t)$. It can be defined as either the perimeter or the area of the cross-section. To approxiamte the derivatives of $F(t)$ (up to third order in practice), we use previous values (up to five) of $F(t)$ to find delta differences in the computation. Also, standard filters, such as Gaussian or median [GW93], can be used to smooth out the values.

To compute $T(t)$, we first define H, a function of the cross-section $G(t)$, as the number of simple polygons in $G(t)$. Note that $G(t - \varepsilon)$ and $G(t)$ are *homotopic* if $G(t - \varepsilon)$ can be deformed continuously into $G(t)$ and vice versa. This property implies that their boundaries are also homotopic, that is, $H(\ G(t)\ ) = H(\ G(t - \varepsilon)\ )$. With that, we arrive at the following definition of $T(t)$:

$$T(t) = \begin{cases} 1, & H(G(t-\varepsilon)) = H(G(t)) \\ 0, & H(G(t-\varepsilon)) \neq H(G(t)) \text{ or } G(t) \text{ degenerates to a point} \end{cases}$$

Since we only need to compare between $G(t - \varepsilon)$ and $G(t)$, only two cross-sections need to be stored throughout the computation.

Component Extraction. From $T(t)$ and the approximate derivatives of $F(t)$, we can determine whether t is a critical point as defined in

Section 3. Figure 9 illustrates a simple example with a critical point at *b*. When a critical point is reached, triangles swept so far that are not part of other components are extracted as a new component. For intersecting triangles in Δ, we could either include them in the new component, or split them up along their intersection with the sweep plane, depending on the application.

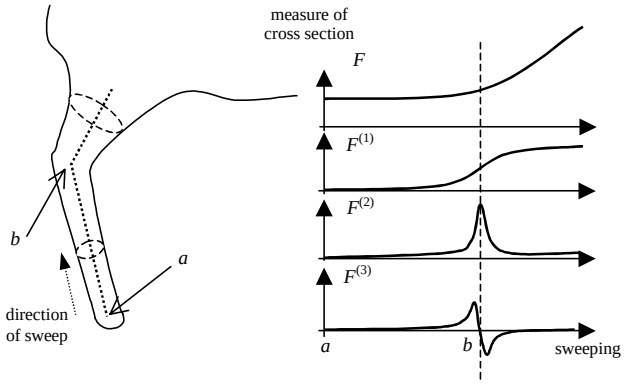


Figure 9: Profile of *F*(*t*)

7 Application to Collision Detection

As a way to show the usefulness of our decomposition technique, we have applied our framework to the problem of building hierarchical data structures for collision detection. The problem of collision detection is to determine whether moving objects (or rays for ray tracing application) are in collision with other objects in a simulated computer environment [LG98].

Collision Detection Systems. RAPID [GLM96] and QuickCD [KHMS98] are state-of-the-art collision detection systems available from the web. Both of these systems are based on constructing hierarchies of tight-fitting bounding volumes of oriented bounding boxes (OBB) and discrete oriented polytopes (*k*-DOPs) for polygon meshes or soup. In [TCL99], Tan *et al.* use outputs of simplification procedures to guide the construction of bounding volume hierarchies. In particular, their UCOLLIDE system builds the top few levels of the bounding volume hierarchy with simplification, and applies RAPID for the lower levels.

Hierarchy Building. Our application to collision detection adopts the same strategy as [TCL99]. In particular, we make use of the components derived from our procedure to build the top few levels of the bounding volume hierarchy, and then apply a native collision framework like RAPID or QuickCD for the lower levels.

The construction of the top few levels of the hierarchy is performed as follows. We start by adding components as the leaves of the bounding volume hierarchy ***B***. These nodes also form the initial set *N*. Let n_i and n_j be members of *N* such that they form the minimal bounding volume (either OBB or *k*-DOP) among all possible pairs in *N*. Create a new node n_k in ***B*** as the parent of n_i and n_j, and replace n_i and n_j in *N* by n_k. Repeat this until *N* has only one node, which is actually the root of ***B***.

Experimental Setup. Our experiment was conducted with an Intel Pentium II 450MHz PC. To compare RAPID and UCOLLIDE with our additional framework, we used the same experimental setup for each test model and environment. The number of simulation steps was set to 5000, the environment was populated with models where the number of triangles was over 200000, and the size of the environment was set to simulate sparse, normal or dense situations. In a sparse environment, the sum of the topmost bounding volumes of all the instances is about 25% of the volume of the environment. For normal and dense environments it is around 50% and 100% respectively.

To compare with QuickCD, we used a similar setup. The environment was populated with a number of static copies of the input model so that the number of triangles in the environment was over 70000. In addition, there was a moving copy of the model with random speed and orientation. The number of simulation steps was set at 5000. Note that fewer copies of the model were used in comparison with RAPID and UCOLLIDE. This is due to the way environments are represented in QuickCD. Copies of the static model are represented in multiple times in the environment, as compared to RAPID or UCOLLIDE, where only position, orientation and direction need to be stored for each copy. In order to prevent disk swapping due to the limit on main memory, only a small number of copies were instantiated.

Results. Some of our decomposition results are shown in Figure 13 (color plate). Each row consists of four images, namely, the derived skeletal tree, two views of the extracted components and the cross-section. Each component is shown in a different color. Note that in our implementation of the space sweeping, we advance sweep plane to the nearest vertex among those vertices that are in the intersecting triangle set Δ and above the current sweep plane *P*(*t*). This is a simplified version of the space sweeping but suffices for our experiments. Also, we use the simple perimeter measure for the geometric function *F*(*t*). The time taken to compute components is only a few seconds for our largest model (the chicken wing model).

The results of the comparisons with RAPID and UCOLLIDE are shown in Figure 10 to 11. In terms of average time taken for each step of simulation, our method is 13-20% (average 16%) and 5-11% (average 8%) faster than RAPID and UCOLLIDE respectively in a sparse environment. In a normal environment, it is faster by 8-22% (average 14%) and 2-15% (average 6.9%), respectively; in a dense environment, it is faster by 8-23% (average 14%) and 1-11% (average 6%), respectively. For comparison with QuickCD (Figure 12), our improvement is 6-26% (average 12%) faster in terms of average time taken for each step of the simulation.

On the whole, results show that better bounding volume hierarchies are constructed with our approach. In terms of total volume of bounding volume hierarchies for each model, our improvement as compared to RAPID is 2-21% (average 11%), to UCOLLIDE 2-11% (average 7%), and to QuickCD 4-12% (average 8%). In addition, they also indicate that our space-sweeping decomposition method performs better than that of the decompositions used in [TCL99] for organic objects.

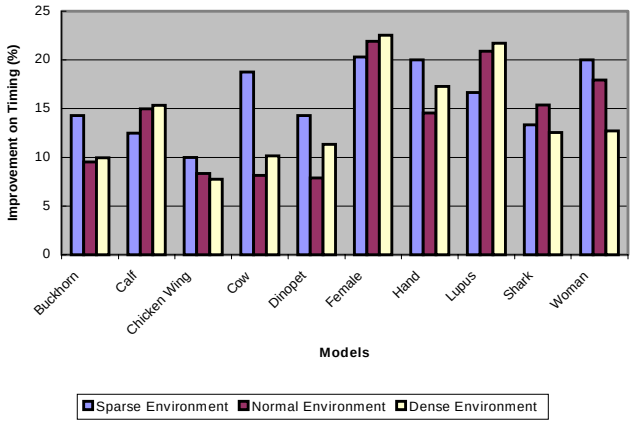


Figure 10: Comparison with RAPID

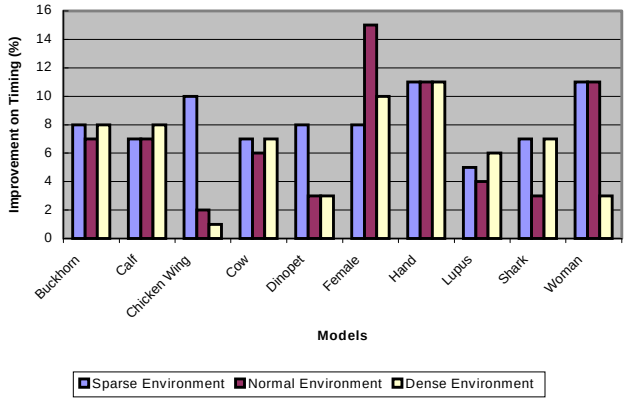


Figure 11: Comparison with UCOLLIDE

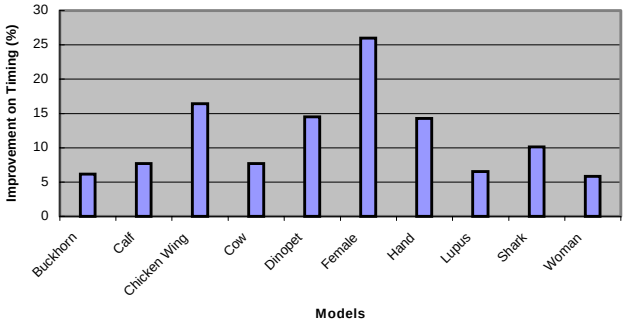


Figure 12: Comparison with QuickCD

8 Concluding Remarks

In this paper, we present a formal notion of components and a framework to decompose objects, represented as polygon meshes, into such components using space sweeping. The framework consists of steps to compute the sweep path, prioritize the sweeping order of segments of the sweep path, calculate geometric and topological features along the sweep path, and extract components of homogeneous features. On the whole, the decomposition is performed efficiently with no user intervention.

Furthermore, we have implemented the framework with application to collision detection, as a demonstration of its effectiveness in improving collision detection timing for state-of-the-art collision detection systems such as RAPID and QuickCD. Possible future work includes applications of the framework to 3D object morphing among others.

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments and suggestions on improving the paper.

References

[BBS99] Borgefors G., di Baja G. S., and Svensson S., "*Decomposition Digital 3D Shape Using a Multiresolution Structure"*, in Bertrand G., Couprie L., and Perroton L., eds., "*DGCI'99, LNCS 1568"*, Springer Verlag, pp. 19-30.

[BIED87] Biederman I., "*Recognition-by Components: A Theory of Human Image Understanding*", "*Psychological Review*", vol. 94(2), pp.115-147, 1987.

[BINF71] Binford T. O., "*Visual Perception by Computer*", "*IEEE conference on Systems and Control*", December 1971.

[BOIS84] Boissonnat J.D., *"Geometric structures for three-dimensional shape reconstruction"*, *"ACM Transactions on Graphics"*, vol. 3, pp. 266-286, 1984.

[CDST95] Chazelle B., Dobkin, D. P., Shouraboura N, and Tal A, "*Strategies for Polyhedral Surface Decomposition an Experimental Study*", "*Proceedings of the eleventh annual symposium on Computational geometry"*, pp. 297-305, 1995.

[CKM99] Culver T., Keyser J., and Manocha D., "*Accurate Computation of the Medial Axis of a Polyhedron*", "*Fifth ACM Symposium on Solid Modeling*", pp. 179-190, 1999.

[CLR90] Cormen T. H., Leiserson C. E., and Rivest R. L., "*Introduction to Algorithms*", *The MIT Press*, pp. 498-513, 1990.

[FS92] Falcidieno B. and Spagnuolo M., "*Polyhedral Surface Decomposition Based on Curvature Analysis*", in Kunii T. L. and Shinagawa Y. eds., "*Modern Geometric Computing for Visualization*", Springer Verlag, pp. 57-72, 1992.

[GLM96] Gottschalk S., Lin M. and Manocha D., "*OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*", "*Proceeding of ACM SIGGRAPH*", pp. 171-180, 1996.

[GS00] Gagvani N. and Silver D., *"Shape-based Volumetric Collision Detection", "IEEE Volume Visualization and Graphics Symposium"*, 2000.

[GSLM99] Gregory A., State A., Lin M., Manocha D., Livingston M. A., *"Interactive Surface Decomposition for Polyhedral Morphing", "The Visual Computer"*, vol. 15(9), pp. 453-470, 1999

[GW93] Gonzalez R. C. and Woods R. E., *"Digital Image Processing"*, Addison-Wesley, pp. 189-195, 1993

[HMMN84] Hertel S., Mehlhorn K., Mantyla M., and Nievergelt J., "*Space Sweep Solves Intersection of Two Convex Polyhedra*", ACTA INFORMATICA 21, pp. 501-519, 1984.

[HOPP96] Hoppe H., "*Progressive Mesh*", "*Proceeding of ACM SIGGRAPH*", pp. 99-108, 1996.

[KHMS98] Klosowski J. T., Held M., Mitchell J. S. B., Sowizral H., and Zikan K., "*Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs*", "*IEEE Trans. on Visualization and Computer Graphics"*, vol. 4(1), pp. 21-36, 1998.

[LG98] Lin M. and Gottschalk S., "*Collision Detection between Geometric Models: A Survey*", "*Proceedings of IMA Conference on Mathematics of Surfaces*", 1998.

[MW99] Mangan A. P., Whitaker R. T., "*Partitioning 3D Surface Meshes Using Watershed Segmentation*", "*IEEE Transactions on Visualization and Computer Graphics*", vol. 5, pp. 308-321, 1999.

[NB77] Nevatia R., Binford T. T., "*Description and Recognition of Curved Objects*", "*Artificial Intelligence*", vol. 8, pp.77-98, 1977.

[NCS94] Nazarian B., Chédot C., and Sequeira J., *"Interactivity and Delaunay Triangulation for the Reconstruction of Tubular Anatomical Structures", "IEEE conference on Engineering in Medicine and Biology Society*", pp. 706-707, 1994.

[PENT86] Pentland A. P., "*Perceptual Organization and the Representation of Natural Form*", "*Artificial Intelligence*", vol. 28, pp. 293-331, 1986.

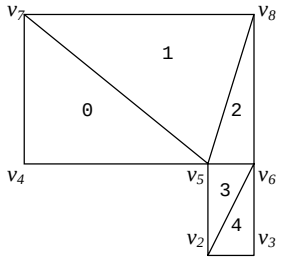
[RR94] Ronfard R., and Rossignac J., "*Triangulating Multiply-Connected Polygons: A Simple, Yet Efficient Algorithm*", "*Computer Graphics Forum*", vol. 13(3), pp. 281-292, 1994.

[TCL99] Tan T. S., Chong K. F., and Low K. L., "*Computing Bounding Volume Hierarchy using Simplified Models*", "*Proceedings of ACM Symposium on Interactive 3D Graphics*", pp. 63-69, 1999.

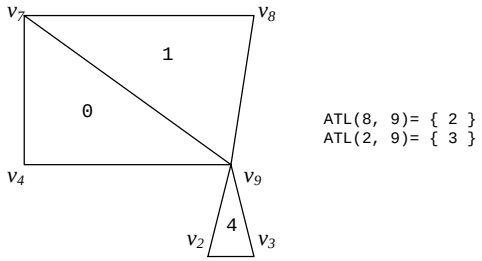
[VL00] Verroust A. and Lazarus F., *"Extracting Skeletal Curves from 3D Scattered Data", "The Visual Computer"*. vol. 16, pp. 15-25, 2000.

[VO95] Vleugels J. and Overmars M., *"Approximating Generalized Voronoi Diagrams in Any Dimension", "Technique Report UU-CS-1995-14"*, Department of Computer Science, Utrecht University, 1995.

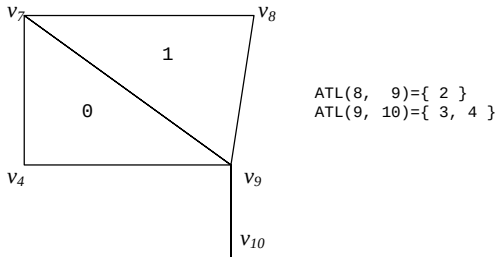
Appendix A: A 2D example of the skeletonization process



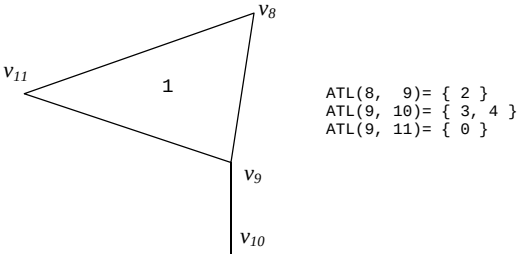
(a) Original mesh



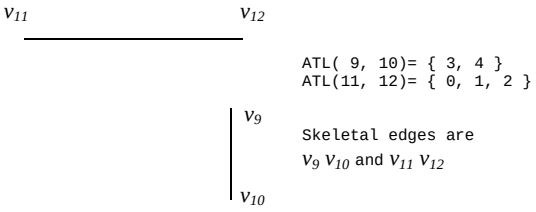
(b) After contracting edge (v_5, v_6) to vertex v_9



(c) After contracting edge (v_2, v_3) to vertex v_{10}



(d) After contracting edge (v_4, v_7) to vertex v_{11}



(e) After contracting edge (v_8, v_9) to vertex v_{12}

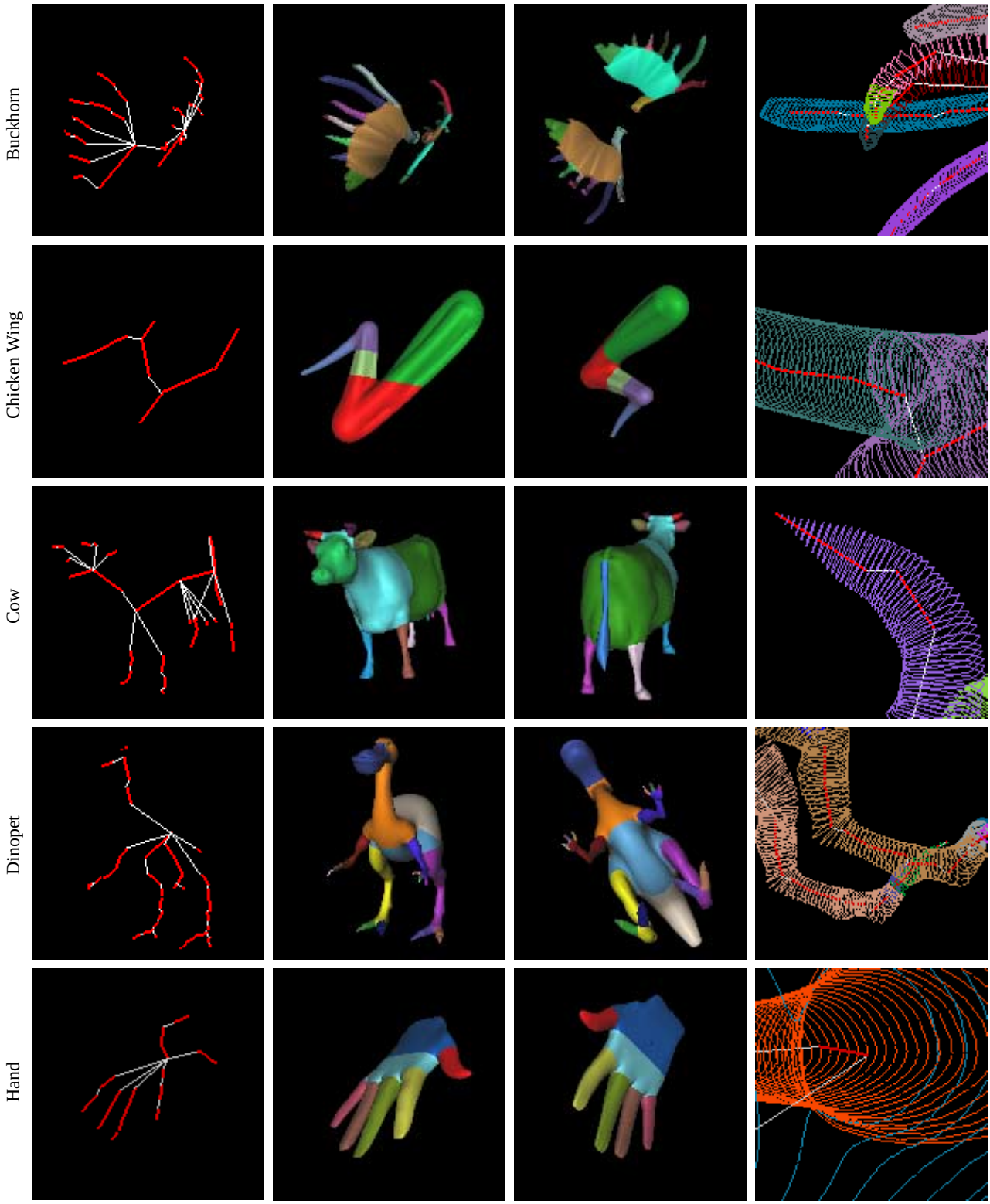


Figure 13: Decomposition Results

. Leftmost column – Skeletal tree of model (the red lines are skeletal edges, whereas white lines are virtual edges)
. Second and third columns – two different views of components, represented in different colors
. Rightmost column – view on some cross-sections of space sweeping