EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow

Juyong Zhang Chunlin Wu Jianfei Cai Jianmin Zheng Xue-cheng Tai

Nanyang Technological University, Singapore

Abstract

This paper considers the problem of interactively finding the cutting contour to extract components from a given mesh. Some existing methods support cuts of arbitrary shape but require careful and tedious input from the user. Others need little user input however they are sensitive to user input and need a postprocessing step to smooth the generated jaggy cutting contours. The popular geometric snake can be used to optimize the cutting contour, but it cannot deal with the topology change. In this paper, we propose a geodesic curvature flow based framework to overcome all these problems. Since in many cases the meaningful cutting contour on a 3D mesh is locally shortest in the sense of some weighted curve length, the geodesic curvature flow is an ideal tool for our problem. It evolves the cutting contour to the nearby local minimum. We should mention that the previous numerical scheme, discretized geodesic curvature flow (dGCF) is too slow and has not been applied to mesh segmentation. With a careful observation to dGCF, we devise here a fast computation scheme called fast geodesic curvature flow (FGCF), which only needs to solve a smaller and easier problem. The initial cutting contour is generated by a variant of random walks algorithm, which is very fast and gives reasonable cutting result with little user input. Experiment results on the benchmark mesh segmentation data set show that our proposed framework is robust to user input and capable of producing good results reflecting geometric features and human shape perception.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms,languages, and systems

1. Introduction

Interactive mesh cutting [LLS*05, Sha08], which involves little user interaction to guide the mesh segmentation process, has received much attention in the recent years. It has many applications such as modeling by examples, 3D morphing, parameterization, texture mapping, and shape matching and reconstruction. In this paper we consider how to cut a meaningful part out from a given triangular mesh with the assistance of little user input.

1.1. Related Work

Many interactive mesh cutting algorithms have been proposed in the literature. In general, they can be classified into two categories: boundary based approaches and region based approaches. In boundary based approaches, the user is often required to provide an initial area that is "close" to the desired cut. The geometric snake and mesh scissoring algorithms [LL02, LLS*04, LLS*05] evolve the initial cutting contour to or find the desired position which is "close" to the initial area. The intelligent scissoring [FKS*04] applies a variant of Dijkstra's algorithm to find the cutting contour that goes within the initial area. Graph Cut algorithm is applied to find the cutting contour in [SBSCO06]. Some early boundary based approaches requires the user to specify a few points on the desired cutting contour and the cut is then accomplished by finding the shortest paths between them [WSHS98, GSL*99, ZSH00], which is also used to design its interactive segmentation tool in [CGF09].

One drawback of the boundary based approaches is that they require great care when specifying the boundary points or boundary areas, especially for complex graphics models. Most recent interactive mesh cutting algorithms take the regional information as the input, which requires a much smaller amount of user efforts. In particular, the user is asked to casually draw two types of strokes to label some ver-

tices or faces as foreground or background seeds, and then the algorithm completes the labeling for all unlabeled vertices or faces. The easy mesh cutting [JLCW06] and the random walks [LHMR08] are two representative region based approaches. The easy mesh cutting [JLCW06] starts with different seed vertices simultaneously and grows the submeshes according to the improved isophotic metric incrementally. The random walks [LHMR08] provides fast mesh segmentation according to the probability value computed by minimizing a Dirichlet energy [SG07].

1.2. Motivation and Our Approach

In this paper, we consider the problem of interactive mesh cutting with the input of foreground and background strokes, which requires least attention from the user. By carefully examining the existing region based approaches [JLCW06, LHMR08], we find that they are not able to achieve robust and effective performance. First, the existing region based approaches are sensitive to the user input. The region growing method used in the easy mesh cutting [JLCW06] heavily depends on the initial seeds' positions and is sensitive to mesh noise. The performance of the random walks [LHMR08] also heavily relies on the seeds' positions. Fig. 1 shows two examples. It can be seen that for the random walks algorithm [LHMR08], different inputs always result in different cutting contours (see Fig. 1(a) and (b), Fig. 1(c) and (d)). The essential reason is that it minimizes a Dirichlet energy and different boundary conditions always result in different harmonic functions.

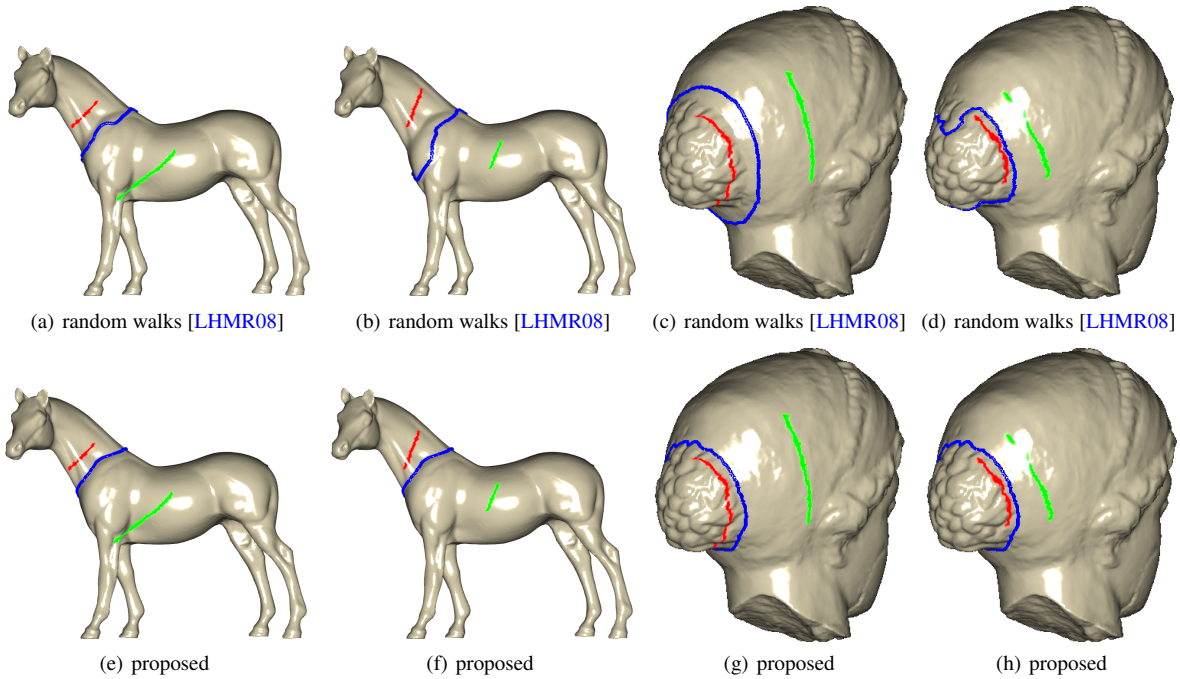
Second, the cutting contours generated by the region based approaches themselves are usually jaggy (e.g Fig. 1(d)). Thus, additional boundary optimization step is often needed to smooth the cutting contour. In fact, the easy mesh cutting [JLCW06] employs a modified snake algorithm to refine the cutting contour. The random walks mesh cutting [LHMR08] uses a feature sensitive smoothing method to smooth the jaggy boundary produced by the random walks algorithm itself. However, these additional boundary optimization methods are supplementary steps, and they are able to change the contour locally for smoothness but incapable of evolving the entire contour to snap to geometry features/edges. In addition, the existing boundary optimization methods have some limitations. It is well known that the geometric snake algorithm [LL02] cannot deal with the topology change problem and introduces parameterization artifacts for keeping the updated contour remaining on the mesh model. The shortest path method based on Dijkstra's algorithm and the graph cut algorithm can be applied here for cutting contour optimization. However, it is hard for them to control the overall contour smoothness while keeping the contour snapping to geometry features. Besides, it is not trivial to find the solution for the graph cut algorithm in a fast manner.

Therefore, it is highly desired to have a "strong" cutting contour optimization method, which can evolve the contour entirely to absorb the non-robust performance of the region based approaches while keeping the contour smooth and snapping to geometry features. The geodesic curvature flow is the one that can meet our goals. The geodesic curvature flow describes how a closed curve evolves to a local optimal one that has minimal weighted curve length. It has been widely used in the applications of image processing [SK07, WT09]. Due to the complexity of 3D surfaces, until recently a feasible approach named *discretized geodesic curvature flow* (dGCF) [WT09] was proposed to compute the geodesic curvature flow on triangular meshes.

Inspired by the dGCF method, in this paper we develop a geodesic curvature flow based interactive mesh cutting framework named mesh snapping. In particular, we use a level set formulation of the geodesic curvature flow and set the cutting contour to be the zero level set of the flow function. By observing the slow processing speed of dGCF, we propose a new and fast computation scheme called *fast geodesic curvature flow* (FGCF) for interactive mesh cutting. In addition, based on the types of seeds specified by the user, the original random walks algorithm is modified to compute a flow function value for each vertex, which is then treated as the initial geodesic curvature flow function for FGCF. By setting a feature sensitive weight to each triangle on the mesh, our FGCF scheme is able to find weighted-length local minimum near the initial contour. The closed curve obtained by FGCF is more coherent with the human perception because of its local minimum property and the feature sensitive weight for each triangle. We also develop a local editing tool to allow the user to slightly edit the cutting contour if he/she is not fully satisfied with the current result. Experimental results show that, compared to the existing interactive mesh segmentation algorithms such as easy mesh cutting, intelligent scissors, mesh scissoring and random walks [JLCW06, FKS*04, LLS*05, LHMR08], our proposed mesh snapping framework is more effective and robust to different user inputs.

Compared with dGCF [WT09], the proposed FGCF introduces an effective weight to each triangle, which makes the segmentations come close to the ones from user studies, and improves the computation of the geodesic curvature flow by not only an efficient initialization via a modified random walks approach but also a fast computation method based on symmetrizing the underlying linear system and reducing the number of unknowns. Our framework is further implemented on GPU that leads to a processing speed near to instantaneous feedback.

The rest of the paper is organized as follows. Section 2 describes the level set formulation of the geodesic curvature flow and points out the challenges of applying the geodesic curvature flow for interactive mesh cutting. Some notations and definitions are also introduced in Section 2. In Section 3, we first review the dGCF algorithm and then present the pro-



(a) random walks [LHMR08] (b) random walks [LHMR08] (c) random walks [LHMR08] (d) random walks [LHMR08]

(e) proposed (f) proposed (g) proposed (h) proposed

Figure 1: *For the random walks algorithm [LHMR08], different inputs always result in different cutting contours, but our proposed mesh snapping framework can cut at the same place although inputs are different. Moreover, our cutting contour is able to snap to geometry features. The foreground and background strokes are in red and green respectively, and the cutting contour is in blue.*

posed FGCF method. In Section 4, we describe a variant of the random walks algorithm, which is used to compute the initial flow function. The local editing algorithm is introduced in Section 5. Experimental results are shown in Section 6. Finally, we summarize the contributions of the paper in Section 7 with some discussions on the limitations.

2. Problem Formulation

2.1. Geodesic Curvature Flow over Smooth Surfaces Using A Level Set Formulation

The geodesic curvature flow describes the curve evolution under a geodesic curvature dependent velocity. It has two totally different frameworks, the Lagrangian framework and the Eulerian framework(also known as level set formulation [OS88]). See [WT09] for details. The Lagrangian framework handles both open and closed curves quite well, but suffers from the difficulty of changing the curve topology. The Eulerian framework works particularly well for closed curve evolution and benefits from its flexible topology adaptivity. These characteristics quite match the requirements of mesh cutting. For example, the cutting contour of mesh cutting is always closed. Thus, we adopt the Eulerian framework for mesh cutting in this research.

Assume $\mathcal{M}$ is a general 2-dimensional manifold embedded in $\mathbb{R}^3$, and ∇, div are the intrinsic gradient and divergence operators on $\mathcal{M}$ respectively [WDCT09]. Suppose that $C \subset \mathcal{M}$ is a curve defined on $\mathcal{M}$, represented by the zero level set of a flow function $\phi : \mathcal{M} \to \mathbb{R}$.

The usual geodesic curvature flow decreases the length of C, i.e. $\int_C dl$. To incorporate the surface feature into our algorithm, here we consider a general geodesic curvature flow which decreases a weighted curve length, where the weight is denoted as $g : \mathcal{M} \to \mathbb{R}^+$, a positive scalar function. Naturally the function g depends on the surface features. By the Co-area Formula [Fed59, MMTD07], the weighted length of C can be derived as [WT09]

$$E(C) = \int_C g dl = \int_{\mathcal{M}} g \, |\nabla \phi| \, \delta(\phi) d\mathcal{M}, \qquad (1)$$

where ϕ is the flow function and $\delta(\cdot)$ is the Dirac function. (1) basically converts an integration over a curve into another one over the surface.

By using variational techniques as in [CBMO02], we obtain the following gradient descent equation (*geodesic curvature flow*)

$$\begin{cases} \frac{\partial \phi}{\partial t} = |\nabla \phi| \mathrm{div}(g \frac{\nabla \phi}{\sqrt{|\nabla \phi|^2 + \beta}}) \\ \phi(t)|_{t=0} = \phi^0 \end{cases}, \qquad (2)$$

where ϕ^0 is an initial flow function, and β is a small positive number introduced to avoid division by zero. This flow was discussed respectively in [CBMO02, SK07, WT09] on implicit (with $g = 1$), parametric and triangulated manifolds. (2) tells that given an initial function ϕ^0, the flow function $\phi(t)$ could be recursively evolved into an optimal one ϕ^*, whose curve C^* represented by the zero level set of ϕ^* has local minimal weighted curve length.

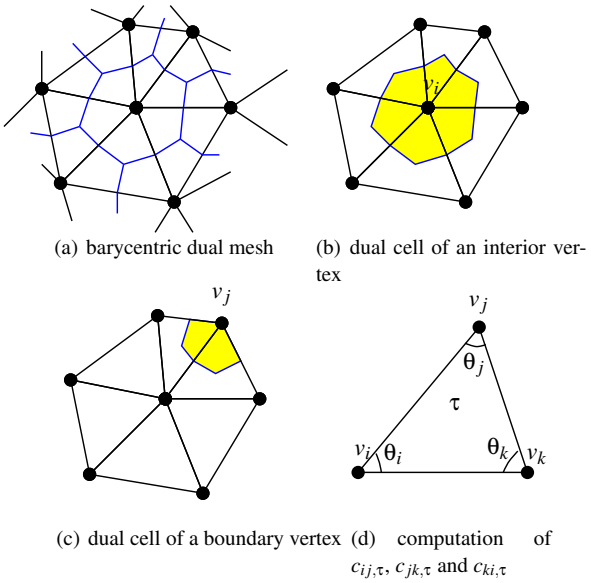
Applying such a geodesic curvature flow framework for segmentation on triangular meshes is not an easy task. First, triangular meshes are not smooth surfaces, and the discretization of the geodesic curvature flow is not straightforward. Second, the initial flow function ϕ^0 is important. Although the geodesic curvature flow framework is able to reliably find a smooth cutting boundary respecting geometry features, it is still a local optimal boundary curve. A poor initial flow function could lead to an undesired cutting boundary. Thus, the initial flow function ϕ^0 should provide a reasonable semantic distance for any point on the surface to the user specified seeds. In addition, the zero level set of ϕ^0 should be somewhere "close" to the desired cut.

2.2. Notation

Before presenting the discretization of geodesic curvature flow and the method to obtain the initial flow function, we first give some notations that will be used throughout the paper. Assume that $M \subset \mathbb{R}^3$ is a compact triangulated surface of arbitrary topology with no degenerate triangles. The set of vertices, the set of edges, and the set of triangles of M are denoted as $V = \{v_i : i = 0, 1, \cdots, |V|-1\}$, $E = \{e_i : i = 0, 1, \cdots, |E|-1\}$, and $T = \{\tau_i : i = 0, 1, \cdots, |T|-1\}$, where $|V|$, $|E|$, and $|T|$ are respectively the numbers of vertices, edges, and triangles. We explicitly denote an edge e whose endpoints are v_i and v_j by $[v_i, v_j]$. Similarly a triangle τ whose vertices are v_i, v_j, v_k is denoted by $[v_i, v_j, v_k]$. If e is an edge of a triangle τ, then we denote it as $e \prec \tau$. Let $N_k(i)$ be the k-ring neighborhood of vertex v_i and $D_1(i)$ be the 1-disk of the vertex v_i.

Now we introduce the concepts of dual meshes and dual cells [MDSB02, WT09] (see Fig. 2). For any triangular mesh surface, a *barycentric dual* is formed by connecting the midpoint of each edge with the barycenters of each of its incident faces, as illustrated in Fig. 2 (a), where the original mesh consists of black lines while the dual mesh is in blue. Using the dual mesh, the *dual cell* of a vertex v_i is defined as part of its 1-disk that is near to v_i in the dual mesh. Fig. 2 (b) shows the dual cell C_i for an interior vertex v_i of the original mesh, and Fig. 2 (c) shows the dual cell for a boundary vertex.

For each vertex v_i, let φ_i denote the usual hat function, which is linear over each triangle and $\varphi_i(v_j) = \delta_{ij}$, $i, j \in V$, where δ_{ij} is the Kronecker delta. The functions $\{\varphi_i : i \in V\}$ have three properties: local support, nonnegativity and partition of unity (see [WT09] for more details). A function u defined over the triangulated surface M is considered to be a *piecewise linear function* if u reaches value u_i at vertex v_i, $i \in V$ and $u(p) = \sum_{0 \le i \le |V|-1} u_i \varphi_i(p)$ for any $p \in M$. Besides, one may have *piecewise constant functions* over M, i.e. a value is assigned to each triangle of M.



(a) barycentric dual mesh (b) dual cell of an interior vertex (c) dual cell of a boundary vertex (d) computation of $c_{ij,\tau}$, $c_{jk,\tau}$ and $c_{ki,\tau}$

Figure 2: *Barycentric dual mesh, dual cells and computation of coefficients.*

3. Discretization of Geodesic Curvature Flow

We now consider the discrete setting, where $\mathcal{M}$ is triangulated to be $M \subset \mathbb{R}^3$. We set ϕ to be a piecewise linear function, which interpolates function values at vertices of M, as defined in Section 2.2. In other words, we only need to compute the value of ϕ at each vertex. For simplicity, the weight function $g(\cdot)$ is set to a piecewise constant function as defined in Section 2.2, i.e. $g(\cdot)$ is a constant for each triangle. Under these settings, the curve C representing the zero level set of ϕ is also piecewise linear and hence the union of some line segments.

In this section, we first review a previous discretization of the geodesic curvature flow equation in the Eulerian framework on triangular meshes, which is named *discretized geodesic curvature flow* (dGCF) [WT09]. By noticing dGCF's slow processing speed that is not suitable for interactive mesh cutting, we modify the dGCF method into a fast computation scheme called *fast geodesic curvature flow* (FGCF) in Subsection 3.2. Finally, the feature sensitive weight function $g(\cdot)$ is described in Subsection 3.3.

3.1. Previous Method: dGCF

The dGCF [WT09] is derived via a semi-implicit finite volume method (FVM) of discretization of (2). In particular, for

each vertex v_i of M, the two sides of (2) are integrated on the dual cell C_i:

$$\int_{C_i} \frac{\partial \phi}{\partial t}\, dA = \int_{C_i} |\nabla\phi| \mathrm{div}(g\widehat{\nabla\phi})\, dA, \tag{3}$$

where $\widehat{\nabla\phi} = \frac{\nabla\phi}{\sqrt{|\nabla\phi|^2+\beta}}$. We first approximate the $|\nabla\phi|$ outside of the div operator by $(|\nabla\phi|)_i = \frac{\sum_{\tau\in D_1(i)} |\nabla\phi|_\tau |\, s_\tau}{\sum_{\tau\in D_1(i)} s_\tau}$, where s_τ is the area of triangle τ and $\nabla\phi|_\tau$ is the gradient on triangle τ, whose computation is referred to [WDCT09]. Then we have

$$\int_{C_i} \frac{\partial \phi}{\partial t}\, dA = \frac{\sum_{\tau\in D_1(i)} |\nabla\phi|_\tau |\, s_\tau}{\sum_{\tau\in D_1(i)} s_\tau} \int_{C_i} \mathrm{div}(g\widehat{\nabla\phi})\, dA. \tag{4}$$

By the divergence theorem, we obtain

$$\int_{C_i} \mathrm{div}(g\widehat{\nabla\phi})\, dA = \sum_{\tau=[v_i,v_j,v_k]\in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau|^2+\beta}} (\phi_i c_{ii,\tau} + \phi_j c_{ij,\tau} + \phi_k c_{ik,\tau}), \tag{5}$$

where $c_{ij,\tau} = \frac{1}{2}\cot\theta_k, c_{ik,\tau} = \frac{1}{2}\cot\theta_j, c_{ii,\tau} = -c_{ij,\tau} - c_{ik,\tau}$ as shown in [MDSB02, WDCT09] (also see Fig. 2 (d)).

Thus with a semi-implicit time integral (from t^n to t^{n+1}), (4) becomes

$$s_i \frac{\phi_i^{n+1} - \phi_i^n}{t^{n+1} - t^n} = \frac{\sum_{\tau\in D_1(i)} |\nabla\phi|_\tau^n | s_\tau}{\sum_{\tau\in D_1(i)} s_\tau} \times \sum_{\tau\in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2+\beta}} (\phi_i^{n+1} c_{ii,\tau} + \phi_j^{n+1} c_{ij,\tau} + \phi_k^{n+1} c_{ik,\tau}), \tag{6}$$

where s_i is the area of the dual cell of v_i.

Denoting $\Phi = (\phi_0, \phi_1, \cdots, \phi_{|V|-1})'$, the above equation is reformulated into a matrix form

$$(S + (t^{n+1} - t^n)\, G(\Phi(t^n))\, H(\Phi(t^n))\,)\, \Phi(t^{n+1}) = S\, \Phi(t^n) \tag{7}$$

where $S = \mathrm{diag}(s_0, s_1, \cdots, s_{|V|-1})$ and $G(\Phi(t^n)) = \mathrm{diag}(\frac{\sum_{\tau\in D_1(0)} |\nabla\phi|_\tau^n | s_\tau}{\sum_{\tau\in D_1(0)} s_\tau}, \frac{\sum_{\tau\in D_1(1)} |\nabla\phi|_\tau^n | s_\tau}{\sum_{\tau\in D_1(1)} s_\tau}, \cdots, \frac{\sum_{\tau\in D_1(|V|-1)} |\nabla\phi|_\tau^n | s_\tau}{\sum_{\tau\in D_1(|V|-1)} s_\tau})$ are two diagonal matrices, and $H(\Phi(t^n)) = (-h_{ij}^n)$ with

$$h_{ij}^n = \begin{cases} \sum_{\tau, e=[v_i,v_j]\prec\tau} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2+\beta}} c_{ij,\tau}, & j \in N_1(i) \\ \sum_{\tau\in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2+\beta}} c_{ii,\tau}, & j = i \\ 0, & \text{others} \end{cases}. \tag{8}$$

As proved in [WT09], matrix $H(\Phi(t^n))$ is symmetric and semi positive-definite. This gives the existence and uniqueness of the solution to (7). See [WT09] for details. Note that although both $G(\Phi(t^n))$ and $H(\Phi(t^n))$ are symmetric, their product of $G(\Phi(t^n))\, H(\Phi(t^n))$ is usually nonsymmetric. Thus, (7) is a nonsymmetric linear system.

3.2. Fast geodesic curvature flow: FGCF

As one can see, the computation complexity of geodesic curvature flow based algorithms depends heavily on how to solve the linear system (7). The dGCF scheme in [WT09] solves (7) directly, which is a nonsymmetric linear system with the number of vertices as the problem dimension for each iteration. Such an approach results in an average of over 20 s to segment one model in our experiments as shown in Table 1, which is unacceptable. In this subsection, we develop a new and fast computation scheme named fast geodesic curvature flow (FGCF) to solve (7), which dramatically decreases the computation cost compared to the dGCF. Our basic idea is to symmetrize the coefficient matrix and reduce the problem dimension.

For the flow function $\phi(t^n)$, we divide all the vertex indices into two sets: K and L, where $K = \{i | \phi_j^n = \phi_i^n, \forall j \in N_1(i)\}$ and $L = \{0, 1, \cdots, |V|-1\} \backslash K$. Specifically, K is the set of vertex indices whose corresponding vertices have zero flow function gradient and L is the index set for the rest of vertices. According to the definitions of $G(\Phi(t^n))$ and K, clearly $G_{ii} = 0$ for $i \in K$. We now discuss how to optimize dGCF in the following two cases: K is empty and K is nonempty.

When K is empty, $G_{ii} > 0, \forall i$. Hence $G(\Phi(t^n))$ is invertible. By multiplying $G^{-1}(\Phi(t^n))$ on both sides of (7), we obtain

$$\begin{aligned} &(G^{-1}(\Phi(t^n))\, S + (t^{n+1} - t^n)\, H(\Phi(t^n))\,)\, \Phi(t^{n+1}) \\ &= G^{-1}(\Phi(t^n))\, S\, \Phi(t^n). \end{aligned} \tag{9}$$

This is a new linear system equivalent to (7). Moreover, the coefficient matrix in (9) becomes symmetric in addition to the sparse and positive-definite properties.

We now consider the case where K is nonempty. As we know, $G_{ii} = 0$ if $i \in K$. Thus, the i-th equation of the system (7) becomes $s_i\ \phi_i(t^{n+1}) = s_i\ \phi_i(t^n)$, which indicates that $\phi_i(t^{n+1}) = \phi_i(t^n)$ for $i \in K$. This means that for those equations whose indices belong to K, there is no need to do the computation since their flow function values remain unchanged. Therefore, by removing those equations, we reduce the number of unknowns and simplify (7).

In particular, we decompose S, $G(\Phi(t^n))$, $H(\Phi(t^n))$ and $\Phi(t^n)$ into the following forms (with index permutations if needed):

$$S = \begin{pmatrix} S_K & 0 \\ 0 & S_L \end{pmatrix},$$

$$G(\Phi(t^n)) = \begin{pmatrix} G_K(\Phi(t^n)) & 0 \\ 0 & G_L(\Phi(t^n)) \end{pmatrix},$$

$$H(\Phi(t^n)) = \begin{pmatrix} H_K(\Phi(t^n)) & B(\Phi(t^n)) \\ B^T(\Phi(t^n)) & H_L(\Phi(t^n)) \end{pmatrix},$$

$$\text{and} \qquad \Phi(t^n) = (\Phi_K(t^n) \quad \Phi_L(t^n))'.$$

Replacing S, $G(\Phi(t^n))$, $H(\Phi(t^n))$ and $\Phi(t^n)$ in (7) with the above expressions and considering that $\Phi_K(t^n) = \Phi_K(t^{n+1}), G_K(\Phi(t^n)) = 0$, we rewrite (7) as

$$\begin{aligned} &(S_L + (t^{n+1} - t^n) G_L(\Phi(t^n)) H_L(\Phi(t^n)))\ \Phi_L(t^{n+1}) \\ &= S_L \Phi_L(t^n) - (t^{n+1} - t^n) G_L(\Phi(t^n)) B^T(\Phi(t^n)) \Phi_K(t^n). \end{aligned} \tag{10}$$

Since the diagonal matrix $G_L(\Phi(t^n))$ is positive-definite, we then multiply $G_L^{-1}(\Phi(t^n))$ to both sides of (10) and obtain

$$\begin{aligned} &(G_L^{-1}(\Phi(t^n)) S_L + (t^{n+1} - t^n) H_L(\Phi(t^n)))\ \Phi_L(t^{n+1}) \\ &= G_L^{-1}(\Phi(t^n)) S_L \Phi_L(t^n) - (t^{n+1} - t^n) B^T(\Phi(t^n)) \Phi_K(t^n). \end{aligned} \tag{11}$$

The coefficient matrix of (11) is also sparse, symmetric and positive-definite. In addition, (11) has a smaller number of unknowns than (7).

So far we have reformulated the original system (7) into (9) and (11) for the two cases of $K = \emptyset$ and $K \neq \emptyset$ respectively. In our implementation, we only need to solve (11) since (9) is a special case of (11) with $K = \emptyset$. As stated in [PTVF92], it is easier to solve a linear system with symmetric positive-definite coefficient matrix than to solve a system with nonsymmetric positive-definite matrix if the number of unknowns are the same. Thus, not to mention the reduced number of unknowns, the new system (11) definitely has lower computational complexity than the original system (7). Moreover, as stated in [WT09], the dGCF has the regularization behavior that the flow function tends to be piecewise constant during the curve evolution and hence the size of K becomes larger and larger. This means that the number of unknowns (the size of L) gets smaller and smaller along the iterations of FGCF. Thus, the complexity of the original linear system solved in the dGCF is further reduced through reducing the dimension of the linear system. See Table 1 for a comparison of processing speed.

3.3. The weight g

The weight function $g(\cdot)$ in (1) is very important since it affects the final result of the geodesic curvature flow, the zero level set of which is expected to respect local geometry features and reflects human shape perception. Thus, we set the weight for each triangle on the mesh according to its geometric properties and the minima rule [HR84], which states that human perception tends to divide a surface into parts along minimum negative curvatures. In particular, for triangle τ_k not containing any seed vertex (specified by the user), we set

$$g|_{\tau_k} = \frac{1}{1 + \sum_{i=1}^{3} \lambda_{k,i} ||N(\tau_k) - N(\tau_{k,i})||^2}, \tag{12}$$

where $\tau_{k,i}$, for $i = 1,2,3$, are the triangles sharing edges with τ_k, $N(\tau_k)$ and $N(\tau_{k,i})$ respectively denote the normal vectors of triangle τ_k and $\tau_{k,i}$, and $\lambda_{k,i}$ is a scaling factor. It can be seen that the weight function $g(\cdot)$ is monotonically decreasing with respect to the absolute normal difference. The scaling factor $\lambda_{k,i}$ is set according to the minima rule: $\lambda_{k,i} = 5$ if the edge shared by τ_k and $\tau_{k,i}$ is a concave edge; otherwise, $\lambda_{k,i} = 1$. For those triangles containing seed vertices, the weight g is set to a big value (10 in our work), since we wish to prevent the cutting contour from passing through these triangles.

4. A Variant of Random Walks on Triangular Mesh

We have presented a fast way to compute the geodesic curvature flow. Now the remaining question is how to find a good initial flow function ϕ^0 (or Φ^0 in vector form), which can provide a reasonable semantic distance for any vertex on the surface to the user specified seeds and the zero level set of which should be somewhere "close" to the desired cut. In this research, we adopt the random walks algorithm to find the initial flow function. This is because the random walks algorithm is extremely efficient and is able to generate reasonable cutting results with little user input.

Random walks algorithm was first proposed by Grady [Gra06] for image segmentation. It is typically used with a simple user interface: the user draws strokes specifying seeds for "foreground" (i.e. the part to be cut out) and "background" (i.e. the rest). The same idea has then been applied on mesh segmentation [LHMR08]. Unlike the work in [LHMR08] which computes a probability value for each triangle, in this paper we propose a variant of random walks algorithm which computes a flow function value for each vertex on mesh. There are a few advantages of computing a value for each vertex instead of each triangle face. First, the number of vertices is roughly one half of the number of the faces and thus this significantly reduces the size of the linear system derived from the random walks. Second, providing each vertex a flow value facilitates the cutting contour going through the interior of the existing triangles by the subsequent FGCF algorithm.

In particular, for a triangular mesh M, we consider that all the edges are undirected and each edge $e = [v_i, v_j]$ is assigned a weight w_{ij}, which stands for the similarity between v_i and v_j. Based on the user input strokes, two sets of seeds, foreground seeds F and background seeds B, are specified, where $F \subset V$ and $B \subset V$, and $F \bigcap B = \emptyset$. We set $\phi_i^0 = 1$ for any $v_i \in F$ and $\phi_i^0 = -1$ for any $v_i \in B$. According to the random walks algorithm [Gra06], for the rest of vertices, we have $\phi_i^0 = \frac{1}{d_i} \sum_{v_j \in N_1(i)} w_{ij}\ \phi_j^0, \quad \forall v_i \in V \setminus (F \cup B)$, where

$d_i = \sum_{v_j \in N_1(i)} w_{ij}$. This leads to a system of linear equations with $\phi_i^0, v_i \in V \setminus (F \cup B)$ as unknowns. Solving the equations gives the initial flow function values for all the vertices. The initial cutting contour is therefore the zero level set of ϕ^0.

In the random walks algorithm, the weights assigned to edges have an important impact on the result. To take into account geometry features and the minima rule [HR84], we define a distance for edge $e = [v_i, v_j]$ as

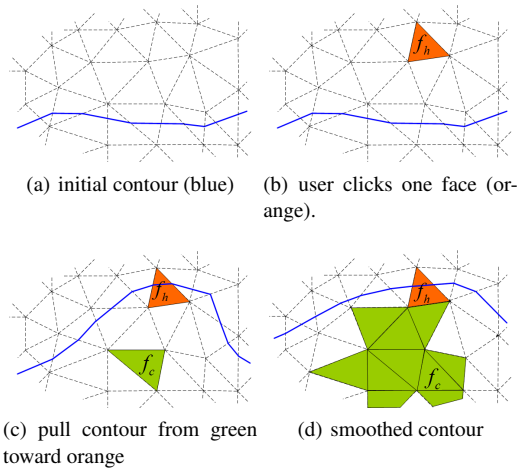
$$d_{ij} = \eta \, ||v_i - v_j|| \cdot ||N(v_i) - N(v_j)|| \qquad (13)$$

where $N(v_i)$ and $N(v_j)$ are respectively the normals of the surface at vertices v_i and v_j, $||v_i - v_j||$ is the Euclidean distance between v_i and v_j, and η is a scaling factor. The scaling factor η is set according to the minima rule: $\eta = 5$ if $e = [v_i, v_j]$ is a concave edge; otherwise $\eta = 1$. Note that before we compute the distance metric (13), the mesh model is first uniformly scaled into one with a unit bounding box. Once the edge distance metrics have been computed, the edge distance d_{ij} is mapped into the weight w_{ij} by a Gaussian function $w_{ij} = exp(-(\frac{d_{ij}}{\overline{d}})^2)$, where $\overline{d}$ is the average value of d_{ij} over the entire mesh.

5. Local Editing for Cutting Contour

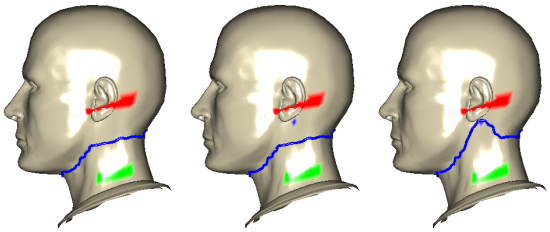
Although the combination of FGCF and the random walks algorithm can produce very robust results, sometimes it might not be exactly the one that a user wants. Different users might have different expectation on the cutting contour. Therefore, it is often necessary to devise a local editing method for the user to locally adjust the cutting contour to meet his expectation. In our system, the user is allowed to use mouse click to indicate where he wants the cutting contour lying near, and then the cutting contour will be automatically pulled toward where he clicked. This process repeats until the user is satisfied with the obtained cutting contour.

In particular, as shown in Fig. 3, assuming that the user is not satisfied with the current cutting contour 3(a) (the blue line), he wants it passing through triangle f_h 3(b). Starting from f_h, the Breadth-First-Search algorithm is first used to find the nearest triangle f_c on the cutting contour, and the number of steps from f_h to f_c is recorded as k. The flow function value of each vertex on the triangles within the k-ring of triangle f_c is subtracted by the flow function value of f_h. In the way, the flow function value of f_h becomes zero and thus the new cutting contour represented as the zero level set of ϕ passes through f_h, as shown in Fig. 3(c). However, this new contour has jaggy shape and other vertices on the cutting contour is far away from the feature positions. To solve these problems, we use a short time-step geodesic curvature flow to pull and smooth the cutting contour with the weights $g(\cdot)$ for the $(k-1)$-ring triangles of f_c (green part in Fig.3(d)) set to a big value (10 in our experiments), which prevents the final contour from passing through this green region. The final smoothed contour is shown in Fig. 3(d).



(a) initial contour (blue) (b) user clicks one face (orange). (c) pull contour from green toward orange (d) smoothed contour

Figure 3: *Illustration of the local editing algorithm.*



(a) (b) (c)

Figure 4: *Local editing example. (a) Initial result; (b) The user clicks a face where he/she wants the cutting contour to lie near; (c) The initial cutting contour is pulled toward where the user clicked.*

Note that the calculation of k-ring neighborhood of a face and the Breadth-First-Search algorithm mentioned above are performed on the dual graph of the mesh.

A local contour editing example is shown in Fig. 4. After inputting two types of seeds (in red and green respectively), the cutting contour (in blue) is then computed as shown in Fig. 4(a). If the user wants to pull the contour toward the "ear", he can select one face where he wants the cutting contour to go through or lie near as shown in Fig. 4(b). The system then uses the above local editing algorithm to compute a new cutting contour shown in Fig. 4(c).

6. Experimental Results

We now summarize the overall process of the developed mesh snapping framework for interactive mesh cutting as follows.

1. The user sketches strokes on the mesh to define two types of seeds.
2. Compute the initial flow function ϕ^0 using our modified random walks algorithm.

3. Starting from ϕ^0, we use the proposed FGCF algorithm and an adaptive time step setting strategy to evolve the zero level set of ϕ toward its nearby local minimum. The adaptive time step setting is similar with the one in [WT09], but the length of the zero level set is computed according to the value of g. The zero level set of the optimal flow function ϕ^* is then treated as the cutting contour.
4. If the user is not satisfied with the current cutting contour, he/she can use the local editing algorithm to edit the cutting contour until he/she is satisfied.

In the following, we provide some experimental results to show that our mesh snapping framework is effective and robust. In all the examples, the two types of seeds are shown in red and green respectively, and the cutting contours are in blue.

First, we test the effectiveness of our mesh snapping framework. We find that the cutting contour of our approach matches the human perception well because it computes the weighted closed geodesics (see the examples in Fig. 5 (a)(b)(c)(d)). We have tested our mesh snapping framework on around twenty models in the ground truth benchmark data set for 3D mesh segmentation [CGF09], where the ground truth results are manually generated by many people. Fig. 5(e) and 5(f) show two examples of the ground truth segmentation results. Note that the ground truth segmentation results contain different contours for cutting different parts. Even for cutting one part, the ground truth results also consist of multiple overlapped contours, which are manually drawn by different people.

We compare our cutting result with the ground truth segmentations using the *Cut Discrepancy* metric [CGF09], which evaluates how well the cutting results match the human-generated segmentations. Specifically, assuming C_1 and C_2 are the sets of all the points on the cutting contours S_1 and S_2, respectively. Denote by $d_G(p_1,p_2)$ the geodesic distance between two points on a mesh and define $d_G(p_1,C_2) = min\{d_G(p_1,p_2),\forall p_2 \in C_2\}$. The *Cut Discrepancy* is then defined as [CGF09]

$$CD(S_1,S_2) = \frac{DCD(S_1 \Rightarrow S_2) + DCD(S_2 \Rightarrow S_1)}{avgRadius} \qquad (14)$$

where *avgRadius* is the average Euclidean distance from a vertex on the surface to centroid of the mesh and $DCD(S_1 \Rightarrow S_2)$ is the mean of $\{d_G(p_1,C_2),\forall p_1 \in C_1\}$. See [CGF09] for details. Over twenty models in the benchmark data set, the average *Cut Discrepancy* between our results and the ground truth results is about 0.005. Such a small cut discrepancy value demonstrates that our cutting results are very close to the ground truth segmentations. It can also be seen by comparing our cutting results in Fig. 5(c) and 5(d) with the corresponding ground truth contours in Fig. 5(e) and 5(f).

Fig. 6 gives a comparison on the cutting contour. It can be seen that the cutting contour of the random walks algorithm

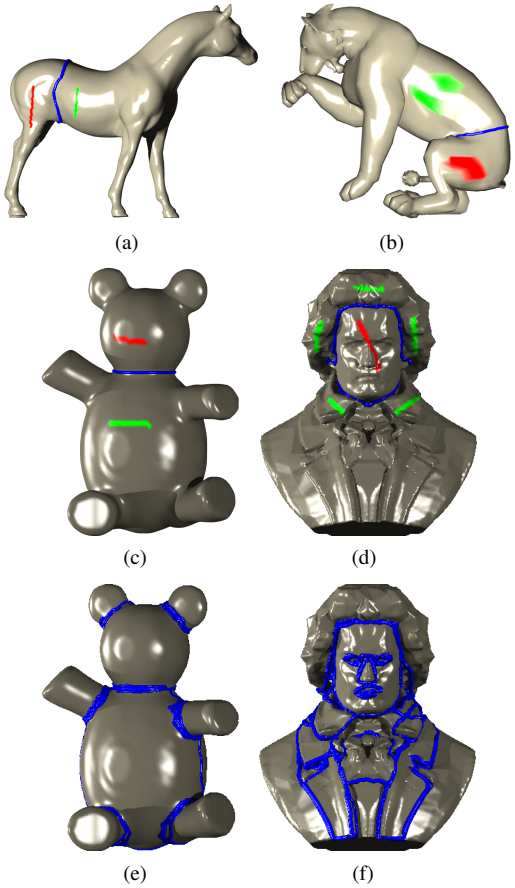
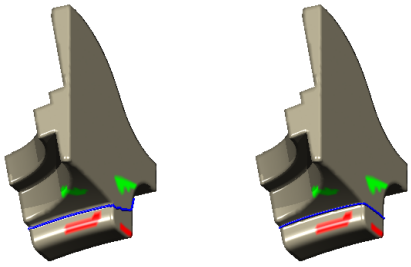


Figure 5: *(a)-(d): our results. (e) and (f): the ground truth segmentations provided by the benchmark data set [CGF09] that are collected from multiple people.*

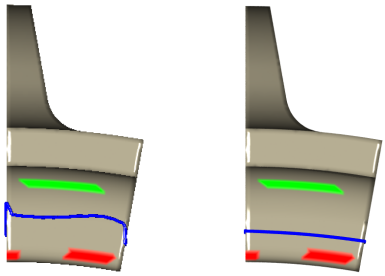
is of jaggy shape since it has no geometric meaning, while our cutting contour is smooth and along geometric edges. Such smooth and geometry feature-snapping properties of the cutting contour can also be observed in Fig. 1 and 5.

One feature we would like to highlight is that our mesh snapping framework can freely deal with the curve topology change since FGCF is a level set based method. Fig.7 shows one example, where the contour evolves from one closed curve at the beginning 7(a), to the middle result 7(b) and finally to two closed curves 7(c). Note that the geometric snake cannot deal with this type of curve position update, which is a well-known shortcoming of the "snake" model [KWT88, MSV95].

Most of the cutting contours shown so far go through concave edges. Fig. 8 shows that the cutting contour of our method can also be along non-concave edges. Although the cutting contours in Fig. 8 are not local minimums in terms of curve length, they are indeed weighted-length local minimums. This is achieved by the weight setting in Section 3.3.

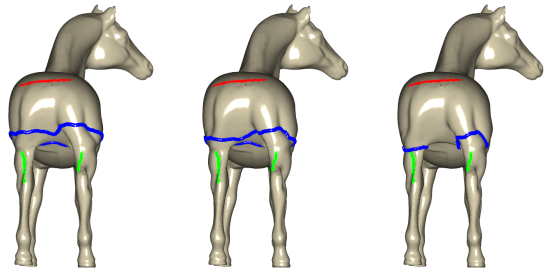


(a) random walks (b) proposed



(c) random walks (d) proposed

Figure 6: *The cutting contours of the random walks algorithm [LHMR08] are of jaggy shape, while our cutting contours are smooth and along geometric edges. The first and the second rows are for two different views.*

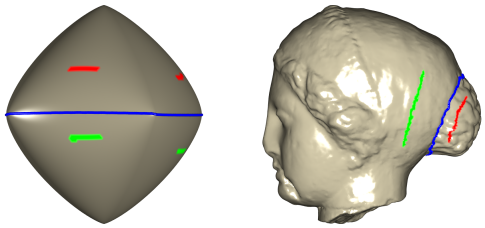


(a) initial contour (b) middle contour (c) final contour

Figure 7: *Due to the level set formulation, our algorithm can easily handle topology changes like splitting.*

Second, we test the robustness of our mesh snapping framework to the user input. As mentioned at the beginning, the existing interactive mesh cutting algorithms such as [JLCW06, LHMR08] are sensitive to the user input in terms of the position or the number of seeds. For example, the result of the random walks algorithm [LHMR08] heavily depends on the position of seeds. Placing user's strokes at different locations results in quite different cutting contours, as shown in Fig. 1(a)(b)(c)(d). In comparison, as shown in Fig. 1(e)(f)(g)(h), our approach produces the same cutting contour for different user inputs.

Third, we measure the efficiency of our proposed method in term of computation time. Table (1) compares the computation time and gives the average number of unknowns



(a) (b)

Figure 8: *Although our algorithm converges into a local minimum, it can cut along non-concave edges because of the feature sensitive weight setting.*

in FGCF. Note that (7) of dGCF is solved by the preconditioned biconjugate gradient method (PBCG) [PTVF92] as [WT09] does. For a fair comparison, we use the corresponding iterative solver for symmetric matrices, i.e. the preconditioned conjugate gradient method (PCG) [PTVF92], to solve (11) of FGCF. In addition, to further improve the processing speed, (11) is also solved by the sparse direct solver Taucs [TcR03], whose complexity is linear with the non-zeros in the coefficient matrix. From Table (1), we can see that FGCF by PCG and FGCF by Taucs are more than two times and five times faster than dGCF respectively. We would like to point out that the number of unknowns in dGCF is always equal to the number of vertices while FGCF has a much smaller number of unknowns as shown in the last column of Table (1).

To reach realtime application, we have also solved the FGCF using the Jacobi-preconditioned Conjugate Gradient algorithm on the GPU [BCL09]. Specifically, we use Nvidia's CUDA programming language with the BLAS library running on an Nvidia Quadro FX 4600 graphics card. In this way, the processing can be typically accomplished within $1 \sim 2$ seconds, as shown in Table (1). All the examples were made on a PC with Intel Core 2.66GHz CPU and 2GB RAM.

Table 1: *Mesh information and running time statistics. The unknowns size in dGCF always equals the number of vertices. The ANUF stands for the average number of unknowns in FGCF iteration.*

Model	# of vertices	dGCF by PBCG(s)	FGCF by PCG(s)	FGCF by TAUCS(s)	FGCF on GPU(s)	ANUF
5(c)	13324	3.273	0.418	0.109	0.093	952
4(a)	15941	5.378	1.677	0.934	0.275	12303
8(a)	16386	6.217	2.052	0.946	0.354	13342
5(d)	25125	10.217	1.419	0.703	0.178	7807
5(b)	29299	28.185	6.741	3.331	0.756	24385
7(c)	48485	33.82	10.934	6.742	1.581	29213
8(b)	67173	62.931	27.365	10.358	4.328	56393

7. Conclusion

In this paper, we have developed a mesh snapping framework for interactive mesh cutting, whose results are robust to the user input and capable of reflecting geometric features

and human shape perception. The major contributions of this paper include the following. First, we apply the geodesic curvature flow function for interactive mesh cutting. To the best of our knowledge, it has not been done before. Second, we propose the FGCF algorithm which greatly reduces the complexity of dGCF. Together with the GPU implementation, our framework can produce the cutting results around $1 \sim 2$ seconds for most of the test models. Third, although FGCF can be used with other exiting mesh cutting algorithms, the marriage of FGCF and the random walk algorithm combines the advantages of the random walk algorithm in terms of simple user interface and fast processing speed and that of FGCF in robustness. The developed local editing tool further incorporates some flexibility into the robust mesh snapping framework.

Since our proposed mesh snapping framework emphasizes on the robustness performance, it does not provide great flexibility for the user to control the final cutting contour. Our local editing tool is only for the user to do some small adjustment locally. Thus, when there is a big gap between the cutting result and user's intention, the local editing tool does not help and the user needs to input new strokes and repeat the entire process. How to optimally tradeoff between robustness and flexibility is still an open question. Another limitation of our mesh snapping framework is that it can only handle a binary segmentation at present. It would be more interesting to extend the current framework for cutting a mesh into multiple parts.

Acknowledgements

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.

References

[BCL09] BUATOIS L., CAUMON G., LÉVY B.: Concurrent Number Cruncher - A GPU implementation of a general sparse linear solver. *International Journal of Parallel, Emergent and Distributed Systems 24* (2009), 205–223.

[CBMO02] CHENG L., BURCHARD P., MERRIMAN B., OSHER S.: Motion of Curves Constrained on Surfaces Using a Level Set Approach. *J. Comput. Phys 175* (2002), 604–644.

[CGF09] CHEN X., GOLOVINSKIY A., FUNKHOUSER T.: A Benchmark for 3D Mesh Segmentation. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3 (2009).

[Fed59] FEDERER H.: Curvature Measures. *Transactions of the American Mathematical Society 93*, 3 (1959), 418–491.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by Example. *ACM Transactions on Graphics (SIGGRAPH)* (2004), 652–663.

[Gra06] GRADY L.: Random Walks for Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11 (2006), 1768–1783.

[GSL*99] GREGORY A., STATE A., LIN M. C., MANOCHA D., LIVINGSTON M. A.: Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 9 (1999), 453–470.

[HR84] HOFFMAN D. D., RICHARDS W.: Parts of recognition. *Cognition* (1984), 65–96.

[JLCW06] JI Z., LIU L., CHEN Z., WANG G.: Easy Mesh Cutting. In *Computer Graphics Forum (Eurographics)* (2006), pp. 219–228.

[KWT88] KASS M., WITKIN A. P., TERZOPOULOS D.: Snakes: Active Contour Models. *International Journal of Computer Vision 1*, 4 (1988), 321–331.

[LHMR08] LAI Y.-K., HU S.-M., MARTIN R., ROSIN P.: Fast Mesh Segmentation using Random Walks. In *ACM symposium on Solid and physical modeling* (2008), pp. 183–191.

[LL02] LEE Y., LEE S.: Geometric Snakes for Triangular Meshes. *Computer Graphics Forum (Eurographics)*, 3 (July 2002), 229–238.

[LLS*04] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.-P.: Intelligent Mesh Scissoring Using 3D Snakes. In *Pacific Graphics* (2004), pp. 279–287.

[LLS*05] LEE Y., LEEA S., SHAMIRB A., COHEN-ORC D., SEIDELD H.-P.: Mesh Scissoring with Minima Rule and Part Salience. *Computer Aided Geometric Design*, 11 (July 2005), 444–465.

[MDSB02] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete Differential-Geometry Operator for Triangulated 2-Manifolds. In *Vis. and Math. III* (2002), Springer Verlag.

[MMTD07] MULLEN P., MCKENZIE A., TONG Y., DESBRUN M.: A Variational Approach to Eulerian Geometry Processing. *ACM Transactions on Graphics (SIGGRAPH) 26*, 3 (2007).

[MSV95] MALLADI R., SETHIAN J. A., VEMURI B. C.: Shape Modeling with Front Propagation: A Level Set Approach. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 2 (1995), 158–175.

[OS88] OSHER S., SETHIAN J. A.: Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *J. Comput. Phys 79*, 1 (1988), 12–49.

[PTVF92] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B.: *Numerical Recipes in C*, second ed. Cambrige University Press, 1992.

[SBSCO06] SHARF A., BLUMENKRANTS M., SHAMIR A., COHEN-OR D.: SnapPaste: An Interactive Technique for Easy Mesh Composition. *The Visual Computer*, 9-11 (2006), 835–844.

[SG07] SINOP A. K., GRADY L.: A Seeded Image Segmentation Framework Unifying Graph Cuts And Random Walker Which Yields A New Algorithm. Proceedings of ICCV, pp. 560–572.

[Sha08] SHAMIR A.: A survey on Mesh Segmentation Techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556.

[SK07] SPIRA A., KIMMEL R.: Geometric curve flows on parametric manifolds. *J. Comput. Phys 223* (2007), 235–249.

[TcR03] TOLEDO S., CHEN D., ROTKIN V.: Taucs: A library of sparse linear solvers. Website, 2003. http://www.tau.ac.il/~stoledo/taucs/.

[WDCT09] WU C., DENG J., CHEN F., TAI X.: Scale-Space Analysis of Discrete Filtering over Arbitrary Triangulated Surfaces. *SIAM Journal on Imaging Sciences 2*, 2 (2009), 670–709.

[WSHS98] WONG K. C.-H., SIU T. Y.-H., HENG P.-A., SUN H.: Interactive Volume Cutting. In *Graphics Interface* (1998).

[WT09] WU C., TAI X.: A Level Set Formulation of Geodesic Curvature Flow on Simplicial Surfaces. *Accepted by IEEE Transactions on Visualization and Computer Graphics* (2009).

[ZSH00] ZOCKLER M., STALLING D., HEGE H.-C.: Fast and Intuitive Generation of Geometric Shape Transitions. *The Visual Computer 16*, 5 (2000), 241–253.