Interactive Mesh Cutting Using Constrained Random Walks

Juyong Zhang, Jianmin Zheng, and Jianfei Cai, *Senior Member, IEEE*

Abstract—This paper considers the problem of interactively finding the cutting contour to extract components from an existing mesh. First, we propose a constrained random walks algorithm that can add constraints to the random walks procedure and thus allows for a variety of intuitive user inputs. Second, we design an optimization process that uses the shortest graph path to derive a nice cut contour. Then a new mesh cutting algorithm is developed based on the constrained random walks plus the optimization process. Within the same computational framework, the new algorithm provides a novel user interface for interactive mesh cutting that supports three typical user inputs and also their combinations: 1) foreground/background seed inputs: the user draws strokes specifying seeds for "foreground" (i.e., the part to be cut out) and "background" (i.e., the rest); 2) soft constraint inputs: the user draws strokes on the mesh indicating the region which the cuts should be made nearby; and 3) hard constraint inputs: the marks which the cutting contour must pass. The algorithm uses feature sensitive metrics that are based on surface geometric properties and cognitive theory. The integration of the constrained random walks algorithm, the optimization process, the feature sensitive metrics and the varieties of user inputs makes the algorithm intuitive, flexible, and effective as well. The experimental examples show that the proposed cutting method is fast, reliable, and capable of producing good results reflecting user intention and geometric attributes.

Index Terms—Computational geometry and object modeling, interaction techniques, geometric algorithms.

1 Introduction

Mesh cutting refers to the process of extracting subparts or components from an existing mesh. It is also known variously as mesh segmentation, mesh partitioning, and mesh scissoring. The cutting operation is of great practical importance in mesh editing [1], [2]. It has been proved useful in many mesh related processing and applications, examples of which include modeling by examples, 3D morphing, parameterization, texture mapping, and shape analysis and understanding.

This paper considers the problem of how to interactively cut out a meaningful sub-mesh from its underlying mesh, which is consistent with user intention, geometric mesh attributes and human shape perception. Human perception and the concept of "meaningful" are content dependent. It is difficult to compute a meaningful partition automatically. Segmentation involving a little user interaction indicating user's interested part is therefore increasingly of interest. Our goal is to develop intuitive and intelligent mesh cutout algorithms or tools that allow users to interactively specify the cutting contour reflecting user intention, geometric features and human perception via flexible and less tedious interactions, and provide instant visual feedback.

1.1 Related work

A lot of research has been done on mesh segmentation in literature. To partition a mesh into meaningful parts, many methods iteratively cluster similar mesh elements or components and then refine the border between the parts to find the segmentation. Examples are $k-$means clustering [3], fuzzy clustering [4], and spectral clustering [5], [6]. Other methods are also proposed, including watershed segmentation [7], edge contraction and space sweeping based decomposition [8], spectral embedding and contour analysis based recursive bisection [6], and hierarchical pose-invariant mesh segmentation [9]. Besides static models, there are also some studies investigating how to segment animating meshes [10]–[12]. Comprehensive references and comparison can be found in the excellent survey papers [1] and [13]. A benchmark for evaluation of 3D mesh segmentation algorithms is provided in [14].

Recently, interactive mesh segmentation, which allows the user to affect or instruct the segmentation, has become popular in computer graphics. The approaches for interactive segmentation of 3D meshes can roughly be classified into three categories:

1) The user is asked to specify a few points on the desired cutting contour and the cut is then accomplished by finding the shortest paths between them [15]–[17]. The methods of this category target the cutting contours and are able to support cuts of arbitrary shape, but they require great care when specifying the points and the points are usually specified in order.

2) The user is asked to provide an initial area that is "close" to the desired cutting contour or that the cut should be made within. The geometric snake (or active contour) and mesh scissoring algorithms [2], [18], [19] evolve the initial area to or find the desired cutting contour which is "close" to the initial area. The minimum ratio cycle (MRC) algorithm discretely finds the optimal contour within a prescribed search domain, which has

the minimal ratio between a contour energy and the length of the contour [20]. The intelligent scissoring [21] applies a variant of Dijkstra's algorithm to find the cutting contour that goes within the initial area. [22] applies the Graph Cut algorithm to find the cutting contour. All these methods allow the user to specify the initial area less precisely. However, as pointed out in [2], [23], the generated contour may not respect the user's intention if the stroke is too short or the back of mesh is too complex. The methods may converge to a local minimum and are hard to control. Graph Cut has the "small cut" behavior. That is, it may return very small segmentations as a result of a small number of seeds.

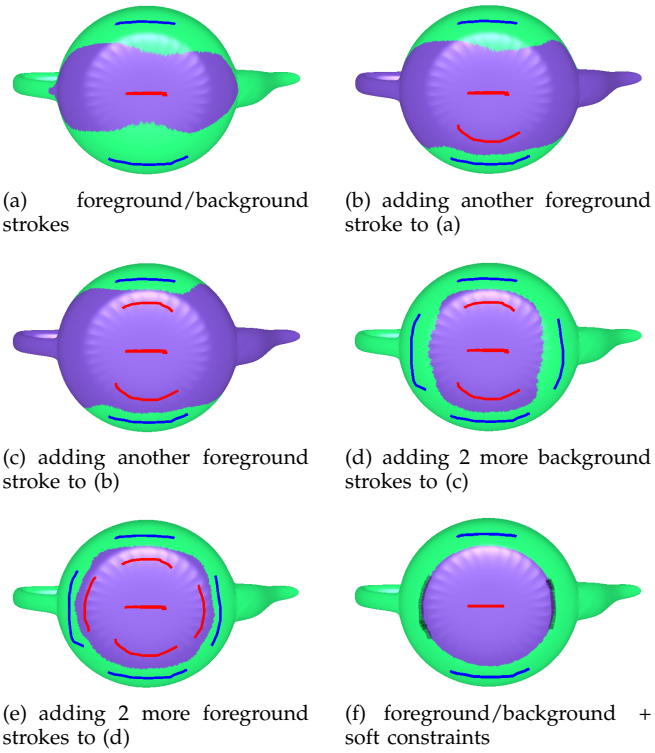
3) The user is asked to provide an initial labeling of some vertices as belonging to the desired part to be segmented (foreground) or to the rest (background), and then the algorithm completes the labeling for all unlabeled vertices. The easy mesh cutting [24] belongs to this category and it starts with different seed vertices and grows several sub-meshes according to an improved isophotic metric incrementally. The "WYSIWYG" mesh decomposition [25] requires the user to specify the feature points and then assigns each face of the mesh to a certain partition based on the distance to that feature point. Similar approaches are also widely used for image segmentation. For example, with a set of given user-defined seeds, for an unseeded pixel, the random walks algorithm [26] determines the probability that a random walk starting at that pixel first reaches each particular seed and then segmentation is formed by assigning the label for which the greatest probability is calculated. The algorithm has been proved very efficient. Since the random walks algorithm is formulated on a graph, it is possible to extend the application of the algorithm to surface meshes [27]. The approaches of this category have the advantage that the user specification is less tedious, but they usually have difficulty in producing accurate results.

No matter whether a segmentation algorithm is automatic or interactive, the problem of how to produce a natural and semantic partition reflecting the human shape perception is of great interest. Theories of the cognitive studies have been used in segmentation algorithms. For example, the minima rule states that human perception tends to divide a surface into parts along minimum negative curvatures [28] and it has been used in the mesh scissoring [2], [19] and the easy mesh cutting [24]. The part salience theory determines the salience of a part based on its relative size, protrusiveness and cut strength [29], [30], and it has been used to reject counter-intuitive segmentation [2] or to guide a cut [6]. Recently, shape analysis based on randomized cuts of meshes is proposed [31], in which a random set of mesh segmentations is generated and the results are used to provide a continuous measure of where natural cuts occur in a mesh.

1.2 Our approach

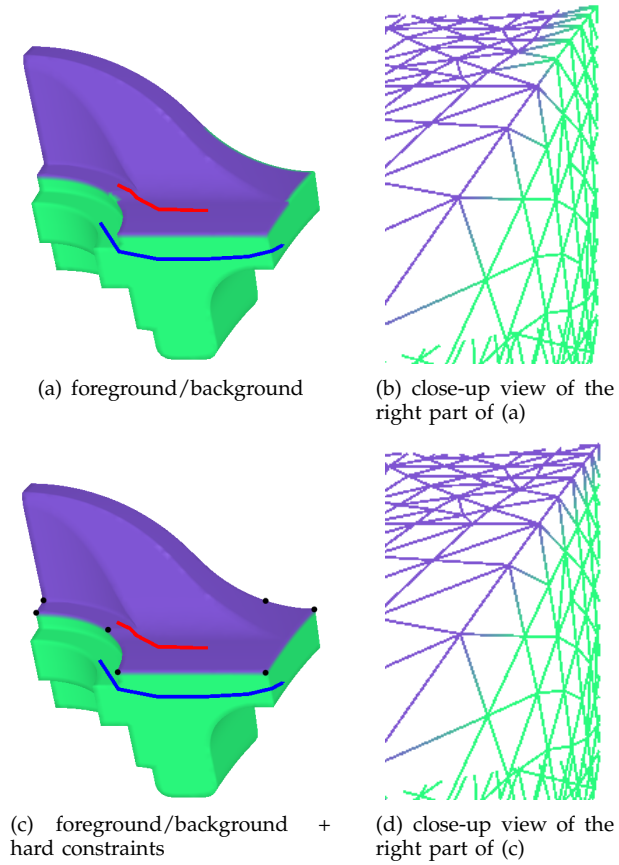
Our work was initially inspired by the easy mesh cutting [24] and the random walks algorithm for image segmentation [26]. When our work completed, we also noticed that the random walks algorithm had been extended for both interactive and automatic mesh segmentation [27]. We appreciated the ease of user's specification for foreground and background in these methods and the speed of the random walks algorithm. However, it is observed that both the easy mesh cutting and the random walks sometimes have difficulty in achieving the user desired cutting, especially when there is no clearly defined boundary in geometry. Even when the desired boundary contains some features, if high accuracy is required, it is also difficult for them to produce a cutting that consistently matches the features. Adding more foreground/background seeds helps to make the foreground and background classification become more accurate, but the process is not so effective. Referring to Figure 1, We try to cut out the bottom circle of the teapot model. Figure 1(a) shows the cutting result by the random walks algorithm [27] with the foreground seeds shown in red and the background seeds shown in blue. We refine the results by gradually adding more foreground and background seeds as shown in Figure 1(b)-(e). This process is a little tedious and the results are still not quite satisfactory. Instead, if we use the method proposed in this paper, we add two strokes (soft constraints) displayed in semi-transparency (see Figure 1(f)) to indicate the region which the cut should be nearby in addition to the initial foreground/background inputs and then the desired result can be quickly found. Figure 2 is another example of using the random walks algorithm. Figure 2(a) and (b) show that the accuracy is difficult to ensure by just specifying foreground and background strokes. However, with a few extra point specifications served as the hard constraints, a neat cutting can be generated, which consistently matches the feature of the desired cutting boundary, as shown in Figure 2(c) and (d). Therefore there is a need for an algorithm to combine user's specification of different levels of ease to provide a simple affordance for fast and accurate cutting.

In this paper, we propose a novel user interface for interactive mesh cutting, which supports the foreground/background seed input, with which the user sketches two strokes to specify which part is foreground and which part is background; the soft constraint input, with which the user draws strokes to show region where the cuts should be made nearby; and the hard constraint input, with which the user places marks to show a set of vertices through which the cuts must go. This is a coarse-to-fine design. In terms of ease of use, the foreground/background seed input needs the least attention, the soft constraint needs only a loose input, and the hard constraint demands a careful input but it assures accuracy. To our knowledge, there is no such user interface for mesh segmentation before. These three



(a) foreground/background strokes
(b) adding another foreground stroke to (a)
(c) adding another foreground stroke to (b)
(d) adding 2 more background strokes to (c)
(e) adding 2 more foreground strokes to (d)
(f) foreground/background + soft constraints

Fig. 1. Incremental foreground/background specifications (a-e) vs foreground/background + soft constraints (f).



(a) foreground/background
(b) close-up view of the right part of (a)
(c) foreground/background + hard constraints
(d) close-up view of the right part of (c)

Fig. 2. Specifying a few points (hard constraints) on the desired cutting contour helps the random walks find more accurate results.

types of user inputs are accomplished by the same computational framework, which includes the constrained random walks algorithm that introduces constraints into the random walks problem and computes the probability for vertices to hint cutting contour information, an optimization process for finding the desired cutting contour, some feature sensitive metrics reflecting the minima rule, and a watershed based preprocessing to speed up computation. The varieties of user inputs make our framework easier and more flexible than the existing interactive mesh segmentation algorithms such as easy mesh cutting, intelligent scissors, mesh scissoring and random walks [2], [21], [24], [27]. Meanwhile, the efficient integration and optimization makes our algorithm fast, reliable and effective.

The main contributions of the paper therefore include: (1) the handling of interactive mesh cutting; (2) the constrained random walks algorithm that adds extra constraints to the conventional random walks and thus allows for useful and intuitive user inputs; and (3) the optimization procedure that uses the shortest graph path to find a nice cutting contour.

1.3 Overview

We first propose a constrained random walk algorithm for computing probability for each vertex of a triangular mesh in Section 2. Then, an optimization process that finds the cutting contour is presented in Section 3. The workflow of our interactive cutout tool is described in Section 4. Section 5 presents experimental results demonstrating the algorithm and Section 6 concludes the paper.

2 Constrained Random Walks for Triangular Meshes

Consider a triangular mesh defined by a tuple $\{V, E, T\}$ of vertices $V = \{v_i \mid v_i \in R^3, i = 1, ..., m\}$, edges $E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$, and faces $T = \{f_{ijk} = (v_i, v_j, v_k) \mid v_i, v_j, v_k \in V, i \neq j, i \neq k, j \neq k\}$. All the edges are undirected and each edge e_{ij} is assigned a value called weight w_{ij} standing for the likelihood that a random walker will move along the edge. Then for each vertex v_i, a scalar value $d_i = \sum_{e_{ij} \in E} w_{ij}$ can be computed. The vertex set consisting of all the vertices lying within the $k-$ring neighborhood of v_i is denoted by $N_k(v_i)$. Assume that we are given a set of vertices $F \subset V$ labeled foreground; a set of vertices $B \subset V$ labeled background, and $F \bigcap B = \emptyset$. Denote by $P(v_i)$ the probability of a random walker starting from vertex v_i arriving at F first, before reaching B. Then for any $v \in F$, $P(v) = 1$ and for any $v \in B$, $P(v) = 0$. For any of the remaining vertices $v_i \in V \setminus (F \cup B)$, we have

$$P(v_i) = \frac{1}{d_i} \sum_{e_{ij} \in E} w_{ij} P(v_j). \quad (1)$$

This leads to a system of linear equations with $P(v_i), v_i \in V \setminus (F \cup B)$ as unknowns. Solving the equations gives the probability of vertex v_i first arriving at F. Based on the computed probability values, classifications can be performed. In particular, for binary segmentation, the vertices with probability value greater than or equal to $\frac{1}{2}$ are classified as the foreground and the vertices with probability value smaller than $\frac{1}{2}$ are classified as the background. This approach is known as the random walks algorithm.

We now impose two types of constraints on some of the vertices except for the foreground and background vertices. The first type is called the *soft* constraint. A vertex on which the soft constraint is imposed has the property that the difference between its probability and $1/2$ is within a small prescribed range $[-\epsilon, \epsilon]$. Let S denote the set of all such vertices. The second type is called the *hard* constraint. A vertex on which the hard constraint is imposed has a probability of $1/2$ and let H denote the set of all vertices with the hard constraint. Then we consider the problem of finding the solution to the equations defined by (1) for $v_i \in V \setminus (F \cup B \cup H)$ subject to the constraints that $P(v) = 1$ for $v \in F$, $P(v) = 0$ for $v \in B$, $P(v) = 1/2$ for $v \in H$, and $|P(v) - 1/2| \leq \epsilon$ for $v \in S$.

It can be proven that the above problem is equivalent to the following minimization problem:

$$\begin{aligned} \min \quad & \sum_{e_{ij} \in E} w_{ij}(P(v_i) - P(v_j))^2 \\ \text{s.t.} \quad & P(v) = 1 \quad \text{for} \quad v \in F \\ & P(v) = 0 \quad \text{for} \quad v \in B \\ & P(v) = 1/2 \quad \text{for} \quad v \in H \\ & |P(v) - 1/2| \leq \epsilon \quad \text{for} \quad v \in S \end{aligned} \tag{2}$$

This is a typical quadratic programming problem. Solving a quadratic programming problem is usually time consuming. More seriously, if the soft constraints are ill-imposed, this quadratic programming problem may have no solution. Therefore, instead of searching for sophisticated quadratic programming solvers, we modify the problem slightly and seek a solution to the following problem, which we call the *constrained random walks problem*:

$$\begin{aligned} \min \quad & \sum_{e_{ij} \in E} w_{ij}(P(v_i) - P(v_j))^2 + \sum_{v_i \in S} \lambda_i (P(v_i) - \frac{1}{2})^2 \\ \text{s.t.} \quad & P(v) = 1 \quad \text{for} \quad v \in F \\ & P(v) = 0 \quad \text{for} \quad v \in B \\ & P(v) = 1/2 \quad \text{for} \quad v \in H \end{aligned} \tag{3}$$

where λ_i is a tradeoff factor controlling the importance of the difference between $P(v_i)$ and $1/2$. The larger λ_i is, the more closely $P(v_i)$ tends to $1/2$.

Differentiating the objective function of (3) with respect to each $P(v_i)$ for $v_i \in V \setminus (F \cup B \cup H)$ and setting it equal to zero, we arrive at

$$P(v_i) = \begin{cases} \frac{1}{d_i + \lambda_i} \sum_{e_{ij} \in E} w_{ij} P(v_j) + \frac{\lambda_i}{d_i + \lambda_i} \frac{1}{2}, & \text{for } v_i \in S \\ \frac{1}{d_i} \sum_{e_{ij} \in E} w_{ij} P(v_j), & \text{for } v_i \in V \setminus (F \cup B \cup H \cup S). \end{cases} \tag{4}$$

This expression has a geometric interpretation (see Figure 3): each vertex with the soft constraint is considered to be connected by a virtual edge with weight λ_i to a virtual neighbor vertex whose probability is $1/2$.

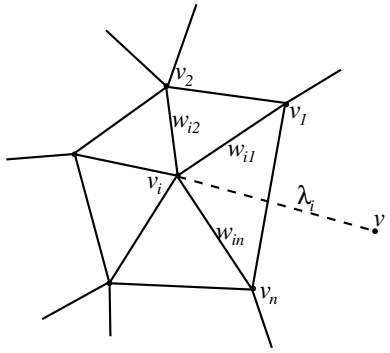


Fig. 3. A vertex v_i with the soft constraint can be considered to have a virtual neighboring vertex v with the probability of 1/2 connected by a virtual edge of weight λ_i.

Equation (4) also gives an estimation of $|P(v_i) - \frac{1}{2}|$ for $v_i \in S$ as follows:

$$\begin{aligned} |P(v_i) - \tfrac{1}{2}| &= \frac{1}{d_i + \lambda_i} \left| \sum_{e_{ij} \in E} w_{ij} P(v_j) - d_i \frac{1}{2} \right| \\ &\leq \frac{1}{d_i + \lambda_i} \sum_{e_{ij} \in E} w_{ij} \left| P(v_j) - \tfrac{1}{2} \right| \leq \frac{1}{2} \frac{d_i}{d_i + \lambda_i} \end{aligned}$$

Therefore, given a small positive number ϵ ($< \frac{1}{2}$), if we let $\lambda_i \geq \left(\frac{1}{2\epsilon} - 1\right) d_i$, we have $\frac{1}{2} \frac{d_i}{d_i + \lambda_i} \leq \epsilon$ which guarantees $|P(v_i) - \frac{1}{2}| \leq \epsilon$. In our experiments, we choose $\lambda_i = 3d_i$, which corresponds to $\epsilon = \frac{1}{8}$.

Let $\mathcal{P}$ be a column vector consisting of all these $P(v_i)(v_i \in V \setminus (F \cup B \cup H))$. Then the equations (4) can be rewritten in matrix form $L\mathcal{P} = \mathcal{C}$ where L is a square coefficient matrix and $\mathcal{C}$ is a column vector. Following the argument in [26], it can be proven that L is a sparse, symmetric, and positive definite matrix. Thus the linear system (4) has a unique solution. Many efficient methods are available for solving such a sparse linear system. It is worth pointing out that many properties of the random walks also remain true for our constrained random walks. For example, it can be found from (4) that $P(v_i)$ falls between 0 and 1 for any $v_i \in V$.

2.1 Edge weights

In the random walks algorithm, the weights assigned to the edges have an important impact on the final results.

For our mesh cutting application, the weights should be properly set so that the segmentation tends to cut the mesh into meaningful pieces.

In graph-based algorithms for image analysis, a common practice is to compute a change in image intensities and then map the change to edge weights. Analogously, here we need to find a distance metric measuring the change of geometric properties along the edges. The feature sensitive metric proposed in [24] is adapted, which considers both the isophotic metric and the minima rule. The isophotic metric was introduced by [32], which is dependent on the length of the path on the surface connecting the two points and also the variation of the surface normals along it. The isophotic metric is a feature sensitive metric on the surface. The minima rule is an approximated perceptual criterion in the cognitive theory [28]. It states that human perception usually divides a surface into parts along negative minima of the principal curvature. For an edge e_{ij} with vertices v_i and v_j, its distance is defined by

$$d_{ij} = w_1||v_i - v_j|| + w_2||n_i - n_j|| + w_3 f(\kappa_{v_i v_j}) \quad (5)$$

where the first two terms account for the isophotic metric and the third term for the minima rule, n_i and n_j are respectively the unit normals of the surface at vertices v_i and v_j , $\kappa_{v_i v_j}$ is the average of the normal curvature at v_i along the line direction $\overrightarrow{v_i v_j}$ and the normal curvature at v_j along $\overrightarrow{v_j v_i}$. The normal curvature for a discrete triangular mesh is calculated based on a formula given in [33] or [34]. The function $f(\kappa)$ is set as

$$f(\kappa) = \begin{cases} \kappa, & \kappa \geq 0 \\ 5|\kappa|, & \kappa < 0 \end{cases} \quad (6)$$

where 5 is added to augment the effect of the negative curvature, which is in accord with the minima rule. The coefficients w_1, w_2 and w_3 are used to control the relative importance of the distance, normal variation and curvature. Usually the size scales are diverse among different 3D models and we note that some terms in equation (5) depend on the size of the models. Thus, before we apply the distance metric (5) to a mesh model, we normalize the model to make the maximum Euclidean distance between two vertices on the mesh be equal to 1. Then we set the values of w_i by making them satisfy the following relations: $\text{avg}(w_1||v_i - v_j||) = 0.1$ $\text{avg}(w_2||n_i - n_j||) = 0.1$ $\text{avg}(w_3 f(\kappa_{v_i v_j}))$ and $\text{avg}(d_{ij}) = 1$, where $\text{avg}(x)$ returns the average value of quantity x over the whole model.

Once the edge distance metrics have been computed, we need to find a function to map the edge metrics d_{ij} to the weights w_{ij}. The larger distance d_{ij} is, the less weight w_{ij} will be assigned to edge e_{ij}. Therefore the function should be decreasing. Following [26], we use the Gaussian function $w_{ij} = exp(-d_{ij}^2)$, which can maximize the entropy of the resulting weights.

3 Optimization for the Cutting Contour

While some algorithms such as in [2], [24] compute new edges for the cutting contour, we restrict ourselves to cutting only along the existing vertices and edges of a mesh in this paper. This has an advantage that the cut mesh is exactly one part of the original mesh, both geometrically and topologically. Note that in man-made models edges of the mesh usually occur along semantic seams and in natural models meshes are usually sufficiently dense. Therefore for these kinds of models our restriction will not affect the results very much and meanwhile it can avoid some unwanted operations.

Once the probability values are computed from the constrained random walks algorithm, we have to design a way to determine the cutting contour. Our basic idea is first to find a contour area, which we denote by G. The contour area is formed by all the candidates for the vertices on the cutting contour and the edges connecting them. The contour area is a part of the original mesh. The second step is then to choose vertices from G for the cutting contour. Since our constrained random walks algorithm gives the probability of a random walker starting from a vertex arriving at the foreground seeds first, before reaching the background seeds, it is reasonable to choose the probability value of $1/2$ as a selection criterion. However, if the closeness of the probability values to $1/2$ is the only criterion, this likely results in a jaggy contour. The top of Figure 4 shows the result generated by this way. Therefore other criteria such as the smoothness of the contour should also be considered for good cutting contours. The purpose of cutting contour optimization is to find the cutting contour that optimizes a certain energy function. The rest of this section describes how to find the contour area and how to determine the cutting contour within the contour area.

Intuitively, if a vertex $v_i \in V \backslash F \cup B \cup H$ with probability $P(v_i) \geq 1/2$ (or $< 1/2$) has at least one neighboring vertex v_j with probability $P(v_j) < 1/2$ (or $P(v_j) \geq 1/2$), it is a candidate for the vertices on the cut contour. All these candidate vertices and the edges connecting them form the contour area G. G has two boundaries B^+ and B^-, which consist of vertices with probability $\geq 1/2$ and $< 1/2$, respectively. Note that for a hard constraint vertex, the weighted average of the probability values of the vertices in its 1-ring neighborhood is not necessarily $1/2$. It is possible that the probability values of all its 1-ring neighboring vertices are greater than $1/2$. Thus, this hard constrained vertex will not be included in G and it will not be in the cutting contour. This contradicts our intention of introducing the hard constraint. To overcome this problem, we expand the candidate set G by moving the two boundaries B^+ and B^- along their respective directions independently until G contains all the hard constraint vertices.

To find a good cutting contour within G, we propose an energy function and try to find the contour that mini-

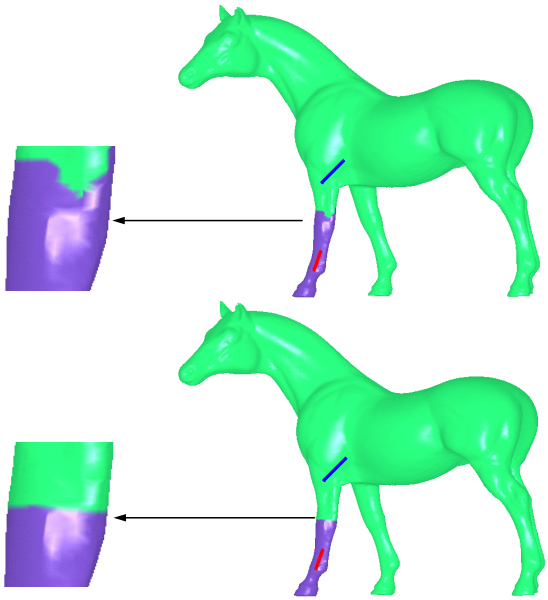


Fig. 4. A non-smooth cutting contour (top) vs a smooth cutting contour (bottom).

mizes the energy function among all possible contours in G, which separate the foreground and the background. For a contour $C = \{v_1 v_2 \cdots v_l\}$ in G, we define its energy function by

$$\sum_{v_i \in C} \{g(|P(v_i) - \frac{1}{2}|) + \alpha\theta_i + \beta(1 - \frac{f(\kappa_{i1}) + f(\kappa_{i2})}{max(f(\kappa_{i1}) + f(\kappa_{i2}))})\} \quad (7)$$

where α and β are the tradeoff factors (the default values for α and β are set to 1), θ_i is the angle (measured in radians) between the projection lines of $v_{i-1}v_i$ and $v_i v_{i+1}$ on the tangent plane of the surface at v_i, $f()$ is the function defined in (6), κ_{i1} and κ_{i2} are the two principal curvatures of the surface at v_i, $\max()$ takes the maximum value over all the vertices of the mesh, and $g()$ is

$$g(|P(v_i) - \frac{1}{2}|) = \begin{cases} -M, & \text{if } v_i \in H \\ |P(v_i) - \frac{1}{2}|, & \text{otherwise} \end{cases}$$

where M is a big positive number. The function $g()$ is designed to ensure that the optimal solution goes through the vertices with hard constraints. In energy function (7), the first term measures how close the vertices' probability is to $1/2$, respecting the probability values; the second term measures the fairness of the polygon; and the third term makes the cutting contour tend to pass the vertices with minimum negative principal curvatures (i.e., in the concave shape areas). According to the minima rule, the vertices with minimum negative principal curvatures are in the interfaces separating object parts.

Finding the solution that minimizes (7) is actually a problem of finding the shortest path in a graph inferred from the triangular mesh, where each vertex is a node and each mesh edge is an edge connecting two nodes. The cost function (7) is defined on each node. However, the problem we are trying to solve has three features that complicate the optimization. In the following, we explain these features and our strategies.

The first feature is that the cutting contour we want to find separates the foreground and the background and thus is often closed. In this case, we take an approach that is similar to the one used in [35] in order to use Dijkstra's algorithm later. For each boundary of G, say B^+, we arbitrarily choose a vertex v_i in it. We apply the breadth-first search algorithm on G starting from v_i until encountering a vertex (say, v_j) in B^-. The path from v_i to v_j found in the breadth-first search algorithm will be used to split G. See the blue path in Figure 5 for illustration. To make the algorithm simpler and easier, we remove those edges (i.e., the dashed edges in Figure 5) that are on one side of the splitting path and are connected to it.

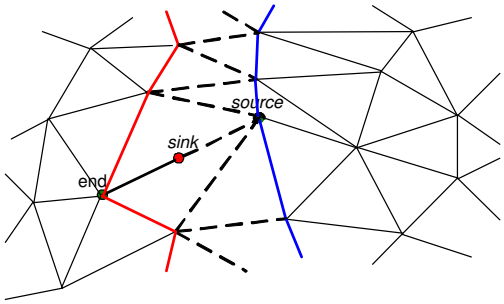


Fig. 5. Splitting the contour area G along the blue path.

The second feature is that when we compute θ_i for each vertex v_i, it depends on which two edges incident to v_i are chosen. That is, θ_i is determined not only by v_i, but also by v_{i-1} and v_{i+1}. To handle all the vertices properly, when we use Dijkstra's algorithm to find the shortest path from "source" to the "end" (see Figure 5), we place a virtual vertex called "sink" on the line between "source" and "end" and replace the dashed line between "source" and "end" by a solid edge for "end" and "sink" and a dashed line for "sink" and "source". Here, the "sink" vertex's probability is set to 0.5 and its two principal curvatures are set to zero. Now we turn to finding the shortest path from "source" to "sink", which is equivalent to our original minimization problem. The introduction of the virtual vertex "sink" makes the cost computation for "source" and "end" simpler.

The third feature is the negative cost when there are hard constraint vertices. In this situation, we cannot simply use Dijkstra's algorithm which assumes all the costs to be nonnegative. Thus we adopt the following strategy to approximate the global optimum. Assume there are m vertices with hard constraints. Dijkstra's algorithm is applied from the "source" vertex until encountering a hard constraint vertex, which we denote by h_1, and we label h_1 as processed. Then Dijkstra's algorithm is applied again from h_1 until meeting an unprocessed hard constraint vertex, which we denote by h_2. Repeat this process until we meet h_m. Then, we find the shortest path between h_m and $sink$. The union of all these pathes

gives the cut contour.

This procedure is applied to every dashed edge in graph G, where the intersection point of the dashed line and the path is used as "source" and the other endpoint of the dashed line is used as "end". The optimal solution is the one that has the minimum energy value. The bottom of Figure 4 is the result generated by this approach. Compared to other optimal methods such as the snake used in [24] which tends to converge to a local minimum, our approach usually returns global optimization. Besides, the time complexity of the modified Dijkstra's algorithm is suitable for interactive mesh cutting.

4 Interactive Mesh Cutting

We are now ready to describe our framework and its implementation for interactive mesh cutting.

When a triangular mesh model is loaded into the system, the user can navigate the mesh, select an appropriate viewpoint, and use the following three basic modes to specify his/her intention:

- foreground/background seed inputs: The user sketches strokes on the screen to define his/her foreground and background.
- soft constraint inputs: The user draws strokes on the screen to show the region which the cutting contour should be made nearby.
- hard constraint inputs: The user places marks on the screen to show the points where the cutting contour should go through.

Each stroke has a user-specified width. Those vertices on the front-face of the mesh will be marked if their image on the screen is covered by the user's strokes. If more than one vertex map to the same screen position, the one with the nearest distance to the user will be marked. The user input mode is also transferred to the corresponding vertices. Except for the hard constraint inputs that allow the user to specify which vertex should be on the cutting contour, the strokes are not necessarily needed to be precise. The user can freely switch one input mode to another and mix the use of them. This provides a lot of flexibility to the user because some models or some parts of the models may need a specific input mode.

Once the user completes the sketching, the constrained random walks algorithm starts to compute probability for each vertex of the mesh, followed by the optimization procedure for finding the cutting contour. Then the cutting contour and the "cutout" mesh are quickly generated.

It should be pointed out that if we have only soft or/and hard constraint inputs, the algorithm actually does not know which part should be cut out. This means the information for the user's intention is not sufficient. In this case, our system will be able to find the cutting contour and automatically set one segment as the foreground and the other as the background by default. The user can input further information to specify which part should be treated as foreground or background.

To implement the cutting with only soft or hard constraint inputs, we use the following strategies. For the soft constraint only, we automatically create two new strokes which are on both sides of the input stroke and have a similar direction as the input one. One of these two new strokes is treated as the foreground seeds and the other as the background seeds. Then the problem becomes the one that has foreground/background seed inputs and a soft constraint input. The constrained random walks algorithm can be used now. Figure 6 (left) shows such an example, where the user only inputs one stroke for soft constraints and the algorithm automatically creates two new strokes which are used as the foreground/background seed inputs. A similar approach can be developed for the hard constraint only in which two input marks are used to define a line and two new strokes along the similar direction on both sides of the line are automatically created. See Figure 6 (right) for an illustration. In our mesh cutting system, these new strokes are created just for the underlying computational purpose and are actually not displayed to avoid confusing the user. Also these "automatically" generated strokes are created only when they are really needed. When there are already some foreground/background seeds, the "automatically" generated strokes will not be created.

If the cut result is not satisfactory, the user can refine the cutting results by interactively sketching more strokes in any of these three input modes.

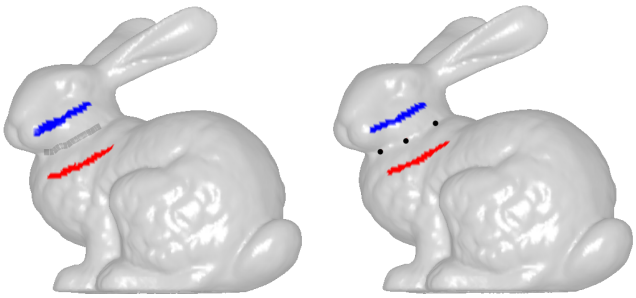


Fig. 6. Automatically generating foreground/background seeds for the soft or hard constraint input only. left: soft constraint input; right: hard constraint input.

4.1 Preprocessing

For interactive applications, instant feedback is very important. Although the constrained random walks algorithm can be implemented by solving a sparse linear system and many efficient solvers for a sparse linear system are available, there is still a need to speed up the computation for very large models. Here we present a preprocessing step based on the watershed method [7], as [36] did.

First, the watershed algorithm is performed to oversegment the mesh. To do so, the total curvature at each vertex is computed. The vertex with local minimum total curvature is assigned a unique label. Loop through

vertices and allow each vertex to descend until a labeled region is encountered.

Second, we perform segmentation using the methods described in preceding sections on the super set of the mesh, in which the vertices are the segmented regions from the watershed segmentation and the edges are the arcs connecting adjacent regions. The weight between two adjacent regions is defined as the sum of weights of those edges crossing the two regions. The outputs of the segmentation are then distributed back to each vertex in the initial mesh and they provide a good approximation that can be used as the initial values for our constrained random walks algorithm. Since the watershed segmentation yields a good superset of the mesh, this preprocessing improves the speed significantly.

5 Experiments and discussions

This section provides some examples to demonstrate the applicability and flexibility of our constrained random walks based mesh cutting algorithm. The experiments were carried out on a 2.67 GHz Intel(R) Core(TM) 2 processor with 2 Gb memory. In these examples, the red strokes, the blue strokes, the black dots, and the semi-transparent grey strokes stand for the foreground input, the background input, the hard constraint input, and the soft constraint input, respectively.

We first tested the running time of our algorithm, without and with the watershed preprocessing step. The test was conducted on six models with different sizes in the number of vertices. Figure 7 shows the models, user's inputs and the cutting results. Since the cutting results with and without the watershed preprocessing are almost the same visually, we just display the results with the watershed preprocessing in Figure 7. Table 1 lists the statistics of the models and the running time. It can be seen that our algorithm (with the watershed preprocessing) can yield the results in real time.

We then compared our algorithm with Lai et al's random walks algorithm [27] and the easy mesh cutting [24] which are closely related to our work. In this comparison, we did not perform the postprocessing step to smooth the cutting contour for both our algorithm and Lai et al's algorithm [27]. From Figure 8, it can be found that without soft and hard constraints our algorithm and Lai et al's algorithm give the similar cutting results. This is not surprising because our algorithm without the soft/hard constraints is just one version of the random walks. One difference between our algorithm and Lai et al's algorithm is that our's random walks is directly applied to the triangular mesh and Lai et al's random walks is applied to the dual graph which treats the faces of the mesh as the graph nodes. While using the dual graph has some advantages, we choose the vertices as the nodes, which facilitates our specifications of hard constraints. Also note that for a triangular mesh with a large number of vertices, the number of vertices is roughly half the number of the faces and this could reduce the size of the linear system derived from the random walks. We also find that the easy mesh cutting algorithm sometimes works badly (see Figure 8(c)). This is because the easy mesh cutting algorithm uses region growing to find the cutting contour. The region growing process selects the smallest weight neighbor adjacent to the current vertices in each iteration. As a result, the region growing heavily depends on the initial seeds position and is sensitive to noises. The vertices with high curvatures or noises usually have large weight values, which prevent the region growing process from growing.

Our interactive cutout algorithm supports three typical inputs and their combinations (see Figure 7). In general, the foreground/background seeds input is easiest to use, the soft constraint input needs some attention and the hard constraint input needs the most attention. The choice of the inputs really depends on the user and applications. For some models, the foreground/backround inputs are sufficient. However, in some situations, it is difficult to obtain satisfactory results by using foreground/backround inputs only. As already shown in Figure 1 and Figure 2, soft and hard constraints can be used with the foreground/background seed inputs to quickly find good cutting. Figures 9, 10 and 11 show another three examples where the soft/hard constraints are used to further guide the cutting. In these examples, the random walks algorithm [27] is also used to find the cutting contours, for which we gradually add more and more foreground and background strokes to refine the cutting, but the results are still not good enough. On the contrary, when we use our constrained random walks algorithm with a few extra soft or/and hard constraints, we can quickly find good cutting results. As pointed out in [23], random walks is quite robust with respect to the number of seeds but sensitive with respect to the seed location. The introduction of soft and hard constraints thus helps to correct the mis-guidance of the foreground/background seeds. This makes our constrained random walks work robustly and stably with respect to various inputs.

Finally, Figure 12 shows some examples of cutting meshes of complex topology using our constrained random walks algorithm.

6 Conclusion

In this paper, we have proposed a constrained random walks algorithm and an optimal path finding algorithm. Based on them, an interactive mesh cutting algorithm is developed which supports three typical user inputs and their combinations, simulating the current leading interactive mesh segmentation algorithms including the easy mesh cutting, Intelligent Scissoring, and the Active Contour Method within the same computational framework. The experiment results demonstrate that the new algorithm is robust, fast, and capable of producing satisfactory results with regard to the user intention and geometric attributes.

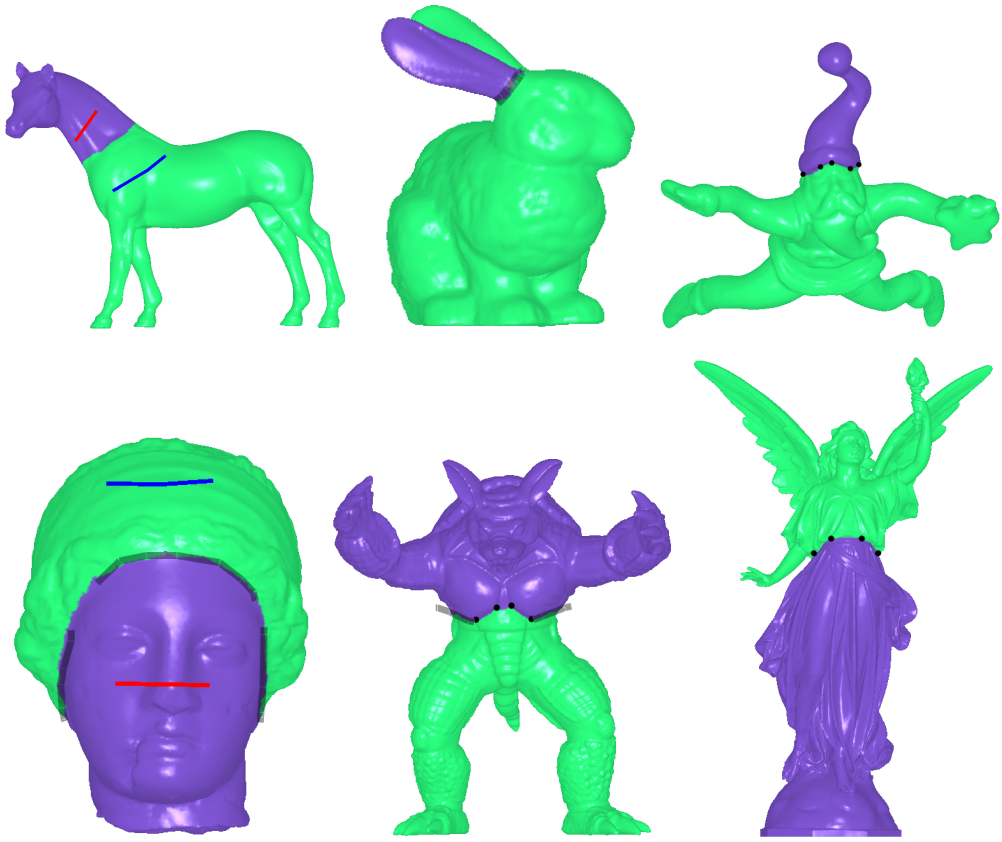
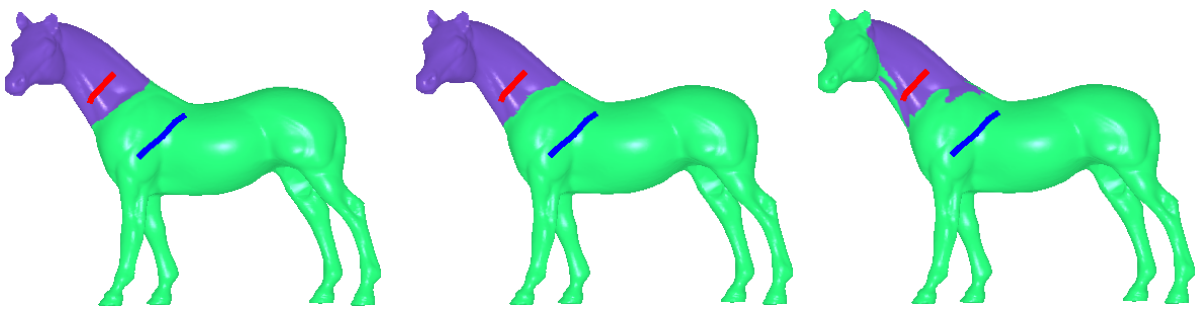


Fig. 7. The cutting results using the proposed algorithm for the examples listed in Table 1, on which the running time test was conducted. These examples also show that the three types of inputs can be used independently or in a combined way.

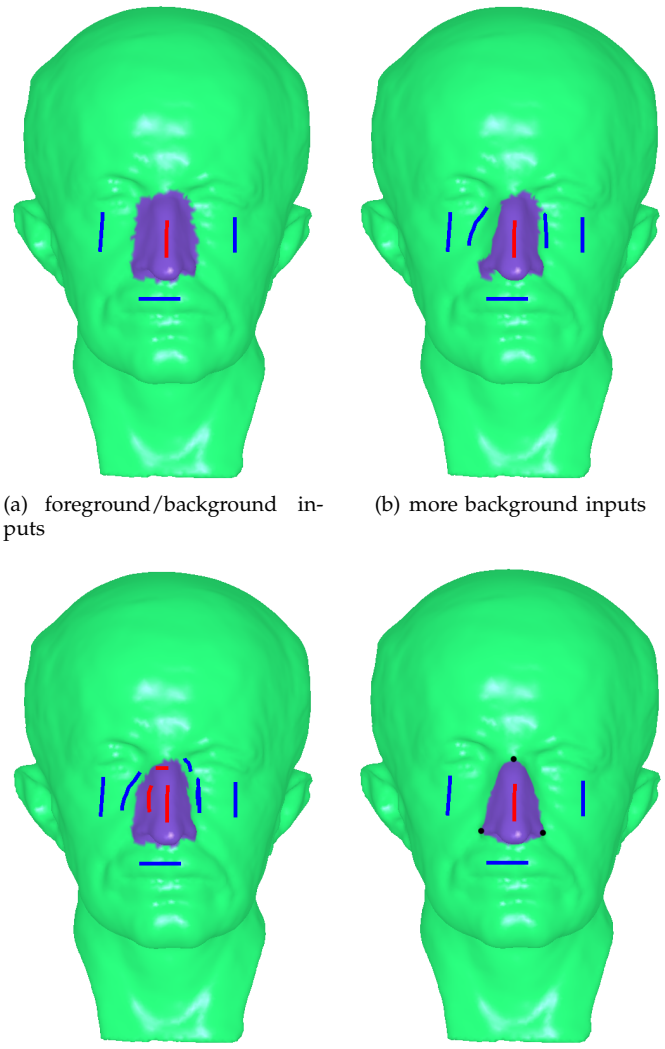
TABLE 1
Running time statistics of our algorithm with and without the watershed preprocessing step.

Model	Vertex Number	Time(s) without preprocessing	Time(s) with preprocessing
Bunny	35947	1.718	0.194
Horse	48485	1.265	0.157
Venus	67173	1.953	0.265
Santa	75781	1.935	0.261
Armadilo	172974	5.984	0.727
Lucy	262909	9.797	1.242



(a) Ours (b) Lai et al's (c) The easy mesh cutting

Fig. 8. When the input consists of only foreground and background seeds, our method usually gives almost the same result as Lai et al's random walks algorithm [27] and a better result than the easy mesh cutting [24].



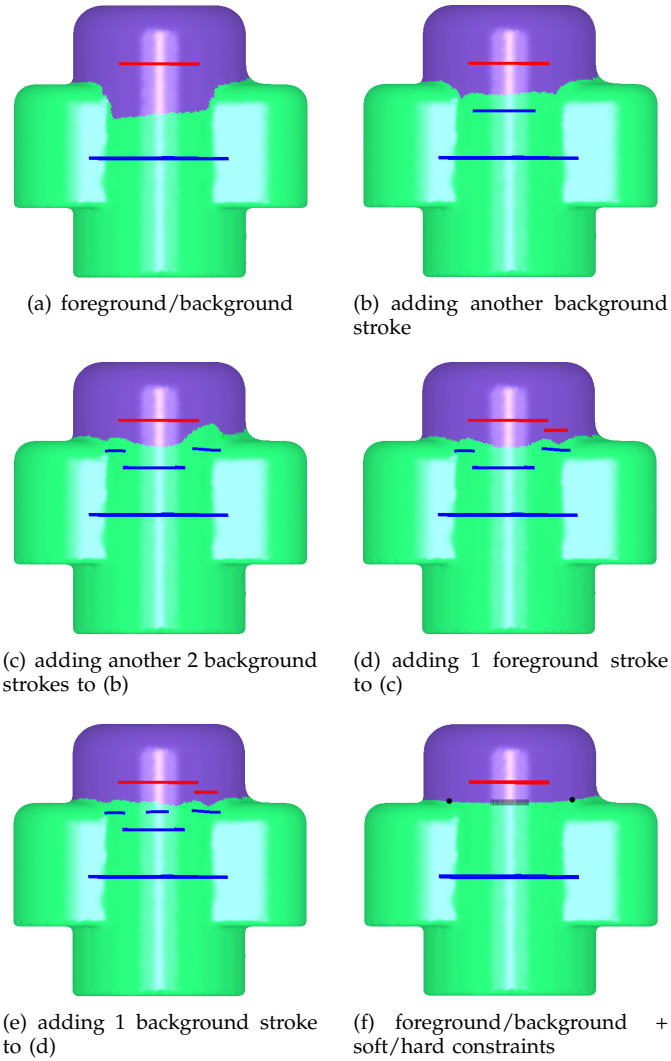
(a) foreground/background inputs

(b) more background inputs

(c) more foreground/background inputs

(d) foreground/background + hard constraints

Fig. 9. While gradually inputting more foreground and background strokes does not guarantee a nice cutting (a-c), specifying three hard constraints at the weak boundary can help to quickly find a satisfactory cutting (d).



(a) foreground/background

(b) adding another background stroke

(c) adding another 2 background strokes to (b)

(d) adding 1 foreground stroke to (c)

(e) adding 1 background stroke to (d)

(f) foreground/background + soft/hard constraints

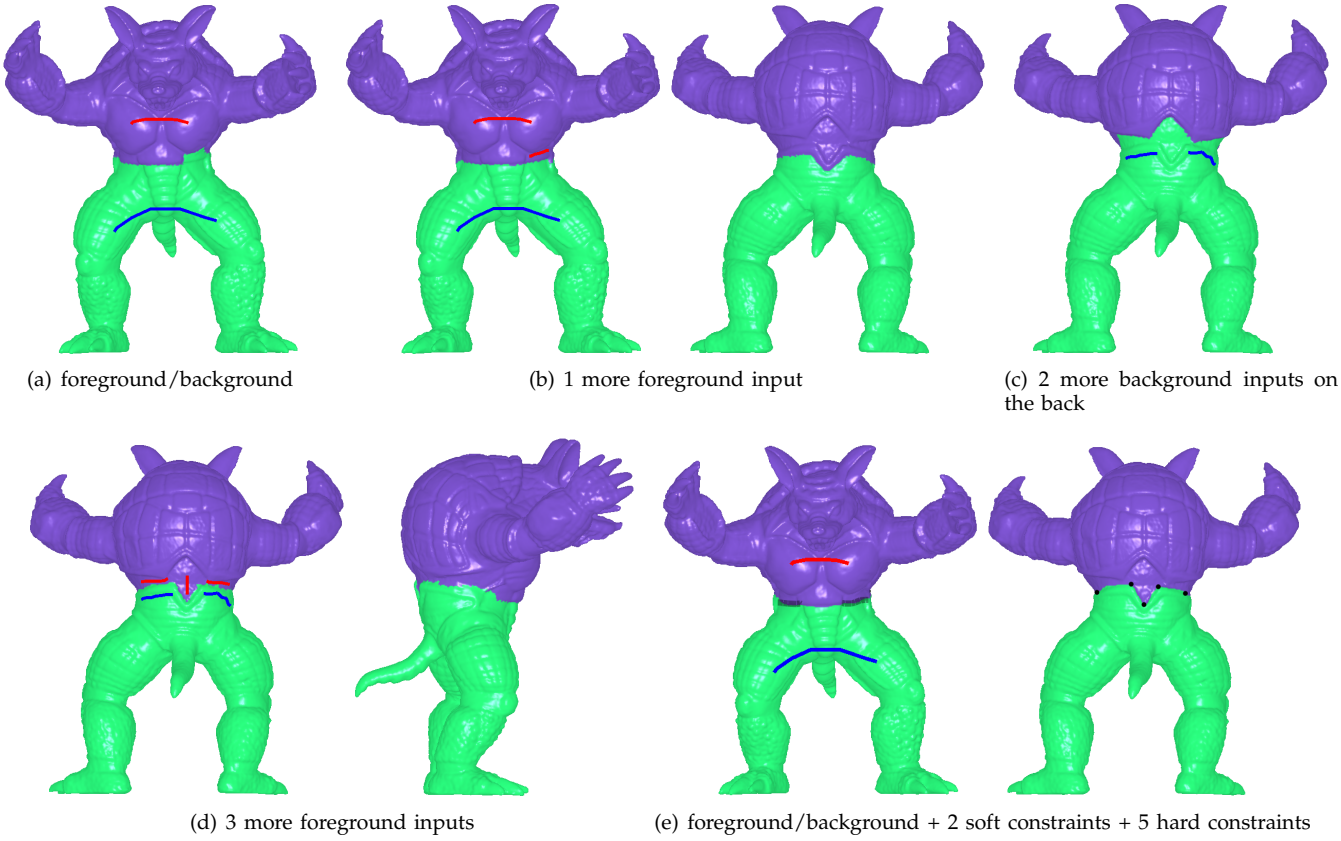
Fig. 10. (a)-(e) show the cutting results by gradually adding foreground/background strokes. (f) shows the result by inputting some soft/hard constraints in addition to the initial foreground/background, which is a satisfactory result reflecting user intention and human shape perception.

Since this paper focuses on segmenting the mesh into two classes, the constrained random walks algorithm is developed and described for such a binary segmenetation. However, it is possible to extend the idea and the algorithm to segment a mesh into several classes. The basic strategy is that for each class of input seeds, we compute the probability of a random walk starting from a node arriving at that class first, before reaching other classes of seeds, and then we classify this node to the class with which the probability is the maximal. Since now we segment the mesh into several classes, when we enter soft or hard constraints, we need to explicitly indicate which class of seeds the soft or hard constraints are associated with. Then the proposed constrained random walks algorithm can be used to compute the probability. In this way, the soft and hard constraint inputs are used to affect the probability distribution. The challenging part in this extension is how to guarantee that the final segmentation contours go through the hard constraint vertices, which is still under investigation.

The contour optimization approach proposed in the paper is quite heuristic. It is of both theoretical and practical interest to devise a more elegant manner for extracting the cutting contour from the results generated from the constrained random walks.

ACKNOWLEDGMENTS

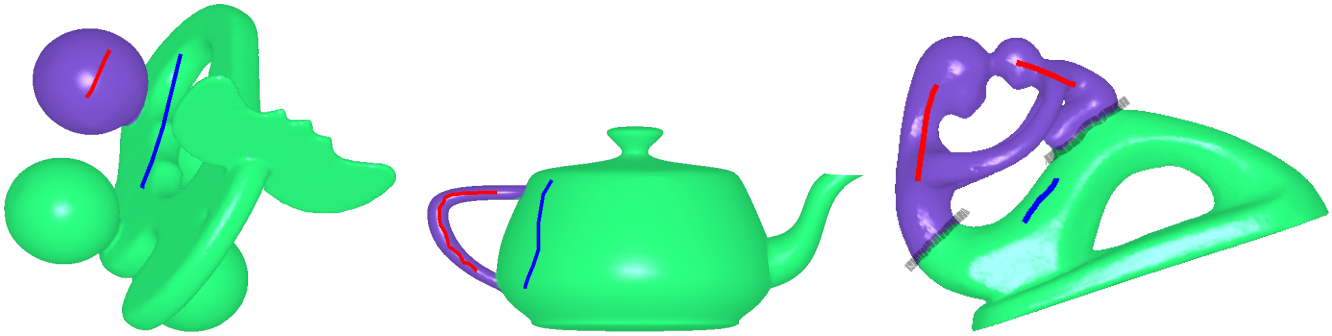
This work is supported by A*STAR SERC TSRP Grant (NO. 062 130 0059) and the ARC 9/09 Grant (MOE2008-T2-1-075) of Singapore.



(a) foreground/background (b) 1 more foreground input (c) 2 more background inputs on the back

(d) 3 more foreground inputs (e) foreground/background + 2 soft constraints + 5 hard constraints

Fig. 11. Cutting result comparison: (a)-(d) vs (e), where (a)-(d) show the results (at front/back/side view) by gradually adding foreground/background strokes, which cannot achieve satisfactory results for the all the views; and (e) shows the result by our proposed algorithm with the initial foreground/background inputs and a few soft/hard constrains.



(a) genus = 1 (b) genus = 1 (c) genus = 4

Fig. 12. More complex meshes.

REFERENCES

[1] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, no. 11, pp. 1–18, Nov 2007.

[2] Y. Lee, S. Leea, A. Shamirb, D. Cohen-Orc, and H.-P. Seideld, "Mesh scissoring with minima rule and part salience," *Computer Aided Geometric Design*, no. 11, pp. 444–465, July 2005.

[3] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of polyhedral surfaces using decomposition," *Computer Graphics Forum*, vol. 21, no. 3, 2002.

[4] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Transactions on Graphics (SIGGRAPH)*, pp. 954–961, 2003.

[5] R. Liu and H. Zhang, "Segmentation of 3D meshes through spectral clustering," in *Proc. of Pacific Graphics*, 2004, pp. 298–305.

[6] ——, "Mesh segmentation via spectral embedding and contour analysis," *Computer Graphics Forum (Eurographics 2007)*, vol. 26, no. 3, pp. 385–394, 2007.

[7] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 308–321, 1999.

[8] X. Li, T. W. Woon, T. S. Tan, and Z. Huang, "Decomposing polygon meshes for interactive applications." Procedding of Symposium on Interactive 3D graphics, 2001, pp. 35–42.

[9] S. Katz, G. Leifman, and A. Tal, "Mesh segmentation using feature point and core extraction," *The Visual Computer*, vol. 21, no. 8-10, pp. 649–658, 2005.

[10] T.-Y. Lee, P.-H. Lin, S.-U. Yan, and C.-H. Lin, "Mesh decomposition using motion information from animation sequences," *Journal of Visualization and Computer Animation*, vol. 16, no. 3-4, pp. 519–

529, 2005.
[11] T.-Y. Lee, Y.-S. Wang, and T.-G. Chen, "Segmenting a deforming mesh into near-rigid components," *The Visual Computer*, vol. 22, no. 9-11, pp. 729–739, 2006.
[12] S. Schaefer and C. Yuksel, "Example-based skeleton extraction," in *Symposium on Geometry Processing*, 2007, pp. 153–162.
[13] M. Attene, S. Katz, M. Mortara, G. Patanè, M. Spagnuolo, and A. Tal, "Mesh segmentation - a comparative study," in *Proc. of Shape Modeling International*, 2006, p. 7.
[14] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Transactions on Graphics (SIGGRAPH)*, 2009.
[15] K. C.-H. Wong, T. Y.-H. Siu, P.-A. Heng, and H. Sun, "Interactive volume cutting," in *Graphics Interface*, 1998, pp. 99–106.
[16] A. Gregory, A. State, M. C. Lin, D. Manocha, and M. A. Livingston, "Interactive surface decomposition for polyhedral morphing," *The Visual Computer*, no. 9, pp. 453–470, December 1999.
[17] M. Zöckler, D. Stalling, and H.-C. Hege, "Fast and intuitive generation of geometric shape transitions," *The Visual Computer*, vol. 16, no. 5, pp. 241–253, 2000.
[18] Y. Lee and S. Lee, "Geometric snakes for triangular meshes," *Computer Graphics Forum (Eurographics 2002)*, no. 3, pp. 229–238, July 2002.
[19] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel, "Intelligent mesh scissoring using 3d snakes," in *Proc. of Pacific Graphics*, 2004, pp. 279–287.
[20] A. Clements and H. Zhang, "Minimum ratio contours on surface meshes," in *Proc. of Shape Modeling International*, 2006, pp. 26–37.
[21] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by example," *ACM Transactions on Graphics (SIGGRAPH)*, pp. 652–663, 2004.
[22] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or, "Snappaste: an interactive technique for easy mesh composition," *The Visual Computer*, no. 9-11, pp. 835–844, September 2006.
[23] A. K. Sinop and L. Grady, "A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm." Proceedings of ICCV, 2007, pp. 560–572.
[24] Z. Ji, L. Liu, Z. Chen, and G. Wang, "Easy mesh cutting," *Computer Graphics Forum (Eurographics 2006)*, vol. 25, no. 3, pp. 219–228, Sep. 2006.
[25] Y.-S. Wang and T.-Y. Lee, "Wysiwyg: Mesh decomposition for static models," in *IIH-MSP '07: Proceedings of the Third International Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 353–356.
[26] L. Grady, "Random walks for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 11, pp. 1768–1783, November 2006.
[27] Y.-K. Lai, S.-M. Hu, R. R.Martin, and P. L.Rosin, "Rapid and effective segmentation of 3d models using random walks," *Computer Aided Geometric Design*, vol. 26, no. 6, pp. 665–679, 2009.
[28] D. D. Hoffman and W. Richards, "Parts of recognition," *Cognition*, pp. 65–96, 1984.
[29] D. Hoffman and M. Singh, "Salience of visual parts," *Cognition*, pp. 29–78, 1997.
[30] H.-Y. S. Lin, H.-Y. M. Liao, and J.-C. Lin, "Visual salience-guided mesh decomposition," *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 46–57, 2007.
[31] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3D mesh analysis," *ACM Transactions on Graphics (SIGGRAPH ASIA)*, Dec. 2008.
[32] H. Pottmann, T. Steiner, M. Hofer, C. Haider, and A. Hanbury, "The isophotic metric and its application to feature sensitive morphology on surfaces," in *Proceedings of ECCV*, 2004, pp. 560–572.
[33] M. Meyer, M. Desbrun, P. Schroder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," *Visualization and Mathematics*, vol. 1, no. 3, pp. 35–57, 2002.
[34] G. Taubin, "Estimating the tensor of curvature of a surface from a polyhedral approximation." Proceedings of ICCV, 1995, pp. 902–907.
[35] J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum, "Drag-and-drop pasting," *ACM Trans. on Graphics (SIGGRAPH)*, no. 3, pp. 631–636, July 2006.
[36] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum, "Lazy snapping," *ACM Transactions on Graphics (SIGGRAPH)*, pp. 303–308, 2004.