Rapid and Effective Segmentation of 3D Models using Random Walks

Yu-Kun Lai [a] Shi-Min Hu [a] Ralph R. Martin [b] Paul L. Rosin [b]

[a] *Tsinghua University, Beijing, China*

[b] *Cardiff University, Wales, UK*

Abstract

3D models are now widely available for use in various applications. The demand for automatic model analysis and understanding is ever increasing. Model segmentation is an important step towards model understanding, and acts as a useful tool for different model 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 model segmentation. This method is extremely efficient, and scales almost linearly with the number of faces, and the number of regions. For models of moderate size, interactive performance is achieved with commodity PCs. We demonstrate that this method can be applied to both triangle meshes and point cloud data. It is easy-to-implement, robust to noise in the model, and yields results suitable for downstream applications for both graphical and engineering models.

Key words: model segmentation, random walks, interactive

1 Introduction

With the development of 3D acquisition techniques, 3D 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 models have not matched the growth in availability of models. The task of model *segmentation* is to decompose a model into a set of disjoint pieces whose union corresponds to the original model. To be useful,

Email addresses: `laiyk03@mails.tsinghua.edu.cn` (Yu-Kun Lai), `shimin@tsinghua.edu.cn` (Shi-Min Hu), `ralph@cs.cf.ac.uk` (Ralph R. Martin), `Paul.Rosin@cs.cf.ac.uk` (Paul L. Rosin).

segmentation must decompose a model into intuitively satisfying 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).

Model segmentation is an important step towards model analysis and understanding. A variety of different applications would benefit from initially dividing the model into regions using an efficient and reliable 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 [35]. In computer graphics, segmentation can be applied in various applications, including mesh simplification [40], collision detection [18], morphing [30,40] and skeleton-driven animation [12].

In general, to be useful, model segmentation should produce results in accordance with cognitive science. As pointed out by Hoffmann [9,10], 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 clues 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, for reverse engineering applications, different surfaces may be separated along sharp edges, which may be convex or concave, or even along smooth edges—different criteria lead to useful segmentations for differing applications.

Recently, Grady [8] 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 each other pixel, 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 each non-seed pixel.

Our work is an extension of the random walks method to the particular problem of 3D model segmentation; previous work has already considered the use of random walks for solving the different problem of 3D mesh *denoising* [31]. 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. The method can also be used for direct segmentation of point cloud data.

As well as a method based on user selection of seeds, we give a generalization of this method to automatic mesh segmentation. Two approaches are discussed.

For coarse-scale segmentation, a few seeds are distributed as far apart as possible, resulting in a segmentation of the model into a few pieces. For fine-scale segmentation, seeds are first placed automatically (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. Our method can also be used for hierarchical segmentation of models, mimicking the way people think.

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; in the latter case, the appropriate 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 the differing kinds of segmentation expected in such cases.

This paper is an extended version of [13]. In particular, we additionally consider (i) how to segment point cloud data, (ii) hierarchical segmentation, and (iii) the robustness of the method in the presence of noise and deformation.

Section 2 briefly reviews related work. Interactive segmentation of both triangle meshes and point cloud data based on random walks is presented in Section 3 and extension to automatic mesh segmentation is presented in Section 4. 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 up-to-date reviews and comparisons of different methods can be found in [1,27].

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. [2]). 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. [30] use k-means clustering to segment the models into meaningful pieces. Katz [12] 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 [14] 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 [12,11,20] or remeshing [14] is typically used. Spectral clustering has also been used [20] with good results, although e.g. the Nyström approximation method may be needed to overcome the performance issues associated with this approach [19].

Unsupervised clustering techniques like the mean shift method can also be applied to mesh segmentation. Shamir [28] extended mean shift analysis to mesh models based on use of a local parameterization method. Later, Yamauchi [38] 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 [21] applied the watershed algorithm to triangle meshes. Li [18] 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 [25] used curve skeletons for hierarchical mesh segmentation. Katz [11] 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 [22] 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.

Segmentation of engineering objects, especially for the purpose of reverse engineering, has also been widely considered. Much of this work deals with point clouds directly instead of triangle meshes due to their wide availability. Sapidis and Besl [26] proposed a method to construct polynomial surfaces from point cloud data, using region growing for segmentation. Benko and Várady [3] proposed a method to directly segment point cloud data for engineering objects based on a series of top-down recursive tests using normal information. Gelfand and Guibas [7] proposed the use of slippage analysis and multi-pass region growing to segment different regions based on slippage signatures. Edelsbrunner [5] discussed how to segment meshes using piecewise linear Morse-Smale theory. This idea has been extended to suit the needs for producing CAD-like segmentation results [36].

Some 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 [16,17]. Funkhouser [6] 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 [29].

Our method is different to any of the above in that it is based on a random walk paradigm. This formulation leads to the need to solve a sparse linear system, which is very efficient. Unlike most interactive methods, interaction is via user choice of a set of seed faces, which is much easier than specifying an approximate cutting boundary. For applications where automatic methods are preferable, 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 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 [8], but due to the differences of source data and aims, certain issues must be resolved.

We assume for now that the given models are triangular meshes; we return to the issue of segmenting point clouds directly later.. 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 should be placed so that one seed lies within each of the final regions desired. Let these seed faces be $s_1, \ldots, s_n$. 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; we discuss shortly how these probabilities are chosen. 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 (if the 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 [8]; this system is sparse as each row of the matrix contains at most 4 non-zero entries, by Eqn 2. Following the argument in [8], the matrix A is positive semi-definite, and the solution to this linear system is uniquely determined. Note that the random walk model described above is in essence equivalent to an electric network model as the distribution of electric potentials at each face, where the probability of moving to a neighboring face corresponds to the conductance. See [4] 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 is guaranteed that each region produced by this segmentation process is contiguous [8].

With the basic framework given by the algorithm above, two significant issues remain: to determine appropriate probabilities for stepping from face to face, and choice of optional preprocessing and postprocessing steps to further improve the results.

3.1 Probability computation

The 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 depend on the purposes and expectations of the resulting mesh segmentation. In the following, we 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:

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

where 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 rather smaller (e.g. $\eta = 0.2$) for convex edges, according to the minima rule.

To handle variations in the dihedral distribution, we normalize d_1 by its average over all mesh 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 variations in differences maps to variations in probability. In our experiments, we have found $\sigma = 1.0$ works well for most 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 aims from segmentation of graphical models. We usually want to segment a mesh into pieces such that can each be fitted with a single analytical surface [35]. 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 which assess 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 [39]. The method basically relies on a covariance analysis of the intersection volume between a ball of radius r and the interior part of the model, locally. 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 using:

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

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. Probabilities are again defined as in Eqn. 6, but with d^* in place of d. The method works well for separating certain smoothly touching regions, as illustrated by the example results in Fig. 1.

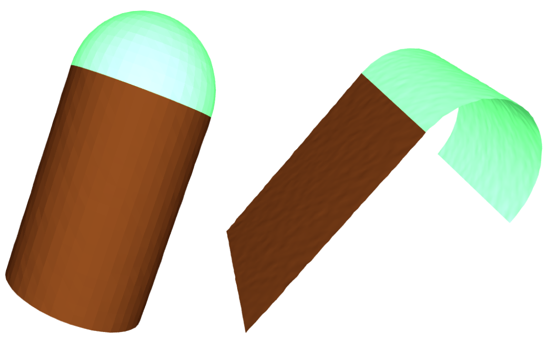

Fig. 1. Segmentation of CAD models without sharp edges.

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 [15] 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, centre)). Projection is done quickly using the Approximate Nearest Neighbors Library [23]. After projection, we find the geodesic path across the mesh between adjacent projected vertices [32], 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.

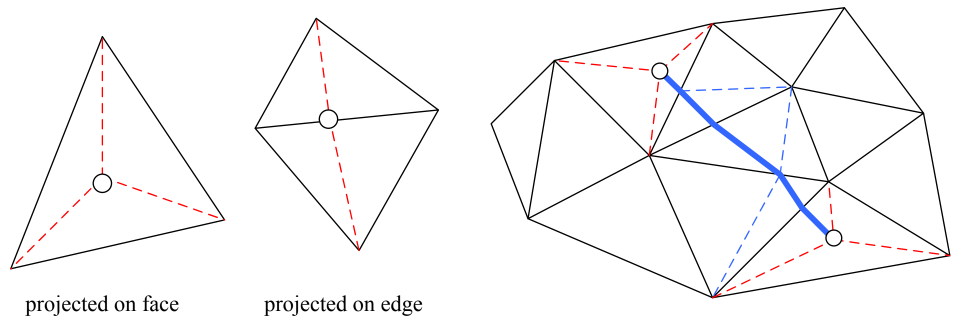

Fig. 2. Projection and local update of the input model. Left, centre: mapping smoothed boundary points onto the surface. Right: mapping smoothed boundary paths onto the surface.

3.3 Direct segmentation of point cloud data

Point cloud data is another representation of 3D geometry. If the data is provided in this form, it may be preferable to avoid explicit triangulation which can be computationally expensive and error-prone. The random walks paradigm can also be applied to direct segmentation of an unstructured point cloud $P = (\mathbf{p}_i \in \mathbb{R}^3)$. Segmentation of point cloud data partitions P into a disjoint set of components. For interactive segmentation, users may select a few seed points, and the point cloud is segmented accordingly. Unlike a triangulated mesh, a point cloud does not have well-defined local connectivity. Thus, as in [24], we use k-nearest neighbors $N(\mathbf{p}_i)$ of each point $\mathbf{p}_i$ to approximate the topological neighborhood. The normal vector $\mathbf{N}_i$ at each point $\mathbf{P}_i$ can be estimated by a local covariance analysis. For each point $\mathbf{p}_i$, the centroid $\bar{\mathbf{p}}_i$ of the local neighborhood can be computed as

$$\bar{\mathbf{p}}_i = \frac{\sum_{\mathbf{p}_j \in N(\mathbf{p}_i)} \mathbf{p}_j}{|N(\mathbf{p}_i)|}, \tag{9}$$

where $|\cdot|$ denotes the number of elements in the set. The covariance matrix is a 3×3 matrix:

$$C_i = \sum_{\mathbf{p}_j \in N(\mathbf{p}_i)} (\mathbf{p}_j - \bar{\mathbf{p}}_i) \cdot (\mathbf{p}_j - \bar{\mathbf{p}}_i)^T. \tag{10}$$

The normal $\mathbf{n}_i$ is estimated as the eigenvector of C_i corresponding to the smallest eigenvalue. Since the eigenvectors have a directional ambiguity, we still need to decide the outward normal at each point. For orientable surfaces, this can be achieved by first assigning the outward direction at a point with one largest coordinate, then propagating directions to neighboring points based on local consistency. This process terminates when all the points have been visited.

To segment point cloud data with random walks, a graph is first constructed. Each point $\mathbf{p}_i$ is connected to every point $\mathbf{p}_j \in N(\mathbf{p}_i)$ (i.e. within the neighborhood). Note that the neighborhood contains the k closest points where k is fixed, guaranteeing that this graph has size proportional to the number of points. Unlike the mesh case, where typically, long thin triangles are avoided by the mesh construction algorithm, for point cloud neighbors, the distance to different points in $N(\mathbf{p}_i)$ may vary significantly, and better results are obtained by incorporating both position and normal variation in the probability computation. Let us denote $d'_1(\mathbf{p}_i, \mathbf{p}_j) = ||\mathbf{p}_i - \mathbf{p}_j||^2$ and $d'_2(\mathbf{p}_i, \mathbf{p}_j) = \frac{\eta}{2} ||\mathbf{n}_i - \mathbf{n}_j||^2$; η is chosen as before. A simple heuristic is used to verify if the connection between neighboring point $(\mathbf{p}_j, \mathbf{n}_j)$ and the center vertex $(\mathbf{p}_i, \mathbf{n}_i)$ is convex. Consider the plane passing through $\mathbf{p}_j$, parallel to the direction $\mathbf{n}_i$, and farthest away from $\mathbf{p}_i$. The connection is considered to be convex if $\mathbf{n}_j$ lies on the opposite side of the plane to $\mathbf{p}_i$, i.e.

$$((\mathbf{p}_j - \mathbf{p}_i) - ((\mathbf{p}_j - \mathbf{p}_i) \cdot \mathbf{n}_i)\, \mathbf{n}_i) \cdot \mathbf{n}_j \geq 0. \tag{11}$$

We have found that this simple heuristic works well in practice.

To adapt to the variation of point sampling density (i.e. d'_1 may have a significant global variation), we compute the average distance $\bar{d_1}(\mathbf{p}_i)'$ in the local neighborhood $N(\mathbf{p}_i)$ while still computing the average distance $\bar{d'_2}$ over the whole model. The probability of moving from $\mathbf{p}_i$ to $\mathbf{p}_j$ is computed as

$$p_{i,j} = \exp\left\{-\frac{d_1'(\mathbf{p}_i, \mathbf{p}_j)}{\sigma_1 \bar{d}_1'(\mathbf{p}_i)}\right\} \cdot \exp\left\{-\frac{d_2'(\mathbf{p}_i, \mathbf{p}_j)}{\sigma_2 \bar{d}_2'}\right\}. \tag{12}$$

This formula bears can be compared to the idea used in bilateral filtering where two contributing factors are combined [34]. $p_{i,j}$ is then normalized to sum to one over $N(\mathbf{p}_i)$. σ_1 and σ_2 control how variations of differences of positions and normals map to variations in probability. We have found $\sigma_1 = \sigma_2 = 1.0$ to be suitable for most examples. An example of direct segmentation of point cloud data is given in Fig. 3, in which a dinosaur model with $56,194$ points is segmented into 7 components.

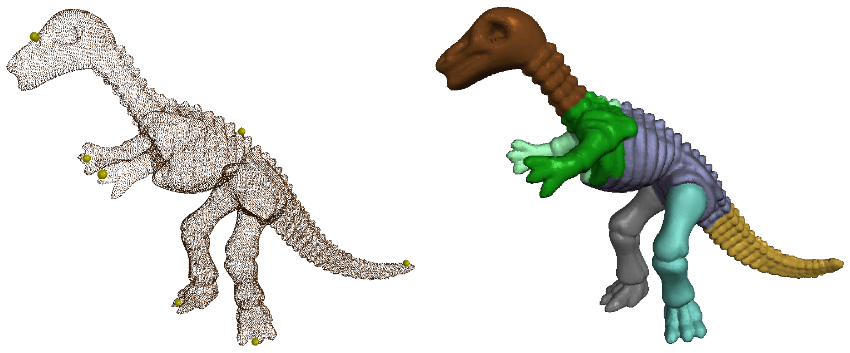

Fig. 3. Left: example of point cloud data. Right: corresponding direct segmentation result.

4 Automatic Segmentation

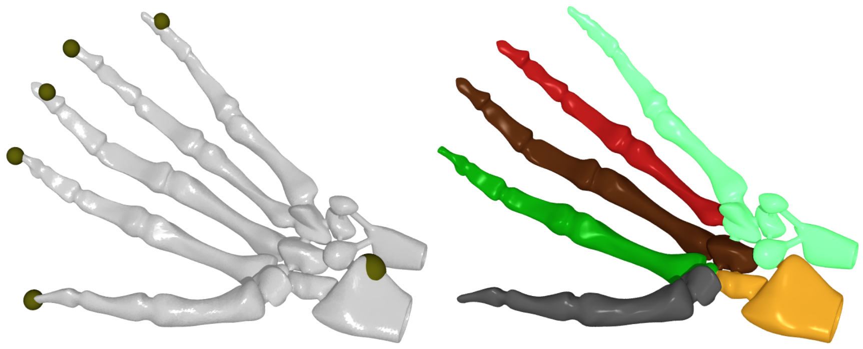

Fig. 4. Left: example of coarse seeding. Right: corresponding segmentation result.

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 takes advantage of our experimental observation that random walk segmentation results are not sensitive to the exact location of the seeds (as we demonstrate 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 face furthest away, in terms of geodesic distance, from the face closest to the centroid of all faces (any such face may be chosen if there are multiple faces equidistant from the centroid). 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

$$s_{n+1} = \arg\max_{f_k \in F} \left\{ \min_{i=1,\ldots,n} D(f_k, s_i) \right\}, \tag{13}$$

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. 4 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 as many parts are separated as possible. For example, given the skeleton example shown in Fig. 4, 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, while giving higher priority to protrusions and regions containing features. Feature sensitive sampling (the first phase of the feature sensitive remeshing method proposed in [15]) suits this need well. Compared with coarse seeding with a large number of seeds, this scheme tends to produce more uniform distribution of initial seeds. The method basically distributes particles over the model optimizing some spring-like energy [37]. 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

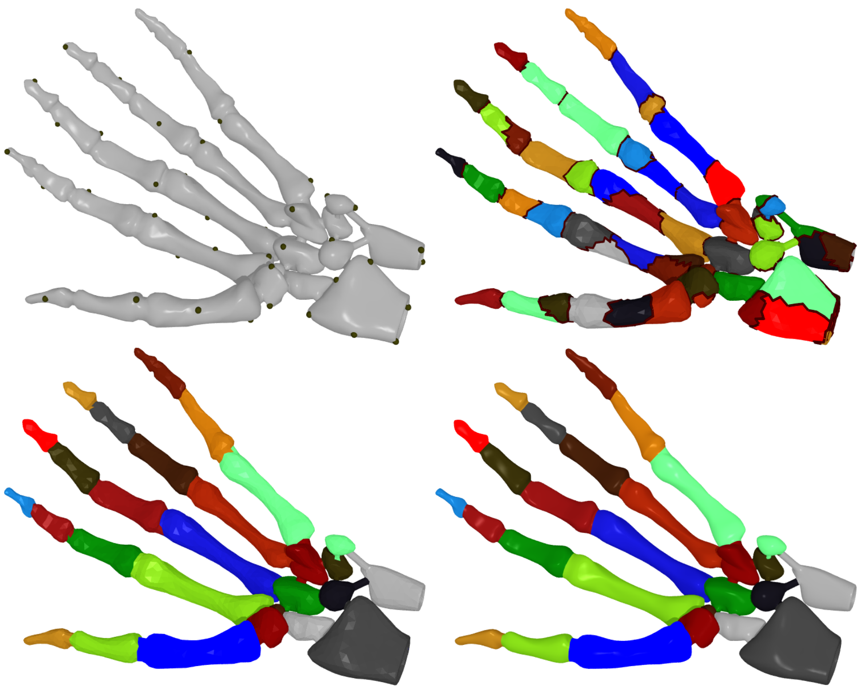

Fig. 5. Example of fine-scale seeding. From top left to bottom right: input model with automatic seed selection; initial (over-) segmented results; result after merging; final result after boundary smoothing and mapping.

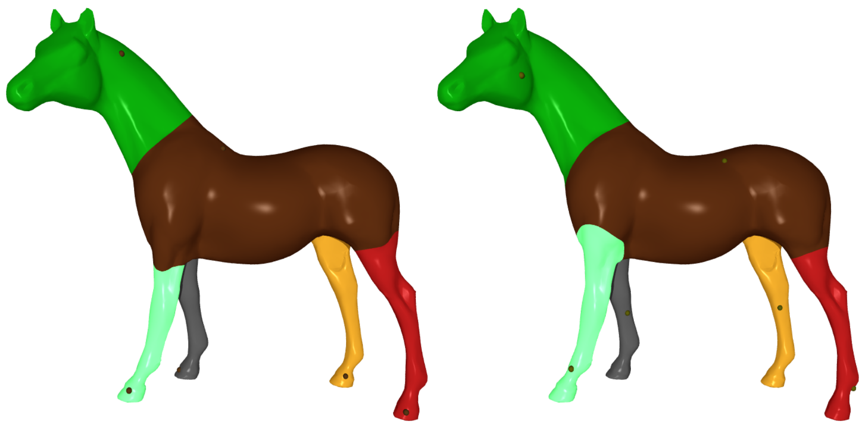

Fig. 6. Segmentation of horse with varying seed locations. Balls represent seed locations.

then define the relative merging cost $c_{i,j}$ as:

$$c_{i,j} = \frac{D_{\partial S_i \cap \partial S_j} / L_{\partial S_i \cap \partial S_j}}{D_{\partial S_i \cup \partial S_j} / L_{\partial S_i \cup \partial S_j}}. \tag{14}$$

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

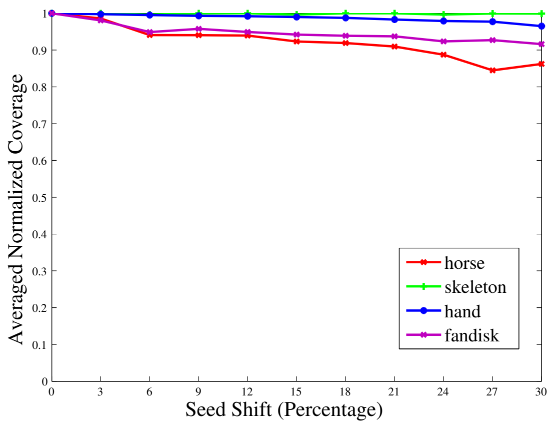

Fig. 7. Stability test results of averaged normalized coverage.

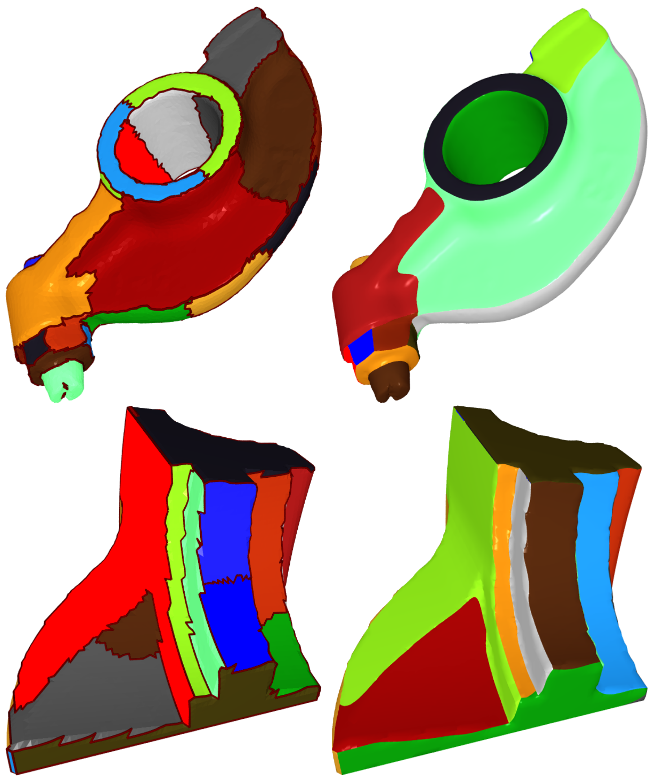

Fig. 8. Segmentation of engineering objects: rocker arm and fan disk. Left: initial segmentation with fine-scale seeding. Right: results after merging and smoothing.

either when a significant increase in $c_{i,j}$ occurs for the current pair, or when we have reached the 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. 5. The initial number of seeds was 60 and the number of segments after merging was 30.

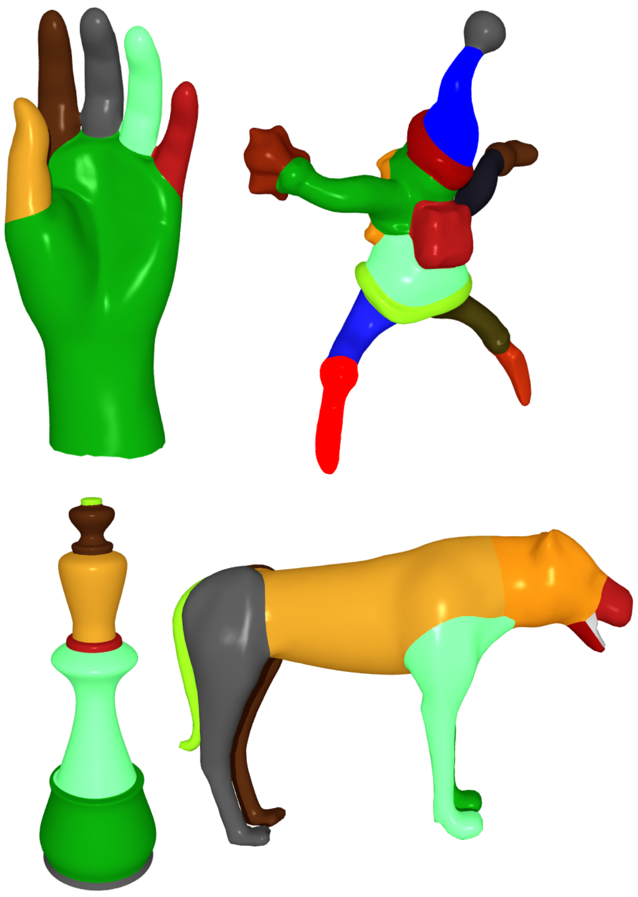

Fig. 9. Examples of mesh segmentation for various graphical models.

5 Experimental Results

5.1 Insensitivity to exact seed locations

Our method is in general insensitive to the exact location of seeds. Fig. 6 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 $3\%, 6\%, 9\%, \dots, 30\%$ of the size of the model. In each test we computed the percentage of faces with the same labels found when using the original seed positions (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. 7, for models like the horse example in Fig. 6 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. 4, 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. For a moderate example like the hand model in Fig. 9, the averaged normalized coverage is above 96.5% for seed shifts of up to 30%.

5.2 Segmentation of graphical and engineering models

Our method can also be applied to meshes representing engineering objects. Fig. 8 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 was 40 for both models. The numbers of segments after merging were 20 for the rocker arm and 25 for the fan disk, respectively. Note that significant normal noise exists near the 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 phases, 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 lies between the values for the hand and the horse examples, being above 91% for shifts up to 30%.

We have also tested our method on various other graphical models, a selection of which are shown in Fig. 9. The hand, Santa, chessman, and cheetah models were segmented into 6, 17, 7 and 10 pieces, respectively. Generally, intuitively reasonable and pleasing segmentation results are produced for such examples.

5.3 Robust segmentation of models with noise

Since our method is based on a probability model, it is in general robust to small-scale noise. For relatively larger-scale additive noise, we may opt to use a simple normal filter that assigns the averaged normal direction of adjacent faces to the center face, and use the filtered normals in probability estimation. By using such normal filtering, segmentation results tend to be more robust to noise, and meanwhile the segmentation results for models without noise are

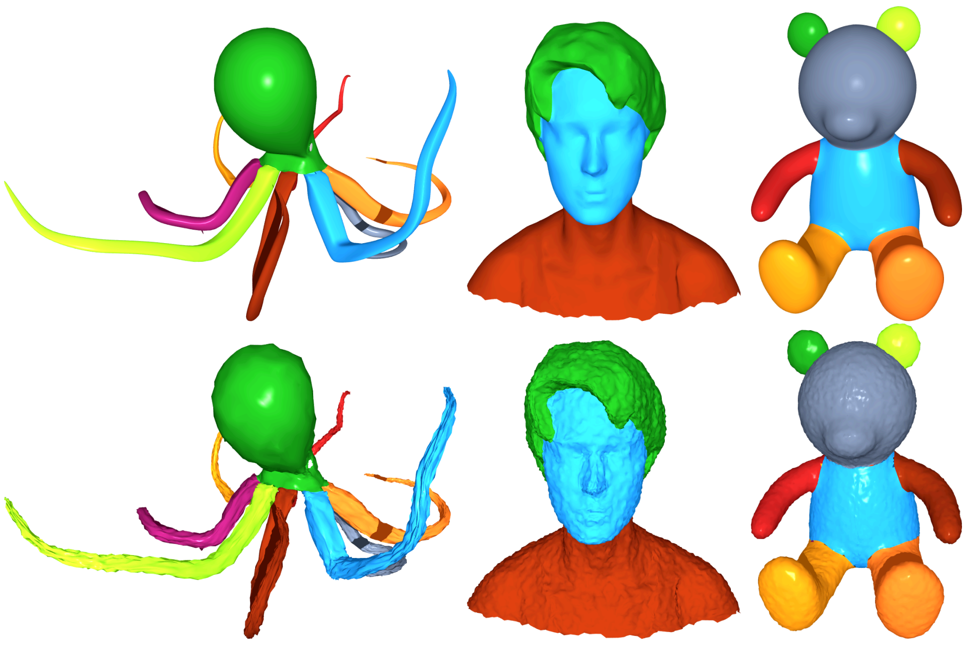

Fig. 10. Examples of segmentation of noisy models. Consistent segmentations are obtained for models without (top) and with (bottom) additive noise.

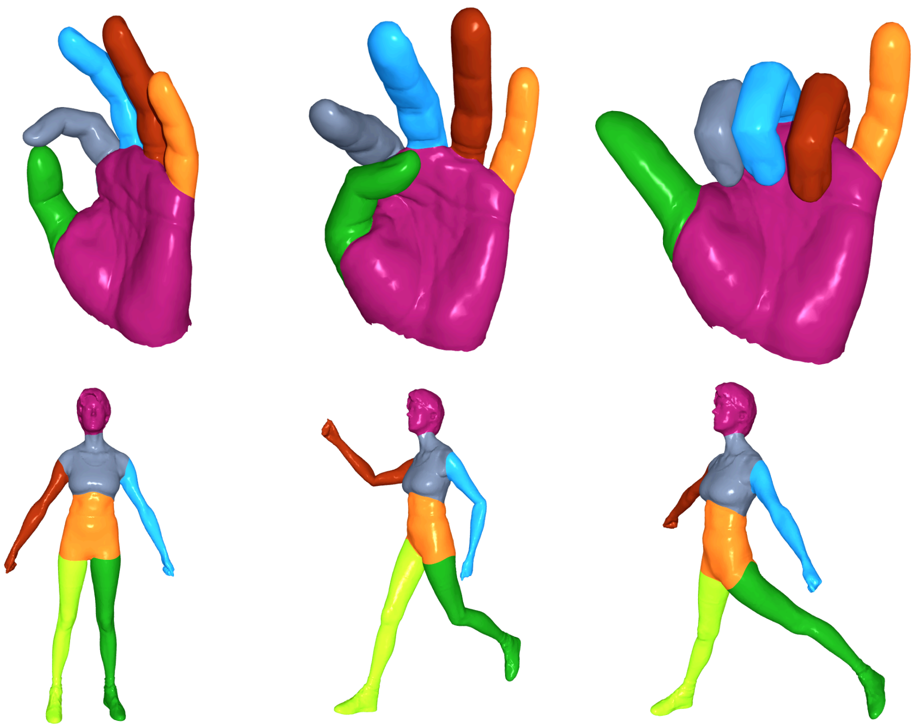

Fig. 11. Examples of consistent segmentation of deformed models.

almost unaffected. Fig. 10 shows that consistent segmentation results were obtained for the original models and models with significant amount of additive noise.

5.4 Consistent segmentation of deformed models

Deformed models usually preserve significant features (but the strengths of features may vary). Thus it can be expected that consistent segmentation results would be obtained for a series of deformed models. Fig. 11 shows two models in a variety of poses, for which consistent segmentation results were obtained even after significant deformation.

5.5 Performance

Compared with state-of-the-art methods, our method is very efficient both in time and memory usage. A detailed comparison of timings for the hand skeleton model remeshed to $10K$, $15K$, $20K$, $30K$ and $40K$ triangles using an implementation of [14] 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 [12,14,20] is dominated by pair-wise distance computations, 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 [11] 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 MATLAB's direct solver (the \ operator) throughout the paper, though sparse linear solver libraries like TAUCS [33] 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. Although the implementation details of MatLab are not publicly available, techniques such as multigrid methods are often able to achieve almost linear time complexity in the number of unknowns for this kind of sparse linear system. Time also increases more or less linearly with the 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. In summary, the overall complexity with respect to the number of faces m and the number of seeds n is approximately $O(mn)$. Segmentation of point clouds has a similar time complexity to the mesh case, where m now represents the number of points.

Note that models with $40K$ triangles or more cannot be processed by the method in [14] 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

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

No. of Triangles	Clustering method [14]	Current method	per 1K triangles
$10K$	129s	0.34s	0.034s
$15K$	303s	0.47s	0.031s
$20K$	532s	0.64s	0.032s
$30K$	1359s	1.00s	0.033s
$40K$	n/a	1.34s	0.034s

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

No. of seeds	6	12	24	48
Time	1.3s	2.1s	3.4s	6.5s

interactive time, suitable for interactive applications that require immediate feedback.

5.6 Hierarchical segmentation

Hierarchical segmentation was first introduced by [12]. It decomposes a given model into several levels of segmentation, in a similar way to how people consider the composition of an object as a natural hierarchy (a human body has arms, arms have hands, hands have fingers, and so on). Hierarchical segmentation may also be helpful to improve the segmentation results in certain cases, since coarser levels of segmentation which capture more significant components may then be used as constraints on the finer levels of segmentation. Once the hierarchy has been computed, a user-controlled slider may be provided to allow the user to browse and find the granularity of segmentation appropriate to his task, as suggested by [2].

Hierarchical segmentation may be obtained by recursively segmenting the regions obtained by coarser levels of segmentation to obtain finer levels of segmentation. Fig. 12 gives hierarchical segmentations of the dinopet ($26,640$ triangles) and eagle ($29,232$ triangles) models into three levels. Coarse seeding is used in each subregion of each level of segmentation to decide the locations of seeds, while the number of seeds in each subregion may be specified by the user. No simplification or remeshing is required for this segmentation. Note that unlike [12], our method can directly deal with larger models (e.g. with $20K$ or more triangles) efficiently. By using a higher resolution, original model, we not only skip the unnecessary step of simplification but also keep sufficient resolution when the segmentation goes down to finer levels.

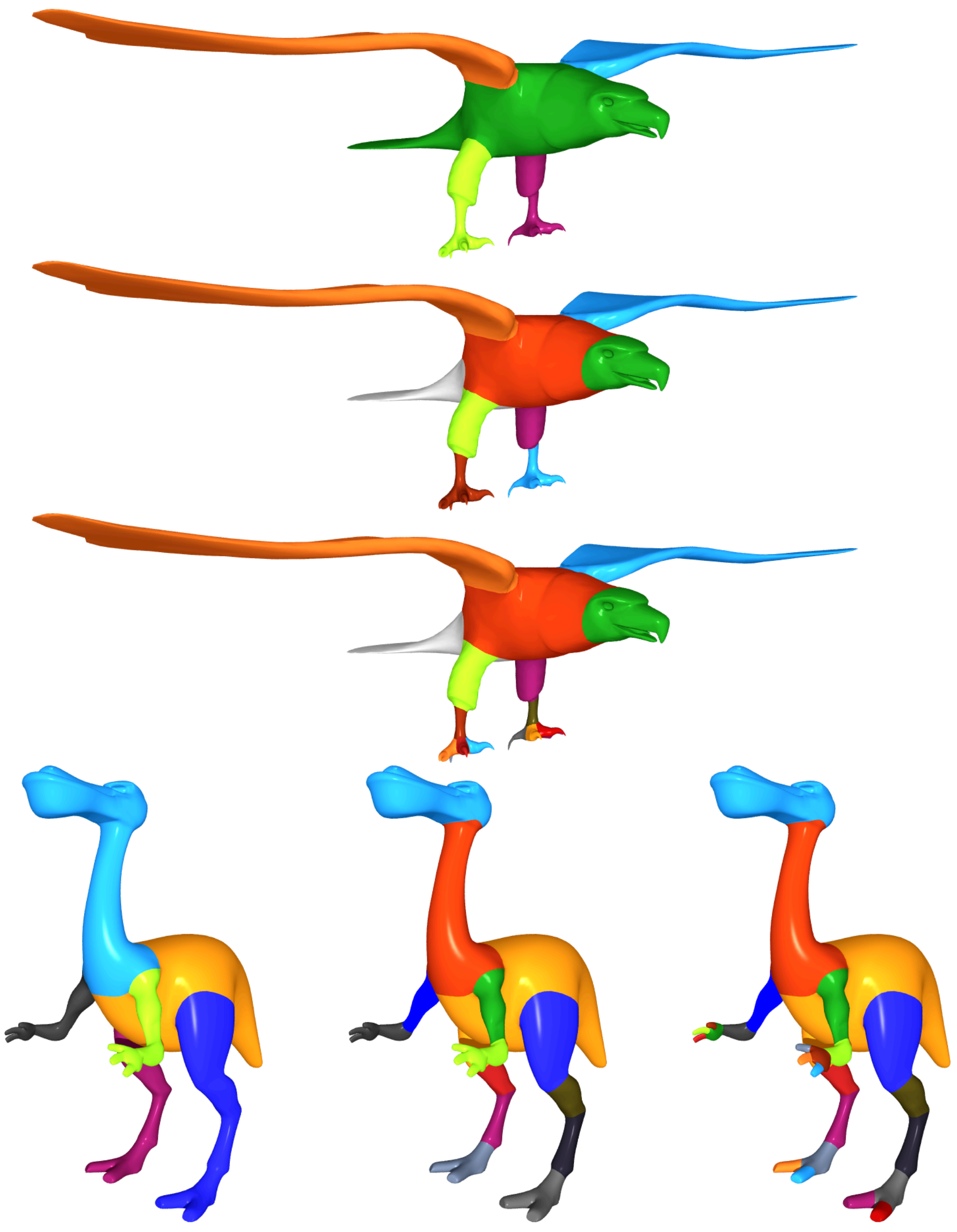

Fig. 12. Examples of hierarchical segmentation.

6 Conclusions

In this paper, we have presented both an interactive and an automatic method of model segmentation based on random walks. We have demonstrated the effectiveness of this method, with both 'natural' graphical models and engineering object models. 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. The method is generally robust to the movement of seeds and the presence of noise. In the future, we may explore the segmentation of models with different geometric textures.

Acknowledgment

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

References

[1] M. Attene, B. Falcidieno, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation—a comparative study. In *Proc. IEEE Int'l Conference on Shape Modeling and Applications*, pages 7–18, 2006.

[2] M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193, 2006.

[3] P. Benko and T. Várady. Direct segmentation of smooth, multiple point regions. In *Proc. Geometric Modeling and Processing*, pages 169–178, 2002.

[4] P. G. Doyle and J. L. Snell. *Random walks and electric networks.* Number 22 in Carus Mathematical Monographs. The Mathematical Association of America, 1984.

[5] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical morse-smale complexes for piecewise linear 2-manifolds. *Discrete Computational Geometry*, 30(1):87–107, 2003.

[6] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by examples. In *Proc. ACM SIGGRAPH*, pages 652–663, 2004.

[7] N. Gelfand and L. J. Guibas. Shape segmentation using local slippage analysis. In *Proc. Eurographics Symposium on Geometry Processing*, pages 219–228, 2004.

[8] L. Grady. Random walks for image segmentation. *IEEE Trans. Pattern Analysis and Machine Inbtelligence*, 28(11):1768–1783, 2006.

[9] D. Hoffmann and W. Richards. Parts of recognition. *Cognition*, 18:65–96, 1984.

[10] D. Hoffmann and M. Singh. Salience of visual parts. *Cognition*, 63:29–78, 1997.

[11] S. Katz, G. Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8–10):865–875, 2005.

[12] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graphics*, 22(3):954–961, 2003.

[13] Y.-K. Lai, S.-M. Hu, R. R. Martin, and P. L. Rosin. Fast mesh segmentation using random walks. In *Proc. ACM Symposium on Solid and Physical Modeling*, pages 183–191, 2008.

[14] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, and R. R. Martin. Feature sensitive mesh segmentation. In *Proc. ACM Symposium on Solid and Physical Modeling*, pages 17–25, 2006.

[15] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, J. Wallner, and H. Pottmann. Robust feature classification and editing. *IEEE Trans. Visualization and Computer Graphics*, 13(1):34–45, 2007.

[16] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Intelligent mesh scissoring using 3d snakes. In *Proc. Pacific Graphics*, pages 279–287, 2004.

[17] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Mesh scissoring with minima rule and part salience. *Computer-Aided Geometric Design*, 22(5):444–465, 2005.

[18] X. Li, T. W. Woon, T. S. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proc. ACM Symposium on Interactive 3D Graphics*, pages 35–42, 2001.

[19] R. Liu, V. Jain, and H. Zhang. Subsampling for efficient spectral mesh processing. *Lecture Notes in Computer Science*, pages 172–184, 2006.

[20] R. Liu and H. Zhang. Segmentation of 3d meshes through spectral clustering. In *Proc. Pacific Graphics*, pages 298–305, 2004.

[21] A. P. Mangan and R. T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Trans. Visualization and Computer Graphics*, 5(4):308–321, 1999.

[22] J. Mitani and H. Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. In *Proc. ACM SIGGRAPH*, pages 259–263, 2004.

[23] D. Mount and S. Arya. ANN: a library for approximate nearest neighbors searching.
url: `http://www.cs.umd.edu/~mount/ann/`, 2005.

[24] M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification and point-sampled surfaces. In *Proc. IEEE Visualization*, pages 163–170, 2002.

[25] D. Reniers and A. Telea. Skeleton-based hierarchical shape segmentation. In *Proc. Shape Modeling International*, pages 179–188, 2007.

[26] N. S. Sapidis and P. J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Trans. Graphics*, 14(3):171–200, 1995.

[27] A. Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 2008. doi:10.1111/j.1467-8659.2007.01103.x.

[28] A. Shamir, L. Shapira, D. Cohen-Or, and R. Goldenthal. Geodesic mean shift. In *Proc. 5th Korea-Israel Conf. Geometric Modeling and Computer Graphics*, pages 51–56, 2004.

[29] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or. Snappaste: an interactive technique for easy mesh composition. *The Visual Computer*, 22(9–11):835–844, 2006.

[30] S. Shlafman, A. Tal, and S. Katz. Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, 21(3):219–229, 2002.

[31] X. Sun, P. L. Rosin, R. R. Martin, and F. C. Langbein. Random walks for mesh denoising. In *Proc. ACM Symposium on Solid and Physical Modeling*, pages 11–22, 2007.

[32] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *Proc. ACM SIGGRAPH*, pages 553–560, 2005.

[33] S. Toledo, D. Chen, and V. Rotkin. TAUCS: A library of sparse linear solvers, ver. 2.2
url: `http://www.tau.ac.il/~stoledo/taucs/`, 2003.

[34] C. Tomasi and R. Manduchi. Bilateral filter for gray and color images. In *Proc. IEEE Int'l Conf. on Computer Vision*, pages 839–846.

[35] T. Várady, R. R. Martin, and J. Cox. Reverse engineering of geometric models—an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[36] Tamás Várady. Automatic extraction of surface structures in digital shape reconstruction. *Computer-Aided Design*, 39(5):379–388, 2007.

[37] A. Witkin and P. Heckbert. Using partickles to sample and control implicit surfaces. In *Proc. ACM SIGGRAPH*, pages 269–277, 1994.

[38] H. Yamachi, S. Lee, Y. Lee, and Y. Ohtake. Feature sensitive mesh segmentation with mean shift. In *Proc. Shape Modeling International*, pages 236–243, 2005.

[39] Y.-L. Yang, Y.-K. Lai, S.-M. Hu, and H. Pottmann. Robust principal curvatures on multiple scales. In *Proc. Eurographics Symposium on Geometry Processing*, pages 223–226, 2006.

[40] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Computers & Graphics*, 26(5):733–743, 2002.