Variational 3D mesh segmentation

Zhang, Juyong

2012

Zhang, J. Y. (2012). Variational 3D mesh segmentation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/48028>

<https://doi.org/10.32657/10356/48028>

NANYANG TECHNOLOGICAL UNIVERSITY

Variational 3D Mesh Segmentation

A Thesis Submitted to the
School of Computer Engineering
of the Nanyang Technological University

by

Zhang Juyong

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy (Ph.D)

2012

Abstract

3D Mesh segmentation refers to the process of extracting subparts or components from an existing mesh or decomposing a mesh model into multiple meaningful components. Mesh segmentation is of great practical importance in many mesh related processing and applications. The challenge with 3D mesh segmentation lies in the fact that the segmentation algorithm is expected to extract meaningful components or decompose a mesh into meaningful parts which are consistent with user intention, geometric mesh attributes and human shape perception. The thesis has made several contributions to address the challenge in 3D mesh segmentation.

First, we consider the problem of interactively finding the cutting contour to extract components from an existing mesh. We propose 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 mesh scissoring within the same computational framework. The experimental examples show that the proposed cutting method is fast, reliable, and capable of producing good results reflecting user intention and geometric attributes.

Second, to overcome the robustness problem of interactive mesh cutting with the input of foreground and background strokes, we propose a geodesic curvature flow based interactive mesh cutting framework. In particular, we propose to marry the existing fast region-based interactive mesh segmentation method with the geodesic curvature flow, where the former is used to generate an initial coarse cutting contour and the latter is applied to evolve the cutting contour to the desired place. Although many existing methods also consist of two separate processes for segmentation and boundary optimization,

the boundary optimization process usually plays a complementary role. On the contrary, our boundary optimization, the proposed geodesic curvature flow, plays a major role here, which is able to evolve the initial coarse contour entirely to absorb the non-robust performance of the region-based approach while keeping the contour smooth and snapping to geometry features. Experimental 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.

Third, we consider the problem of automatically decomposing a 3D mesh into meaningful parts. We present a variational mesh decomposition algorithm that is accurate and efficient. The algorithm extends the Mumford-Shah model to 3D meshes that simultaneously handle segmentation and boundary smoothing, which are usually two separate processes in most previous work. The efficiency is achieved by solving the Mumford-Shah model through a saddle-point problem that is solved by a fast primal-dual method. While the proposed algorithm can automatically segment a mesh into meaningful parts, user interaction is allowed by incorporating user’s inputs into the variational model to reflect user’s special intention. Also, a strategy is proposed to automatically determine the number of segments that the mesh should be decomposed into. Experimental results show that the proposed algorithm outperforms competitive segmentation methods when evaluated on the Princeton Segmentation Benchmark.

Acknowledgments

I thank my advisors, Prof. Cai Jianfei and Prof. Zheng Jianmin, for providing fertile ground on which this thesis could grow. Especially, I want to thank that they gave me privilege to do any research I like, and the challenging problems they put forward during the discussions pushed me go further and benefited me a lot. Their encouragement and support strengthen my faith to keeping on the research path, and I hope that our cooperation will continue in the future.

I am deeply indebted to my friend Dr. Wu Chunlin, in particular, for helping me study optimization techniques and numerical PDEs, and numerous discussions which tremendously helped to clarify the theoretical part of this thesis.

Many thanks to the seniors: Guan Wei, Jiang Xing, Yang Wenxian, and Zhang Yu. They gave me many useful suggestions and advices on how to do research and how to spend the boring Ph.D student life. I am thankful to other colleagues: Cheng Xiangang, Duan Qi, Duan Lixin, Guo Tiantian, Liu Yong, Lu Haifeng, Wu Min, and Zhuang Jinfeng for their kind support and assistance. They have helped provide a supportive and enjoyable research environment.

Last but not the least, I would like to extend deepest appreciations to my beloved families and girlfriend for their lasting encouragement and support that make my heart strong and make my life beautiful.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
List of Tables	xi
1 Introduction	1
1.1 Background	1
1.2 Objectives and Scope	3
1.3 Contributions & Thesis Organization	4
2 Literature Review	7
2.1 Segmentation Type	7
2.2 Attributes and Partitioning Criteria	8
2.3 Segmentation Techniques	9
2.3.1 Interactive Mesh Segmentation	10
2.3.2 Automatic Mesh Segmentation	13
2.4 Other related works	17
3 Flexible Interactive Mesh Cutting	20
3.1 Motivation and Our Approach	20
3.2 Constrained Random Walks for Triangular Meshes	23
3.2.1 Edge weights	26
3.3 Cutting Contour Optimization	28
3.4 Interactive Mesh Cutting	32
3.4.1 Preprocessing	34
3.5 Experiments and Discussions	35

4	Mesh Snapping: Robust Interactive Mesh Segmentation	42
4.1	Motivation and Our Approach	42
4.2	Problem Formulation	46
4.2.1	Geodesic Curvature Flow over Smooth Surfaces	46
4.2.2	Notation	47
4.3	Discretization of Geodesic Curvature Flow	49
4.3.1	Previous Method: dGCF	49
4.3.2	Fast geodesic curvature flow: FGCF	51
4.3.3	The weight g	53
4.4	Preprocessing and postprocessing	54
4.4.1	Initialization	54
4.4.2	Local Editing for Cutting Contour	55
4.5	Experimental Results	57
5	Automatic Mesh Decomposition	64
5.1	Motivation and Our approach	64
5.2	Mumford-Shah Image Segmentation	67
5.3	Mumford-Shah Mesh Decomposition	69
5.3.1	Multichannel Function $\mathbf{f}(x)$	71
5.3.2	Discretization of the Mumford-Shah Model	74
5.3.3	Fast Primal-Dual Method	75
5.4	User Interaction	77
5.4.1	Mumford-Shah Model with Constraints	77
5.4.2	Constraint Propagation	78
5.5	Determination of the Number of Segments	79
5.6	Experimental Results and Discussions	81
5.6.1	Effects of Key Ingredients	82
5.6.2	Overall Performance of the Algorithm	85
5.7	Limitations	92
6	Conclusions	94
6.1	Summary	94
6.2	Future Directions	96

References	99
List of Publications	109

List of Figures

3.1	Incremental foreground/background specifications (a-e) vs foreground/background + soft constraints (f).	21
3.2	Specifying a few points (hard constraints) on the desired cutting contour helps the random walks find more accurate results.	22
3.3	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	26
3.4	A non-smooth cutting contour (top) vs a smooth cutting contour (bottom).	29
3.5	Splitting the contour area G along the blue path.	31
3.6	Automatically generating foreground/background seeds for the soft or hard constraint input only. left: soft constraint input; right: hard constraint input.	35
3.7	The cutting results using the proposed algorithm for the examples listed in Table 3.1, 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.	36
3.8	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 [1] and a better result than the easy mesh cutting [2].	37
3.9	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).	39
3.10	More complex meshes.	39

3.11	(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. . .	40
3.12	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 all the views; and (e) shows the result by our proposed algorithm with the initial foreground/background inputs and a few soft/hard constrains.	41
4.1	For the random walks algorithm [1], 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.	43
4.2	Barycentric dual mesh, dual cells and computation of coefficients.	48
4.3	Illustration of the local editing algorithm.	56
4.4	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.	57
4.5	(a)-(d): our results. (e) and (f): the ground truth segmentations provided by the benchmark data set [3] that are collected from multiple people. . .	59
4.6	The cutting contours of the random walks algorithm [1] 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.	60
4.7	Due to the level set formulation, our algorithm can easily handle topology changes like splitting.	61
4.8	Although our algorithm converges into a local minimum, it can cut along non-concave edges because of the feature sensitive weight setting.	61

5.1	Segmentation results collected from applying our proposed algorithm to the Princeton Segmentation Benchmark [Chen et al. 2009]. One mesh is shown for each category. The segmentation results match human perception well in not only the cutting boundaries but also the number of segments.	64
5.2	Plots of the eigenvectors of the proposed Laplacian matrix corresponding to the first eight nonzero eigenvalues. Note that for the fifth model, we purposely render the back side of the model to make the dark part visible.	72
5.3	Plots of the eigenvectors of the simple Laplacian matrix corresponding to the first eight nonzero eigenvalues.	72
5.4	User’s input changes the segmentation result.	77
5.5	Numbers of segments and the second order differences of RatioCut values.	80
5.6	Plots of the eigenvectors of our proposed Laplacian matrix corresponding to the first twenty nonzero eigenvalues.	83
5.7	Comparison of the eigenvectors of the simple Laplacian matrix and our proposed Laplacian matrix. We can see that our proposed Laplacian matrix better reflects the structure of the underlying models.	84
5.8	Comparison of segmentation with various choices.	86
5.9	Quantitative evaluation of segmentation results in terms of Rand Index and Consistency Error (smaller values stand for better results). For all the methods, we perform evaluations according to the protocols of [Chen et al. 2009], using all human segmentations in the Princeton Segmentation Benchmark. “SB19” and “SB3” stand for the learning based method with training set sizes of 19 and 3 (out of 20 models for each category), respectively. “M-S”, “Rand Cuts” and “Shape Diam” represent our proposed method, Randomized Cuts and Shape Diameter Function, respectively. Note that Rand Cuts is given the dominant number of segments in the human segmentations for each model. Our method significantly outperforms the two state-of-the-art geometry based methods while worse than the learning based method with large training size.	87

5.10	Visual comparisons to other segmentation methods for Table, Octopus, Hand, Human, Fourleg, Chair, Mech. Top row: results of our method. Middle row: results of Randomized Cuts, with number of segments defined as the dominant number of segments in the human segmentations for each model. Bottom row: results of Shape Diameter Function.	89
5.11	Segmentation results of the proposed algorithm on four non-benchmark models. The cutting contours are along geometry features.	90
5.12	Segmentation results of the proposed algorithm on two CAD models. Our method cuts the models along the sharp edges.	90
5.13	The influence of parameter μ in Eq. (Eq. 5.4) on the segmentation result. Big μ results in good segmentation in (a) while small μ gives the disappointing results with jaggy boundaries in (b). A must-link stroke (in blue) is drawn in (c) to obtain the rectified result in (d).	92
5.14	(a)-(d): The segmentation results of our method with the number of segments being 4, 5, 6, 7, respectively. (e)-(h): samples of the corresponding human segmentation results. (i)-(p) Color plots of the first eight nonzero eigenvectors of the Laplacian matrix for a fish model. Our method fails at this smooth model which contains some tiny segments (fins) and lacks clear geometry edges.	93

List of Tables

3.1	Running time statistics of our algorithm with and without the watershed preprocessing step.	37
4.1	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.	63
5.1	Rand Index scores for human segmentations, SB19, SB3, MS (our method), Randomized Cuts, and Shape Diameter Function. The Rand Index scores are measured against all human segmentations in the Princeton Benchmark.	88
5.2	Mesh information and running time statistics. The GPU implementation improves the processing speed by around ten times.	91

Chapter 1

Introduction

1.1 Background

Digital geometry processing is a research area that defines mathematical foundations for computational method to process geometry and develops tools and algorithms for efficient manipulation of geometric data. The field of digital geometry processing was formally established at nearly ten years ago on the vision that geometry would become the fourth type of digital medium after sound, image, and video. The past research efforts have successfully established some theoretical and algorithmic foundations to deal with this very special “signal”, i.e., geometry. This facilitates 3D geometry to play a more and more important role in many applications, including e-commerce, entertainment, industrial design, physical simulation, medicine, and education.

Mesh segmentation is a basic operation in geometry processing. It refers to the process of extracting subparts or components from an existing mesh or decomposing a mesh model into multiple meaningful components. It is also known variously as mesh cutting, mesh decomposition, mesh partitioning, and mesh scissoring. Mesh segmentation not only provides semantic information of a model for shape understanding and recognition, but also assists many geometric processing tasks [4, 5].

The segmentation operation has been proved quite useful in many mesh related processing and applications. For example, it has been used to create new models by

assembling different parts of existing mesh models [6]. Mesh decomposition followed by recognition and matching of sub-parts have been shown helpful in shape matching and retrieval, and shape reconstruction [7, 8]. Part matching can also be utilized for morphing [9]. In addition, object part decomposition can facilitate object skeleton creation [10, 11, 12, 13, 14], which is useful for deformations and animation. Bounding boxes defined around object parts can assist in fast collision detection calculations [15]. Other applications include 3D morphing [9], metamorphosis [9, 16], parameterization, texture mapping [17], simplification [18], 3D shape retrieval [8, 19], compression [20], animation, and shape analysis and understanding.

Noncountable 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 [21], fuzzy clustering [11], and spectral clustering [22, 23]. Other methods are also proposed, including watershed segmentation [24], edge contraction and space sweeping based decomposition [15], spectral embedding and contour analysis based recursive bisection [23], and hierarchical pose-invariant mesh segmentation [25]. Besides static models, there are also some studies investigating how to segment animating meshes [26, 27, 28]. Comprehensive references and comparison can be found in the excellent survey papers [4] and [5]. A benchmark for evaluation of 3D mesh segmentation algorithms is provided in [3].

In addition to automatic mesh segmentation, there are also many interactive mesh segmentation algorithms [1, 2, 6, 29, 30, 31], which involve user interactions to guide the segmentation process. Compared with automatic mesh segmentation, interactive methods are easy to achieve the results that match user intention since they allow the user to instruct or control the segmentation at the cost of requiring a certain amount of user effort.

1.2 Objectives and Scope

The challenge with 3D mesh segmentation lies in the fact that the segmentation algorithm is expected to extract meaningful components or decompose a mesh into meaningful parts which are consistent with user intention, geometric mesh attributes and human shape perception, but the concept of “meaningful” is tied to shape understanding and human knowledge that are difficult to model mathematically, making it difficult to design a “perfect” automatic algorithm. In general, a good segmentation algorithm should be able to output the results that satisfy the following criteria.

- First, the elements within the same segment should have high similarity.
- Second, the association between different segments should be low. Geometrically, the segment boundary should be tight and smooth.
- Third, the segment boundary should be kind of sharp or semi-sharp. Based on cognitive science, the human visual system perceives segment boundaries at negative minima of the principal curvatures, which is known as the minima rule [32].
- Fourth, segmentation should reflect significant features and small-scale fluctuation should be ignored. In particular, the part salience theory provides three factors to determine the salience of a part: the relative size, the boundary strength and the degree of protrusion [33].

The ultimate goal of this research is to design and develop more robust and accurate algorithms satisfying all the criteria mentioned above. We start with the relatively simple scenario of interactive mesh cutout, which is to interactively cut out a meaningful sub-mesh from its underlying mesh with the assistance of user inputs. The challenge is how 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. In particular, we first tackle the flexibility problem of interactive mesh segmentation, i.e., the existing methods are often trapped in tedious editing due to lack of flexible interface for the user to convey his intention to the underlying segmentation engine. Second, we attack the robustness problem of interactive mesh segmentation, i.e., the existing methods are sensitive to precise placement of strokes within the segments, where different user inputs resulting in different segmentation results. Last but not the least, we consider the more challenging scenario of automatic mesh decomposition, which is to automatically decompose a 3D mesh into a set of disjoint, meaningful parts.

1.3 Contributions & Thesis Organization

The thesis has made several contributions to the research of 3D mesh segmentation, including a flexible interactive mesh segmentation algorithm, a robust interactive mesh segmentation method, and a unifying segmentation and boundary optimization framework.

To overcome the problem of lacking flexibility for interactively finding the cutting contours to extract components from an existing mesh, we propose a novel user interface for interactive mesh cutting. Our proposed interface 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 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 [1, 2, 6, 29].

To overcome the reliability problem of interactive mesh cutting with the input of foreground and background strokes, we propose a geodesic curvature flow based interactive mesh cutting algorithm. In particular, we propose to marry the existing fast region-based interactive mesh segmentation method with the geodesic curvature flow, where the former is used to generate an initial coarse cutting contour and the latter is applied to evolve the cutting contour to the desired place. Although many existing methods also consist of two separate processes for segmentation and boundary optimization, the boundary optimization process usually plays a complementary role. On the contrary, our method employs a much stronger boundary evolving process. In other words, the proposed geodesic curvature flow for boundary optimization plays a major role here, which is able to evolve the initial coarse contour entirely to absorb the non-robust performance of the region based approach while keeping the contour smooth and snapping to geometry features. In addition, we also develop a fast method to compute the geodesic curvature flow on triangular meshes.

Furthermore, we consider the challenging problem of automatically decomposing a 3D mesh into meaningful parts. Unlike most of the existing methods, which handle the segmentation and the boundary optimization as two separate processes, we propose a variational mesh decomposition framework that unifies the two separate processes so

as to achieve more accurate and efficient performance. In particular, the accuracy is achieved by the proposed mesh segmentation algorithm using the convexified version of the Mumford-Shah model based on total variation. The efficiency is achieved by solving the Mumford-Shah model through a saddle-point problem that is solved by a fast primal-dual method. While the proposed algorithm can automatically segment a mesh into meaningful parts, user interaction is allowed by incorporating user's inputs into the variational model to reflect user's special intention. Also, a strategy is proposed to automatically determine the number of segments that the mesh should be decomposed into.

The rest of the thesis is organized as follows. Chapter 2 introduces the related work on both automatic mesh segmentation and interactive mesh segmentation, and the classifications of the existing methods according to different criteria. Chapters 3, 4, and 5 present the proposed flexible interactive mesh cutting algorithm, the proposed robust interactive mesh cutting approach, and the proposed unified mesh decomposition framework, respectively. Chapter 6 concludes the thesis and discusses the directions of future work.

Chapter 2

Literature Review

In this chapter, we give a review on related work. We begin by defining the two types of segmentation. Then, we introduce the common attributes and partitioning criteria used in different segmentation algorithms. After that, we review the existing segmentation techniques including interactive and automatic mesh segmentation algorithms.

2.1 Segmentation Type

There are two primarily distinct types of mesh segmentation: part-type segmentation and surface-type segmentation. Although there are no strict definitions for them, the part-type segmentation mainly aims to partition a mesh into meaningful parts, mostly volumetric parts, while the surface-type segmentation is to partition a mesh surface into patches under some criteria such as planarity, constant curvature or other patch properties.

Surface-type segmentation is often applied in the applications of texture mapping [17, 34, 35], building charts [36], geometry-image creation [37], remeshing and simplification [8, 18, 38, 39, 40]. On the other hand, part-type segmentation decomposes a 3D object into sub-meshes, where the semantic is in the sense of human perception and human understanding [32, 33, 41]. The part-type segmentation is widely used in appli-

cations such as modeling, shape matching and retrieval, shape reconstruction, skeleton creation, and collision detection. In this thesis, we focus on part-type segmentation.

2.2 Attributes and Partitioning Criteria

No matter which algorithm is used for mesh segmentation, one important and indispensable component is the criteria to decide which elements belong to the same segment. These criteria are usually based on mesh attributes and the constraints imposed on the partitioning process.

Although mesh attributes used in an algorithm depends on the segmentation application, there are some commonly used attributes including planarity of various forms, higher degree geometric proxies, difference in normals of vertices or dihedral angles between faces, curvature, geodesic distance, slippage, symmetry, convexity and concavity, medial axis and shape diameter, and motion characteristics.

The attribute of planarity assists segmentation goals such as simplification, parametrization, and texture mapping. The attribute of higher degree geometric proxies refers to non-planar geometric proxies including spheres, cylinders, cones, rolling ball blends, triangle strips and cones that can be used to cluster non-planar regions. Curvatures typically calculated by discrete approximations or local fitting is a popular attribute for many mesh segmentation algorithms. Another popular attribute is the average geodesic distance, which is the average over all the geodesic distances from each point to all other points on the mesh. The attribute of symmetry is utilized in [42] to segment a mesh into components. Medial axis and medial axis transform are important topological attributes and carry the structure and size information of an object. Shape diameter function [43] that measures the local diameter of an object at its boundary points gives a good distinction between thick and thin parts of the object, which has been use for part-type partition-

ing [31, 43]. Several methods [26, 44, 45] have used motion characteristics of vertices for segmentation in the case that the mesh being segmented is dynamic or animated.

To achieve a meaningful segmentation, in addition to various attributes, the theories of the cognitive studies have also been adopted in the partitioning criteria. For example, the minima rule states that human perception tends to divide a surface into parts along minimum negative curvatures [32], which has been used in mesh scissoring [29, 46], easy mesh cutting [2], and random walks algorithm [1]. The part salience theory determines the salience of a part based on its relative size, protrusiveness and cut strength [33, 47], and it has been used to reject counter-intuitive segmentation [29] or to guide a cut [23].

In mesh segmentation algorithms, three types of constraints are often used to restrict the final segments. This first one is the cardinality constraint that is to limit the number of elements in each segment to be balanced, so as to avoid too small or too large partitions. The cardinality constraint is typically realized by restricting the number of elements in each segment to be in some ranges or restricting the ratio between the maximum and the minimum numbers of elements in all segments. The second constraint is the geometric constraint that restricts the geometric properties of each segments such as area, diameter, and perimeter. The third one is the topological constraint that constrains each segment to be topologically equivalent to disk or to be a single connected component.

2.3 Segmentation Techniques

Techniques developed for mesh segmentation often come from related fields such as image segmentation, finite element mesh partitioning, unsupervised machine learning, and others. For example, fast geodesic curvature flow for interactive mesh segmentation can be traced back to geodesic active contour for image segmentation [48]. Spectral clustering for mesh segmentation [22] has similar idea to the spectral clustering in machine learning [49]. In this section, we mainly survey the different techniques used for interactive and

automatic algorithms in part-type mesh segmentation. It provides better understanding to the strengths and weaknesses of each technique, which lead to further improvements.

2.3.1 Interactive Mesh Segmentation

Automatic decomposition of a mesh into meaningful semantic components that match human intuition is a hard problem, as it is difficult to define a measure that captures the semantic information of a given shape, and human perception on segmentation is subjective even for the same object. Segmentation involving a little user interaction indicating user’s intention is therefore increasingly of interest. Recently, many interactive mesh segmentation algorithms [1, 2, 6, 9, 46] have been proposed. According to the type of user inputs, the existing interactive mesh segmentation algorithm can be roughly classified into the following categories.

- (i) 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 [50, 9, 16], which is also used as an interactive segmentation tool for labeling ground truth in [3]. The methods in this category directly targetting the cutting contours 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.
- (ii) 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 [29, 46, 51] 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 [52]. The intelligent scissoring [6] applies a variant of

Dijkstra’s algorithm to find the cutting contour that goes within the initial area. [53] 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 [29, 54], 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.

- (iii) 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 un-labeled vertices. The easy mesh cutting [2] 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 [55] requires the user to specify the feature points and then assign 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 [56] 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 to be 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 [1] by minimizing a Dirichlet energy [54]. The approaches of this category have the advantage that the user specification is less tedious, but they usually have difficulty in producing accurate results.

- (iv) The user is asked to draw one or more strokes across a desired cut, and the algorithm automatically returns a best cutting contour running through all the strokes. The method of cross-boundary brushes [30] belongs to this category, which finds the cutting contour among the isolines of a harmonic field driven by the user-specified strokes. However, it cannot process non-manifold mesh surfaces which do not have good harmonic fields, and its underlying region-growing algorithm always results in local minima.
- (v) The user draws a single stroke on the desired region, and the algorithm returns the desired cutting part. Paint Mesh Cutting [31] belongs to this type, which is achieved by efficient local graph-cut based optimization using the Gaussian mixture models (GMM) with the shape diameter function (SDF) metric [43]. However, it suffers from a few drawbacks due to the nature of the graph-cut optimization. For example, it is difficult to cut out a partial part from a smooth surface, and the user might have to specify many strokes in order to cut out some semantic parts from highly-detailed regions.

In addition to the above classification, according to the information contained in the user input, the existing interactive mesh segmentation methods can also be classified into two main categories: boundary-based approaches and region-based approaches. The above categories (i) and (ii) belong to boundary-based approaches, and the categories (iii) and (v) belong to region-based approaches, while category (iv) is something in-between since its user inputs contain mixed boundary and regional information. 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 region-based approaches, the most

popular user input is to casually draw two types of strokes to label some vertices or faces as foreground or background seeds, and then the algorithm completes the labeling for all unlabeled vertices or faces. The easy mesh cutting [2] and the random walks [1] are two representative region-based approaches, which has been discussed above. A comprehensive comparisons between these interactive mesh segmentation algorithms can be found in [57].

2.3.2 Automatic Mesh Segmentation

Compared with interactive methods, automatic mesh segmentation has a much longer history and numerous automatic approaches have been proposed in literature. In general, according to the techniques used, they can be classified into a few categories including clustering-based segmentation, top down approaches, spectral analysis based methods, skeleton based methods, and some other automatic mesh segmentation algorithms.

In clustering based methods, segmentations rely on hierarchical clustering or iterative clustering. For example, the K-means approach iteratively selects the representatives and performs clustering [21], while the resultant boundaries between segments are often jagged or even not accurate. Mangan et al. applied watershed segmentation algorithm for triangular meshes in [24]. Unsupervised clustering techniques like the mean shift clustering are also used to segment meshes [58, 59], but they often result in over-segmentation. Generally, this type of method searches for local optimum of each segment separately in a way of region growing, which may sometimes create unsatisfactory global results. Besides, the choice of initial representative might affect the final result significantly.

Opposite to the clustering based methods, top down approaches start from the whole object and create segments by partitioning it into two or more parts. This process continues in each part recursively until the desired number of segments is reached, or a certain tolerance is met for each part. This idea has been used in [11], where a fuzzy

clustering is performed in a hierarchical way from coarse to fine to find fuzzy components and then the boundaries between the components are computed using graph cuts. In [12], such top down scheme is used to create both a skeleton and a partitioning of 3D models simultaneously. The recursive partitioning measures the quality of the approximated skeleton, and the concavity measure of the object is controlled by a tolerance. The partitioning is created by searching for the best path that cuts the object to create an approximate convex decomposition.

Spectral analysis is another popular technique used for segmentation. Since the pioneer work [60] states the relationship between the combinatorial characteristics of a graph and the algebraic properties of its Laplacian, spectral property of Laplacian matrix has been applied in many areas such as data clustering [49], image segmentation [61], image matting [62], etc. They are all based on a common process: denoting $\{e_0, e_1, \dots, e_{n-1}\}$ as the eigenvectors of Laplacian matrix L , by embedding the graph G into the space R^d using the first d eigenvectors, one can reduce the combinatorial graph partitioning problem to a geometric space-partitioning problem. In [22], a symmetry affinity matrix W is constructed, which can be viewed as the adjacency matrix of a complete graph whose nodes are the mesh faces. K-means algorithm is employed to cluster the eigenvectors of the adjacent matrix W . It has also been extended in [23], which projects the mesh into a plane and then the outer contour of the 2D spectral embedding of the mesh is used to guide the segmentation.

There are some segmentation algorithms based on skeleton. In fact, skeleton extraction and part-type segmentation are strongly linked problems, where the two operations help and guide each other. Several methods first extract a skeleton and then impose a partitioning of the object based on a partitioning of the skeleton. In [15], an approximation of the skeleton of the mesh is first extracted. Then, a plane perpendicular to the skeleton branches is swept over the mesh and critical points are identified. Each critical

skeleton point is used to define a cut using the sweeping plane which segments the mesh to different parts. In this scheme, the segmentation is implicitly achieved by the creation of cuts. In [63], the object is approximated using bead-like primitives by first extracting a voxelized skeleton and then partitioning it. The top down method [12] mentioned above also involves skeleton creation.

There are some other ideas for mesh segmentation. For example, in [40], Ioana M. Boier-Martin converts the problem of 3D mesh segmentation into 2D image segmentation by using geometry images [64] to represent the mesh and imposing the partitioning of the image on the mesh segmentation. [1] applies random walks for mesh segmentation, where the seeds can be automatically generated based on certain assumptions. Core extraction [25] transforms the mesh vertices into a pose-insensitive representation, extracts prominent feature points and then core components, and finally refines boundaries to follow the natural seams of the mesh. The algorithm partitions segments hierarchically, and stops automatically when the current segment has no feature points or when the fraction of vertices contained in the convex hull is above a threshold. Fitting primitives [19] starts with every face in a separate segment. At each iteration, the best fitting geometric primitive (plane, cylinder, or sphere) is computed to approximate the faces in every pair of adjacent segments, and the best fitting pair is chosen for merger. Except the proposed interactive mesh segmentation method, they also put forward an automatic mesh segmentation algorithm based on minima rule and snake [51]. It first extracts features to find candidate contours based on the minima rule [32, 33]. Open contour is then automatically completed to form a loop around a specific part of the mesh. Finally snakes are used to smooth the cuts which define a part-type segmentation of the object.

Recent progress includes Shape Diameter Function [43], Randomized Cuts and Normalized Cuts [65], and learning-based segmentation [66]. Shape Diameter Function [43] is a measure of the diameter of an object's volume in the neighborhood of a point on

the surface. It produces a vector for each face indicating the probability to be assigned to each of the clusters and then the graph-cut algorithm is used to refine the segmentation by minimizing an energy function that combines the probabilistic vectors with boundary smoothness and concaveness. The general strategy of Randomized Cuts [65] is to randomize mesh segmentation algorithms to produce a function that captures the probability that an edge lies on segmentation boundary and to produce a ranked set of the most consistent cuts based on how many cuts overlap with others in a randomized set. In Normalized Cuts [65], every face of the mesh starts in its own segment, and segments are hierarchically merged in an order determined by the area-normalized cut cost. The algorithm terminates when a user-specified number of segments has been reached. Consistent segmentation and co-segmentation of a set of shapes are getting popular in the last two years. In [67], they proposed a method to segment a set of models consistently by graph clustering. Another related work is [68], which computes a consistent segmentation across all shape.

Learning-based segmentation [66] is a knowledge-driven approach, where knowledge is not coming from user interaction but from training data set in a machine learning framework. It formulates the objective as a Conditional Random Field model with terms assessing the consistency of faces with labels and terms between labels of neighboring faces. The objective function is learned from a collection of labeled training meshes. It can give very accurate segmentation results if the training set is large and accurate. One additional advantage is that it does not need any parameter tuning, which is very useful for large scale automatic mesh segmentation.

In summary, most of the existing algorithms segment a mesh based on some geometric criteria such as concavity, skeleton topology, curvature, geodesic distances, and shape diameter. These geometric quantities basically provide low-level, local geometric cues. In contrast, the learning-based segmentation method segments a mesh from a collection

of labeled training meshes. It has the advantage of learning higher-level, global cues, but requires per-category training. The state-of-the-art automatic segmentation is still far from satisfactory in terms of quality and speed, especially in the situation where the segment number is required to be determined automatically. In fact, most of the existing methods require the users to specify the number of segments. Only a few algorithms such as [25, 23, 43] can determine the number automatically.

2.4 Other related works

In this section, we highlight some techniques which are closely related to the algorithms we are going to present in the subsequent chapters.

Active contour and level set methods [69] are often used for contour evolution, where a user is generally asked to place a contour near the desired boundary and the algorithm evolves the boundary to a local energy minimum. Different terms can be used in the energy functional to achieve different effects or incorporate different domain knowledge. These methods have been extended to curve evolution on surfaces [70, 71, 72]. In [70], Kimmel studied geodesic curvature flow on parametric manifolds with applications in image processing. He proposed to bend invariant scale-space concepts of images painted on surfaces via geodesic curvature flow and presented numerical methods to solve the problem. It was also extended to parametric surfaces with even more applications in [73]. Under the level set framework, curve evolution over implicit surfaces was well studied by Cheng et al. in [71]. In Cheng's work, both the surface and the curve are represented as level sets of functions defined in R^3 , and the flow equations are solved in a narrow band of 3D Cartesian grids near the surface. Recently, Wu and Tai [72] proposed to use level set method to solve the geodesic curvature flow problem on triangulated surfaces. The geodesic curvature flow equation is first presented on general smooth manifolds via

curve energy minimization. It is then discretized by semi-implicit finite volume method on triangular mesh.

The random walks algorithm [56] is a popular technique for image segmentation due to its simplicity and fast processing speed. Given a small number of pixels with user-defined (or pre-defined) labels, one can analytically and quickly determine the probability that a random walker starting at each unlabeled pixel will first reach one of the pre-labeled pixels. By assigning each pixel to the label for which the greatest probability is calculated, a high-quality image segmentation may be obtained. Random walks has also been successfully applied in image matting [74] and mesh segmentation [1].

The Mumford-Shah model is a classical model proposed in [75] to optimally approximate a general function by piecewise smooth functions. It has been extended to image segmentation [76] and multiclass labeling [77, 78] through its special multi-phase piecewise constant form. The key merit of the Mumford-Shah model lies in the fact that it incorporates the similarity within one segment and the association between segments into one variational model. However, due to the non-convex energy function, the classical model has difficulty in getting the global minima. Later on, instead of resorting to sophisticated methods to find the minima of non-convex problems, many efforts have been made to reformulate the energy based on TV-regularizers to get a convex problem [77, 78, 79], and several optimization techniques have been developed to solve the convexified model like fast primal-dual method [77] based on primal-dual method [80], Douglas-Rachford Method [78] based on Douglas-Rachford splitting approach [81].

We would also like to give some discussions for the graph cut optimization algorithm [82], which is often used to find an optimization solution or to find the final cutting contour. For example, in fuzzy clustering [11], graph cut is applied to find the minimum cut within the fuzzy area as the final cutting contour. In Paint Mesh Cutting [31], graph-cut is used to optimize the Gaussian mixture models (GMM) on the

shape diameter function (SDF) metric of the shape to find the cut. The essential idea of graph cut is to formulate the energy minimization problem as a max-flow/min-cut problem on a well-designed graph. Although it performs well in many situations, there are a few limitations. For example, since the algorithm always returns the smallest cut, it has the “small cut” problem. Additionally, the K-way graph cuts problem is NP-Hard, requiring the use of heuristics to obtain a solution.

Chapter 3

Flexible Interactive Mesh Cutting

In this chapter, we consider how to combine multiple types of intuitive user inputs to provide more flexibilities for the user to guide the underlying segmentation algorithm to segment 3D meshes in an easy way.

3.1 Motivation and Our Approach

This research was initially inspired by the easy mesh cutting [2] and the random walks algorithm for image and mesh segmentation [1, 56]. 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 3.1, We try to cut out the bottom circle of the teapot model. Figure 3.1(a) shows the cutting result by the random walks algorithm [1] 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 3.1(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 chapter, we add two strokes (soft constraints) displayed in semi-transparency (see Figure 3.1(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 3.2 is another example of using the random walks algorithm. Figure 3.2(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 3.2(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.

Figure 3.1: Incremental foreground/background specifications (a-e) vs foreground/background + soft constraints (f).

Figure 3.2: Specifying a few points (hard constraints) on the desired cutting contour helps the random walks find more accurate results.

In this chapter, 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 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 intelligent scissors, mesh scissoring, easy mesh cutting, and random walks [6, 46, 2, 1]. Meanwhile, the efficient integration and optimization makes our algorithm fast, reliable and effective.

The main contributions of this work 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. We first propose a constrained random walk algorithm for computing probability for each vertex of a triangular mesh in Section 3.2. Then, an optimization process that finds the cutting contour is presented in Section 3.3. The work flow of our interactive cutout tool is described in Section 3.4. Section 3.5 presents experimental results demonstrating the algorithm.

3.2 Constrained Random Walks for Triangular Meshes

Consider a triangular mesh defined by a tuple $\{V, E, T\}$ of vertices $V = \{v_i \mid v_i \in \mathbb{R}^3, i = 1, \dots, m\}$, 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 $F \cap B = \emptyset$. 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

$$P(v_i) = \frac{1}{d_i} \sum_{e_{ij} \in E} w_{ij} P(v_j). \quad (\text{Eq. 3.1})$$

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 (Eq. 3.1) 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:

$$\begin{aligned} \min \quad & \sum_{e_{ij} \in E} w_{ij} (P(v_i) - P(v_j))^2, \\ \text{s.t.} \quad & P(v) = 1 \quad \text{for } v \in F \\ & P(v) = 0 \quad \text{for } v \in B \\ & P(v) = 1/2 \quad \text{for } v \in H \\ & |P(v) - 1/2| \leq \epsilon \quad \text{for } v \in S. \end{aligned} \quad (\text{Eq. 3.2})$$

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*:

$$\begin{aligned}
 \min \quad & \sum_{e_{ij} \in E} w_{ij} (P(v_i) - P(v_j))^2 + \sum_{v_i \in S} \lambda_i (P(v_i) - \frac{1}{2})^2, \\
 \text{s.t.} \quad & P(v) = 1 \quad \text{for } v \in F \\
 & P(v) = 0 \quad \text{for } v \in B \\
 & P(v) = 1/2 \quad \text{for } v \in H
 \end{aligned} \tag{Eq. 3.3}$$

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 (Eq. 3.3) 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

$$P(v_i) = \begin{cases} \frac{1}{d_i + \lambda_i} \sum_{e_{ij} \in E} w_{ij} P(v_j) + \frac{\lambda_i}{d_i + \lambda_i} \frac{1}{2}, & \text{for } v_i \in S \\ \frac{1}{d_i} \sum_{e_{ij} \in E} w_{ij} P(v_j), & \text{for } v_i \in V \setminus (F \cup B \cup H \cup S). \end{cases} \tag{Eq. 3.4}$$

This expression has a geometric interpretation (see Figure 3.3): 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 $1/2$.

Equation (Eq. 3.4) also gives an estimation of $|P(v_i) - \frac{1}{2}|$ for $v_i \in S$ as follows:

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

Therefore, given a small positive number ϵ ($< \frac{1}{2}$), if we let $\lambda_i \geq (\frac{1}{2\epsilon} - 1) d_i$, 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 $\epsilon = \frac{1}{8}$.

Figure 3.3: 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 .

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 (Eq. 3.4) 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 [56], it can be proven that L is a sparse, symmetric, and positive definite matrix. Thus the linear system (Eq. 3.4) 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 (Eq. 3.4) that $P(v_i)$ falls between 0 and 1 for any $v_i \in V$.

3.2.1 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 [2] is adapted, which considers both the isophotic metric and the minima rule. The isophotic metric was introduced by [83],

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 [32]. 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

$$d_{ij} = w_1 \|v_i - v_j\| + w_2 \|n_i - n_j\| + w_3 f(\kappa_{v_i v_j}), \quad (\text{Eq. 3.5})$$

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 [84] or [85]. The function $f(\kappa)$ is set as

$$f(\kappa) = \begin{cases} \kappa, & \kappa \geq 0 \\ 5|\kappa|, & \kappa < 0 \end{cases} \quad (\text{Eq. 3.6})$$

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 the equation (Eq. 3.5) depend on the size of the models. Thus, before we apply the distance metric (Eq. 3.5) 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 [56], we use the Gaussian function $w_{ij} = \exp(-d_{ij}^2)$, which can maximize the entropy of the resulting weights. We want to emphasize that the three terms in Eq. 3.5 complement each other because they represent different features on mesh. The first one is Eulerian distance between two vertices, second term is the differential value of distance, while the third one is the second differential of distance. These three terms cover the most important features for mesh segmentation.

3.3 Cutting Contour Optimization

While some algorithms such as in [2, 29] compute new edges for the cutting contour, we restrict ourselves to cutting only along the existing vertices and edges of a mesh in this research. 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 3.4

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.

Figure 3.4: A non-smooth cutting contour (top) vs a smooth cutting contour (bottom).

Intuitively, if a vertex $v_i \in V \setminus F \cup B \cup H$ 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 minimizes 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

$$\sum_{v_i \in C} \left\{ g\left(\left|P(v_i) - \frac{1}{2}\right|\right) + \alpha\theta_i + \beta\left(1 - \frac{f(\kappa_{i1}) + f(\kappa_{i2})}{\max(f(\kappa_{i1}) + f(\kappa_{i2}))}\right) \right\} \quad (\text{Eq. 3.7})$$

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 (Eq. 3.6), 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

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

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 the energy function (Eq. 3.7), 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 (Eq. 3.7) 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 (Eq. 3.7) 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 [86] 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 3.5 for illustration. To make the algorithm simpler and easier, we remove those edges (i.e., the dashed edges in Figure 3.5) that are on one side of the splitting path and are connected to it.

Figure 3.5: 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 3.5), 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 is denoted 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 is denoted 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 3.4 is the result generated by this approach. Compared to other optimal methods such as the snake used in easy mesh cutting [2], 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.

We should point out that different from Snake model [51] used in many interactive mesh segmentation method for smoothness, our Dijkstra algorithm based method can guarantee that the final cutting contour passes through hard constraint vertices, and find the global minimal of energy functional (Eq. 3.7).

3.4 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 images on the screen are 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 use them according to his/her preference. This provides flexibility to the user because some models or 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 3.6 (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 3.6 (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.

3.4.1 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 [24], as [87] did.

First, the watershed algorithm is performed to over-segment the mesh. To do so, the total curvature at each vertex is computed. The vertex with local minimum total

Figure 3.6: Automatically generating foreground/background seeds for the soft or hard constraint input only. left: soft constraint input; right: hard constraint input.

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.

3.5 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.

Figure 3.7: The cutting results using the proposed algorithm for the examples listed in Table 3.1, 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.

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 3.7 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 3.7. Table 3.1 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.

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

Table 3.1: Running time statistics of our algorithm with and without the watershed preprocessing step.

Figure 3.8: 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 [1] and a better result than the easy mesh cutting [2].

We then compared our algorithm with Lai et al's random walks algorithm [1] and the easy mesh cutting [2] 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 [1]. From Figure 3.8, it can be found that without soft and hard constraints our algorithm and Lai et al's algorithm give similar cutting results. This is not surprising because our algorithm without the soft/hard constraints works just as random walks. One difference between our algorithm and Lai et al's algorithm is that our method 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 specification 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 3.8(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 bound the region growing process.

Our interactive cutout algorithm supports three typical inputs and their combinations (see Figure 3.7). In general, the foreground/background seeds input is the 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/background inputs are sufficient. However, in some situations, it is difficult to obtain satisfactory results by using foreground/background inputs only. As already shown in Figure 3.1 and Figure 3.2, soft and hard constraints can be used with the foreground/background seed inputs to quickly find good cutting. Figures 3.9, 3.11 and 3.12 show another three examples where the soft/hard constraints are used to further guide the cutting. In these examples, the random walks algorithm [1] is also used to find the cutting contours, for which we add gradually 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 [54], 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.

Figure 3.9: 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).

Figure 3.10: More complex meshes.

Finally, Figure 3.10 shows some examples of cutting meshes of complex topology using our constrained random walks algorithm.

Figure 3.11: (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.

Figure 3.12: 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 all the views; and (e) shows the result by our proposed algorithm with the initial foreground/background inputs and a few soft/hard constraints.

Chapter 4

Mesh Snapping: Robust Interactive Mesh Segmentation

In the previous chapter, we have proposed an algorithm which provides a flexible method for interactive mesh segmentation. With unlimited soft/hard constraints, the flexible method in Chapter 3 can always obtain satisfactory results. However, soft/hard constraints require more user efforts and many existing methods only employ the simple foreground/background inputs. In this chapter, we study the problem of interactive mesh segmentation with only foreground and background strokes. By noticing that the existing method with such a simple input often produces non-robust results, we propose a solution to improve the robustness of interactive mesh segmentation in terms of insensitive to user inputs and snapping to geometry features.

4.1 Motivation and Our Approach

In this work, 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 [1, 2], 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 [2] heavily depends on the initial seeds' positions and is sensitive to mesh noise. The performance of the random walks [1] also heavily relies on the seeds' positions. Figure 4.1 shows two examples. It can be seen that for the random walks algorithm [1], different inputs always result in different cutting contours (see figure 4.1(a) and (b), figure 4.1(c) and (d)). The essential reason is that it minimizes a Dirichlet energy and different boundary conditions always result in different harmonic functions.

Figure 4.1: For the random walks algorithm [1], 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.

Second, the cutting contours generated by the region-based approaches themselves are usually jaggy (e.g., figure 4.1(d)). Thus, additional boundary optimization step is

often needed to smooth the cutting contour. In fact, the easy mesh cutting [2] employs a modified snake algorithm to refine the cutting contour. The random walks mesh cutting [1] 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 [51] 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 [72, 73]. Due to the complexity of 3D surfaces, until recently a feasible approach named *discretized geodesic curvature flow* (dGCF) [72] was proposed to compute the geodesic curvature flow on triangular meshes.

Inspired by the dGCF method, this work develops 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 that allows 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 [1, 2, 6, 29], our proposed mesh snapping framework is more effective and robust to different user inputs.

Compared with dGCF [72], the proposed FGCF introduces an effective weight to each triangle, which makes the segmentation 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, leading to a processing speed near to instantaneous feedback.

The rest of the chapter is organized as follows. Section 4.2 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 4.2. In Section 4.3, we first review the dGCF algorithm and then present the proposed FGCF method. In Section 4.4, we describe the preprocessing

and postprocessing steps. A variant of the random walks algorithm is used to compute the initial flow function, and the local editing algorithm is introduced. Finally, experimental results are shown in Section 4.5.

4.2 Problem Formulation

4.2.1 Geodesic Curvature Flow over Smooth Surfaces

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 [88]). See [72] for detail. 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 that $\mathcal{M}$ a general 2-dimensional manifold embedded in $\mathbb{R}^3$, and ∇, div are the intrinsic gradient and divergence operators on $\mathcal{M}$, respectively [89]. Suppose that $C \subset \mathcal{M}$ is a curve defined on $\mathcal{M}$, represented by the zero level set of a flow function $\phi : \mathcal{M} \rightarrow \mathbb{R}$.

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 $g : \mathcal{M} \rightarrow \mathbb{R}^+$, a positive scalar function. Naturally the function g depends on the surface features. By the Co-area Formula [90, 91], the weighted length of C can be derived as [72]

$$E(C) = \int_C g dl = \int_{\mathcal{M}} g |\nabla \phi| \delta(\phi) d\mathcal{M}, \quad (\text{Eq. 4.1})$$

where ϕ is the flow function and $\delta(\cdot)$ is the Dirac function. Eq. 4.1 basically converts an integration over a curve into another one over the surface.

By using variational techniques as in [71], we obtain the following gradient descent equation (*geodesic curvature flow*)

$$\begin{cases} \frac{\partial \phi}{\partial t} = |\nabla \phi| \operatorname{div} \left(g \frac{\nabla \phi}{\sqrt{|\nabla \phi|^2 + \beta}} \right) \\ \phi(t)|_{t=0} = \phi^0 \end{cases}, \quad (\text{Eq. 4.2})$$

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 [71, 72, 73] on implicit (with $g = 1$), parametric and triangulated manifolds. Eq. 4.2 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.

4.2.2 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 chapter. 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

Figure 4.2: Barycentric dual mesh, dual cells and computation of coefficients.

of M are denoted as $V = \{v_i : i = 0, 1, \dots, |V| - 1\}$, $E = \{e_i : i = 0, 1, \dots, |E| - 1\}$, and $T = \{\tau_i : i = 0, 1, \dots, |T| - 1\}$, respectively, 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 on M and $D_1(i)$ be the 1-disk of the vertex v_i .

Now we introduce the concepts of dual meshes and dual cells [72, 84] (see figure 4.2). 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 figure 4.2 (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. Figure 4.2 (b) shows the dual cell C_i for an interior vertex v_i of the original mesh, and figure 4.2 (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 [72] for more detail). 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 $u(p) = \sum_{0 \leq i \leq |V|-1} u_i \varphi_i(p)$ 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 .

4.3 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 4.2.2. 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 4.2.2, 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) [72]. 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 4.3.2. Finally, the feature sensitive weight function $g(\cdot)$ is described in Subsection 4.3.3.

4.3.1 Previous Method: dGCF

The dGCF [72] is derived via a semi-implicit finite volume method (FVM) of discretization of (Eq. 4.2). In particular, for each vertex v_i of M , the two sides of (Eq. 4.2) are integrated on the dual cell C_i :

$$\int_{C_i} \frac{\partial \phi}{\partial t} dA = \int_{C_i} |\nabla \phi| \operatorname{div}(g \widehat{\nabla} \phi) dA, \quad (\text{Eq. 4.3})$$

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 [89]. Then we have

$$\int_{C_i} \frac{\partial\phi}{\partial t} dA = \frac{\sum_{\tau \in D_1(i)} |\nabla\phi|_\tau s_\tau}{\sum_{\tau \in D_1(i)} s_\tau} \int_{C_i} \text{div}(g\widehat{\nabla\phi}) dA. \quad (\text{Eq. 4.4})$$

By the divergence theorem, we obtain

$$\int_{C_i} \text{div}(g\widehat{\nabla\phi}) dA = \sum_{\tau=[v_i, v_j, v_k] \in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau|^2 + \beta}} (\phi_i c_{ii,\tau} + \phi_j c_{ij,\tau} + \phi_k c_{ik,\tau}), \quad (\text{Eq. 4.5})$$

where $c_{ij,\tau} = \frac{1}{2} \cot \theta_k$, $c_{ik,\tau} = \frac{1}{2} \cot \theta_j$, and $c_{ii,\tau} = -c_{ij,\tau} - c_{ik,\tau}$ as shown in [84, 89] (also see figure 4.2 (d)).

Thus with a semi-implicit time integral (from t^n to t^{n+1}), (Eq. 4.4) becomes

$$s_i \frac{\phi_i^{n+1} - \phi_i^n}{t^{n+1} - t^n} = \frac{\sum_{\tau \in D_1(i)} |\nabla\phi|_\tau^n s_\tau}{\sum_{\tau \in D_1(i)} s_\tau} \times \sum_{\tau \in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2 + \beta}} (\phi_i^{n+1} c_{ii,\tau} + \phi_j^{n+1} c_{ij,\tau} + \phi_k^{n+1} c_{ik,\tau}), \quad (\text{Eq. 4.6})$$

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

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

$$(S + (t^{n+1} - t^n) G(\Phi(t^n)) H(\Phi(t^n))) \Phi(t^{n+1}) = S \Phi(t^n) \quad (\text{Eq. 4.7})$$

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

$$h_{ij}^n = \begin{cases} \sum_{\tau, e=[v_i, v_j] \prec \tau} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2 + \beta}} c_{ij,\tau}, & j \in N_1(i) \\ \sum_{\tau \in D_1(i)} \frac{g|_\tau}{\sqrt{|\nabla\phi|_\tau^n|^2 + \beta}} c_{ii,\tau}, & j = i \\ 0, & \text{others} \end{cases}. \quad (\text{Eq. 4.8})$$

As proved in [72], matrix $H(\Phi(t^n))$ is symmetric and semi positive-definite. This gives the existence and uniqueness of the solution to (Eq. 4.7). See [72] for detail. Note that although both $G(\Phi(t^n))$ and $H(\Phi(t^n))$ are symmetric, their product $G(\Phi(t^n)) H(\Phi(t^n))$ is usually nonsymmetric. Thus, (Eq. 4.7) is a nonsymmetric linear system.

4.3.2 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 (Eq. 4.7). The dGCF scheme in [72] solves (Eq. 4.7) 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 4.1, which is unacceptable. In this subsection, we develop a new and fast computation scheme named fast geodesic curvature flow (FGCF) to solve (Eq. 4.7), 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 $L = \{0, 1, \dots, |V| - 1\} \setminus K$. 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 (Eq. 4.7), we obtain

$$(G^{-1}(\Phi(t^n)) S + (t^{n+1} - t^n) H(\Phi(t^n))) \Phi(t^{n+1}) = G^{-1}(\Phi(t^n)) S \Phi(t^n). \quad (\text{Eq. 4.9})$$

This is a new linear system equivalent to (Eq. 4.7). Moreover, the coefficient matrix in (Eq. 4.9) 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 (Eq. 4.7) 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 (Eq. 4.7).

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):

$$\begin{aligned} S &= \begin{pmatrix} S_K & 0 \\ 0 & S_L \end{pmatrix}, \\ G(\Phi(t^n)) &= \begin{pmatrix} G_K(\Phi(t^n)) & 0 \\ 0 & G_L(\Phi(t^n)) \end{pmatrix}, \\ H(\Phi(t^n)) &= \begin{pmatrix} H_K(\Phi(t^n)) & B(\Phi(t^n)) \\ B^T(\Phi(t^n)) & H_L(\Phi(t^n)) \end{pmatrix}, \\ \text{and } \Phi(t^n) &= (\Phi_K(t^n) \quad \Phi_L(t^n))'. \end{aligned}$$

Replacing S , $G(\Phi(t^n))$, $H(\Phi(t^n))$, and $\Phi(t^n)$ in (Eq. 4.7) with the above expressions and considering that $\Phi_K(t^n) = \Phi_K(t^{n+1})$, $G_K(\Phi(t^n)) = 0$, we rewrite (Eq. 4.7) as

$$\begin{aligned} & (S_L + (t^{n+1} - t^n)G_L(\Phi(t^n))H_L(\Phi(t^n))) \Phi_L(t^{n+1}) \\ &= S_L \Phi_L(t^n) - (t^{n+1} - t^n)G_L(\Phi(t^n))B^T(\Phi(t^n))\Phi_K(t^n). \end{aligned} \tag{Eq. 4.10}$$

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 (Eq. 4.10) and obtain

$$\begin{aligned} & (G_L^{-1}(\Phi(t^n))S_L + (t^{n+1} - t^n)H_L(\Phi(t^n))) \Phi_L(t^{n+1}) \\ &= G_L^{-1}(\Phi(t^n))S_L \Phi_L(t^n) - (t^{n+1} - t^n)B^T(\Phi(t^n))\Phi_K(t^n). \end{aligned} \tag{Eq. 4.11}$$

The coefficient matrix of (Eq. 4.11) is also sparse, symmetric, and positive-definite. In addition, (Eq. 4.11) has a smaller number of unknowns than (Eq. 4.7).

So far we have reformulated the original system (Eq. 4.7) into (Eq. 4.9) and (Eq. 4.11) for the two cases of $K = \emptyset$ and $K \neq \emptyset$ respectively. In our implementation, we only need to solve (Eq. 4.11) since (Eq. 4.9) is a special case of (Eq. 4.11) with $K = \emptyset$. As stated in [92], 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 (Eq. 4.11) definitely has lower computational complexity than the original system (Eq. 4.7). Moreover, as stated in [72], 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 4.1 for a comparison of processing speed.

4.3.3 The weight g

The weight function $g(\cdot)$ in (Eq. 4.1) 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 [32], 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

$$g|_{\tau_k} = \frac{1}{1 + \sum_{i=1}^3 \lambda_{k,i} \|N(\tau_k) - N(\tau_{k,i})\|^2}, \quad (\text{Eq. 4.12})$$

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.

4.4 Preprocessing and postprocessing

4.4.1 Initialization

We have on 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, which has been discussed in Chapter 3. This is because the random walks algorithm is extremely efficient and is able to generate reasonable cutting results with little user input.

Unlike the work in [1] which computes a probability value for each triangle, this work proposes 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. 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 [32],

we define a distance for edge $e = [v_i, v_j]$ as

$$d_{ij} = \eta \|v_i - v_j\| \cdot \|N(v_i) - N(v_j)\|, \quad (\text{Eq. 4.13})$$

where $N(v_i)$ and $N(v_j)$ are the normals of the surface at vertices v_i and v_j , respectively, $\|v_i - v_j\|$ 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 (Eq. 4.13), 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 $w_{ij} = \exp(-(\frac{d_{ij}}{\bar{d}})^2)$, where $\bar{d}$ is the average value of d_{ij} over the entire mesh.

4.4.2 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 the 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 figure 4.3, assuming that the user is not satisfied with the current cutting contour (a) (the blue line), he wants it passing through triangle f_h (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 figure (c). However, this new contour has jaggy shape and other vertices on the cutting contour are 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 figure (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 figure (d). 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.

Figure 4.3: Illustration of the local editing algorithm.

A local contour editing example is shown in figure 4.4. After inputting two types of

seeds (in red and green, respectively), the cutting contour (in blue) is then computed as shown in figure (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 figure (b). The system then uses the above local editing algorithm to compute a new cutting contour shown in figure (c).

Figure 4.4: 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.

4.5 Experimental Results

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

- (i) The user sketches strokes on the mesh to define two types of seeds.
- (ii) Compute the initial flow function ϕ^0 using our modified random walks algorithm.
- (iii) 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 to the one in [72], 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.

- (iv) 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 drawn 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 figure 4.5 (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 [3], where the ground truth results are manually generated by many people. Figures (e) and (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 segmentation using the *Cut Discrepancy* metric [3], which evaluates how well the cutting results match the human-generated segmentations. Specifically, assuming C_1 and C_2 to be 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 [3]

$$CD(S_1, S_2) = \frac{DCD(S_1 \Rightarrow S_2) + DCD(S_2 \Rightarrow S_1)}{avgRadius}, \quad (\text{Eq. 4.14})$$

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 [3] for detail. Over the 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 figure (c) and (d) with the corresponding ground truth contours in figures (e) and (f).

Figure 4.5: (a)-(d): our results. (e) and (f): the ground truth segmentations provided by the benchmark data set [3] that are collected from multiple people.

Figure 4.6 gives a comparison on the cutting contour. It can be seen that the cutting contour of the random walks algorithm 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 figure 4.1 and 4.5.

Figure 4.6: The cutting contours of the random walks algorithm [1] 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.

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. Figure 4.7 shows one example, where the contour evolves from one closed curve at the beginning (a), to an intermediate result (b), and finally to two closed curves (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 [93, 94].

Most of the cutting contours shown so far go through concave edges. Figure 4.8 shows that the cutting contour of our method can also be along non-concave edges. Although

Figure 4.7: Due to the level set formulation, our algorithm can easily handle topology changes like splitting.

the cutting contours in figure 4.8 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 4.3.3.

Figure 4.8: Although our algorithm converges into a local minimum, it can cut along non-concave edges because of the feature sensitive weight setting.

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 [1, 2] 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 [1] heavily depends on the position of seeds. Placing user’s strokes at different locations results in quite different cutting contours, as shown in figures 4.1(a)(b)(c)(d). In comparison, as shown in figures 4.1(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 (4.1) compares the computation time and gives the average number of unknowns in FGCF. Note that (Eq. 4.7) of dGCF is solved by the preconditioned biconjugate gradient method (PBCG) [92] as [72] does. For a fair comparison, we use the corresponding iterative solver for symmetric matrices, i.e., the preconditioned conjugate gradient method (PCG) [92], to solve (Eq. 4.11) of FGCF. In addition, to further improve the processing speed, (Eq. 4.11) is also solved by the sparse direct solver Taucs [95], whose complexity is linear with the non-zeros in the coefficient matrix. From Table (4.1), 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 (4.1).

To reach realtime performance, we solved the FGCF using the Jacobi-preconditioned Conjugate Gradient algorithm on the GPU [96]. 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 (4.1). All examples were performed on a PC with Intel Core 2.66GHz CPU and 2GB RAM.

In the last, we want to talk about the limitations of our proposed method. Since our proposed mesh snapping framework emphasizes on the robustness performance, it

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

Table 4.1: 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.

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.

Automatic Mesh Decomposition

So far, we have presented our solutions to improve the flexibility and the robustness in interactive binary mesh segmentation. In this chapter, we tackle a even more challenging problem, i.e., automatic mesh decomposition.

5.1 Motivation and Our approach

Figure 5.1: Segmentation results collected from applying our proposed algorithm to the Princeton Segmentation Benchmark [Chen et al. 2009]. One mesh is shown for each category. The segmentation results match human perception well in not only the cutting boundaries but also the number of segments.

We considers the problem of decomposing a 3D mesh into a set of disjoint, meaningful

parts. Figure 5.1 shows such decomposition examples where 19 mesh models taken from the Princeton Segmentation Benchmark [3], a benchmark for evaluating 3D mesh segmentation algorithms, are automatically decomposed into several parts that are depicted by various colors. Mesh decomposition is also known as mesh segmentation and mesh partitioning. It is a fundamental operation in geometry processing and computer graphics. It not only provides semantic information about the model for shape understanding and recognition, but also assists many geometric processing tasks such as skeleton extraction [11], 3D morphing [9], texture mapping [17], and modeling by examples [6].

The challenge with automatic mesh decomposition lies in the fact that the decomposition algorithm is expected to segment a mesh into meaningful parts which are consistent with user intention, geometric mesh attributes and human shape perception, but the concept of “meaningful” and human perception are content dependent. In general, a good segmentation algorithm should at least be able to output the results that satisfy the following criteria. First, the elements within the same segment should have high similarity. Second, the association between different segments should be low. Geometrically, the segment boundary should be tight and smooth. Third, the segment boundary should match human perception. Based on cognitive science, the human visual system perceives segment boundaries at negative minima of the principal curvatures, which is known as the minima rule [32]. Fourth, segmentation should reflect significant features and small-scale fluctuation should be ignored. In particular, the part salience theory provides three factors to determine the salience of a part: the relative size, the boundary strength and the degree of protrusion [33]. Many previous segmentation algorithms were designed to meet the criteria via two separate processes: segmentation and boundary smoothing, or partially satisfy the criteria through one process. For instance, the fuzzy clustering method [11] first segments a mesh into components and then refines the boundaries in the fuzzy area, and the mesh scissoring [29] concentrates on the contours for cutting using

the minima rule and part salience. For the best performance, however, all these criteria should be taken into account within one process.

The above observation motivates us to propose a new method for mesh decomposition. The mathematical tool at the heart of the new method is the Mumford-Shah model (M-S model) that has proven successful in image segmentation [75]. The piecewise constant Mumford-Shah model contains two terms: data term and regularization term. The data term measures the consistency of each segment and the regularization term measures the boundary length, which makes the model suitable for our purpose of unifying the two processes: segmentation and boundary smoothing. However, extending the Mumford-Shah model from image segmentation to 3D mesh segmentation is not trivial for two main reasons: (i) Unlike images, meshes are irregular in connectivity and sampling; and (ii) While the image intensity is directly used in the M-S model for image segmentation, it is not clear what should be used in the M-S model for mesh segmentation. The contributions of this research are as follows:

- We present a mesh segmentation algorithm using the convexified version of the Mumford-Shah model based on total variation. We extract spectral attributes from the eigenvectors of the dual graph Laplacian matrix for the mesh and adapt the Mumford-Shah model to the spectral attributes of the mesh. The spectral attributes reflect global information of the underlying mesh. The minimal rule is respected in constructing the Laplacian matrix. The Mumford-Shah model simultaneously handles the two processes: partitioning and boundary smoothing.
- We propose a fast numerical method to solve the convexified version of the Mumford-Shah model, which involves solving two sub-problems. One has an explicit solution and the other is converted to a saddle-point problem that can be solved by the fast primal-dual method. Examples show that the numerical algorithm can quickly converge to the solution.

- We present a way to allow the user to interactively express his/her intention that some regions must be in the same segment. The Mumford-Shah model is modified to incorporate the user's intention.
- We also propose a method to determine the number of segments that the model should be decomposed into. By incorporating this method, the proposed Mumford-Shah mesh segmentation can automatically decompose a mesh into meaningful parts.

We test our method with various models and the experimental results show that our method is efficient and able to produce segmentation reflecting geometric attributes of the models and human perception. We also evaluate our method on the Princeton Segmentation Benchmark. Our method outperforms competitive geometry-based segmentation methods and is comparable to the learning based segmentation [66] with a training size of 6. Figure 5.1 shows some segmentation results obtained automatically from our method requiring no prior information, no given number of segments and no training.

The rest of the chapter is organized as follows. We first review the Mumford-Shah model for image segmentation in Section 5.2. The proposed Mumford-Shah model based mesh decomposition is presented in Section 5.3. Additional ingredients of incorporating user inputs and automatically determining the number of segments are described in Section 5.4 and Section 5.5, respectively. Experimental results are shown in Section 5.6. Finally, we discuss the limitations of our proposed method in Section 5.7.

5.2 Mumford-Shah Image Segmentation

To help understand the principle of our algorithm, this section gives a brief description of the Mumford-Shah image segmentation.

Given an image $I : \Omega \rightarrow R$ with bounded domain $\Omega \subset R^2$, the Mumford-Shah image segmentation problem is to find a partition $\Omega = \bigcup_{i=1}^k \Omega_i$ where Ω_i are pairwise disjoint and numbers c_i for Ω_i , which are the solution to the following optimization problem [75]

$$\inf_{\Omega_i, c_i} \sum_{i=1}^k \left(\int_{\Omega_i} (I(x) - c_i)^2 dx + \frac{\mu}{2} |\partial \Omega_i| \right) \quad (\text{Eq. 5.1})$$

where μ is a constant, and $\partial \Omega_i$ and $|\partial \Omega_i|$ represent the boundary and the boundary length of segment Ω_i , respectively. The basic idea of (Eq. 5.1) is to minimize the variation within segments and the length of the boundary between segments as well. However, even if the optimal constants c_i and the number of segments have been known a priori, in general this is still a difficult non-convex problem.

When $k = 2$ and c_1, c_2 have been known, problem (Eq. 5.1) can be re-written as

$$\inf_{\Omega_1 \subset \Omega} \left\{ \int_{\Omega_1} (I(x) - c_1)^2 dx + \int_{\Omega \setminus \Omega_1} (I(x) - c_2)^2 dx + \frac{\mu}{2} (|\partial \Omega_1| + |\partial(\Omega \setminus \Omega_1)|) \right\}. \quad (\text{Eq. 5.2})$$

Nikolova et al. [79] showed that problem (Eq. 5.2) is equivalent to finding a scalar function $u(x)$ for the following convex minimization problem:

$$\min_{0 \leq u \leq 1} \left\{ \int_{\Omega} \{u(x)(I(x) - c_1)^2 + (1 - u(x))(I(x) - c_2)^2\} dx + \mu \int_{\Omega} |\nabla u| dx \right\}. \quad (\text{Eq. 5.3})$$

Moreover, it has been shown that if $u(x)$ is a minimizer of (Eq. 5.3), then for any $\eta \in (0, 1)$, the set $\{x \in \Omega : u(x) > \eta\}$ is the minimizer of (Eq. 5.2), implying that the solution to (Eq. 5.2) can be obtained by thresholding u at an arbitrary threshold between 0 and 1. While $u(x)$ can be considered to be the characteristic function for region Ω_1 , $1 - u(x)$ is the one for region $\Omega \setminus \Omega_1$.

In practice, c_1 and c_2 are unknown. Note that if Ω_1 is fixed, the values of c_1 and c_2 that minimize (Eq. 5.2) are

$$c_1 = \frac{1}{|\Omega_1|} \int_{\Omega_1} I(x) dx, \quad c_2 = \frac{1}{|\Omega \setminus \Omega_1|} \int_{\Omega \setminus \Omega_1} I(x) dx.$$

Thus a simple approach to finding the solution is a two-step scheme that first computes c_1 and c_2 according to these formulae and then updates Ω_1 by solving (Eq. 5.3) [79].

Based on the convex formulation (Eq. 5.3), several extensions have been developed for multi-region image segmentation [78, 97].

5.3 Mumford-Shah Mesh Decomposition

Our decomposition problem can be stated as follows: Given a triangular mesh M and a positive integer k , find a disjoint partitioning of M into $M_1, M_2, \dots, M_k$ such that the elements within the same segment have high similarity and the association between different segments is low. We measure the similarity within each segment through the variance of some feature values and the association between different segments is measured by the weighted boundary length.

We now propose to solve the decomposition problem using the Mumford-Shah model. Suppose we have defined a multichannel function $\mathbf{f}(x)$ over mesh M , which, similar to the RGB function for an image, is a vector function representing some attributes of x over M and will be described in Section 5.3.1. The decomposition problem can be accomplished by solving the following minimization problem:

$$\min_{\mathbf{u} \in K, \chi_i} \left\{ \int_M \langle \mathbf{u}(x), \mathbf{s}(x) \rangle + \mu g(x) |\nabla_M \mathbf{u}(x)| d\sigma \right\}, \quad (\text{Eq. 5.4})$$

where K is the set of vector functions $\mathbf{u} = (u_1, \dots, u_k)^T : M \rightarrow R^k$ satisfying that for all $x \in M$ and $i \in [1 \dots k]$, $u_i(x) \geq 0$ and $\sum_{i=1}^k u_i(x) = 1$; $\mathbf{s}(x) = (s_1(x), \dots, s_k(x))^T$ is a k -dimensional vector with $s_i(x) = (\mathbf{f}(x) - \chi_i)^T (\mathbf{f}(x) - \chi_i)$ indicating the affinity of x with segment M_i measured by the difference between $\mathbf{f}(x)$ and vector χ_i that is associated with M_i ; $\langle, \rangle$ is the inner dot operator; the second term in (Eq. 5.4) is a weighted multi-channel total variation formulation with edge detection function $g(x)$; and μ is a tradeoff factor balancing the two terms. This is the convexified version of the Mumford-Shah

model. The first term of (Eq. 5.4) is the data term that is to ensure the segmentation complying with segment coherence (i.e., small variance). If the affinity of x with segment M_i is large, $u_i(x)$ will tend to be small in order to minimize the energy functional of (Eq. 5.4). Thus $u_i(x)$ can be viewed as the probability of x being assigned to segment M_i and $\mathbf{u}(x)$ can be used as a classification function for the segmentation. The second term of (Eq. 5.4) is the regularization term that is to constrain the boundary between different segments to be as short as possible. We also introduce an edge detection function into the second term. The edge detection function is designed to return values between 0 and 1 (see Section 5.3.2 for details) and a small value corresponds to a likely edge. Including the edge detection function is to favor the segmentation along the curves where the edge detection function has small values.

The decomposition algorithm proceeds in three steps:

step 1. Fix $\mathbf{u}$ and find the value of χ_i that minimizes (Eq. 5.4). The value reads:

$$\chi_i = \frac{\int_M u_i(x) \mathbf{f}(x) d\sigma}{\int_M u_i(x) d\sigma}$$

which corresponds to the mean of $\mathbf{f}(x)$ for segment M_i .

step 2. Fix χ_i and solve the minimization problem (Eq. 5.4) for $\mathbf{u}$. If the update of $\mathbf{u}$ is less than a prescribed value, go to step 3; otherwise, go to step 1.

step 3. Once $\mathbf{u}$ is finally obtained, each vertex v is labeled by $\ell(v) = \arg \min_{i \in \{1, \dots, k\}} \|\mathbf{u}(v) - \mathbf{e}^i\|$, where $\mathbf{e}^i = [0, \dots, 1, \dots, 0]^T$ with the i -th entry being 1, and is then classified into $M_{\ell(v)}$.

For completeness of the algorithm, we next need to discuss how to define an appropriate multichannel function $\mathbf{f}(x)$ for mesh M , how to discretize the continuous Mumford-Shah model (Eq. 5.4), and how to numerically solve (Eq. 5.4), which are elaborated below.

5.3.1 Multichannel Function $f(x)$

Note that the success of the M-S model relies on the assumption that the underlying data are approximately piecewise constant. Also, it is revealed in [3] that algorithms based on non-local shape features produce segmentations that most closely resemble ones made by humans. However, for 3D mesh models, the commonly used geometric quantities like curvature, vertex normal, face normal, and geodesic distance are local features and they are not necessarily piecewise constant either. As shown in [98], appropriate shape features should be defined. Our approach is to define the multichannel function from the eigenvectors of a dual graph Laplacian matrix. The Laplacian matrix encodes the similarity between adjacent nodes into its entries, and its eigenvectors reflect some global information (such as spectral attributes) of the underlying mesh model [99, 100]. A similar definition of Laplacian matrix for mesh segmentation was introduced in [22]. We should emphasize that the Laplacian matrixes for mesh segmentation in our method and [22] are different from traditional Laplacian matrix [99], which reflects the differential properties of underlying mesh.

Now we define the Laplacian matrix. We choose the weight of two adjacent triangles based on their dihedral angle and whether the edge shared by the two triangles is concave or convex. Let τ_i and τ_j be two adjacent triangles of M , sharing an edge e . Their dihedral angle is denoted by $dih(\tau_i, \tau_j)$. We define $d_1(\tau_i, \tau_j)$ to measure the difference of τ_i and τ_j

$$d_1(\tau_i, \tau_j) = \eta[1 - \cos(dih(\tau_i, \tau_j))] = \frac{\eta}{2} \|N(\tau_i) - N(\tau_j)\|^2 \quad (\text{Eq. 5.5})$$

where $N(\tau_l)$ is the normal vector of triangle τ_l , and η is a constant used to give higher priority to a concave edge. We set $\eta = 1.0$ for a concave edge and a relatively small number (e.g. 0.2) for a convex edge, to follow the minima rule. The difference $d_1(\tau_i, \tau_j)$ is then normalized by the average over all edges, $\overline{d_1}$, which gives the normalized difference

Figure 5.2: Plots of the eigenvectors of the proposed Laplacian matrix corresponding to the first eight nonzero eigenvalues. Note that for the fifth model, we purposely render the back side of the model to make the dark part visible.

Figure 5.3: Plots of the eigenvectors of the simple Laplacian matrix corresponding to the first eight nonzero eigenvalues.

$d(\tau_i, \tau_j) = \frac{d_1(\tau_i, \tau_j)}{d_1}$. Then a weight is heuristically defined to describe the similarity of triangles τ_i and τ_j and is assigned to their common edge e :

$$w_{ij} = |e| \exp\{-d(\tau_i, \tau_j)\} \quad (\text{Eq. 5.6})$$

where $|e|$ is the edge length. Finally the Laplacian matrix $L = [L_{ij}]$ is defined, where

$$L_{ij} = \begin{cases} -w_{ij}, & i \neq j \text{ and } \tau_i \text{ and } \tau_j \text{ share a common edge} \\ \sum_k w_{ik}, & j = i \\ 0, & \text{otherwise} \end{cases} \quad (\text{Eq. 5.7})$$

Actually L is the Laplacian matrix of the dual graph of M .

It is easy to check that Laplacian matrix L is symmetric and its smallest eigenvalue is 0. Assume that the eigenvectors of L are $\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{|T|-1}$ corresponding to the eigenvalues in ascending order. Here we select eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_{k-1}$ corresponding to the $k - 1$ smallest nonzero eigenvalues for k -partitioning. These eigenvectors form a $|T| \times (k - 1)$ matrix $[\mathbf{v}_1, \dots, \mathbf{v}_{k-1}]$. Each row of the matrix corresponds to a triangle of M . This suggests that we define the multichannel function

$$\mathbf{f}(\tau_i) = [\mathbf{v}_1(i), \mathbf{v}_2(i), \dots, \mathbf{v}_{k-1}(i)]$$

where $\mathbf{v}_j(i)$ represents the i -th entry of $\mathbf{v}_j$. In this way, we introduce $(k-1)$ -channel data to each triangle on the mesh. Since the data are from the eigenvectors of the Laplacian matrix, $\mathbf{f}$ can be viewed as a spectral attribute of the mesh.

Remark 1. We chose to use the dual graph Laplacian matrix, rather than the mesh Laplacian matrix itself. This results in the multichannel function $\mathbf{f}$ to be actually a dual function, which benefits our numerical solver presented later in Section 5.3.3.

Remark 2. As pointed out in [101], the eigenvectors are approximation of the characteristic functions for each component of the mesh. Figure 5.2 shows plots of the eigenvectors of the Laplacian matrix L corresponding to the first eight nonzero eigenvalues for a bear model. Each eigenvector is linearly mapped such that the maximum and minimum values of the entries are 1 and 0. The value of the i -th entry is interpreted as the grey value assigned to the i -th triangle. We can see that each of these eigenvectors tends to binary segment the bear model. This feature implies that our definition of the multichannel function is feasible. It is worth pointing out that our Laplacian matrix is different from the simple cotangent based Laplacian. Our expression of the Laplacian better measures the similarity between adjacent triangles and has been used in random walks based segmentation algorithms [1]. As a comparison, Figure 5.3 provides plots of the eigenvectors of the simple Laplacian matrix corresponding to the first eight nonzero eigenvalues.

Remark 3. In graph partitioning, the spectral decomposition of the Laplacian matrix is often used to approximately minimize the RatioCut model [102]

$$RatioCut(M_1, \dots, M_k) = \frac{1}{2} \sum_{i=1}^k \frac{W(M_i, \overline{M_i})}{|M_i|} \quad (\text{Eq. 5.8})$$

where $M_1, \dots, M_k$ are a k -partition of a graph, $\overline{M_i}$ is the complement of M_i , $W(M_i, \overline{M_i})$ is an association value of sets M_i and $\overline{M_i}$, and $|M_i|$ is the size of M_i . The objective function (Eq. 5.8) tries to achieve a “balance” of the clusters in terms of the size. This implies that the use of eigenvectors might avoid the influence of small-scale fluctuation.

5.3.2 Discretization of the Mumford-Shah Model

Let φ_i denote a hat function that is linear on each triangle of M and $\varphi_i(v_j) = \delta_{ij}$ for $v_j \in V$, where δ_{ij} is the Kronecker delta. The functions $\{\varphi_i : i = 0, \dots, |V| - 1\}$ have three properties: local support, nonnegativity and partition of unity. Any piecewise linear function $f(x)$ defined over M , which has value f_i at vertex v_i and is linear on each triangle, can be written as $f(x) = \sum_{0 \leq i \leq |V|-1} f_i \varphi_i(x)$ for any $x \in M$. To discretize the M-S segmentation energy functional of (Eq. 5.4), we restrict $\mathbf{u}$ to be a piecewise linear function over M : $\mathbf{u}(x) = \sum_{v_i \in V} \mathbf{u}(v_i) \varphi_i(x)$. Then the M-S segmentation energy functional of (Eq. 5.4) can be discretized into

$$\sum_{\tau \in T} \left(\sum_{v_i \prec \tau} \langle \mathbf{u}(v_i), \int_{\tau} \varphi_i(x) \mathbf{s}(x) d\sigma \rangle + \mu \int_{\tau} g_{\tau} |\nabla_M \mathbf{u}(x)| d\sigma \right). \quad (\text{Eq. 5.9})$$

For triangle τ , the edge detection function is set to be $g_{\tau} = \frac{1}{1 + \sum_{i=1}^3 \lambda_i \|N(\tau) - N(\tau_i)\|^2}$ where τ_i , $i = 1, 2, 3$, are the triangles sharing edges with τ , and λ_i is a scaling factor, which is set respecting the minima rule: $\lambda_i = 5$ if the edge shared by τ and τ_i is a concave edge; otherwise, $\lambda_i = 1$.

Moreover, the gradient of $\mathbf{u}(x)$ is

$$\nabla_M \mathbf{u}(x) = \sum_{v_i \in V} \mathbf{u}(v_i) \nabla_M \varphi_i(x) \quad (\text{Eq. 5.10})$$

where $\nabla_M \varphi_i(x)$ is regarded as a row vector, i.e., a 1×3 matrix, which can be calculated using the natural piecewise parametrization (see [89] for details). It follows that $\nabla_M \mathbf{u}(x)$ is a constant $k \times 3$ matrix on each triangle. Thus (Eq. 5.9) becomes

$$\sum_{\tau \in T} \left(\sum_{v_i \prec \tau} \langle \mathbf{u}(v_i), \int_{\tau} \varphi_i(x) \mathbf{s}(x) d\sigma \rangle + \mu g_{\tau} a_{\tau} |\nabla_M \mathbf{u}(\tau)| \right) \quad (\text{Eq. 5.11})$$

where a_{τ} is the area of triangle τ .

5.3.3 Fast Primal-Dual Method

To find $\mathbf{u}$ that minimizes the functional (Eq. 5.11), a gradient descent method [103] may be used. However, explicit schemes often lead to numerical instability, slow convergence for large meshes, and insufficient control over global behavior. In this section, we adapt a fast primal-dual algorithm to meshes to quickly and stably minimize the objective functional. The primal-dual algorithm can be highly paralleled and thus the GPU technology can be used to speed up the computation.

Using the Cauchy-Schwartz inequality, we can reformulate the second term of (Eq. 5.11) to be

$$\begin{aligned} \sum_{\tau \in T} \mu g_{\tau} a_{\tau} |\nabla_M \mathbf{u}(\tau)| &= \max_{\mathbf{d}(\tau) \in D, \forall \tau} \mu \sum_{\tau \in T} g_{\tau} a_{\tau} \langle \nabla_M \mathbf{u}, \mathbf{d}(\tau) \rangle \\ &= \max_{\mathbf{d}(\tau) \in D} \mu \sum_{\tau=[v_i, v_j, v_k] \in T} g_{\tau} a_{\tau} \langle \mathbf{u}(v_i) \nabla_M \varphi_i(\tau), \mathbf{d}(\tau) \rangle \\ &\quad + \langle \mathbf{u}(v_j) \nabla_M \varphi_j(\tau), \mathbf{d}(\tau) \rangle + \langle \mathbf{u}(v_k) \nabla_M \varphi_k(\tau), \mathbf{d}(\tau) \rangle \\ &= \max_{\mathbf{d}(\tau) \in D} \mu \sum_{v_i \in V} \langle \mathbf{u}(v_i), \sum_{\tau \in D_1(v_i)} g_{\tau} a_{\tau} \nabla_M \varphi_i(\tau), \mathbf{d}(\tau) \rangle \end{aligned}$$

where $D = \{\mathbf{d}(\tau) = (\mathbf{d}_1, \dots, \mathbf{d}_k)^T \in R^{k \times 3} : \|\mathbf{d}(\tau)\|_F \leq 1, \tau \in T\}$, and $\mathbf{d}(\tau)$ is a dual variable defined for each triangle of the mesh. Here the operations $\langle \nabla_M \mathbf{u}, \mathbf{d}(\tau) \rangle$ and $\langle \mathbf{u}(v_i) \nabla_M \varphi_i(\tau), \mathbf{d}(\tau) \rangle$ are understood as follows:

$$\begin{aligned} \langle A, B \rangle &= \sum_{1 \leq m \leq k, 1 \leq n \leq 3} a_{mn} b_{mn}, \\ \langle a, B \rangle &= \left(\sum_{1 \leq n \leq 3} a_n b_{1n}, \sum_{1 \leq n \leq 3} a_n b_{2n}, \dots, \sum_{1 \leq n \leq 3} a_n b_{kn} \right)^T \end{aligned}$$

for $a \in R^{1 \times 3}$ and $A, B \in R^{k \times 3}$, respectively, and no confusion will be caused by writing out the arguments in the bracket $\langle, \rangle$.

Let $C_i = \sum_{\tau \in D_1(v_i)} \int_{\tau} \varphi_i(x) \mathbf{s}(x) d\sigma$. Minimizing (Eq. 5.11) becomes a saddle-point problem:

$$\min_{\mathbf{u} \in K} \max_{\mathbf{d}(\tau) \in D} g(\mathbf{u}, \mathbf{d}) \quad (\text{Eq. 5.12})$$

where

$$g(\mathbf{u}, \mathbf{d}) = \sum_{v_i \in V} \langle \mathbf{u}(v_i), C_i + \mu \sum_{\tau \in D_1(v_i)} g_{\tau} a_{\tau} \nabla_M \varphi_i(\tau), \mathbf{d}(\tau) \rangle.$$

Since $\mathbf{u}$ is uniquely defined by its values at vertices, it is considered as a primal variable. Thus the saddle-point problem has its primal objective $p(\mathbf{u}) := \max_{\mathbf{d} \in D} g(\mathbf{u}, \mathbf{d})$ and its dual objective $p_d(\mathbf{d}) := \min_{\mathbf{u} \in K} g(\mathbf{u}, \mathbf{d})$. [104] has shown that g has a saddle point $(\mathbf{u}^*, \mathbf{d}^*)$ and

$$\min_{\mathbf{u} \in K} p(\mathbf{u}) = p(\mathbf{u}^*) = g(\mathbf{u}^*, \mathbf{d}^*) = p_d(\mathbf{d}^*) = \max_{\mathbf{d} \in D} p_d(\mathbf{d}). \quad (\text{Eq. 5.13})$$

Based on the above analysis, we can see that a straightforward approach to solving (Eq. 5.12) is to alternately apply the projected gradient decent method on the primal variable $\mathbf{u}$ and the projected gradient ascent method on the dual variable $\mathbf{d}$ [78]. In fact, [77] has presented the Fast Primal-Dual (FPD) method, which is a variant of Popov's saddle point method [105] with provable convergence. We adapt it to our problem, which is outlined in Algorithm 1.

Algorithm 1 FPD method

Choose initial values $\mathbf{u}^0$, $\bar{\mathbf{u}}^0$ and $\mathbf{d}^0$
 Choose primal step $\tau_p > 0$, dual step $\tau_d > 0$, and termination tolerance $\epsilon > 0$
while $\frac{p(\mathbf{u}^k) - p_d(\mathbf{d}^k)}{p_d(\mathbf{d}^k)} > \epsilon$ **do**
 $\mathbf{d}^{k+1}(\tau) \leftarrow \Pi_D(\mathbf{d}^k(\tau) + \tau_d(\mu g_\tau a_\tau \nabla_M \bar{\mathbf{u}}^k))$ for $\tau \in T$
 $\mathbf{u}^{k+1}(v_i) \leftarrow \Pi_K(\mathbf{u}^k(v_i) - \tau_p(C_i$
 $+ \mu \sum_{\tau \in D_1(v_i)} g_\tau a_\tau < \nabla_M \varphi_i(\tau), \mathbf{d}^{k+1}(\tau) >))$ for $v_i \in V$
 $\bar{\mathbf{u}}^{k+1} \leftarrow 2\mathbf{u}^{k+1} - \mathbf{u}^k$
 $k \leftarrow k + 1$
end while

In Algorithm 1, the initial primal variable $\mathbf{u}^0$ is randomly chosen, $\bar{\mathbf{u}}^0$ is the same as $\mathbf{u}^0$, and the initial dual variable $\mathbf{d}^0$ is set to the divergence of the initial primal variable. $\Pi_K(\cdot)$ is the operator that projects the primal variable $\mathbf{u}$ on the set K and $\Pi_D(\cdot)$ is the operator that projects the dual variable $\mathbf{d}$ on the dual constraint set D .

5.4 User Interaction

Though automatic decomposition is preferred, it is also desirable to allow users to express their intention or their preference in some situations, especially when there exist several decomposition possibilities. For example, if we decompose a bear model into five parts, our algorithm generates the result shown in Figure (a). However, if we want the head to be connected to the body, user’s input such as a stroke passing through the head and the body shown in Figure (b) would help. Therefore we introduce constraints into the Mumford-Shah model to incorporate such user’s inputs and the new result is shown in Figure (c).

Figure 5.4: User’s input changes the segmentation result.

5.4.1 Mumford-Shah Model with Constraints

If the user draws a stroke to express the intention that some vertices must be in the same segment, we can achieve this “must-link” constraint by enforcing $\mathbf{u}$ at those vertices to be the same. Then the Mumford-Shah model becomes a minimization problem with some equality constraints. For example, assume that m vertices are required to be in the same segment. This will introduce $m - 1$ independent constraints. If we let $\mathbf{U} = [\mathbf{u}(v_0), \dots, \mathbf{u}(v_{|V|-1})]^T$, the k -th constraint that v_i and v_j belong to the same segment

can be expressed as $\mathbf{z}_k^T \mathbf{U} = 0$ where $\mathbf{z}_k$ is a $|V|$ -dimensional vector with only two nonzero entries: $\mathbf{z}_k(i) = 1$ and $\mathbf{z}_k(j) = -1$. Grouping all the $\mathbf{z}_k$ gives $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_{m-1}]$ and all the constraints become

$$\mathbf{Z}^T \mathbf{U} = 0. \quad (\text{Eq. 5.14})$$

Incorporating these constraints into the Mumford-Shah model of (Eq. 5.9) leads to a constrained Mumford-Shah model. Using Lagrange multipliers, we can change the constrained model into an unconstrained optimization problem:

$$\min_{\mathbf{u} \in K} \max_{\mathbf{w}} \left\{ \sum_{\tau \in T} \left(\sum_{v_i \prec \tau} \langle \mathbf{u}(v_i), \int_{\tau} \varphi_i(x) \mathbf{s}(x) d\sigma \rangle + \mu \int_{\tau} g_{\tau} |\nabla_M \mathbf{u}(x)| d\sigma \right) + \langle \mathbf{w}, \mathbf{Z}^T \mathbf{U} \rangle \right\}, \quad (\text{Eq. 5.15})$$

where $\mathbf{w}$ are the Lagrangian multipliers. This minimization problem can also be solved using the fast primal-dual method described in Section 5.3.3. If there are several sets of such requirements, we can express all these constraints in a similar way.

5.4.2 Constraint Propagation

It is observed that constraints (Eq. 5.14) only guarantee that those m vertices specified by user's inputs are in the same segment and they have little influence on the neighborhood of the user's inputs. Moreover, we wish that user's inputs could be reflected in the two terms of the Mumford-Shah model. However, in our method, the data term is determined by the Laplacian matrix's eigenvectors, which are fixed if the edge weights w_{ij} are fixed. To overcome this problem, we propose to increase the weights for those edges near the user's inputs in order to propagate the influence of the user's inputs. Specifically, for each indicated vertex, we get its farthest vertex on the mesh. The user indicated vertices are treated as foreground seeds and those farthest vertices are treated as background seeds. Then we use random walks algorithm [56] to compute a probability value $p(v_k)$ for each vertex v_k on the mesh. For edge e bounded by vertices v_i and v_j , if $\frac{p(v_i) + p(v_j)}{2} > 0.9$, which means that v_i and v_j are very close to the user indicated vertices, we re-set the

edge weight of e to 1. In this way, a small region around the user indicated vertices will be strongly connected because the similarity weights between them are set to the largest value 1.

5.5 Determination of the Number of Segments

The Mumford-Shah mesh decomposition presented in preceding sections depends on the number of segments. This section proposes a heuristic method to compute a number based on the stability of the RatioCut values for the number of segments. It can be done in a pre-processing stage and could be added to the Mumford-Shah mesh decomposition to provide an automatic algorithm. The experiments show that the number computed by the method matches the one given by users fairly well.

Our observation is that if a good k -partitioning has been done and one more segment is to be added, the existing segments have to be split and merged to form a new segmentation, which usually causes the association between segments to increase rapidly and the sizes of segments to lose balance. As a result, the RatioCut value of (Eq. 5.8) where the association value $W(M_i, \overline{M_i})$ is computed as the length of M_i 's boundary will have a sudden change. This motivates us to propose a brute-force approach: perform segmentation for various numbers of segments, compute the RatioCut value for the segmentation results, and choose $h - 1$ as the number of segments if h -partitioning causes a sudden change in RatioCut values. We can rank this value, and then the second/third best choice can also be determined.

Note that producing good segmentations for every possible number of segments is a time-consuming process. Considering our goal here is to find the number of segments, instead of using sophisticated segmentation methods for high quality segmentation, we employ a fast algorithm to obtain a reasonable segmentation, based on which we perform RatioCut value stability analysis. As explained in Section 5.3.1, the Laplacian matrix

L contains global information of the underlying mesh. [49, 106] determines the cluster number by searching for a drop in the magnitude of the eigenvalues of the Laplacian matrix. [107] rotates the eigenvectors of the Laplacian matrix and finds the number of segments which provides the best alignment with the canonical coordinate system. We notice that each eigenvector is roughly considered to be an indicator for some segment of mesh M . Therefore we use the K-means algorithm to cluster all the triangles based on their multichannel data from the $k - 1$ smallest (nonzero) eigenvectors to obtain a rough k segments, which can be done in a very fast speed. Then the RatioCut value is computed for each k , which is denoted by $RC(k)$. Finally the number of segments is chosen to be $\arg \max_i \{|RC(i) - 2RC(i - 1) + RC(i - 2)|\} - 1$, which maximizes the second order difference.

Figure 5.5: Numbers of segments and the second order differences of RatioCut values.

Figure 5.5 shows three possible numbers of segments and their corresponding second order differences of RatioCut values.

5.6 Experimental Results and Discussions

This section provides experimental results to validate our proposed algorithm. The experiments are conducted in two aspects. Since the proposed Mumford-Shah mesh segmentation algorithm contains various technical components, the first aspect is to test the effects of the key ingredients. The second aspect is to evaluate the performance of the algorithm as a whole applied to various 3D mesh models and to perform comparison with other methods as well.

Our proposed algorithm requires to select a value for tradeoff parameter μ in Eq. (Eq. 5.4). The larger μ is, the more significant the weighted boundary term is. Different μ values could result in different segmentation results. As the data and regularization terms are affected by the mesh model, the number of segments, and the magnitude of geometric features of the model, it is usually difficult to select a value that is suitable for all situations. However, considering that μ is expected to be independent of the model scale and also noticing that when the number of segments increases, the value of the regularization term increases and the value of the data term decreases, we here provide an empirical formula to compute μ :

$$\mu = \bar{\mu} \times \text{Number-of-segments} \times \frac{E_f}{E_r}$$

where $E_f = \int_M \langle \mathbf{u}(x), \mathbf{s}(x) \rangle d\sigma$ is the data term and $E_r = \int_M g(x) |\nabla_M \mathbf{u}(x)| d\sigma$ is the regularization term given an initial $\mathbf{u}$; and $\bar{\mu}$ is a “normalized” parameter. Our empirical value for $\bar{\mu}$ is 0.02, which works well for many situations. In fact, we used this choice for all the models in the Princeton Segmentation Benchmark. If needed, users can still adjust $\bar{\mu}$ around 0.02, which is more convenient than tuning μ in a wide range.

It is worth pointing out that the Mumford-Shah segmentation does not guarantee the connectedness of the segmentation. That is, though the segmentation produces k segments for k -partition, the resulting regions belonging to the same segment may not

be connected geometrically, which results in more than k disjoint regions. In our experiments, in case this situation occurs, we recursively merge the segment with the smallest number of the vertices to its neighboring segment that has the largest number of vertices until the number of the disjoint regions is equal to k .

5.6.1 Effects of Key Ingredients

Our proposed algorithm contains several key ingredients. First, we use the eigenvectors of a dual Laplacian matrix to define the attributes for the mesh elements instead of using purely local feature descriptions. Remark 2 of Section 5.3.1 has mentioned that the eigenvectors approximate the characteristic functions of components of a mesh. Figures 5.2, 5.6, and 5.7(b) visualize the eigenvectors of the proposed Laplacian matrix corresponding to the first few nonzero eigenvalues, from which it can be seen that the eigenvectors convey segment information. It is also observed that usually the first few smallest (nonzero) eigenvectors correspond to relatively large structures of a mesh. Considering the computational cost, we choose only the first $(k - 1)$ nonzero eigenvectors to define our multichannel function if the target number of segments is k , which is similar to the approach of [22] that chooses k eigenvectors of a normalized affinity matrix.

Second, different from the simple Laplacian matrix, our Laplacian matrix is constructed from the weights that consider the coplanarity between adjacent triangles and the local convexity or concavity of the edge shared by the triangles as well. As a result, each eigenvector of our Laplacian matrix tends to be more suitable for binary segmentation and concentrate more on meaningful parts of the model. Figures 5.2 and 5.3, and Figure 5.7 show such comparison.

Third, the Mumford-Shah model contains a regularization term that constrains the boundary between segments to be as short as possible. This has an effect of smoothing. Furthermore, we introduce an edge detection function into the regularization term

Figure 5.6: Plots of the eigenvectors of our proposed Laplacian matrix corresponding to the first twenty nonzero eigenvalues.

(a) Eigenvectors of the simple Laplacian matrix

(b) Eigenvectors of our proposed Laplacian matrix

Figure 5.7: Comparison of the eigenvectors of the simple Laplacian matrix and our proposed Laplacian matrix. We can see that our proposed Laplacian matrix better reflects the structure of the underlying models.

to encourage the segmentation to align with feature edges. Figure 5.8(b) show some results of segmentation without the regularization term. The boundaries of segments are apparently not smooth. Adding the regularization term improves the smoothness of the boundaries, but without the edge detection function the resulting boundaries of segments may not align well with the geometric features of the models, as depicted in Figure 5.8(c). For a comparison, we also display the segmentation results using the M-S model applied to the eigenvectors of the simple Laplacian matrix in Figure 5.8(a). Obviously, by integrating all the key ingredients mentioned above, our proposed algorithm produces excellent segmentation results as shown in Figure 5.8(d).

5.6.2 Overall Performance of the Algorithm

Overall performance on benchmark dataset. We apply our proposed algorithm on the entire 3D Segmentation Benchmark dataset [3], which contains 19 categories of meshes (20 models per category), segmentation results by human users, source code for computing evaluation scores, and the results of many existing segmentation methods. One snapshot of the visual results in each category produced by our algorithm is shown in Figure 5.1. Note that our algorithm is general and fully automatic, requiring no prior information, no given number of segments and no training. From Figure 5.1, we can see that the results of our algorithm match human perception well in not only the cutting boundaries but also the number of segments. In addition, the cutting contours are along geometric features.

We further compare our algorithm with the two state-of-the-art geometry based methods: Randomized Cuts [65] and Shape Diameter Function [43], where Randomized Cuts requires given number of segments and Shape Diameter Function determines the number of segments automatically. We also include the latest learning based method [66] for reference, although it is a different type of approach. For all the methods, we perform

(a) The results using the simple Laplacian matrix

(b) The results without the regularization term

(c) The results without including edge detection function $g(x)$

(d) The results with our proposed M-S model.

Figure 5.8: Comparison of segmentation with various choices.

evaluations according to the protocols of [3], using all human segmentations in the Princeton Segmentation Benchmark. Following the comparison presentation in [66], we show the scores of two evaluation metrics, Rand Index and Consistency Error, in Figure 5.9 and list the detailed Rand Index scores for each category in Table 5.1, which also include the segmentation results of human users. The definitions of Rand Index and Consistency Error can be referred to [3], and smaller values suggest better segmentation results. Note that Rand Cuts is given the dominant number of segments in the human segmentations for each model.

Figure 5.9: Quantitative evaluation of segmentation results in terms of Rand Index and Consistency Error (smaller values stand for better results). For all the methods, we perform evaluations according to the protocols of [Chen et al. 2009], using all human segmentations in the Princeton Segmentation Benchmark. “SB19” and “SB3” stand for the learning based method with training set sizes of 19 and 3 (out of 20 models for each category), respectively. “M-S”, “Rand Cuts” and “Shape Diam” represent our proposed method, Randomized Cuts and Shape Diameter Function, respectively. Note that Rand Cuts is given the dominant number of segments in the human segmentations for each model. Our method significantly outperforms the two state-of-the-art geometry based methods while worse than the learning based method with large training size.

Among the three geometry based methods, M-S, Randomized Cuts and Shape Diameter Function, from Figure 5.9 and Table 5.1, we can see that our proposed M-S algorithm achieves the best performance, significantly outperforming the other two. Note that the

Object Categories	Bench Mark	SB 19	SB 3	MS	Rand Cuts	Shape Diam
Human	13.5	11.9	14.7	11.1	13.1	17.9
Cup	13.6	9.9	10.0	20.4	21.9	35.8
Glasses	10.1	13.6	14.2	9.4	10.1	20.4
Airplane	9.2	7.9	10.2	11.1	12.2	9.2
Ant	3.0	1.9	2.6	2.2	2.5	2.2
Chair	8.9	5.4	6.6	10.9	18.4	11.1
Octopus	2.4	1.8	2.2	2.5	6.3	4.5
Table	9.3	6.2	11.1	10.3	38.3	18.4
Teddy	4.9	3.1	5.6	3.2	4.5	5.7
Hand	9.1	10.4	15.8	7.9	9.0	20.2
Plier	7.1	5.4	10.5	8.9	11.0	37.5
Fish	15.5	12.9	13.5	29.6	29.7	24.8
Bird	6.2	10.4	18.6	9.4	10.7	11.5
Armadillo	8.3	8.0	8.6	8.7	9.2	9.0
Bust	22.0	21.4	39.3	25.1	23.2	29.9
Mech	13.1	10.0	24.0	13.1	27.7	23.8
Bearing	10.4	9.7	32.7	16.6	12.4	11.9
Vase	14.4	16.0	25.3	12.5	13.3	23.9
FourLeg	14.9	13.3	16.3	14.4	17.4	16.1
Average	10.3	9.4	14.8	12.0	15.3	17.6

Table 5.1: Rand Index scores for human segmentations, SB19, SB3, MS (our method), Randomized Cuts, and Shape Diameter Function. The Rand Index scores are measured against all human segmentations in the Princeton Benchmark.

scores of human segmentations are not perfect. This is due to the variations of the segmentation results among different users. Figure 5.10 further shows the visual comparisons of the three geometry based methods, which further demonstrate the superior performance of our proposed method.

As for the learning based method [66], we do not intend to have a comprehensive comparison with our method since they are different types of approaches. The learning based method is an excellent work, which can perform not only segmentation but also labelling while requiring category-specified training. In terms of the segmentation results, in general the average performance of the learning based method is better than ours when

the training set size is large. However, when the training set size is small, say no more than 30% of the entire category size, the performance of our algorithm is comparable or better on average. Note that the human model in Figure 5.10 is actually a failure case for the learning based method [66] while our method can segment it well.

Figure 5.10: Visual comparisons to other segmentation methods for Table, Octopus, Hand, Human, Fourleg, Chair, Mech. Top row: results of our method. Middle row: results of Randomized Cuts, with number of segments defined as the dominant number of segments in the human segmentations for each model. Bottom row: results of Shape Diameter Function.

Cutting contour smoothness. The visual results in Figures 5.1 and 5.10 already show that our algorithm produces smooth cutting contour, along geometry features. This is mainly because of the boundary term in the M-S model, which essentially pulls the cutting contours toward the geodesic curves [108]. In addition to the benchmark dataset, we also test our algorithm on other models in Figures 5.8(d) and 5.11 and two CAD models in Figure 5.12, which contain sharp edges. We can see that our algorithm can cut along the sharp edges and produce smooth cutting contour.

Segmentation efficiency. As described in Section 5.3.3, a fast primal-dual algorithm has been devised to efficiently solve the M-S model for mesh decomposition. To

Figure 5.11: Segmentation results of the proposed algorithm on four non-benchmark models. The cutting contours are along geometry features.

Figure 5.12: Segmentation results of the proposed algorithm on two CAD models. Our method cuts the models along the sharp edges.

further accelerate the processing speed, we implement our algorithm on GPU using the NVIDIA CUDA framework with Quadro FX 4600 graphics card. All the experiments are run on a PC with Intel Core 2.66GHz CPU and 2GB RAM. Table 5.2 lists the computing time for the models in Figures 5.8(d) and 5.12. It can be seen that with our GPU implementation, the processing speed is improved by around ten times and the segmentation of a middle-size model can be done in a few seconds even though our GPU implementation is not optimized yet. Further speed improvement might be possible since over thirty times acceleration has been reported in [77].

Model (Figure)	# of vertices	# of triangles	# of segments	CPU (s)	GPU (s)
Horse (5.8(d))	48485	96966	6	50.47	4.16
Hand (5.8(d))	53054	105860	6	71.45	4.41
Santa (5.8(d))	75781	151558	6	80.48	6.54
Bunny (5.8(d))	34834	69451	12	91.68	8.63
Octahedron ((a))	16386	32768	8	21.78	2.21
Fandisk ((b))	6475	12946	3	4.81	0.60

Table 5.2: Mesh information and running time statistics. The GPU implementation improves the processing speed by around ten times.

Number of segments. All the visual results shown in the previous figures have demonstrated that the number of segments selected by our method proposed in Section 5.5 matches human perception well. Considering that the method of Shape Diameter Function [43] can also automatically choose the number of segments, we further conduct the following experiments for comparison. In particular, for each mesh model we compare the selected number of segments with each of the human segmentations. If the predicted number is equal to a human segmentation, we count it as one; otherwise, we count it as zero. For the entire 4300 human segmentations on 380 different meshes, the number of segments we choose coincides with 1087 human segmentations while Shape Diameter Function scores 741. In fact, the number of matches for our method is quite

high, considering that there exist significant variations in the number of segments among different human segmentations for each model.

5.7 Limitations

As all the existing geometry-based segmentation methods, our proposed algorithm requires fine tuning of the parameters and is sensitive to the selection of the parameters. For example, the parameter μ in Eq. (Eq. 5.4) plays an important role in the tradeoff of the weighted boundary term. The larger μ is, the more significant the weighted boundary term is. Different μ values could result in quite different segmentation results, as illustrated in Figure. 5.13, where big μ results in smooth cutting contours while small μ gives the disappointing results with jaggy boundaries. On the other hand, too big μ value may cause large variation within one segment. If the parameter μ is not good for a particular model, we can use user's input to rectify it. As shown in Figures (c) and (d), a must-link stroke rectifies the result as we expect.

Figure 5.13: The influence of parameter μ in Eq. (Eq. 5.4) on the segmentation result. Big μ results in good segmentation in (a) while small μ gives the disappointing results with jaggy boundaries in (b). A must-link stroke (in blue) is drawn in (c) to obtain the rectified result in (d).

Another major limitation is that our method fails at some cases, despite its superior average performance over the entire benchmark dataset. Figure 5.14 gives one example, where our results do not match the human segmentations. This is mainly because the

fish model is very smooth lacking clear geometry edges and some expected segments (the fins) are of small size, while our method assumes that the size of each segment should be considerable. By observing the first eight nonzero eigenvectors of the Laplacian matrix for the fish model, we find that the eigenvectors themselves do not contain much expected segment information. To handle this and small sized components problem, combining high-level cues or local feature with eigenvectors would be a possible solution.

Figure 5.14: (a)-(d): The segmentation results of our method with the number of segments being 4, 5, 6, 7, respectively. (e)-(h): samples of the corresponding human segmentation results. (i)-(p) Color plots of the first eight nonzero eigenvectors of the Laplacian matrix for a fish model. Our method fails at this smooth model which contains some tiny segments (fins) and lacks clear geometry edges.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we have studied the fundamentals of 3D mesh segmentation and reviewed the existing interactive and automatic methods for mesh segmentation. We have proposed a flexible interactive mesh segmentation algorithm, a robust interactive mesh segmentation method, and a unified automatic mesh decomposition framework.

For flexible interactive mesh segmentation, 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 mesh scissoring method within the same computational framework. The experimental results demonstrate that the new algorithm is flexible, fast, and capable of producing satisfactory results with regard to the user intention and geometric attributes.

For robust interactive mesh segmentation, we have developed a mesh snapping framework, whose results are robust to the user input and capable of reflecting geometric features and human shape perception. In particular, we have applied the geodesic curvature flow function for interactive mesh cutting. To the best of our knowledge, it has not been done before. We have also proposed 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. In addition, 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 certain flexibility into the robust mesh snapping framework.

For automatic mesh decomposition, we have studied three fundamental issues in mesh decomposition: how to accurately and efficiently decompose a mesh into meaningful parts, how to incorporate user's inputs to influence the segmentation, and how to automatically determine the number of segments that a mesh should be decomposed into. First, a Mumford-Shah model based mesh decomposition algorithm is presented. The algorithm simultaneously handles segmentation and boundary smoothing and is able to accurately and efficiently partition a mesh into a prescribed number of components. Second, a constrained Mumford-Shah model is formulated to incorporate user's inputs and an approach is presented to compute the segmentation that reflects user's intention. Third, an automatic approach is proposed to compute the best segment number for a given model. These works extensively utilize the spectral information of the mesh represented by the eigenvectors of the Laplacian matrix of the mesh, aim at segmentation with high similarity within each segment and low association among different segments, and take human perception into consideration. Consequently, the proposed algorithms outperform most geometry-based segmentation algorithms in terms of quality and speed. Extensive experiments show that our algorithms are able to produce segmentation results that match human's perception.

6.2 Future Directions

Despite their superior performances comparing to counterpart methods, our proposed algorithms also have some limitations. Although it can always obtain satisfactory results with unlimited soft/hard constraints by using the method in Chapter 3, soft/hard constraints require more user efforts, which are not convenient comparing to foreground/background input methods. The performance of the algorithm in Chapter 4 depends on the initial cutting contour because of its local optimization property. For example, it may not give good segmentation results to the segment whose boundaries are on convex areas and concave areas like the the faces of fandisk model. Besides, the segmentation results also depend on the edge detection weights. For CAD models, some of the boundaries between segments are on the concave areas, which fails the boundary length term in the model. Accordingly, the algorithm in Chapter 5 is not suitable for this type of models.

Our flexible interactive mesh cutting work focuses on segmenting the mesh into two classes, the constrained random walks algorithm is developed for such a binary segmentation. 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 vertex arriving at that class first, before reaching other classes of seeds, and then we classify this vertex 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 are within the soft constraint areas and go through the hard constraint vertices.

Our proposed robust interactive mesh segmentation method emphasizes on the robustness performance, but it does not provide great flexibility for the user to control the final cutting contour. The 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. It is interesting to combine our flexible cutting work with the robust segmentation work. However, how to optimally tradeoff between robustness and flexibility is still an open question.

Our proposed M-S computational framework for mesh decomposition still has space for improvement. First, the eigenvectors of the mesh model sometimes do not provide enough segment information, which results in unsatisfactory performance as shown in Fig. 5.14. It is of great interest to find a more effective feature to replace Laplacian eigenvectors, where the new feature should have higher similarity within segment than eigenvectors. Second, it would be beneficial to introduce high-level, global information through learning into the M-S model to further improve the decomposition performance in terms of quality and robustness. Third, the proposed must-link constraints incorporated in the M-S model might not be sufficient for the user to express his intention. For example, in some cases, there is a need to add constraints that some vertices or triangles must not be in the same segment, which can be called as cannot-link constraints. It is interesting to study how to introduce appropriate user interface for incorporating interactive techniques in mesh decomposition. Last but not the least, although the method we proposed for determining the number of segments gives promising results, it is worth further investigation to look for better approaches that can achieve more accurate prediction in a fast manner.

We would also like to point out that L_1 related models have achieved great success in image processing and compressed sensing, while its application on mesh is still limited.

Our proposed total variation based M-S model can be seen as a pioneer work in this direction. We believe that L_1 based models can be successfully applied in many mesh processing applications. Besides, our proposed fast primal-dual algorithm on mesh is suitable to solve L_1 related models for mesh processing, such as mesh restoration and image segmentation on meshes.

Our works belong to geometry-based mesh segmentation algorithms, which is a bottom-up method. From the human perception point of view, we should use not only low-level features, such as color, texture and shape, but also prior knowledge to decompose an object into several parts. Although some cognition criteria like minima rule has been adopted to design the weights, but it is far from the high-level semantic cues that could be obtained through training. Integration low-level geometry features with high-level knowledge from training data would be a promising way to achieve breakthrough in mesh decomposition. A similar idea on image segmentation [109] has already been proposed and superior performance has been reported.

References

- [1] Yu-Kun Lai, Shi-Min Hu, Ralph R. Martin, and Paul L. Rosin, “Rapid and effective segmentation of 3D models using random walks,” *Computer Aided Geometric Design*, vol. 26, no. 6, pp. 665–679, 2009.
- [2] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang, “Easy mesh Cutting,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 25, no. 3, pp. 283–291, 2006.
- [3] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser, “A Benchmark for 3D Mesh Segmentation,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, no. 3, 2009.
- [4] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patanè, Michela Spagnuolo, and Ayellet Tal, “Mesh Segmentation - A Comparative Study,” in *Shape Modeling and Applications*, 2006, pp. 14–25.
- [5] Ariel Shamir, “A survey on Mesh Segmentation Techniques,” *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539–1556, 2008.
- [6] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin, “Modeling by Example,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 23, no. 3, pp. 652–663, 2004.
- [7] Sylvain Petitjean, “A survey of methods for recovering quadrics in triangle meshes,” *ACM Computing Surveys*, vol. 34, no. 2, pp. 211–262, June 2002.

- [8] Emanuel Zuckerberger, Ayellet Tal, and Shymon Shlafman, “Polyhedral surface decomposition with applications,” *Computers and Graphics*, vol. 26, no. 5, pp. 733–743, 2002.
- [9] Arthur Gregory, Andrei State, Ming C. Lin, Dinesh Manocha, and Mark A. Livingston, “Interactive surface decomposition for polyhedral morphing,” *Visual Computer*, vol. 15, no. 9, pp. 453–470, December 1999.
- [10] Silvia Biasotti, Simone Marini, Michela Mortara, and Giuseppe Patanè, “An overview on properties and efficacy of topological skeletons in Shape Modelling,” in *Shape Modeling International*, 2003, pp. 245–256.
- [11] Sagi Katz and Ayellet Tal, “Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 22, no. 3, pp. 954–961, 2003.
- [12] Jyh-Ming Lien, John Keyser, and Nancy M. Amato, “Simultaneous Shape Decomposition and Skeletonization,” in *ACM symposium on Solid and physical modeling*, 2006, pp. 219–228.
- [13] Michela Mortara, Giuseppe Patanè, and Michela Spagnuolo, “From geometric to semantic human body models,” *Computers and Graphics*, vol. 30, no. 2, pp. 185–196, 2006.
- [14] Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, Bing-Yu Chen, and Ming Ouhyoung, “Domain connected graph: the skeleton of a closed 3D shape for animation,” *Visual Computer*, vol. 22, no. 2, pp. 117–135, 2006.
- [15] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang, “Decomposing polygon meshes for interactive applications,” in *Symposium on Interactive 3D graphics*, 2001, pp. 35–42.
- [16] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege, “Fast and intuitive generation of geometric shape transitions,” *Visual Computer*, vol. 16, no. 5, pp. 241–253, 2000.

REFERENCES

- [17] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot, “Least squares conformal maps for automatic texture atlas generation,” *ACM Transactions on Graphics (SIGGRAPH)*, vol. 21, pp. 362–371, July 2002.
- [18] Cohen-Steiner David, Alliez Pierre, and Mathieu Desbrun, “Variational Shape Approximation,” *ACM Transactions on Graphics (SIGGRAPH)*, pp. 905–914, 2004.
- [19] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo, “Hierarchical mesh segmentation based on fitting primitives,” *Visual Computer*, vol. 22, no. 3, pp. 181–193, 2006.
- [20] Zach Karni and Craig Gotsman, “Spectral Compression of Mesh Geometry,” in *SIGGRAPH*, 2000, pp. 279–286.
- [21] Shymon Shlafman, Ayellet Tal, and Sagi Katz, “Metamorphosis of Polyhedral Surfaces using Decomposition,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 21, pp. 219–228, 2002.
- [22] Rong Liu and Hao Zhang, “Segmentation of 3D Meshes through Spectral Clustering,” in *Pacific Conference on Computer Graphics and Applications*, 2004, pp. 298–305.
- [23] Rong Liu and Hao Zhang, “Mesh Segmentation via Spectral Embedding and Contour Analysis,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 26, no. 3, pp. 385–394, 2007.
- [24] Alan P. Mangan and Ross T. Whitaker, “Partitioning 3D Surface Meshes Using Watershed Segmentation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 308–321, 1999.
- [25] Sagi Katz, George Leifman, and Ayellet Tal, “Mesh segmentation using feature point and core extraction,” *Visual Computer*, vol. 21, no. 8-10, pp. 649–658, 2005.
- [26] Tong-Yee Lee, Ping-Hsien Lin, Shaur-Wei Yan, and Chun-Hao Lin, “Mesh decomposition using motion information from animation sequences,” *Journal of Visualization and Computer Animation*, vol. 16, no. 3-4, pp. 519–529, 2005.

REFERENCES

- [27] Tong-Yee Lee, Yu-Shuen Wang, and Tai-Guang Chen, “Segmenting a deforming mesh into near-rigid components,” *Visual Computer*, vol. 22, no. 9-11, pp. 729–739, 2006.
- [28] Scott Schaefer and Can Yuksel, “Example-based skeleton extraction,” in *Symposium on Geometry Processing*, 2007, pp. 153–162.
- [29] Yunjin Lee, Seungyong Leea, Ariel Shamirb, Daniel Cohen-Or, and Hans-Peter Seidel, “Mesh scissoring with minima rule and part salience,” *Computer Aided Geometric Design*, vol. 22, no. 11, pp. 444–465, July 2005.
- [30] Youyi Zheng and Chiew-Lan Tai, “Mech Decomposition with Cross-Boundary Brushes,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 29, no. 2, 2010.
- [31] Lubin Fan, Ligang Liu, and Kun Liu, “Paint Mesh Cutting,” *Computer Graphic Forum (Proc. Eurographics)*, vol. 30, no. 2, 2011.
- [32] Donald D. Hoffman and Whitman Richards, “Parts of recognition,” *Cognition*, vol. 18, pp. 65–96, 1984.
- [33] D.D. Hoffman and Manish Singh, “Salience of visual parts,” *Cognition*, vol. 63, pp. 29–78, 1997.
- [34] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe, “Texture mapping progressive meshes,” in *SIGGRAPH*, 2001, pp. 409–416.
- [35] Eugene Zhang, Konstantin Mischaikow, and Greg Turk, “Feature-based surface parameterization and texture mapping,” *ACM Transactions on Graphics*, vol. 24, no. 1, pp. 1–27, 2005.
- [36] Kun Zhou, John Snyder, Baining Guo, and Heung-Yeung Shum, “Iso-charts: Stretch-driven Mesh Parameterization using Spectral Analysis,” in *Symposium on Geometry Processing*, 2004, pp. 47–56.
- [37] Pedro V. Sander, Zoë J. Wood, Steven J. Gortler, John Snyder, and Hugues Hoppe, “Multi-Chart Geometry Images,” in *Symposium on Geometry Processing*, 2003, pp. 146–155.

REFERENCES

- [38] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle, “Multiresolution analysis of arbitrary meshes,” in *SIGGRAPH*, 1995, pp. 173–182.
- [39] Michael Garland, Andrew Willmott, and Paul S. Heckbert, “Hierarchical face clustering on polygonal surfaces,” in *Symposium on Interactive 3D graphics*, 2001, pp. 49–58.
- [40] Ioana M. Boier-Martin, “Domain Decomposition for Multiresolution Analysis,” in *Symposium on Geometry Processing*, 2003, pp. 31–41.
- [41] Irving Biederman, “Recognition-by-components: A theory of human image understanding,” *Psychological Review*, vol. 94, pp. 115–147, 1987.
- [42] Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz, and Thomas A. Funkhouser, “A planar-reflective symmetry transform for 3D shapes,” *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 549–559, 2006.
- [43] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or, “Consistent mesh partitioning and skeletonisation using the shape diameter function,” *Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.
- [44] Mirko Sattler, Ralf Sarlette, and Reinhard Klein, “Simple and efficient compression of animation sequences,” in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2005, pp. 209–217.
- [45] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek, “Ray Tracing Animated Scenes using Motion Decomposition,” *Comput. Graph. Forum*, vol. 25, no. 3, pp. 517–525, 2006.
- [46] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel, “Intelligent Mesh Scissoring Using 3D Snakes,” in *Computer Graphics and Applications, Pacific Conference*, 2004, pp. 279–287.
- [47] Hsueh-Yi Sean Lin, Hong-Yuan Mark Liao, and Ja-Chen Lin, “Visual Saliency-Guided Mesh Decomposition,” *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 46–57, 2007.

- [48] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro, “Geodesic Active Contours,” *International Journal of Computer Vision*, vol. 22, no. 1, pp. 61–79, 1997.
- [49] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss, “On Spectral Clustering: Analysis and an algorithm,” in *NIPS*, 2001, pp. 849–856.
- [50] Kevin Chun-Ho Wong, Tommy Yu-Hang Siu, Pheng-Ann Heng, and Hanqiu Sun, “Interactive volume cutting,” in *Graphics Interface*, 1998, pp. 99–106.
- [51] Yunjin Lee and Seungyong Lee, “Geometric Snakes for Triangular Meshes,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 21, no. 3, pp. 229–238, July 2002.
- [52] Andrew Clements and Hao Zhang, “Minimum Ratio Contours on Surface Meshes,” in *Proc. of Shape Modeling International*, 2006, pp. 26–37.
- [53] Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or, “Snap-Paste: an interactive technique for easy mesh composition,” *Visual Computer*, vol. 22, no. 9-11, pp. 835–844, September 2006.
- [54] Ali Kemal Sinop and Leo Grady, “A Seeded Image Segmentation Framework Unifying Graph Cuts And Random Walker Which Yields A New Algorithm,” in *International Conference on Computer Vision (ICCV)*, 2007, pp. 560–572.
- [55] Yu-Shuen Wang and Tong-Yee Lee, “Wysiwyg: Mesh Decomposition for Static Models,” in *International Conference on International Information Hiding and Multimedia Signal Processing*, 2007, pp. 353–356.
- [56] Leo Grady, “Random Walks for Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1768–1783, November 2006.
- [57] Min Meng, Lubin Fan, and Ligang Liu, “A comparative evaluation of foreground/background sketch-based mesh segmentation algorithms,” *Computers & Graphics*, vol. 35, no. 3, pp. 650–660, 2011.
- [58] Ariel Shamir, Lior Shapira, and Daniel Cohen-Or, “Mesh analysis using geodesic mean-shift,” *Visual Computer*, vol. 22, pp. 99–108, February 2006.

REFERENCES

- [59] Hitoshi Yamauchi, Seungyong Lee, Yunjin Lee, Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel, “Feature Sensitive Mesh Segmentation with Mean Shift,” *Shape Modeling and Applications, International Conference on*, pp. 238–245, 2005.
- [60] Fan R. K. Chung, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*, American Mathematical Society, 1997.
- [61] Jianbo Shi and Jitendra Malik, “Normalized Cuts and Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.
- [62] Anat Levin, Alex Rav-Acha, and Dani Lischinski, “Spectral Matting,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 10, pp. 1699–1712, 2008.
- [63] Roni Raab, Craig Gotsman, and Alla Sheffer, “Virtual Woodwork: Making Toys from Geometric Models,” *International Journal of Shape Modeling*, vol. 10, no. 1, pp. 1–30, 2004.
- [64] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe, “Geometry Images,” in *SIGGRAPH*, 2002, pp. 355–361.
- [65] Aleksey Golovinskiy and Thomas Funkhouser, “Randomized Cuts for 3D Mesh Analysis,” *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA)*, vol. 27, no. 5, December 2008.
- [66] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh, “Learning 3D Mesh Segmentation and Labeling,” *ACM Transactions on Graphics (SIGGRAPH)*, vol. 29, no. 4, 2010.
- [67] Aleksey Golovinskiy and Thomas A. Funkhouser, “Consistent segmentation of 3d models,” *Computers and Graphics*, vol. 33, no. 3, pp. 262–269, 2009.
- [68] Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng, “Style-content separation by anisotropic part scales,” *ACM Trans. Graph.*, vol. 29, no. 6, pp. 184, 2010.
- [69] J.A.Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 1999.

- [70] Ron Kimmel, “Intrinsic Scale Space for Images on Surfaces: The Geodesic Curvature Flow,” *CVGIP: Graphical Model and Image Processing*, vol. 59, no. 5, pp. 365–372, 1997.
- [71] L.T. Cheng, P. Burchard, B. Merriman, and S. Osher, “Motion of curves constrained on surfaces using a level-set approach,” *Journal of Computational Physics*, vol. 175, pp. 604–644, 2002.
- [72] Chunlin Wu and Xuecheng Tai, “A Level Set Formulation of Geodesic Curvature Flow on Simplicial Surface,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 647–662, 2009.
- [73] A. Spira and R. Kimmel, “Geometric curve flows on parametric manifolds,” *Journal of Computational Physics*, vol. 223, no. 1, pp. 235–249, 2007.
- [74] Dheeraj Singaraju and René Vidal, “Interactive image matting for multiple layers,” in *Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [75] David Mumford and Jayant Shah, “Optimal approximations by piecewise smooth functions and associated variational problems,” *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989.
- [76] Luminita A. Vese and Tony F. Chan, “A Multiphase Level Set Framework for Image Segmentation Using the Mumford and Shah Model,” *International Journal of Computer Vision*, vol. 50, no. 3, pp. 271–293, 2002.
- [77] T. Pock, D. Cremers, H. Bischof, and A. Chambolle, “An Algorithm for Minimizing the Piecewise Smooth Mumford-Shah Functional,” in *International Conference on Computer Vision (ICCV)*, Kyoto, Japan, 2009.
- [78] Jan Lellmann and Christoph Schnoerr, “Continuous Multiclass Labeling Approaches and Algorithms,” Tech. rep., Univ. of Heidelberg, Feb. 2010.
- [79] Mila Nikolova, Selim Esedoglu, and Tony F. Chan, “Algorithms for Finding Global Minimizers of Image Segmentation and Denoising Models,” *SIAM Journal of Applied Mathematics*, vol. 66, no. 5, pp. 1632–1648, 2006.

REFERENCES

- [80] Antonin Chambolle, “An Algorithm for Total Variation Minimization and Applications,” *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1-2, pp. 89–97, 2004.
- [81] Jim Douglas and H.H.Rachford, “On the numerical solution of heat conduction problems in two and three space variables,” *Trans. of the AMS*, vol. 82, no. 2, pp. 421–439, 1956.
- [82] Yuri Boykov and Marie-Pierre Jolly, “Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images,” in *International Conference on Computer Vision (ICCV)*, 2001, pp. 105–112.
- [83] Helmut Pottmann, Tibor Steiner, Michael Hofer, Christoph Haider, and Allan Hanbury, “The Isophotic Metric and Its Application to Feature Sensitive Morphology on Surfaces,” in *Proceedings of ECCV*, 2004, pp. 560–572.
- [84] Mark Meyer, Mathieu Desbrun, Peter Schroder, and Alan H. Barr, “Discrete differential-geometry operator for triangulated 2-manifolds,” *Visualization and Mathematics*, vol. 1, no. 3, pp. 35–57, 2002.
- [85] Gabriel Taubin, “Estimating the tensor of curvature of a surface from a polyhedral approximation,” in *International Conference on Computer Vision (ICCV)*, 1995, pp. 902–907.
- [86] Jiaya Jia, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum, “Drag-and-drop pasting,” *ACM Transactions on Graphics (SIGGRAPH)*, vol. 25, no. 3, pp. 631–636, July 2006.
- [87] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum, “Lazy Snapping,” *ACM Transactions on Graphics (SIGGRAPH)*, vol. 23, no. 3, pp. 303–308, 2004.
- [88] Stanley Osher and James A. Sethian, “Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations,” *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

REFERENCES

- [89] Chunlin Wu, Jiansong Deng, Falai Chen, and Xuecheng Tai, “Scale-Space Analysis of Discrete Filtering over Arbitrary Triangulated Surfaces,” *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 670–709, 2009.
- [90] Herbert Federer, “Curvature Measures,” *Transactions of the American Mathematical Society*, vol. 93, no. 3, pp. 418–491, 1959.
- [91] Patrick Mullen, Alexander McKenzie, Yiying Tong, and Mathieu Desbrun, “A Variational Approach to Eulerian Geometry Processing,” *ACM Transactions on Graphics (SIGGRAPH)*, vol. 26, no. 3, 2007.
- [92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press, second edition, 1992.
- [93] Michael Kass, Andrew P. Witkin, and Demetri Terzopoulos, “Snakes: Active contour models,” *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.
- [94] Ravi Malladi, James A. Sethian, and Baba C. Vemuri, “Shape Modeling with Front Propagation: A Level Set Approach,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 2, pp. 158–175, 1995.
- [95] Sivan Toledo, Doron chen, and Vladimir Rotkin, “Taucs: A library of sparse linear solvers,” Website, 2003, <http://www.tau.ac.il/~stoledo/taucs/>.
- [96] Luc Buatois, Guillaume Caumon, and Bruno Lévy, “Concurrent number cruncher - A GPU implementation of a general sparse linear solver,” *International Journal of Parallel, Emergent and Distributed Systems*, vol. 24, pp. 205–223, 2009.
- [97] Thomas Pock, Antonin Chambolle, Daniel Cremers, and Horst Bischof, “A convex relaxation approach for computing minimal partitions,” in *Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 810–817.
- [98] Tamal K. Dey, Joachim Giesen, and Samrat Goswami, “Shape segmentation and matching with flow discretization,” in *Proc of 8th International Workshop on Algorithms and Data Structures (WADS)*, 2003, pp. 25–36.

REFERENCES

- [99] Bruno Lévy and Richard Hao Zhang, “Spectral Geometry Processing,” in *ACM SIGGRAPH Course Notes*, 2010.
- [100] Tamal K. Dey, Pawas Ranjan, and Yusu Wang, “Convergence, stability, and discrete approximation of Laplace spectra,” in *Proc of ACM/SIAM Sympos. Discrete Algorithms (SODA)*, 2010, pp. 650–663.
- [101] Ulrike von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.
- [102] Lars W. Hagen and Andrew B. Kahng, “New spectral methods for ratio cut partitioning and clustering,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 11, no. 9, pp. 1074–1085, 1992.
- [103] Amaël Delaunoy, Ketut Fundana, Emmanuel Prados, and Anders Heyden, “Convex Multi-Region Segmentation on Manifolds,” in *International Conference on Computer Vision (ICCV)*, 2009.
- [104] R. Tyrrell Rockafellar, *Convex Analysis*, Princeton University Press, 1970.
- [105] L. D. Popov, “A modification of the Arrow-Hurwicz method for search of saddle points,” *mathematical Notes*, vol. 28, no. 5, pp. 845–848, 1980.
- [106] Marzia Polito and Pietro Perona, “Grouping and dimensionality reduction by locally linear embedding,” in *NIPS*, 2001, pp. 1255–1262.
- [107] Lihi Zelnik-Manor and Pietro Perona, “Self-Tuning Spectral Clustering,” in *NIPS*, 2004.
- [108] Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng, and Xue-Cheng Tai, “Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 29, no. 2, pp. 517–526, 2010.
- [109] Eran Borenstein and Shimon Ullman, “Combined top-down/bottom-up segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 12, pp. 2109–2125, 2008.

List of Publications

Journal

1. Juyong Zhang, Jianmin Zheng, Chunlin Wu and Jianfei Cai. Variational Mesh Decomposition. *ACM Transactions on Graphics*. 2012. Accepted.
2. Thi Nhat Anh Nguyen, Jianfei Cai, Juyong Zhang and Jianmin Zheng. Robust Interactive Image Segmentation Using Convex Active Contours. *IEEE Transactions on Image Processing*. 2012. Accepted.
3. Juyong Zhang and Chunlin Wu, Fast Optimization for Multichannel Total Variation Minimization With Non-quadratic Fidelity, *Signal Processing*, vol 91(8), 1933-1940, 2011
4. Chunlin Wu, Juyong Zhang, Yuping Duan and Xue-cheng Tai, Augmented Lagrangian Method for Total Variation based Image Restoration and Segmentation over Triangulated Surfaces, *Journal of Scientific Computing*, vol 50(1), 145-166, 2011
5. Chunlin Wu, Juyong Zhang, and Xue-cheng Tai, Augmented Lagrangian Method for Total Variation Restoration with Non-quadratic Fidelity, *Inverse Problems and Imaging*, vol 5(1), 237-261, 2011
6. Juyong Zhang, Jianmin Zheng and Jianfei Cai, Interactive Mesh Cutting Using Constrained Random Walks, *IEEE Transactions On Visualization and Computer Graphics*, vol 2(2), 44-67, 2010
7. Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng and Xue-cheng Tai, Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow, *Computer Graphics Forum (In Proc. of Eurographics 2010)*, vol 29(2), 517-526, 2010
8. Wei Guan, Jianfei Cai, Juyong Zhang and Jianmin Zheng, Progressive Coding and Illumination and View Dependent Transmission of 3D Meshes Using R-D Optimization, *IEEE Transactions on Circuits and Systems for Video Technology*, vol 20(4), 575-586, 2010.

Conference

1. Thi Nhat Anh Nguyen, Jianfei Cai, Juyong Zhang and Jianmin Zheng. Constrained Active Contours for Boundary Refinement in Interactive Image Segmentation. *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2012.
2. Juyong Zhang, Jianmin Zheng and Jianfei Cai, A Diffusion Approach to Seeded Image Segmentation, *in the proceeding of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun, 2010
3. Juyong Zhang, Jianfei Cai, Wei Guan, Jianmin Zheng and Susanto Rahardja, Re-examination of applying wavelet based progressive image coder for 3D Semi-regular mesh compression, *in the proceeding of International Conference on Multimedia & Expo (ICME)*, Jun, 2008
4. Wei Guan, Juyong Zhang, Jianfei Cai and Jianmin Zheng, Illumination and View Dependent Progressive Transmission of Semi-regular Meshes with Subdivision Connectivity, *in the proceeding of the International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, Dec, 2007