Fast Mesh Segmentation using Random Walks

Yu-Kun Lai*
Tsinghua University
Beiing, China

Shi-Min Hu†
Tsinghua University
Beijing, China

Ralph R. Martin‡
Cardiff University
Wales, UK

Paul L. Rosin§
Cardiff University
Wales, UK

Abstract

3D mesh models are now widely available for use in various applications. The demand for automatic model analysis and understanding is ever increasing. Mesh segmentation is an important step towards model understanding, and acts as a useful tool for different mesh processing applications, e.g. reverse engineering and modeling by example. We extend a random walk method used previously for image segmentation to give algorithms for both interactive and automatic mesh segmentation. This method is extremely efficient, and scales almost linearly with increasing number of faces. For models of moderate size, interactive performance is achieved with commodity PCs. It is easy-to-implement, robust to noise in the mesh, and yields results suitable for downstream applications for both graphical and engineering models.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms, languages, and systems;

Keywords: mesh segmentation, random walks, interactive

1 Introduction

With the development of 3D acquisition techniques, 3D mesh models are now widely available and used in various applications. The demand for model analysis and understanding is thus ever increasing. However, techniques for intelligent automated processing of large mesh models have not matched the growth in availability of models. The task of mesh *segmentation* is to decompose a mesh model into a set of disjoint pieces whose union corresponds to the original model. To be useful, segmentation must decompose a mesh into meaningful pieces (e.g. limbs and torso of an animal) or ones which satisfy other desirable criteria (e.g. each piece is bounded by sharp edges or small radius blends).

Mesh segmentation is an important step towards model analysis and understanding. A variety of different applications could benefit from preprocessing the mesh using an efficient and reliable mesh segmentation method. In the field of reverse engineering of CAD models, segmentation plays an important role in splitting a model into pieces, each of which may then be fitted with a single analytical surface [Várady et al. 1997]. In computer graphics, segmentation can be applied in various applications, including mesh simplification [Zuckerberger et al. 2002], collision detection [Li et al. 2001], morphing [Shlafman et al. 2002; Zuckerberger et al. 2002] and skeleton-driven animation [Katz and Tal 2003].

*e-mail:laiyk03@mails.thu.edu.cn
†e-mail:shimin@tsinghua.edu.cn
‡e-mail:ralph@cs.cf.ac.uk
§e-mail:Paul.Rosin@cs.cf.ac.uk

The basis for mesh segmentation derives from cognitive science. As pointed out by Hoffmann [Hoffmann and Richards 1984; Hoffmann and Singh 1997], the human visual system perceives region boundaries at negative minima of principal curvature, or concave creases—this observation is known as the *minima rule*. The depth of the concavity directly affects the salience of region boundaries. Such concave feature regions together with other information are important cues for segmentation. Moreover, it can be observed that only significant features are important to segmentation; small-scale fluctuations should be ignored, even if they represent sharp creases. On the other hand, in the case of reverse engineering, different surfaces are separated along sharp edges, which may be convex or concave, or even along smooth edges—different criteria are applicable to segmentation for such uses.

Recently, Grady [2006] proposed an interactive algorithm for image segmentation based on the use of random walks. The main ideas are as follows: a set of seed pixels is first specified by the user. For all other pixels, using an efficient process, we determine the probability that a random walk starting at that pixel first reaches each particular seed, given some definition of the probability of stepping from a given pixel to each neighbor. The segmentation is formed by assigning the label of the seed first reached to the non-seed pixel.

Our work is an extension of the random walks method to the particular problem of mesh *segmentation*; previous work has already considered the use of random walks for solving the different problem of mesh *denoising* [Sun et al. 2007]. By using different methods of assigning probability distributions, we are able to segment both engineering object models and graphical models. Our results are reliable even when the models are noisy or have small-scale textures that should be ignored.

As well as a method based on user selection of seeds, we give a generalization of this method to automatic mesh segmentation, which automatically places seeds (usually with more seeds than the required number of regions) using feature sensitive isotropic point sampling. Our two-pass method segments the mesh using these initial seeds, and then merges the regions found based on similarities of neighboring regions.

Compared to other methods, our proposed method has the following advantages. Our method:

- provides results of comparable quality to state-of-the-art methods, but is significantly more efficient, making it especially suitable for interactive applications or applications that require segmentation of large models, or large numbers of models,
- can be used both interactively or automatically, and in particular the optimal number of regions can be deduced automatically,
- is robust to noise and small-scale texture that may be present in real scanned models,
- is applicable to both CAD models and graphical models, and
- is easy-to-implement.

Section 2 briefly reviews related work. Interactive and automatic mesh segmentation algorithms based on random walks are pre-

sented in Sections 3 and 4 respectively. Experimental results are given in Section 5, with conclusions and discussions in Section 6.

2 Related Work

Compared to the problem of image segmentation, research into mesh segmentation is much more recent; however, it is now an active research topic, due to the wide range of potential applications. A complete survey of mesh segmentation is beyond the scope of the paper, but an up-to-date review and comparison of different methods can be found in [Attene et al. 2006a].

Based on the different aims, existing mesh segmentation algorithms can be generally categorized into two classes. The first class is aimed at applications such as reverse engineering of CAD models (e.g. [Attene et al. 2006b]). Such methods segment a mesh model into patches each of which is a best fit to one of a given class of mathematical surfaces, e.g. planes, cylinders, etc. The second class tries to segment typically 'natural objects' into meaningful pieces, as expected by a human observer. Our algorithm is mainly aimed at solving problems of the latter class, but with certain modifications, it is also able to handle engineering objects reasonably well.

Most state-of-the-art work on mesh segmentation is based on iterative clustering. Shlafman et al. [2002] use k-means clustering to segment the models into meaningful pieces. Katz [Katz and Tal 2003] improved on this by using fuzzy clustering and minimal boundary cuts to achieve smoother boundaries between clusters. Top-down hierarchical segmentation has also been used to segment objects with a natural hierarchy of features. Lai [Lai et al. 2006] suggested combining integral and statistical quantities derived from local surface characteristics, producing more meaningful results on meshes with noise or repeated patterns. One of the most prominent drawbacks of such algorithms is the necessity to compute pairwise distances, making it expensive or even prohibitive to handle large models directly. To handle models with e.g. more than 10,000 faces, mesh simplification [Katz and Tal 2003; Katz et al. 2005; Liu and Zhang 2004] or remeshing [Lai et al. 2006] is typically used. Spectral clustering has also been used [Liu and Zhang 2004] with good results, but this approach also suffers from performance problems, although Nyström has given an approximation method for accelerating it [Liu et al. 2006].

Unsupervised clustering techniques like the mean shift method can also be applied to mesh segmentation. Shamir [Shamir et al. 2004] extended mean shift analysis to mesh models based on use of a local parameterization method. Later, Yamauchi [Yamachi et al. 2005] applied mean shift clustering to surface normals. Such methods tend to oversegment a model into more pieces than expected or desired.

Other methods for mesh segmentation also exist. Mangan and Whitaker [1999] applied bobsledding watershed algorithm to triangle meshes. Li [Li et al. 2001] proposed using skeletonization based on edge contraction and space sweeping to perform mesh decomposition. Visually appealing results are obtained; however, their results depend mainly on large-scale features, and do not always capture salient geometric features. Recently, Reniers and Telea [2007] used curve skeletons in hierarchical mesh segmentation. Katz [Katz et al. 2005] proposed a segmentation algorithm based on multidimensional scaling and extraction of feature points and cores. The method is able to produce consistent results when regions of a mesh are placed in differing relative poses. However, an expensive method is used to find feature points, which limits the complexity of models that can be efficiently handled, even after simplification. Mitani [Mitani and Suzuki 2004] proposed a technique for making paper models from meshes. This can be considered to be a specialized mesh segmentation method that produces naturally developable triangle strips.

In the sense of segmentation of engineering objects, especially for the purpose of reverse engineering, there exist quite a few works. Most of such work deals with point cloud directly instead of triangle meshes due to its wide availability. Sapidis and Besl [Sapidis and Besl 1995] proposed a method to construct polynomial surfaces from point cloud data, using region growing for segmentation. Benkő and Várady [2002] proposed a method to directly segment point cloud data of engineering objects based on a serial of top-down recursive tests. Gelfand and Guibas [2004] proposed to use slippage analysis and multi-pass region growing to segment different regions based on different slippage signatures. Edelsbrunner [Edelsbrunner et al. 2003] proposed to segment meshes with piecewise linear Morse-Smale theory. This idea has been extended to suit the needs for producing CAD-like segmentation results [Várady 2007].

Certain work explicitly considers the problem of interactive mesh segmentation. Lee proposed a method to segment models using user-guided or automatically extracted cut lines based on 3D snakes [Lee et al. 2004; Lee et al. 2005]. Funkhouser [Funkhouser et al. 2004] provided an intuitive interactive segmentation tool to find optimal cuts guided by user-drawn strokes, and applied it to a modeling system based on stitching parts extracted from a model database. An interactive segmentation method based on graph-cut was proposed by Sharf, again for use in a cut-and-paste system [Sharf et al. 2006].

Our method is different to any of the above in that it is based on a random walk paradigm. The formulation leads to the need to solve a sparse linear system, which is very efficient. Unlike most interactive methods, user interaction is provided by specifying a set of seed mesh faces, which is much easier than specifying a rough cutting boundary. In applications where automatic methods are preferred, we use a two-stage method that first oversegments the mesh using a set of automatically chosen seeds, and then merges these initial regions to give the final regions.

3 Interactive Segmentation

In this section, we will discuss our algorithm for interactive mesh segmentation using random walks; extension to an automatic mesh segmentation algorithm will be discussed in the next section. The basic idea of the algorithm is in spirit similar to the corresponding method for image segmentation [Grady 2006], but due to the differences of source data and aims, certain issues must be resolved.

We assume that the given models are triangular meshes. Random walk mesh segmentation proceeds as follows: assume that the user picks n faces as seeds, where n is the number of final regions desired; seeds are placed so that one seed 'obviously' lies within each of the final regions the user desires. We denote the seeds by $s_1, \ldots, s_n$. Other faces are non-seed faces, denoted by $f_1, \ldots, f_m$. We associate a probability with each of the three edges $e_{k,i}$ of each non-seed face f_k, denoted by $p_{k,1}$, $p_{k,2}$ and $p_{k,3}$ respectively. These correspond to the probabilities that a random walk will move across a particular edge to the corresponding neighbor. These probabilities satisfy the following equation:

$$\sum_{i=1}^{3} p_{k,i} = 1. \tag{1}$$

For $i = 1, 2, 3$, denote the face sharing $e_{k,i}$ with f_k by $f_{k,i}$. For a particular seed face s_l, denote the probability of a random walk

starting from a particular face f_k arriving at s_l first, before reaching other seeds, as $P^l(f_k)$, for $l = 1, \ldots, n$. $P^l(s_l) = 1$ and $P^l(s_k) = 0$ for any $k \neq l$. As the number of steps considered increases, in the limit, the following equation holds for each non-seed face f_k (for each $l = 1, 2, \ldots n$):

$$P^l(f_k) = \sum_{i=1}^{3} p_{k,i} P^l(f_{k,i}). \tag{2}$$

For a particular seed face s_l, the $P^l(f_k)$ form a column vector of length m (denoted by P^l) that needs to be computed, and we have m equations of the form given in Eqn. 2. We may rewrite Eqn. 2 in matrix form as $A_{m\times m}P^l = B^l$, where A and B^l can be deduced from Eqn. 2. Most values in B^l are zeros. However, from Eqn. 2, for a non-seed face f_k adjacent to a seed face, the corresponding $P^l(f_{k,i})$ is not a variable, but a constant, either 0 (if not the l^{th} seed) or 1 (for l^{th} seed). Since the l^{th} seed has at most (and normally, exactly) three neighbors, B^l also has at most (and normally) three non-zero values.

Note that A is independent of the choice of l. Thus we may put the P^l together and form a matrix $P_{m\times n}$ with rows $P_{l,k} = P^l(f_k)$, to give $AP = B$, where $B = (B_1, \ldots, B_n)$. This sparse linear system has the same general nature as the one in [Grady 2006]; this system is sparse as each row of the matrix contains at most 4 non-zero entries, as shown in Eqn 2. Following the argument in [Grady 2006], the matrix A is positive semi-definite, and the solution to this linear system is uniquely determined. Please note that random walk model described above is in essence equivalent to electric network model as the distribution of electric potentials at each face, where the probability to move to neighboring face corresponds to the reciprocal of resistors (i.e. conductances). See [Doyle and Snell 1984] for a thorough study.

We now define that a given face belongs to the region attached to seed s_l if a random walk starting at that face has a higher probability of reaching this seed than any other seed. Thus, after computing $P^l(f_k)$ for $l = 1, \ldots, n$ and $k = 1, 2, \ldots, m$, we assign the label for seed s_l to those non-seed faces f_k which satisfy

$$P^l(f_k) = \max_{t=1,\ldots,n} P^t(f_k). \tag{3}$$

It can be shown that each region produced by this segmentation process is guaranteed to be contiguous.

Given the basic framework given by the algorithm above, two significant issues remain: to determine appropriate probabilities for stepping from face to face, what optional preprocessing and post-processing steps may be needed to further improve the results.

3.1 Probability computation

Choice of suitable probability assignments, i.e. $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, for each face i is essential for the random walk approach to give good mesh segmentation results. Appropriate probabilities are affected by the types of models, due to the different purposes of mesh segmentation. In the following, we will address 'natural' graphical models and engineering object models separately.

3.1.1 Graphical models

Mesh segmentation of graphical models should split a model into meaningful pieces. The most important information for segmentation comes from the *minima rule*, as used by many segmentation algorithms, where significant (concave) features are considered as important hints. For a given face f_i, we define a *difference function* $d(f_i, f_{i,k})$ which measures the difference in some specific geometric property between f_i and one of its neighboring faces $f_{i,k}$, $k = 1, 2, 3$. For graphical models, we define this function to mainly depend on a function d_1 measuring the dihedral angle:

$$d_1(f_i, f_{i,k}) = \eta\left[1 - \cos\left(\mathrm{dihedral}(f_i, f_{i,k})\right)\right] = \frac{\eta}{2}\left|\left|\mathbf{N}_i - \mathbf{N}_{i,k}\right|\right|^2, \tag{4}$$

where $\mathrm{dihedral}(f_l, f_m)$ represents the dihedral angle between adjacent faces f_l and f_m, and $\mathbf{N}_l$ is the normal to face f_l. η is used to give higher priority to concave edges: we set $\eta = 1.0$ for concave edges and a relatively small number (e.g. 0.2) for convex edges, according to the minima rule.

To handle variations in the dihedral distribution, we normalize d_1 by its average over all edges, $\bar{d}_1$, giving as the overall difference function d:

$$d(f_i, f_{i,k}) = \frac{d_1(f_i, f_{i,k})}{\bar{d}_1}. \tag{5}$$

Given a definition for the difference function at hand, the probability distribution is now computed as

$$p_{i,k} = |e_{i,k}| \exp\left\{-\frac{d(f_i, f_{i,k})}{\sigma}\right\}, \tag{6}$$

where $|e_{i,k}|$ is the edge length of the corresponding common edge, and σ is used to control how variation of differences maps to variations in probability. In our experiments, we have found $\sigma = 1.0$ works well for most of cases. $p_{i,k}$ is then normalized to sum to one over each face. An exponential function is used above as a convenient way of mapping differences in $(0, \infty)$ to probabilities in $(0, 1)$, where a high difference corresponds to a low probability

3.1.2 Engineering models

Segmentation of engineering object meshes differs in its aims from segmentation of graphical models. We usually want to segment such a mesh into pieces that can each be fitted with some analytical surface [Várady et al. 1997]. In many typical cases (but not all), Gaussian and mean curvatures should be almost uniform over a segment, which is a different requirement from the case of graphical models.

Again, we use d_1 to measure the change of normals between adjacent faces; however, for engineering object mesh segmentation, we set $\eta = 1.0$ for both convex and concave edges, since they are equally important for the segmentation of such models. Moreover, we introduce two further difference measures for the variation of Gaussian and mean curvatures. To begin with, we need to estimate the Gaussian and mean curvatures on both sides of a given edge. Such curvature estimates are known to be sensitive to noise, so we use robust estimators for this purpose—we use PCA-based integral invariants in ball neighborhoods [Yang et al. 2006]. The method basically relies on a covariance analysis of the intersection volume between a ball of radius r and the volume of interior part of the given model. Since the method actually computes principal tensors at a regular mesh point, we may adapt this method to directly interpolate the principal curvatures at the center of each face rather than at each vertex. We denote Gaussian and mean curvatures at face f_i as $K(f_i)$ and $H(f_i)$ respectively. If the model is relatively clean, we may set r to be 1 to 2 times the average edge length of the model. For noisy models, to make the result robust, we must use a larger radius r at the cost of sacrificing the ability to accurately locate some boundaries.

The difference functions for Gaussian and mean curvatures are now defined as

$$\begin{aligned} d_2(f_i, f_{i,k}) &= |K(f_i) - K(f_{i,k})| \\ d_3(f_i, f_{i,k}) &= |H(f_i) - H(f_{i,k})| \,. \end{aligned} \tag{7}$$

The overall difference function is defined by combining d_1, d_2 and d_3 to be:

$$d^*(f_i, f_{i,k}) = \max \left\{ \frac{d_1(f_i, f_{i,k})}{\bar{d}_1}, \frac{d_2(f_i, f_{i,k})}{\bar{d}_2}, \frac{d_3(f_i, f_{i,k})}{\bar{d}_3} \right\}, \tag{8}$$

where $\bar{d}_1$, $\bar{d}_2$ and $\bar{d}_3$ are average values of the corresponding difference function over the whole model. Note that the maximum of these three is used instead of their weighted average: the peak responses of any component are significant, and this approach also avoids the difficulty of choosing appropriate weights. Given d^*, the probability is again defined using Eqn. 6, but with d^* in place of d. The method works well for separating smoothly touching regions, as illustrated in Fig. 1.

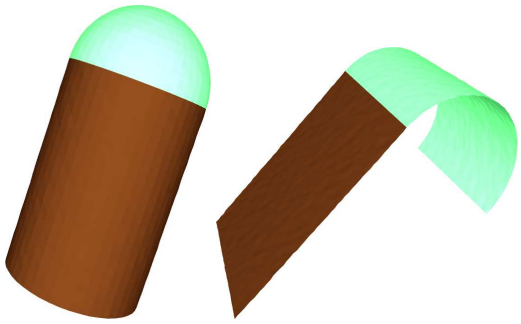

Figure 1: *segmentation of CAD models without sharp edges.*

3.2 Preprocessing and postprocessing

Although the method as described is much faster than any method based on iterative clustering, for very large models, it may be preferable to simplify or remesh the models to a more practical size (e.g.$10,000$-$20,000$ faces) for efficiency. This is also reasonable, since extra detail in models actually provides little extra help in segmentation. Segmentation can be computed using the faces of the reduced model. This step is optional for the overall pipeline.

After random walk segmentation, each segment is represented by a contiguous set of faces. The boundaries may be somewhat jagged, partly due to noise and other variations in local properties near the separating edges, and partially due to the limited resolution of the mesh. We use feature sensitive smoothing as proposed in [Lai et al. 2007] to smooth the segment boundaries while keeping them snapped to features. This amounts to optimizing a discretized spline-in-tension energy in the feature sensitive metric. The boundaries generally form a complicated graph, so branching points are first detected and each boundary segment between branching points is smoothed independently.

The smoothed boundaries are represented as a set of connected points; however each point generally will not be located at any vertex of the initial mesh. We suggest updating the input mesh model slightly so that the smoothed boundaries map to a sequence of edges in the updated mesh. To do so, we first project each point on the smoothed boundary onto the input mesh model. The resulting point may be located at a vertex, on an edge, or within a face. In the latter two cases, we split the related faces to make this point a vertex of the revised mesh (as illustrated in Fig. 2(left)). Projection is done quickly using the approximate nearest neighbors library [Mount and Arya 2005]. After projection, we find the geodesic path across the mesh between adjacent projected vertices [Surazhsky et al. 2005], and split each face crossed by the geodesic into two. To ensure that the resulting mesh remains a triangular mesh, quad faces induced by this splitting are further split into two triangles.

Such local updates can be performed efficiently. Since the geodesic computation requires a data structure that cannot be easily adapted for dynamic updating of the mesh structure, we use the assumption that adjacent points on the smoothed boundaries are usually close to each other, build a small patch of the input mesh that covers both projected points, and compute the geodesics on such small patches. An example of such splitting is shown in Fig. 2(right). The blue edges correspond to those which must be added so that geodesic edges become edges of the mesh. Thick blue edges correspond to edges that are part of the smoothed boundary. After this process, the smoothed boundaries can be directly mapped to edges of the modified input model, and the segmentation results after smoothing may be represented by assigning a label to each face of the modified input mesh.

4 Automatic Segmentation

The random walk segmentation method may be adapted to work automatically. In this case, a set of seeds is automatically selected, generally with more seeds than the number of finally expected clusters. For segmentation of graphical models, we usually require a coarse segmentation, which does not need a dense set of seeds; for engineering object meshes, or when a detailed segmentation is preferred, more seeds may be necessary. Our interface allows users to specify both an approximate number of seeds, and to place specific seeds before or after automatic selection.

The random walk algorithm described above is used to segment the model. There will in general be more resulting pieces than desired, and so a further merging process is used to combine these oversegmented pieces into the final segments. This approach works well in practice, as it is based on our experimental observation that random walk segmentation results are not sensitive to the exact location of the seeds (as demonstrated later).

4.1 Coarse-Scale Seeding

If it is desired to segment the model into large pieces representing large-scale structures, we should generally evenly distribute a sparse set of seeds, so that only the most significant features or protrusions are captured. Based on the observation that the segmentation results are generally insensitive to the exact location of seeds, we use a clustering method similar to that used in k-means clustering segmentation.

The first seed face is selected as the (or a) face furthest away, in terms of geodesic distance, from the face closest to the centroid of all faces. We then iteratively add new seed faces one by one. For any two faces f_i and f_j, a *path* from f_i to f_j is a contiguous sequence of faces starting from f_i and ending at f_j. For any path, we may compute the sum of the difference measures d (or d^* if appropriate), and select the minimal sum among all possible paths, denoting it by $D(f_i, f_j)$. Assume $s_1, \ldots, s_n$ are n faces already selected as seeds. The next seed face s_{n+1} is determined by

$$s_{n+1} = \arg\max_{f_k \in F} \left\{ \min_{i=1,\ldots,n} D(f_k, s_i) \right\}, \tag{9}$$

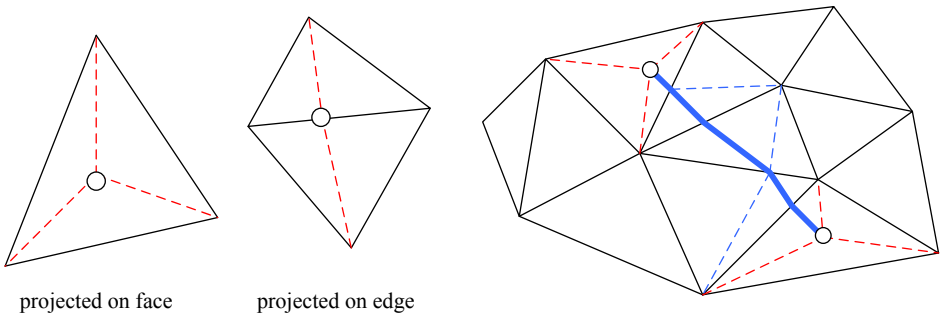

Figure 2: *Projection and local update of the input model. Left: mapping smoothed boundary points onto the surface; right: mapping smoothed boundary paths onto the surface.*

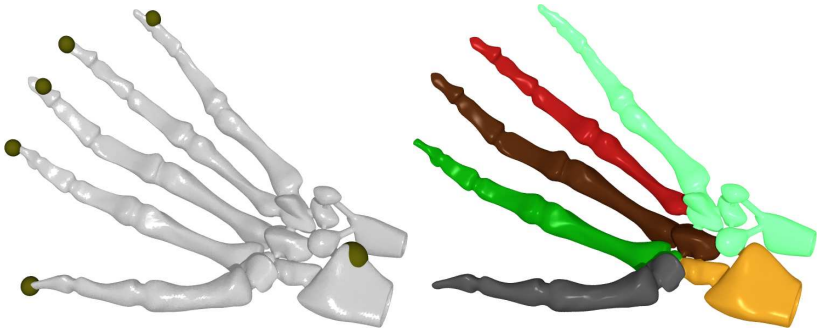

Figure 3: *Example of coarse seeding (left) and corresponding segmentation result (right).*

where F is the set of all the faces. This process terminates when a significant decrease of D occurs between the newly selected s_{n+1} to the nearest neighboring seed, whereupon s_{n+1} is discarded. Note that this computation is efficient, since we only need to solve a few single-source shortest distance problems starting from each seed face. Using Dijkstra's algorithm gives a complexity of $O(nm \log m)$, where n and m are the number of seed faces and the total number of faces, respectively.

Fig. 3 shows an example of coarse seeding. The left figure gives the positions of seeds (the colored balls indicating the seed locations) and the right figure is the corresponding segmentation result.

4.2 Fine-Scale Seeding

In certain cases, we would like to segment the model into smaller pieces, where significant parts are segmented as much as possible. For example, given the skeleton example shown in Fig. 3, we may want to segment to further detail than simply 5 fingers. Fine-scale seeding based on automatic seed distribution and merging is then more appropriate.

4.2.1 Automatic seed selection

We should pick a set of random faces that are in general evenly distributed over the surface, and gives higher priority to protrusions and regions containing features. Feature sensitive sampling (the first phase of feature sensitive remeshing proposed in [Lai et al. 2007]) suits this need well. The method basically distributes particles over the model optimizing some spring-like energy [Witkin and Heckbert 1994]. After distribution, we pick those faces with particles in them as seeds. For our purpose, we may use a sufficiently large number of sampling faces (e.g. 20–200 for most models). Again assuming that n is the number of seeds, and m is the total number of faces, the time complexity is $O(n \log m)$, since nearest neighbor queries are performed using kd-tree acceleration. Placement of initial seeds is typically very fast (much less than a second). Note that if the original number of seeds is not large enough to cover all the significant features, our semi-automatic interface allows users to add further seeds where desired.

4.2.2 Merging

Using such an approach, we expect many segments to have multiple seeds, which naturally leads to over-segmentation. However, as segmentation results are in general not sensitive to the exact placement of seeds, we may simply merge the resulting segments to give suitable final regions. We perform merging as an iterative process. To define the relative merging cost between two adjacent segments S_i and S_j, we first denote by $\partial S_i \cap \partial S_j$ the common boundary of the two segments, and by $\partial S_i \cup \partial S_j$ the combination of the two boundaries. We integrate the difference measure d (or $d*$ if appropriate) along the common boundary, and denote it by $D_{\partial S_i \cap \partial S_j} = \sum_{e \in \partial S_i \cap \partial S_j} |e| d_e$, where $|e|$ and d_e are the length of edge e and the difference measure d (or $d*$) between the two faces adjacent to e. We also define the overall length of common boundary as $L_{\partial S_i \cap \partial S_j} = \sum_{e \in S_i \cap S_j} |e|$. $D_{\partial S_i \cup \partial S_j}$ and $L_{\partial S_i \cup \partial S_j}$ can be defined similarly. We then define the relative merging cost $c_{i,j}$ as:

$$c_{i,j} = \frac{D_{\partial S_i \cap \partial S_j} / L_{\partial S_i \cap \partial S_j}}{D_{\partial S_i \cup \partial S_j} / L_{\partial S_i \cup \partial S_j}}. \tag{10}$$

For each adjacent pair of segments S_i and S_j, we compute the merging cost $c_{i,j}$ and put the pairs into a priority queue. The merging process proceeds by picking the pair with minimal merging cost, merging them into one segment and updating the priority queue accordingly. This process can be terminated either when

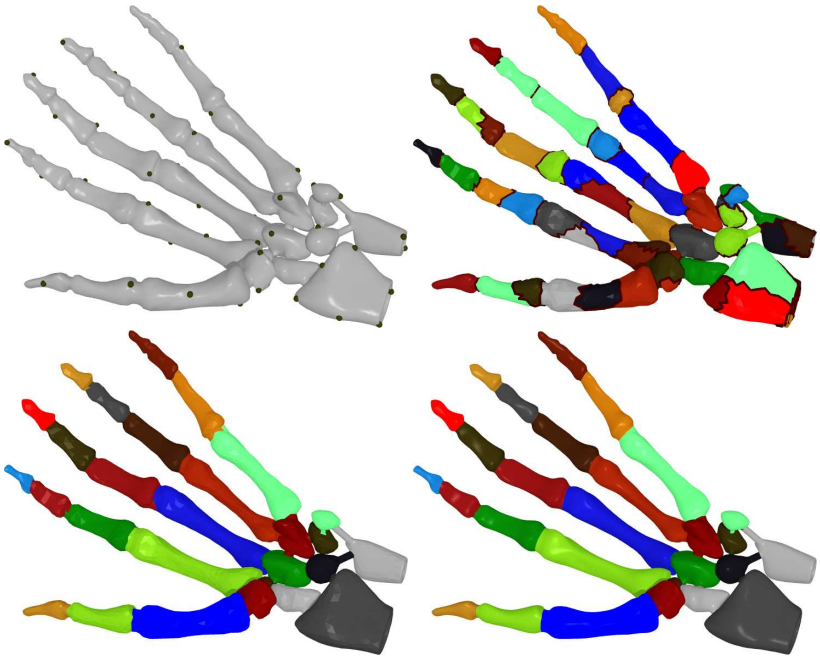

Figure 4: *Example of fine-scale seeding. From top left to bottom right: input model with automatic seed selection; initial (over-) segmented results; result after merging; final result after boundary smoothing and mapping.*

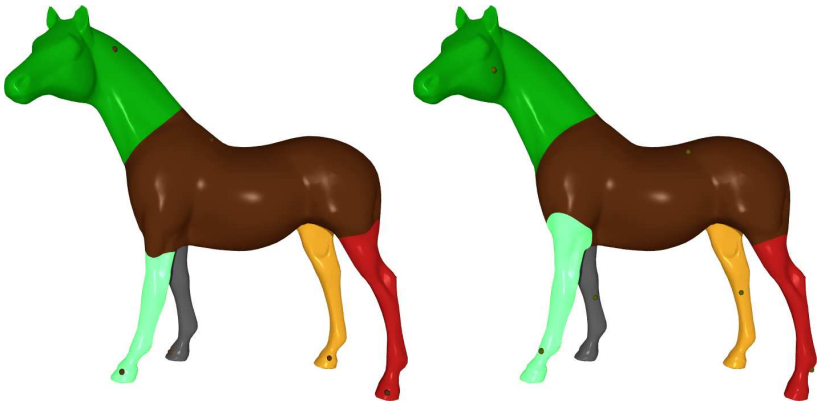

Figure 5: *Segmentation of horse with varying seed locations. Balls represent seed locations.*

there exists a significant increase in $c_{i,j}$ for the current pair, or when we have reached a final number of regions desired by the user. In our experiments, the merging process usually stops with minimal relative cost of about 0.5.

Experimental results of automatic segmentation before and after merging are shown in Fig. 4. The initial number of seeds is 60 and the number of segments after merging is 30.

5 Experimental Results

Our method is in general insensitive to the exact location of seeds. Fig. 5 shows an example of segmenting a horse model. Note that there are no clearly defined boundaries between the legs and the body, so changing the positions of the seeds has a slight effect on the final result; however, even if the seed positions are changed significantly, the results are similar, and snap to some local features.

A more accurate test of stability with respect to choice of seed faces was also performed. Given some maximal seed shift radius r, we allow all seeds to move randomly to any face within a geodesic distance of r from their original seed position (but we restrict new seed positions to be within the same segment found by segmentation using the original seeds). We performed tests using maximal shifts of $3\%, 6\%, 9\%, \ldots, 30\%$ of the size of the model. In each test we computed the percentage of faces with the same labels found when using the initial seeds (we call this the *normalized coverage*). To obtain a robust result, we performed 100 trials for each shift distance and averaged the normalized coverage. As illustrated in Fig. 6, for models like the horse example in Fig. 5 where no clearly defined boundaries exist between segments, the normalized coverage decreases gradually with increasing shift. Even for shifts of up to 30%, the averaged normalized coverage is above 84%. For models like the hand skeleton example in Fig. 3, where significant features exist between segments, the averaged normalized coverage is above 99.6% for shifts up to 30%, which means almost identical results are produced even with significant change of seed locations.

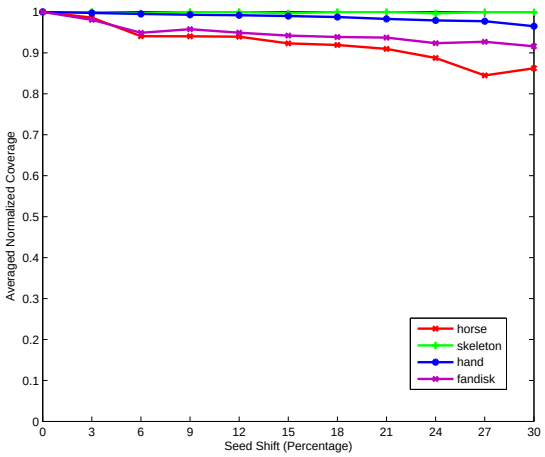

Figure 6: *Stability test results of averaged normalized coverage.*

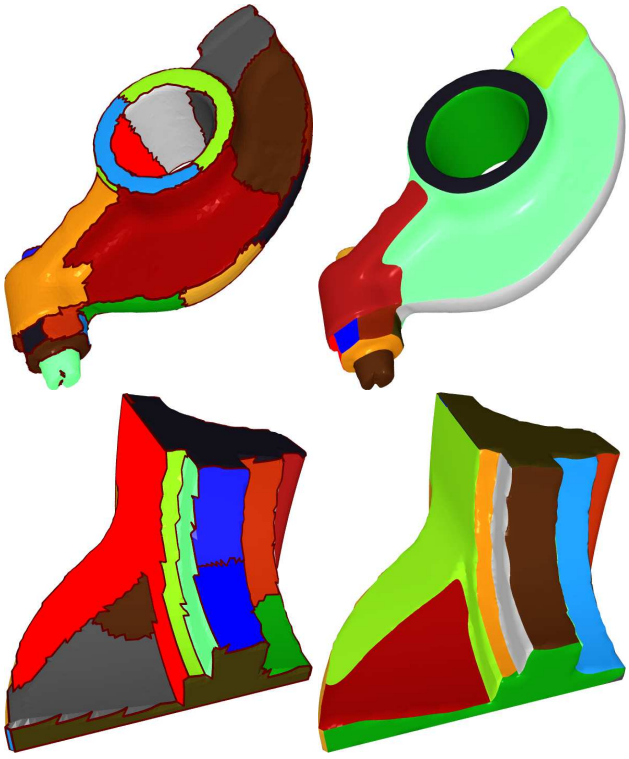

Figure 7: *Segmentation of engineering objects: rocker arm and fan disk. Left: initial segmentation with fine-scale seeding. Right: results after merging and smoothing.*

For a moderate example like the hand model in Fig. 8, the averaged normalized coverage is above 96.5% for seed shifts of up to 30%.

Our method can also be applied to meshes representing engineering objects. Fig. 7 gives the results of segmenting the well-known rocker arm and fan disk models. On the left are the segmentation results with fine-scale seeding, while on the right are the corresponding results after merging and smoothing. The number of seeds before merging is 40 for both models. The numbers of segments after merging are 20 for the rocker arm and 25 for the fan disk, respectively. Note that significant normal noise exists at sharp features of the input model, making the initial segmentation results rather jagged. These initial results also suffer from oversegmentation. After the merging and smoothing phase, however, results are significantly improved. The averaged normalized coverage for the fan disk (an engineering model) is also given in Fig. 7; in this case the normalized coverage tends to lie between the values for the hand and the horse examples, being above 91% for shifts up to 30%.

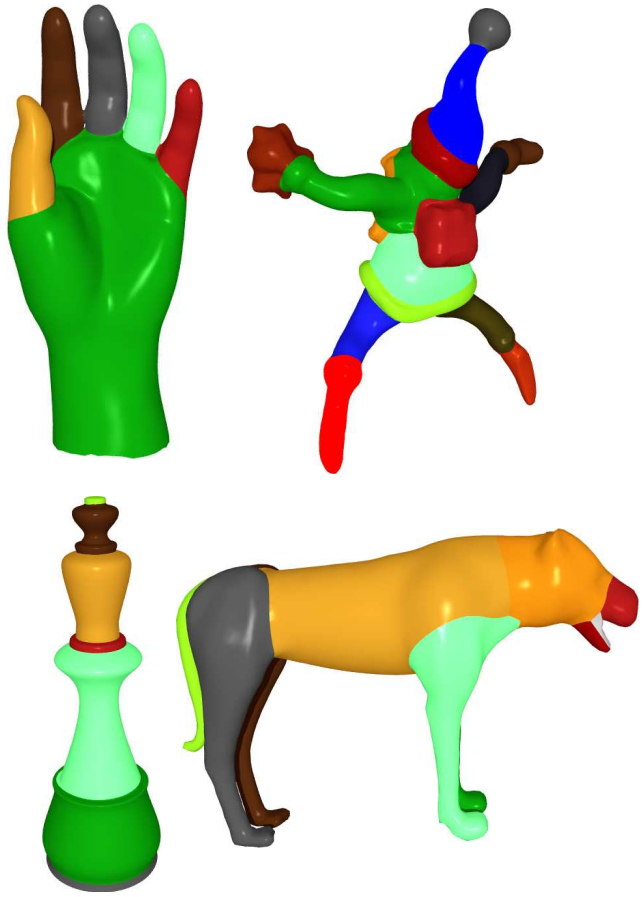

Figure 8: *Examples of mesh segmentation of various graphical models.*

We have also tested our method on various other graphical models, a selection of which are shown in Fig. 8. Hand, Santa, chessman, cheetah models are segmented into 6, 17, 7 and 10 pieces, respectively. Generally, intuitively reasonable and pleasing segmentation results are produced for such examples.

Moreover, compared with state-of-the-art methods, our method is very efficient both in time and memory usage. A detailed comparison of timings for a model remeshed to $10K$, $15K$, $20K$, $30K$ and $40K$ triangles with our method and an implementation of [Lai et al. 2006] is presented in Table 1; 6 seeds were used in each of these experiments, which were carried out on an Intel Core2Duo 2GHz laptop with 2GB RAM. Note that the computational time for k-means clustering based methods [Katz and Tal 2003; Lai et al. 2006; Liu and Zhang 2004] is dominated by pair-wise distance computations, and leading to a complexity of $O(m^2 \log m)$ time and $O(m^2)$ memory, where m is the number of faces. The method in [Katz et al. 2005] utilizes non-linear multidimensional scaling, which is even slower. The computations in our current method are dominated by solving the sparse linear system. We used

Table 1: *Timing comparison of a k-means clustering based method and our current method.*

Number of triangles	Clustering method [Lai et al. 2006], seconds	Current method, seconds	Current method, seconds/K triangles
$10K$	129	0.34	0.034
$15K$	303	0.47	0.031
$20K$	532	0.64	0.032
$30K$	1359	1.00	0.033
$40K$	n/a	1.34	0.034

Table 2: *Timings for the same $40K$-triangle model with differing numbers of seeds.*

No. of seeds	6	12	24	48
Timing (seconds)	1.3	2.1	3.4	6.5

MATLAB's direct solver ("backslash" operator) throughout the paper, though sparse linear solver libraries like TAUCS [Toledo et al. 2003] could also be used. Experimental results in Table 1 show that the times used per triangle are almost constant as the number of triangles varies. Time also increases more or less linearly with an increasing number of seeds, as shown by the example in Table 2. Clearly, such a linear bound is expected, as the linear system for each seed can be solved independently. (The practical timings show that the performance is actually faster than linear). In summary, the overall complexity with respect to the number of faces m and the number of seeds n is bounded by $O(mn)$.

Note that models with $40K$ triangles or more cannot be processed by the method in [Lai et al. 2006] without simplification or remeshing, due to memory limitations. Our method only requires $O(m + n)$ memory to store the sparse linear equations and thus does not have such memory limitations. For relatively small models with $10K$ triangles, the current method is more than 300 times faster, while for models as large as $30K$ triangles, it is more than $1,000$ times faster. For models of moderate size, the segmentation can be carried out in interactive time, suitable for interactive applications that require immediate feedback.

6 Conclusions

In this paper, we have presented both an interactive and an automatic method of mesh segmentation based on random walks. We have demonstrated the effectiveness of this method, with both 'natural' graphical model meshes and engineering object model meshes. The results are pleasing, and the method is sufficiently efficient to be useful in interactive applications, and in applications that require segmentation of large models, or a large collection of models. In future, we intend to explore extension of our method to hierarchical segmentation of models.

Acknowledgment

The models in this paper are courtesy of AIM@SHAPE Repository. The authors would like to thank Qian-Yi Zhou for his help on geodesic computation. This work was supported by the National Basic Research Project of China (Project Number 2006CB303102), the Natural Science Foundation of China (Project Number 60673004) and the National High Technology Research and Development Program of China (Project Number 2007AA01Z336).

References

ATTENE, M., FALCIDIENO, B., KATZ, S., MORTARA, M., PATANE, G., SPAGNUOLO, M., AND TAL, A. 2006. Mesh segmentation—a comparative study. In *Proc. IEEE Conference on Shape Modeling and Applications*, 7–18.

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3, 181–193.

BENKŐ, P., AND VÁRADY, T. 2002. Direct segmentation of smooth, multiple point regions. In *Proc. Geometric Modeling and Processing*, 169–178.

DOYLE, P. G., AND SNELL, J. L. 1984. *Random walks and eletric networks*. No. 22 in Carus Mathematical Monographs. The Mathematical Association of America.

EDELSBRUNNER, H., HARER, J., AND ZOMORODIAN, A. 2003. Hierarchical morse-smale complexes for piecewise linear 2-manifolds. *Discrete Computational Geometry 30*, 1, 87–107.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by examples. In *Proc. ACM SIGGRAPH*, 652–663.

GELFAND, N., AND GUIBAS, L. J. 2004. Shape segmentation using local slippage analysis. In *Proc. Eurographics Symposium on Geometry Processing*, 219–228.

GRADY, L. 2006. Random walks for image segmentation. *IEEE Trans. Pattern Analysis and Machine Inbtelligence 28*, 11, 1768–1783.

HOFFMANN, D., AND RICHARDS, W. 1984. Parts of recognition. *Cognition 18*, 65–96.

HOFFMANN, D., AND SINGH, M. 1997. Salience of visual parts. *Cognition 63*, 29–78.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graphics 22*, 3, 954–961.

KATZ, S., LEIFMAN, G., AND TAL, A. 2005. Mesh segmentation using feature point and core extraction. *The Visual Computer 21*, 8–10, 865–875.

LAI, Y.-K., ZHOU, Q.-Y., HU, S.-M., AND MARTIN, R. R. 2006. Feature sensitive mesh segmentation. In *Proc. ACM Symposium on Solid and Physical Modeling*, 17–25.

LAI, Y.-K., ZHOU, Q.-Y., HU, S.-M., WALLNER, J., AND POTTMANN, H. 2007. Robust feature classification and editing. *IEEE Trans. Visualization and Computer Graphics 13*, 1, 34–45.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2004. Intelligent mesh scissoring using 3d snakes. In *Proc. Pacific Graphics*, 279–287.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2005. Mesh scissoring with minima rule and part salience. *Computer-Aided Geometric Design 22*, 5, 444–465.

LI, X., WOON, T. W., TAN, T. S., AND HUANG, Z. 2001. Decomposing polygon meshes for interactive applications. In *Proc. ACM Symposium on Interactive 3D Graphics*, 35–42.

LIU, R., AND ZHANG, H. 2004. Segmentation of 3d meshes through spectral clustering. In *Proc. Pacific Graphics*, 298–305.

LIU, R., JAIN, V., AND ZHANG, H. 2006. Subsampling for efficient spectral mesh processing. *Lecture Notes in Computer Science*, 172–184.

MANGAN, A. P., AND WHITAKER, R. T. 1999. Partitioning 3d surface meshes using watershed segmentation. *IEEE Trans. Visualization and Computer Graphics 5*, 4, 308–321.

MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. In *Proc. ACM SIGGRAPH*, 259–263.

MOUNT, D., AND ARYA, S., 2005. ANN: a library for approximate nearest neighbors searching.
url: `http://www.cs.umd.edu/~mount/ann/`.

RENIERS, D., AND TELEA, A. 2007. Skeleton-based hierarchical shape segmentation. In *Proc. Shape Modeling International*, 179–188.

SAPIDIS, N. S., AND BESL, P. J. 1995. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Trans. Graphics 14*, 3, 171–200.

SHAMIR, A., SHAPIRA, L., COHEN-OR, D., AND GOLDENTHAL, R. 2004. Geodesic mean shift. In *Proc. 5th Korea-Israel Conf. Geometric Modeling and Computer Graphics*, 51–56.

SHARF, A., BLUMENKRANTS, M., SHAMIR, A., AND COHEN-OR, D. 2006. Snappaste: an interactive technique for easy mesh composition. *The Visual Computer 22*, 9–11, 835–844.

SHLAFMAN, S., TAL, A., AND KATZ, S. 2002. Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum 21*, 3, 219–229.

SUN, X., ROSIN, P. L., MARTIN, R. R., AND LANGBEIN, F. C. 2007. Random walks for mesh denoising. In *Proc. ACM Symposium on Solid and Physical Modeling*, 11–22.

SURAZHSKY, V., SURAZHSKY, T., KIRSANOV, D., GORTLER, S., AND HOPPE, H. 2005. Fast exact and approximate geodesics on meshes. In *Proc. ACM SIGGRAPH*, 553–560.

TOLEDO, S., CHEN, D., AND ROTKIN, V., 2003. TAUCS: A library of sparse linear solvers, ver. 2.2
url: `http://www.tau.ac.il/~stoledo/taucs/`.

VÁRADY, T., MARTIN, R. R., AND COX, J. 1997. Reverse engineering of geometric models—an introduction. *Computer-Aided Design 29*, 4, 255–268.

VÁRADY, T. 2007. Automatic extraction of surface structures in digital shape reconstruction. *Computer-Aided Design 39*, 5, 379–388.

WITKIN, A., AND HECKBERT, P. 1994. Using particles to sample and control implicit surfaces. In *Proc. ACM SIGGRAPH*, 269–277.

YAMACHI, H., LEE, S., LEE, Y., AND OHTAKE, Y. 2005. Feature sensitive mesh segmentation with mean shift. In *Proc. Shape Modeling International*, 236–243.

YANG, Y.-L., LAI, Y.-K., HU, S.-M., AND POTTMANN, H. 2006. Robust principal curvatures on multiple scales. In *Proc. Eurographics Symposium on Geometry Processing*, 223–226.

ZUCKERBERGER, E., TAL, A., AND SHLAFMAN, S. 2002. Polyhedral surface decomposition with applications. *Computers & Graphics 26*, 5, 733–743.