Decomposing Polygon Meshes by Means of Critical Points

Yinan Zhou and Zhiyong Huang

Department of Computer Science, School of Computing
National University of Singapore, Singapore 117543
{zhouyina, huangzy}@comp.nus.edu.sg

Abstract

Polygon mesh is among the most common data structures used for representing objects in computer graphics. Unfortunately, a polygon mesh does not capture high-level structures, unlike a hierarchical model. In general, high-level abstractions are useful for managing data in applications. In this paper, we present a method for decomposing an object represented in polygon meshes into components by means of critical points. The method consists of steps to define the root vertex of the object, define a function on the polygon meshes, compute the geodesic tree, compute critical points, decide the decomposition order, and extract components using backwards flooding. We have implemented the method. The preliminary results show that it works effectively and efficiently. The decomposition results can be useful for applications such as 3D model retrieval and morphing.

1. Introduction

Polygon mesh is the most widely available form of three-dimensional models. Geometrically, polygon mesh is a set of faces that share edges, or a set of vertices that are connected to neighbouring vertices by edges. The information it stores is the vertices geometric positions and their connectivity. The meshes come mostly from digitization of models or more rarely from real objects. They have many advantages. They are simple and fast for rendering. They can provide good approximation of real world objects. They have, however, no explicit higher-level structure; they exist simply as a collection of connected polygons – *a bucket of polygons*.

One way to impose a higher-level structure on such a mesh is to partition it into a set of connected components. Psychological studies also show that human shape perception is partly based on decomposition, and any complex object can be regarded as an arrangement of simple primitives, or components [2]. Presumably, the components should represent something about the underlying structure of the object itself — the *semantic* meaning of the object's parts and subparts. However, when no structure, other than the local connectedness of the polygons, is specified, any algorithm seeking to partition such meshes must infer some structure from the local shape (e.g., geometry) of the mesh itself.

In this paper, we propose an algorithm to decompose an object into connected components, which accord human shape perception, and more importantly, are beneficial to interactive graphics applications such as mesh editing, shape interpolation, establishing corresponding points for morphing and animation.

To decompose a polygon mesh using only the connectivity information of vertices, we address three key issues:

1. What features can be used to delineate components reliably?
2. How to detect these features? and
3. How to do the decomposition based on the features detected?

Our method consists of steps to define the root vertex of the object, define a function on the polygon meshes, compute the geodesic tree, compute critical points, decide the decomposition order, and extract components using backwards flooding.

The remainder of this paper is organized as follows. Section 2 reviews related previous works and the problems that arise in these works. Section 3 discusses in detailed of our work. First, a notion of component is proposed and followed by an overview of our decomposition method (Subsections 3.1 and 3.2). Then, we describe critical points detection (Subsections 3.3 to 3.7) and the decomposition method (Subsections 3.8 and 3.9). In Section 4, we present the decomposition results. We show the results of the geodesic tree, critical points and components, in automatically and interactively selected root, respectively. It is clear that the interactively selected root is more accurate. Thus, it gives better decomposition results. We also show the running time on a Pentium 4, 1.6 GHz, 256MB memory computer. We conclude the paper in Section 5 with a brief discussion of one future work on morphing.

2. Related Work

In this section, we will review the existing work on decomposing a 3D object. They can be roughly divided as methods based on computer vision, volume data, and polygon mesh representations.

2.1 Computer Vision-Based Decomposition

Decomposition of shape is studied extensively in the computer vision community. Approaches, such as decompose 2D digital shapes are proposed in the literature. In [2], shapes are decomposed into simple

components, "geometrical ions", geons. The geons consist, in that article, of generalized cones. The shape is segmented and the obtained parts approximated with the best fitting geon. This can be seen as analogous to the usage of phonemes in the lexical access during speech perception. For example, only 44 phonemes are needed to code all the words in English. For 2D images, 36 geons are needed.

Nevatia and Binford [13] used hierarchies of cylinder-like modeling primitives to describe natural forms. Pentland [14] extended the modeling primitives from cylinder to a family including cubes, cylinders, spheres, diamonds and pyramidal shapes as well as the round-edged shapes intermediate between these standard shapes. He proposed the idea of super-quadric, a superset of the modeling primitives currently in common use.

These works and their extensions were used in 2D images and range data. But it is trivial to be applied to 3D polygon meshes.

2.2 Volume Data Decomposition

Decomposition of 3D volume digital shapes based on a hierarchical decomposition method was discussed in [16]. The internal cores of the parts are detected on the distance transform of the object and the corresponding parts obtained by a region growing process from them. As any distance transform can be used, the decomposition is stable under rotation. The parts can be seen as nearly convex and elongated parts. The elongated parts are either necks or protrusions. A man-like object was decomposed into two nearly convex parts, the head and the body; one neck, the neck; and four protrusions, the two arms and the two legs.

2.3 Polygon Mesh Decomposition

Falcidieno and Spagnuolo [5] proposed an algorithm to classify polygon surface into curvature regions. Chazelle et al. [3] addressed the problem of dividing polyhedra into collection of convex pieces. The individual faces are each convex, but if one considers the optimal partitioning (e.g., smallest number of pieces), the problem is NP-complete. Mangan and Whitaker [12] extended the 2D image segmentation technique, morphological watersheds, to segment 3D surfaces into patches, where each patch has a relatively consistent curvature. These methods need to compute third- and fourth-order derivatives, which are very sensitive to noise, and work well only on relatively high-resolution data.

Gregory et al. [6] proposed decomposing the boundary of a polyhedron into the same number of morphing patches from the interactively defined feature nets. This approach needs user interaction and is complicated to implement.

Tan et al. [18] achieved good results in decomposing objects through the use of vertex-based simplification. Their 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.

Li et al. [11] proposed a framework for decomposing polygon meshes into components by treating them as surfaces of a volume. They adapt the idea of edge contraction to build skeleton of the object, then use space sweeping to decompose it. The skeleton generated by edge contraction is unconnected and virtual links need to be added. One of the important steps is the construction of skeleton. The methods to extract 1D skeleton from mesh object have been studied in [1, 12, 19]. The cost to build skeleton is high. The recent major progress includes [7, 15].

Our approach is mostly similar to Li et al. [11] on that we both use critical points information as the stop condition of decomposition. However, they differ in the following aspects: in the method of Li et al., critical points are detected in the process of sweeping along the skeleton. The sweeping plane's orientation is adjusted during the sweeping; in our method, the critical points are detected using the mesh surface information, the geodesic distance, before the decomposition process.

3. Our Work

In this section, we will start to present our work in detail. First, we will define a notion of component followed by an overview of our decomposition method (Subsections 3.1 and 3.2). Then, we will describe critical points detection (Subsections 3.3 to 3.7). Finally, we will describe the decomposition method (Subsections 3.8 to 3.9)

3.1 Definition of Component

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. Li et al. [11] points out the two important distinguishing characteristics are geometry and topology. In our work, we capture the information using critical points of a geodesic tree.

Notion of Component: A component C is a subset of the object O, such that it is bounded by borderlines consisting of *critical* (*local extremum* or *saddle) points*.

i.e. $C = \bigcup_{l_s}^{l_e} v$ for $v \in Obj$, where l_s and l_e are starting and ending borderlines formed by c*ritical points (local extremum or saddle vertices)*. A function μ is defined on vertices of the object to derive the critical points. In our work, we define the function μ in this way: (1) we automatically compute or interactively pick a vertex as the *root vertex* $\boldsymbol{r}$, $\mu(\boldsymbol{r})=0$; (2) starting from the root $\boldsymbol{r}$, compute the geodesic tree which covers all vertices of the object; (3) for any vertex $\boldsymbol{v}$, its function value $\mu(\boldsymbol{v})$ is defined as the geodesic distance of shortest path to root vertex $\boldsymbol{r}$.

With this notion, the problem of decomposing an object into components is thus transformed to one that locates critical points and uses them to form the borderlines.

3.2 Overview of Our Approach

The input of our algorithm is one connected 3D object represented in triangle meshes and the outputs are several connected components represented in triangle meshes. The overview of our method is listed as follows:

Given a 3D object represented in triangle meshes (Figure 1 (a)),

1. Preprocess input data to insure even distribution of vertices and reasonable dense sampling rate (Figure 1 (b)).
2. Automatically or interactively select a root vertex for the object (Figure 1 (c)).
3. Define a function μ to capture geometric information of the object. We use geometric distance in this work.
4. Start from the root vertex defined in step 2, construct a distance map and an oriented graph call *geodesic tree* (Figure 1 (d)) connect all vertices in the object. The geodesic tree is obtained by computing shortest paths between each vertex and the root vertex. The distance map is the length of these shortest paths.
5. Compute the *vertex sign change* for each vertex in the mesh using geodesic distance between each vertex $\boldsymbol{v}$ to the root and the geodesic distance from $\boldsymbol{v}$'s neighbours to root.
6. By examine the *vertex sign change*, detect critical points (*local extremum* and *saddle points*) which corresponding to the extreme end of components and joints where topology change occurs (Figure 1 (e)).
7. Define the decomposition order by ordering the critical vertices using geodesic distance (Figure 1 (f)).
8. The decomposition is finally obtained by using a *backwards flooding* technique. Start from each local extremum in the order of step 7, construct a connected shortest path graph on the object surface until a saddle vertex is met. For all the vertices in the graph, mark them to be the vertices of one component (Figure 1 (g)).

Table 1. Overview of our method in steps

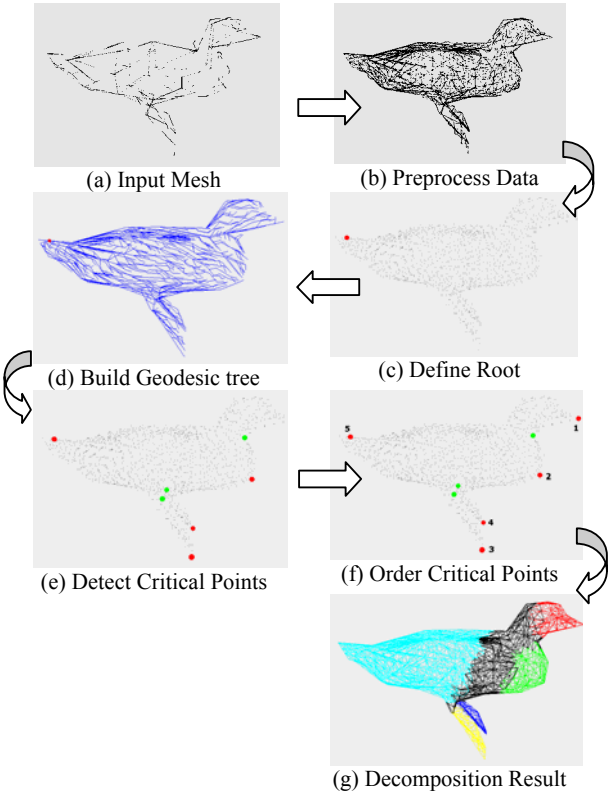


Figure 1. Illustration of decomposition in steps

3.3 The Root Vertex of an Object

The root vertex of the geodesic tree can be any vertex on the object. However, its choice will influence the decomposition results. Intuitively, using an extremum vertex located at the tip of a longest branch or the polar extremity as the root vertex will better describe an object's structure. Thus, it gives out a better decomposition result. In our work, we select the root vertex both automatically and interactively (illustrated in Figure 2 (a) and (b)). In the former case, we apply a heuristic based on an algorithm to find a diameter of a tree. A diameter of a tree is a pair of vertices separated by the largest possible distance in the tree. Intuitively, the two vertices of a diameter are located at the tip of a longest branch of the tree. Detailed description of the algorithm is as following:

Procedure to Determines a Tree Diameter

Step 1: Pick a vertex x at random from the pool of mesh vertices.
Step 2: Build shortest-path tree root at x and find y such that $dis_x(y)$ is maximal.
Step 3: Build shortest-path tree root at y and find z such that $dis_y(z)$ is maximal.
Step 4: Output (y,z) as a diameter. z will be a suitable root vertex.

Table 2. Algorithm to determine a tree diameter

The comparison of the final decomposition results using these two ways of root allocation can be found in Section 4. The results are shown to be equally satisfactory.

(a) (b)

Figure 2. Automatically (a) and interactively (b) selected root vertex: (a) usually located at the tip of a longest branch and (b) at polar extremity

3.4 A Function on Vertices

The critical points we use to decompose the object are generated by plugging each vertex v into a function μ and examine the value μ (v). It is important that the function μ is carefully defined. Hilaga et al. [8] and Takahashi et al. [17] used the height function as μ, so that any vertex is associated its own z-coordinate in a suitable coordinate system. The height function turns out to be a successful function μ in their application because for terrain data, the critical vertices correspond to peaks, passes or pits of the terrain. Height function is not suitable for our approach since there is relatively hard to set up a base plane for arbitrary object. Another approach uses curvature as the function μ. It may be applicable for some objects; but it is still not suitable for our purposes. Because a stable calculation of curvature is difficult on a noisy surface, and small undulations may result in a large change of curvature, causing error in the detection of saddle vertices.

Through experiment we find geodesic distance, the distance from point to point on a surface, can be a suitable function μ for genus-0 polygon mesh.

Methods for computing an accurate geodesic distance have been well studied [4, 9], however, when computing the accurate geodesic distance using these methods, the computational cost is high. Considering the trade off between computational cost and accuracy, we employ a relatively simple method in which geodesic distance is approximated by Dijkstra's algorithm based on edge length (described in Subsection 3.5).

3.5 Computing the Geodesic Distance

The construction of Shortest Path Tree is based on Dijkstra's shortest path algorithm. As for the classical Dijkstra's algorithm, we maintain a priority queue Q with respect to the shortest distance from each vertex v to root ($dis(v)$) and use a breadth-first search. This priority queue contains the vertices adjacent to the already visited vertices, whose distances have not yet been decided.

Dijkstra's Algorithm for Computing Geodesic Distance

Step 1: Initialize $dis(v) = \infty$ for all vertices.
Step 2: Select a root vertex r, set $dis(r) = 0$, and insert r to Q.
Step 3: Dequeue the vertex v with smallest $dis(v)$ from Q.
Step 4: For each vertex v_i adjacent to v, if $dis(v) + length(v, v_i) < dis(v_i)$, update $dis(v_i) = dis(v) + length(v, v_i)$ and insert (or update) v_i to Q.
Step 5: Repeat Step 3 and 4 until Q is empty.

Table 3. Computing the geodesic distance

The input of the algorithm is a 3D triangulated mesh (set of vertices, the connectivity information of the vertices and a root vertex). The output will be a tree, the *geodesic tree*, storing the geodesic distance from each vertex to the root vertex. One example of the geodesic tree is shown in Figure 3.

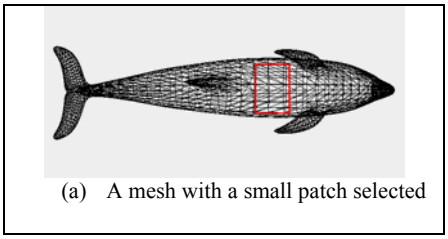
Figure 3. Example of a geodesic tree

3.6 Vertex Sign Change

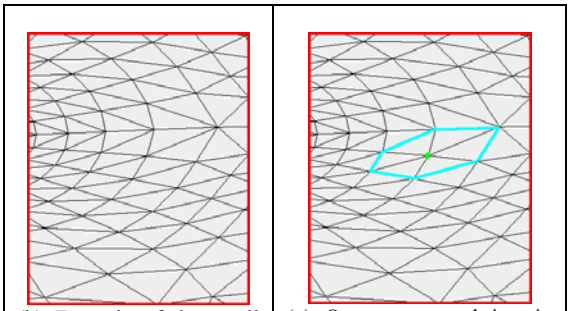
Zooming into the mesh object, we can find each vertex has its neighbours form a closed circle around it (Figures 4).

Knowing the adjacency between faces of the polyhedron, for each vertex, we can figure out its neighbours and organize these neighbours in a circle. Let $v_1, v_2, \ldots, v_k$ be the k neighbours of v enumerated around v and $dis(v_1)$, $dis(v_2)$, ..., $dis(v_k)$, $dis(v)$ be the distance from root vertex to vertex $v_1, v_2, \ldots, v_k$, v. We consider the number of sign changes, sgc(v), in the sequence ($dis(v_1)$-$dis(v)$, $dis(v_2)$-$dis(v)$, ..., $dis(v_k)$-$dis(v)$).

If $dis(\boldsymbol{v}_i)=dis(\boldsymbol{v})$, we skip this *ith* neighbour vertex by assume there is no sign change.



(a) A mesh with a small patch selected



(b) Zoom-in of the small patch

(c) One vertex and its six neighbouring vertices

Figure 4. The neighbours of a vertex

Figure 5 shows two vertices $\boldsymbol{v}$ with sgc($\boldsymbol{v}$) both equal to 4.

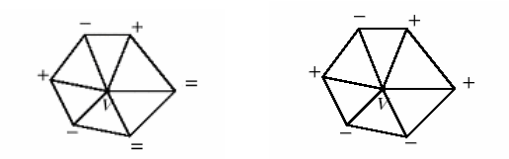
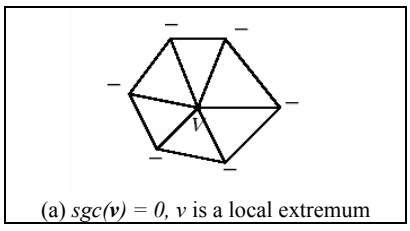


Figure 5. Sign changes around vertex *v*

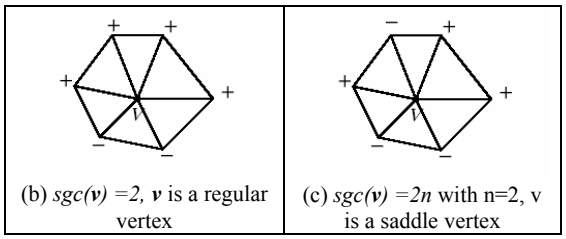
A '+' ('-') indicates a $\boldsymbol{v}$ neighbour, $\boldsymbol{vk}$, such that $dis(\boldsymbol{vk})$ is greater (smaller) than $dis(\boldsymbol{v})$. $sgc(\boldsymbol{v})$ counts the number of '+' and '-' sequences. The *sgc* of $\boldsymbol{v}$ tells how *dis* is changing around $\boldsymbol{v}$. It actually records part of the topology of the whole object.

3.7 Critical Vertices and Topology Change

There are 3 possible value for $sgc(\boldsymbol{v})$, namely, $sgc(\boldsymbol{v})=0$, $sgc(\boldsymbol{v})=2$, and $sgc(\boldsymbol{v})=2n$, $n=2,3\ldots$ which corresponding to *3* types of vertices (Figure 6). We define a vertex $\boldsymbol{v}$ with $sgc(\boldsymbol{v}) = 2$ as *regular*, otherwise *critical.*



(a) $sgc(\boldsymbol{v}) = 0$, v is a local extremum



(b) $sgc(\boldsymbol{v}) = 2$, $\boldsymbol{v}$ is a regular vertex

(c) $sgc(\boldsymbol{v}) = 2n$ with n=2, v is a saddle vertex

Figure 6. Counter of Sign Changes and Classification of Vertices

Local Extremum: If all the neighbours of $\boldsymbol{v}$ are nearer (further) to the root than $\boldsymbol{v}$, this vertex $\boldsymbol{v}$ is a local extremum.

Regular Vertex: There exists a clear cutting line between the sign of $\boldsymbol{v}$'s neighbours'. Half of $\boldsymbol{v}$'s neighbours are nearer to root vertex than $\boldsymbol{v}$, and the rest half are further away than $\boldsymbol{v}$. We define this vertex $\boldsymbol{v}$ as a regular vertex. Most of the vertices in a mesh object are regular vertices.

Saddle Vertex: A critical vertex, which is not an extremum, is defined to be a saddle vertex. For a saddle vertex $\boldsymbol{v}$, $\boldsymbol{v}$'s neighbours' signs change irregularly. Saddle vertices are at the joints of object, where topology change occurs.

One example is shown in Figure 7 of the local extremum, regular and saddle vertices.

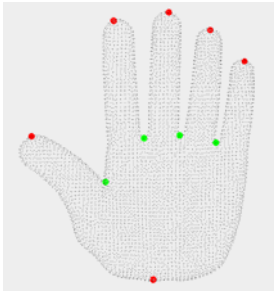


Figure 7. Critical vertices (red—local extrema, green—saddle vertices, gray—regular vertices)

3.8 Process the Mesh Data before Decomposition

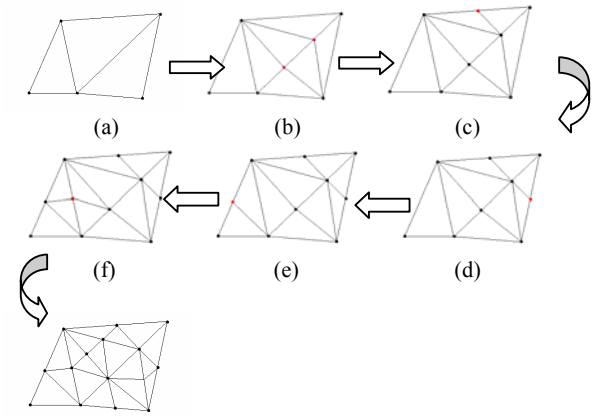
The decomposition will be accurate only when the sampling rate of the vertices is fine enough (especially around the critical vertices) to represent the distribution of the distance function well (detail explanation can be found in Subsection 3.9). It is therefore sometimes necessary to resample the vertices until all edge lengths are less than a threshold σ.

We design an algorithm to do the subdivision of mesh as shown in Figures 8. Figure9 shows a mesh object before and after the subdivision process. Given an object represented in triangle meshes, we build up a priority queue EQ according to the length of the edges of each triangle mesh.

Subdivision of a Mesh:

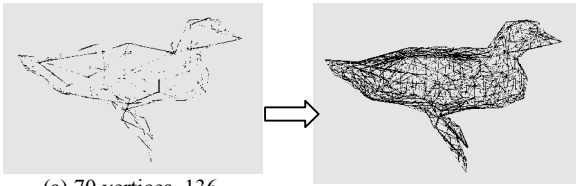
Step1: For each face of a mesh, test its three edges. If length of the edge > threshold, insert this edge into the EQ.
Step2: Dequeue the first element of EQ which has the longest edge ***e*** with length κ. Insert new vertices to divide this edge into κ/ threshold sub-edges.
Step3: Find out the two adjacent triangles of ***e***. Split them into 2κ/ threshold smaller triangles.
Step4: Test all the edges of the newly constructed triangles, if the edge length is greater than threshold, insert it into EQ.
Step 5: Repeat Step 2 to 4 until EQ is empty.

Table 4. The subdivision in steps



(g) The final result

Figure 8. Resampling and subdivision of a mesh



(a) 70 vertices, 136 triangles

(b) 916 vertices, 1828 triangles

Figure 9. Polygon mesh before and after preprocessing

3.9 Decomposition and Backwards Flooding

After the previous steps, the results we get are the critical points; there is no skeleton information. To derive the borderlines of components, we proposed a backwards flooding technique. The order of flooding is determined by the geodesic distance of a descending order. The process is listed in Table 5.

Note that, for automatically allocated root, since the root is the end of the diameter tree, it is a candidate local maximum and we include it in the starting points of backwards flooding. However, the interactively selected root vertex, a local minimum, will not be used as a starting point of backwards flooding.

Backwards Flooding

Step 1: Initialize C(***v***) = -1 about all vertices. Set *componentNo* = 0.
Step 2: Sort the local maxima to be in descending order using the geodesic distance from root.
Step 3: Select local maximum ***m*** which is furthest from root with C(***m***) = -1.
Step 4: Build a shortest path graph with root at ***m***. The construction stops when meets a saddle vertex or a vertex ***q*** with C(***q***) ≠ -1.
Step 5: Mark all the vertices in this graph to be the vertices of component *componentNo*. Increase *componentNo* by 1.
Step 6: Repeat steps 3 to 5 until all local maxima have been visited.

Table 5. The algorithm of backwards flooding

The stopping condition we used for backwards flooding is to meet a saddle point, and then mark the vertices in the flooding path to be the vertices of one component. The vertices we are referring to are the original vertices in the mesh, so the resolution of flooding decomposition is dependent on the resolution of mesh. Thus we may increase the resolution. Preprocess will not affect the critical points detection. For the efficiency reason, we may delay preprocess to be carried out after detected the critical points. Another more efficient way may process only the faces nearer to the saddle vertices.

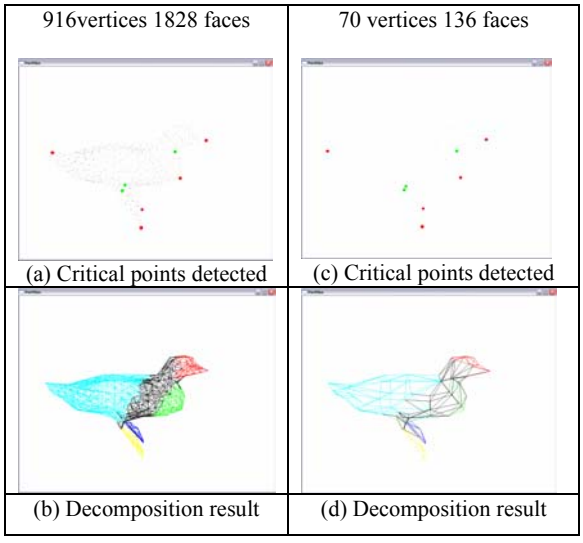
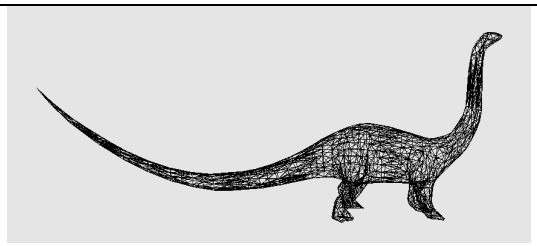


Figure 10. Decomposition results of different resolution meshes. With a higher resolution, decomposition is more accurate

4. Implementation and Results

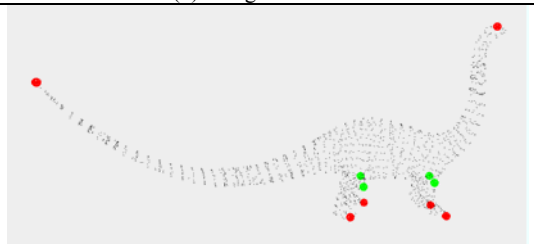
We have implemented our method. Some of our decomposition results are shown in Figures 11 to 14 (color plate). In each figure, we show the original mesh model. Then, three sub-figures show the results of the geodesic tree, critical points and components, in automatically and interactively selected root, respectively. It is clear that the interactively selected root is more accurate. Thus, it gives better decomposition results. In table 6, we show the running time of some examples on a Pentium 4, 1.6 GHz, 256MB memory computer.



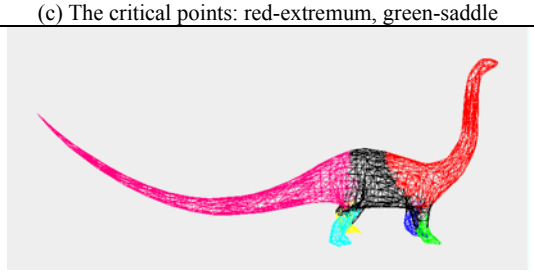
(a) Original model

Automatically selected root

(b) The geodesic tree



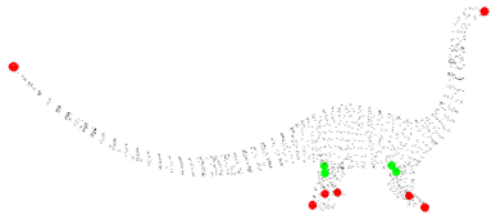
(c) The critical points: red-extremum, green-saddle



(d) Decomposition result: components shown in different colors

Interactively selected root

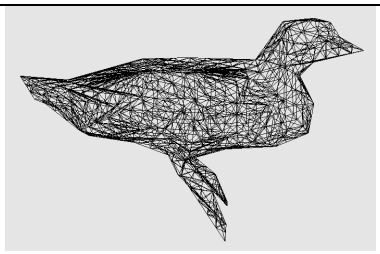
(e) The geodesic tree



(f) The critical points: red-extremum, green-saddle

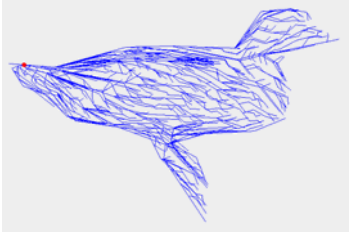
(g) Decomposition result: components shown in different colors

Figure 11. Result of dinosaur model

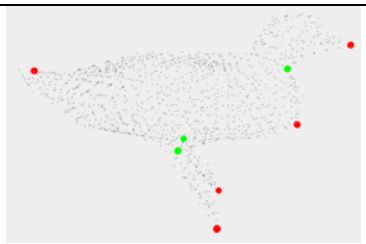


(a) Original model

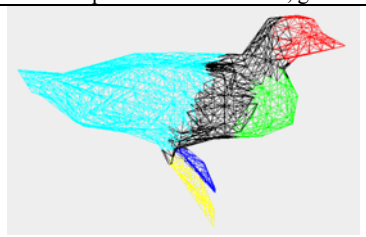
Automatically selected root



(b) The geodesic tree



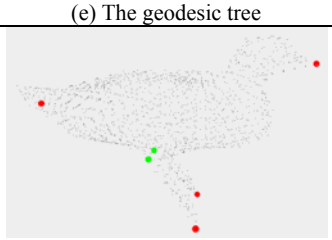
(c) The critical points: red-extremum, green-saddle



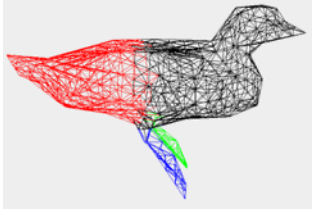
(d) Decomposition result: components shown in different colors

Interactively selected root

(e) The geodesic tree

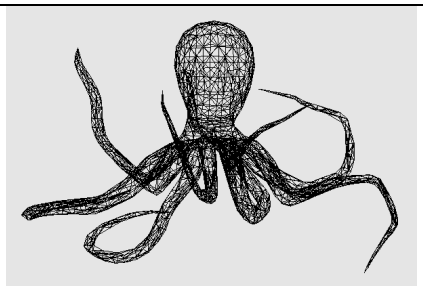


(f) The critical points: red-extremum, green-saddle



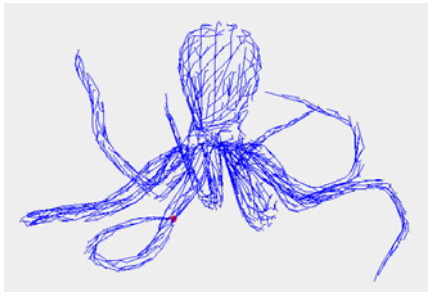
(g) Decomposition result: components shown in different colors

Figure 12. Result of duck model

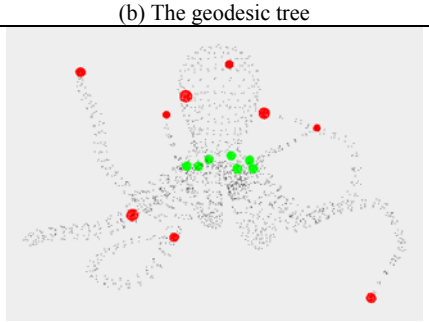


(a) Original model

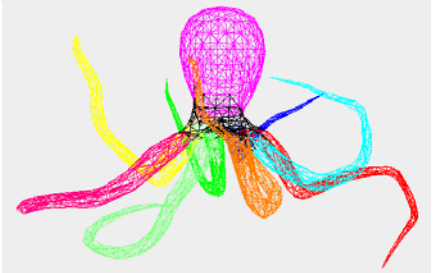
Automatically selected root



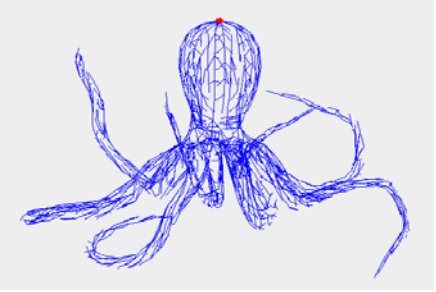
(b) The geodesic tree



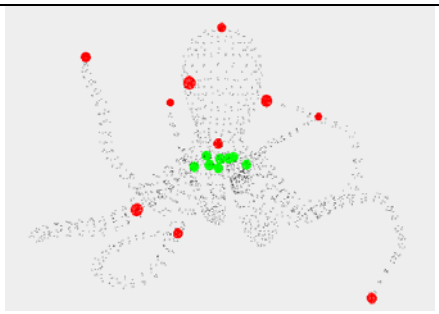
(c) The critical points: red-extremum, green-saddle



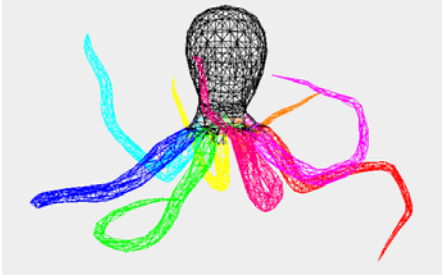
(d) Decomposition result: components shown in different colors



(e) The geodesic tree

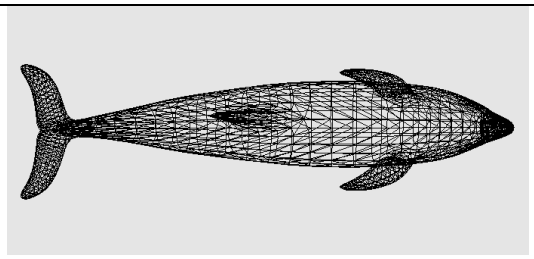


(f) The critical points: red-extremum, green-saddle



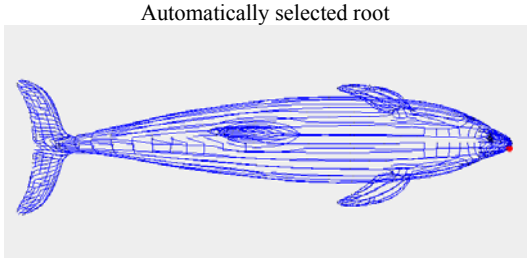
(g) Decomposition result: components shown in different colors

Figure 13. Result of an octopus model

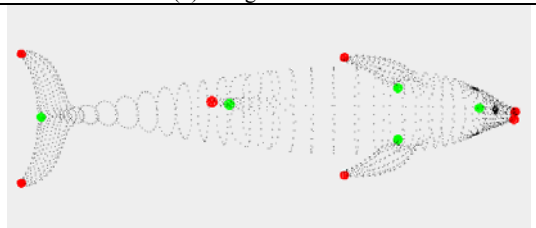


(a) Original model

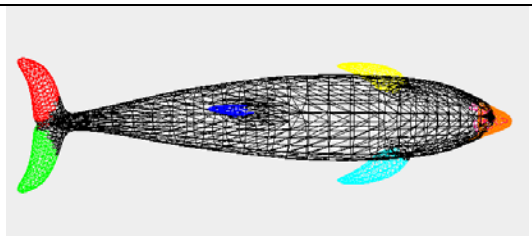
Automatically selected root



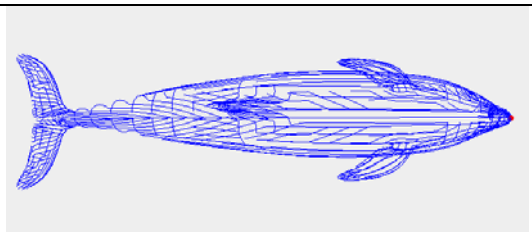
(b) The geodesic tree



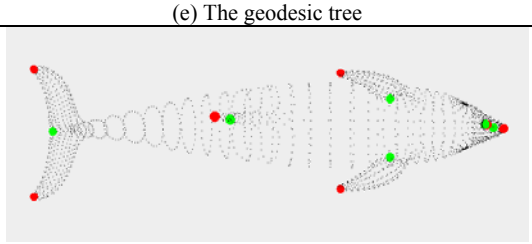
(c) The critical points: red-extremum, green-saddle



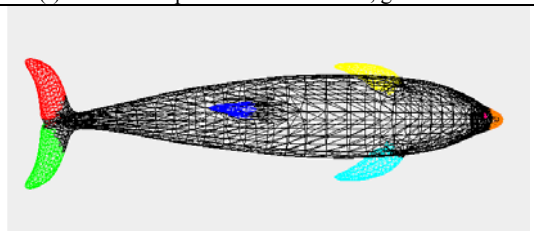
(d) Decomposition result: components shown in different colors



(e) The geodesic tree



(f) The critical points: red-extremum, green-saddle



(g) Decomposition result: components shown in different colors

Figure 14. Result of a fish model

	Dino-saur	**Dolphin**	**Duck**	**Octopus**
No. of vertices	1551	2101	916	1909
No. of faces	3098	4198	1828	3814
Decomposition time (second)				
Automatic allocation of root	2.00	2.00	1.00	3.00
Interactively assigned root	1.00	1.00	<1.00	2.00

Table 6. Decomposition time with preprocess done before critical points detection on Pentium 4, 1.6 GHz, 256MB memory computer

5. Conclusion and Future Work

We propose a method for decomposing an object represented in polygon meshes into components by means of critical points. We have implemented it and conducted experiments on a few 3D objects. On the whole, it is effective and efficient. The results are satisfactory.

For its applications, one of them is 3D model retrieval. With the components of a 3D model, a component hierarchy is derived and the retrieval problem can be addressed as a tree matching. Another application is component-based morphing. For instance, the morphing system of Gregory et al. [6] could take advantage of our decomposition result.

6. Acknowledgment

This first author is supported by NUS undergraduate student scholarship. This work is also partly supported by NUS grant S252-000-090-112.

References

[1] Attali D. and Montanvert A. (1997) Computing and Simplifying 2D and 3D Continuous Skeletons. Computer Vision and Image Understanding Vol. 67, No. 3, September, pp. 261-273,1997

[2] Biederman I. (1987) Recognition-by Components: A Theory of Human Image Understanding. Psychological Review, Vol. 94, No. 2, 1987, pp.115-147.

[3] Chazelle B., Dobkin D. P., Shouraboura N, and Tal A, (1995) Strategies for Polyhedral Surface Decomposition an Experimental Study, Proc. Symp. Computational Geometry, 1995, pp. 297-305.

[4] Chen J. and Han Y.(1990) Shortest Paths on Polyhedron. Proc. Symp. Theory of Computing, 1990, pp.360-369.

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

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

[7] Katz S., A. Tal A., "Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts", SIGGRAPH 2003.

[8] Hilaga M., Shinagawa Y., Kohmura T., and Kunii T., Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. Proc. SIGGRAPH 2001, pp. 203-212.

[9] Lanthier M., Maheshwari A. and Sack J. (1999) Approximating Weighted Shortest Paths on Polyhedral Surfaces. Proc. Symp. Computational Geometry, 1999, pp.274-283.

[10] Lazarus F. and Verroust A.(1999) Level Set Diagrams of Polyhedral Objects. Solid Modeling and Applications 1999. Ann Arbour, 1999, June.

[11] Li X.T., Woon T.W., Tan T.S. and Huang Z.Y. (2001) Decomposing Polygon Meshes for Interactive Applications. In Proceedings of ACM Symposium on Interactive 3D Graphics, 2001. pp.35-42, pp.243.

[12] Mangan A. P. and Whitaker R. T. (1999) Partitioning 3D Surface Meshes Using Watershed Segmentation. IEEE Transactions on Visualization and Computer Graphics. Vol. 5, 1999. pp. 308-321.

[13] Nevatia R. and Binford T. O. (1977) Description and Recognition of Curved Objects. Artificial Intelligence, Vol. 8, February, 1977, pp.77-98.

[14] Pentland A. P. (1986) Perceptual Organization and the Representation of Natural Form. Artificial Intelligence, Vol. 28, May, 1986, pp. 293-331.

[15] Shlafman S., Tal A., Katz S., "Metamorphosis of Polyhedral Surfaces using Decomposition", Eurographics 2002, Volume 21, Number 3.

[16] Svensson S. (1999) Decomposition Digital 3D Shape Using a Multiresolution Structure. In Discrete Geometry for Computer Imagery Conference, Springer-Verlag. Italy, 1999, pp. 19-30.

[17] Takahashi S. , Shinagawa Y. and Kunii T. L. (1997) A feature-based approach for smooth surfaces. In Solid Modeling, May 1997. Atlanta, Georgia, ACM.

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

[19] Zhou Y., Kaufman A and Toga A.W. (1998) Three-dimensional Skeleton and Centerline Generation Based on An Approximate Minimum Distance Field. The Visual Computer Vol.14, 1998 pp.303-314.