MINIMUM RATIO CONTOURS FOR MESHES

by

Andrew Clements
B.Comp., University of Guelph, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2006

APPROVAL

Name: Andrew Clements

Degree: Master of Science

Title of thesis: Minimum Ratio Contours For Meshes

Examining Committee: Dr. Binay Bhattacharya
Chair

Dr. Richard Zhang, Senior Supervisor

Dr. Torsten Möller, Supervisor

Dr. Daniel Weiskopf, Supervisor

Dr. Ghassan Hamarneh, SFU Examiner

Date Approved: Jun. 26/06

Abstract

We present a novel minimum ratio contour (MRC) algorithm, for discretely optimizing contours on the surface of triangle meshes. We compute the contour having the minimal ratio between a numerator and a denominator energy. The numerator energy measures the bending and salience (feature adaptation) of a contour, while the denominator energy measures contour length. Given an initial contour, the optimal contour within a prescribed search domain is sought. The search domain is modeled by a weighted acyclic edge graph, where nodes in the graph correspond to directed edges in the mesh. The acyclicity of this graph allows for an efficient computation of the MRC. To further improve the result, the algorithm may be run on a refined mesh to allow for smoother contours that can cut across mesh faces. Results are demonstrated for postprocessing in mesh segmentation. We also speculate on possible global optimization methods for computing a MRC.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

It could be argued that nothing is more basic than perception. Looking around our environs, our eyes perceive a multitude of objects. These objects have many qualities; in describing a person we might note that he has a large nose, small ears, two hands, with five fingers on each hand, and so on. Collectively, we can refer to these characteristics as *features*. But *what* is the definition of a feature?

Looking through the Longman Modern English Dictionary [54], we find the following definition for a *feature*: "a prominent aspect of something". Now, *something* could be *anything*, and this is rather disconcerting. In computer graphics and computer vision, we are typically interested in scenes composed of objects. Therefore we refine our problem, and *something* becomes a physical object. Our revised problem is now to locate features of physical objects.

After formulating the input to our problem, we still lack details as to what the outputs should be. Our clue, is that a feature must be *prominent* on a physical object. Were we to think for a moment, we would certainly be able to think of prominent features of everyday objects. A person's nose may be prominent. Another person's nose may barely be discernable, but his ears may be very prominent. In reality, the amount or type of prominence recognized by differing people may be related to their past experiences. Obviously, any feature extraction algorithm that works without prior knowledge would not be able to model this. However, it is still of great interest to develop algorithms that rely on the intrinsic properties of features.

Our discussion so far still in no way offers insight into how to precisely quantify features, and leaves us in an unsatisfactory state of affairs. Perhaps if we looked up 'prominent'

in a dictionary, we would obtain a more specific description of a feature? We could even keep looking up additional terms, but by the circular definitions of English words, we would eventually end up with a set of words which are synonyms for prominent, such as (but not limited to): salient, outstanding, striking, and spectacular. This poses a conundrum, since there is no precise definition of a feature. It is not surprising that researchers have struggled to provide a precise mathematical definition of a feature, when its English definition is rather vague.

What would a mathematical model of a feature entail? What guidance for determining features can be used? If we attempt to mimic what a human would perceive as features, we should incorporate perceptual theories from psychology when searching for features, such as the Gestalt laws [29] and the minima rule [18]. Unfortunately, while these theories attempt to describe general properties of what and how humans perceive features, they do not provide a computational or mathematical description of them. Thus it is often left to the implementor to determine the precise mathematical model of a feature.

To implement a computer algorithm capable of detecting features on physical objects, a model of the physical objects needs to be taken as input. We choose triangle meshes, since they are the standard format for describing 3D objects in computer graphics. The returned features, or output of the algorithm, are represented by closed contours. Contours are attractive since they divide a mesh (surface) into two distinct regions, or parts, resulting in a mesh segmentation. The segmentation of an object into parts can then be used as input to further processing tasks, such as object recognition. A theory for decomposing objects into parts is due to Hoffman and Singh [19] and is based on the minima rule. The minima rule provides intuition as to where the cut boundaries (as perceived by humans) between different object parts should occur. More details relating to the minima rule, and perceptual considerations taken into account, are given in Section 3.3 and Section 3.4.

1.1 Applications

Obtaining a segmentation of a mesh into patches, where the patches are delimited by contours, is an important task in geometry processing. The segmented patches can be used as input to subsequent mesh processing algorithms, such as mesh parameterization, mesh morphing, 3D shape matching and recognition, and mesh editing, to name a few. One of the

motivations for applying mesh segmentation to these problems, is that often a global computation over the entire mesh is either too computationally expensive, or that the problem can be more accurately performed on a patch by patch basis. For instance, large distortion typically results when performing a global parameterization of a complicated, closed manifold mesh. By dividing the mesh into patches, it is possible to obtain lower distortion when parameterizing each individual patch [15].

In applications such as spectral mesh compression [25] and mesh watermarking [42], where expensive eigenvector computations for mesh Laplacian operators are required, the division of a large mesh into smaller patches is necessary from a computational point of view. The eigenvector computations can then be carried out on a patch by patch basis. Although it is almost always desirable to perform mesh segmentation along boundaries deemed as features, the applications mentioned so far do not make this requirement absolute.

For the problem of object recognition however, it becomes necessary that the segmentation boundaries correspond to visually significant features. This is the case for the proposed work, which is designed to aid in feature extraction and segmentation for the purpose of object recognition. In this context, the patches that segment a mesh correspond to the visually salient parts of the object that that mesh describes. The algorithm presented returns contours which delimit the meaningful parts of an object; such contours are a natural and practical choice, by virtue of the fact that they divide a mesh into two regions.

The proposed work is just one of many steps necessary for performing a typical object recognition task such as shape matching [47, 48]. An example of a pipeline to carry out the task of shape matching is the following: an initial given input mesh is segmented, either automatically [28, 38], or semi-automatically (such as in mesh scissoring algorithms [17, 36]). The initial segmentation yields a set of contours which, using the proposed work in this thesis, are then used to post-process the segmented contours. The refinement of the contours during post-processing ensures a perceptually meaningfull segmentation. This refined segmentation can then be used to construct a *skeleton* of the mesh model, which represents the topological relations between segmented model parts. A skeleton is represented by a graph, where nodes in the graph correspond to segmented parts. Edges are inserted into the graph to link nodes whose segmented parts are adjacent. Skeletons derived from mesh models can then be used as input to shape matching algorithms [47] where, given a query model, a search is performed within a database of models in order to return models which are similar to the query model.

Skeletons are versatile structures, and can also be used as inputs for other geometry processing applications. For instance, they can be used to perform such tasks as collision detection [37], mesh morphing [47], and mesh animation [7]. Additionally, with the use of skeletons, it is also possible to perform some applications on an individual part basis, such as component based shape matching [47], or component based morphing [57].

1.2 Motivation

Snakes, or active contours, have been applied widely in image analysis for feature extraction and segmentation [2, 8, 9, 27]. More recently, they have been adapted for use on triangle meshes [4, 35, 41, 22]. The snake method most frequently uses gradient descent to reach a local minimum of a boundary cost function, which is typically given by a summation of the external and internal energy of the snake. The cost function represents a *total energy*, which, if optimized globally, yields a trivial solution (an elaboration on this topic is provided in Section 2.1). Thus the fact that snakes can often adapt to image or geometric features well is more of an artifact of the local nature of the optimization procedure and not necessarily the result of a snake having reached a state that is close to the global minimum [21].

This thesis advocates the use of an *energy ratio* as the cost function, using a global optimization approach which is constrained within a prescribed search region. Although the energies making up the ratio can vary, we focus on the mesh segmentation problem and define the numerator weight to combine the internal (for smoothness) and external (for feature adaptation) energies of a contour, and the denominator weight as contour length.

To illustrate the differences between the snake energy and the minimum ratio energy, consider several contours which segment fingers from a hand, as shown in Figure 1.1. The contour labelled F is the contour which best segments the ring finger (second finger from the left) from the rest of the hand. This contour passes through two large concave, feature regions (between adjacent fingers), while the connecting parts travel across the convex regions on the front and back of the finger. Notice that the segments passing through the convex regions (which connect the feature regions) are as short as possible while maintaining smoothness.

A snake initialized with contour E, may descend into the desired final contour F, or depending on initialization, the snake may traverse in the other direction, stopping at contour D. If the initial contour started at C instead of E, then the snake would settle in the locally

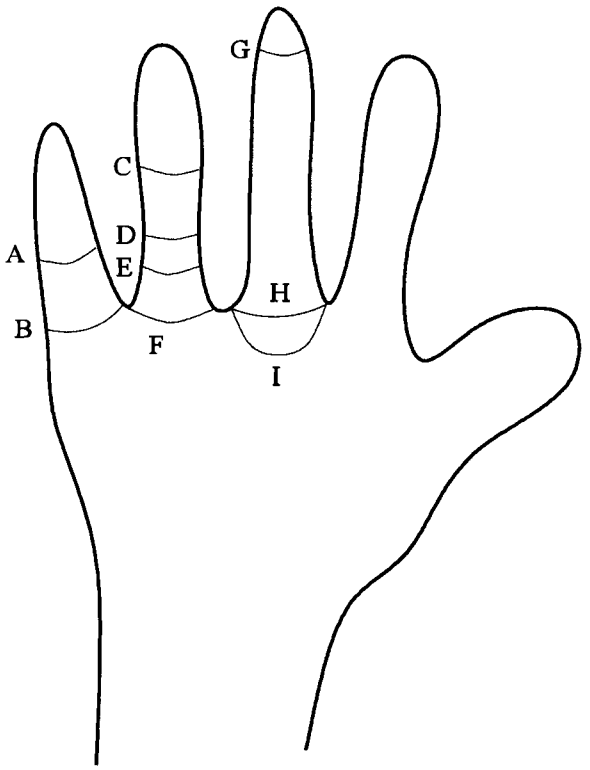


Figure 1.1: A hand, showing several contours which segment fingers from the hand in different places. Differences between the minimum ratio contour (MRC) algorithm and snakes are explained in the accompanying text.

optimal position at contour D, and is therefore prone to detecting local minima. If the minimum ratio contour (MRC) algorithm developed in this thesis (introduced in Section 1.3) is used with a search space which includes the optimal contour F, then the desired optimal contour F would be found irrespective of which initial contour - C,D, or E - is used.

In another situation, consider a snake starting at contour A on the pinky finger (the leftmost finger). Such a snake would travel towards the end of finger, and then disappear at the end of the finger. Again, with a sufficient width search space, the minimum ratio contour algorithm would find the desired contour B.

To counter such undesirable behaviour of a snake, a balloon like force can be added [10], so that it seeks to continually expand outwards, until the force attracting it to features is stronger than the inflationary force. Thus the snake initialized on the middle finger (third finger from the left) at G would grow, and travel down the length of the finger. The

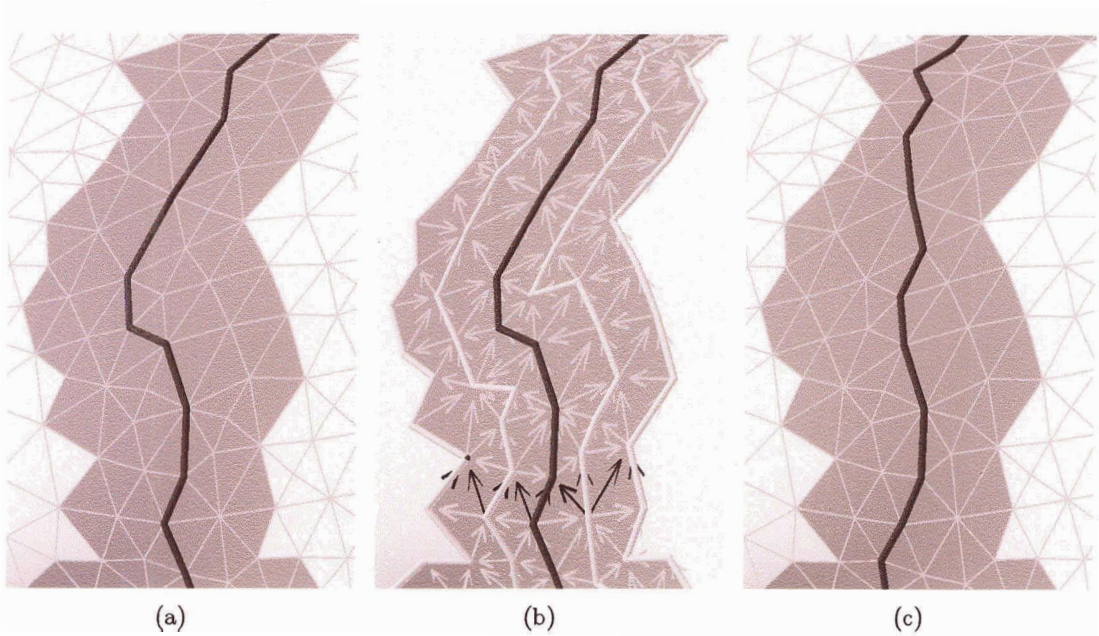


Figure 1.2: Major steps of the minimum ratio contour (MRC) algorithm. (a) Given an initial contour (dark gray), a search band around it with a user-defined width is found. (b) An acyclic graph is constructed, where nodes in the graph correspond to directed mesh edges (as shown). Arcs in the search graph are inserted between consecutive directed mesh edges. The dark directed edges are duplicated, and form an edge cut across the search space. The edge cut is used to specify the source and destination nodes in the acyclic graph. (c) Optimal, minimum ratio contour. It is found by solving a series of minimum ratio path problems in the acyclic graph.

inflationary force counters the length energy, reducing the effect of keeping the snake short. However, based on the snake's chosen initial elastic parameters, it is prone to *overexpansion*, resulting in a contour similar to I, instead of the desired final contour H. Such overexpansion can typically occur when segmenting parts from three dimensional models, since the resulting contours may have large sub-segments which must travel through featureless areas. The snake would expand past the desired contour in these regions until its inflationary force is matched by the length and bending force.

1.3 Brief overview of our approach

The minimum ratio contour (MRC) algorithm proposed uses graph based minimization to determine the optimal contour on the surface of a mesh. Here, the optimal contour

within a constructed search graph is sought which minimizes the defined energy. In contrast to continuous snaking approaches [4, 8, 9, 35] (which can be discretized), we begin by discretizing the continuous case, and obtain a graph which approximates the continuous case. Unlike snaking approaches, we advocate minimizing a ratio of energies, that is, a ratio of a numerator energy to a denominator energy. The numerator energy measures the bending and salience (feature adaptation) of a contour, while the denominator energy measures its length. The optimal (minimal ratio) contour, is the one that has the most numerator energy per unit length. This graph based approach has the advantage that it decouples contour optimization from feature extraction and selection.

One of the main challenges in applying the minimum ratio idea to contour optimization on an irregular-grid surface mesh is to come up with an appropriate search space, which is a weighted graph. We wish to ensure that the globally optimal cycle is simple (no self-intersections), efficient to compute, and represents a smooth and perceptually meaningful segmentation. These are accomplished with the construction of graphs based on the ratio energy definition. The algorithm proceeds in two stages. First an ambient graph is constructed from the given mesh, and then an acyclic search graph is derived from the ambient graph, and a search is carried out in the acyclic graph to determine the optimal contour.

In the first stage, an ambient graph is constructed from a given mesh, where cycles in the graph correspond to contours on the mesh. Two different graph structures may be built: the first restricts contours to only pass through the edges and vertices of the mesh, and is referred to as the *regular* case. The second allows contours to cut across mesh faces to increase the resolution and smoothness of contours, and is referred to as the *refined* case. In these graphs, every node is associated with a directed edge or line segment on the surface of the mesh, and directed arcs are inserted between nodes which correspond to consecutive directed edges (or line segments) on the mesh. The terms nodes and arcs are used exclusively with the constructed graphs, to distinguish them from vertices and edges, which are only associated with meshes. The energy of the transition from one node (mesh edge) to another node (mesh edge) along a directed arc is related to the bending and salience between the two edges.

The next stage of the minimum ratio contour algorithm is to construct an acyclic search graph. This graph contains a subset of the nodes and arcs of one of either the regular or refined ambient graphs constructed above. Solving a series of minimum ratio path problems in the acyclic graph, we obtain the optimal minimum ratio contour (MRC) within the acyclic

search graph.

Figure 1.2 illustrates the major steps of the MRC algorithm, for the regular case. Given an initial contour on a mesh, a search space consisting of a band (with a user specified width) surrounding the initial contour is first constructed, as shown in Figure 1.2(a). Within the search space, a weighted acyclic graph is constructed whose nodes are directed mesh edges, as shown in Figure 1.2(b). Next, a minimal edge cut of the search band is found, as shown in Figure 1.2(b). Each mesh edge along the cut is duplicated, resulting in pairs of corresponding source and destination edges. The problem of computing a minimum ratio cycle in the search space is reduced to a related problem of finding a series of minimum ratio paths in the constructed acyclic graph. A minimum ratio path (MRP) problem is solved for each pair of corresponding source and destination nodes in the acyclic graph. The path with the minimum ratio corresponds to the cycle in the search graph that has the minimum ratio energy. The minimum ratio contour in the search space is shown in Figure 1.2(c). The entire optimization process, with additional details, is presented in Chapter 4.

1.4 Contributions

The main contribution of this work is to offer a minimum ratio approach for performing the task of feature extraction and segmentation on triangle meshes — this has not been attempted before. One of the main challenges has been to find an algorithm which is efficient and that can incorporate hard constraints, both of which are not possible with a general minimum ratio algorithm. We also need to ensure that the returned contours take into account perceptual considerations. Both of these requirements are met with our novel graph construction and energy definition. The drawbacks of snakes, such as descending into local minima, and the limitations of the classical linear energy minimization, are overcome by minimizing a ratio of energies instead. A comparison between snakes and minimum ratio techniques are provided in Section 2.1 and Section 2.2.

In addition to having a different energy to minimize, the proposed algorithm differs from previous implementations of mesh snakes in that a discrete, graph-based search is used to find a globally optimal contour within a restricted search space. The approach operates directly on the surface mesh and not in a parameter domain [35]. It is efficient, flexible, and general enough to allow the generation of both closed and open curves, curves cutting across mesh faces, and the incorporation of both hard and soft constraints. Topological changes

can also be detected and handled quite easily. It is also intended that the current approach will spur development of graph optimization methods to the field of geometry processing. To summarize, the main features of this work are:

- **Application of Minimum Ratio to Meshes:** The minimum ratio technique is adapted for use on triangle meshes for the purpose of feature extraction. There is no previous work in this area.
- **Energy Definition:** The energy of a contour is defined as a ratio. Its definition incorporates the minima rule from psychology, models the Gestalt laws of proximity and continuity, and is scale invariant. Unlike snake methods, a non-trivial solution is the minimizer.
- **Efficiency:** By reducing the space of admissible contours searched, through the construction of an acyclic graph, an efficient linear time sub-routine (finding a negative path in the acyclic graph) is presented for aiding in the determination of the minimum ratio contour within the given search space. This contrasts with general minimum ratio techniques, which are much more computationally expensive.

Additionally, the MRC algorithm presented satisfies a number of desirable properties. It should be noted that neither snakes [4, 8, 9, 35, 41, 22] nor general minimum ratio techniques [14, 21, 52] alone can satisfy *all* of the following requirements:

- Efficient computation of the optimal contour in the search space
- A non-trivial contour is the minimizer of the defined energy
- Efficient detection and control of collisions and topology change
- No mesh parameterization is needed and thus any parameterization artifacts (as in [35]) will not be present in our approach.
- Can incorporate hard and soft constraints
- Can produce both open or closed curves
- User intervention is trivial if desired
- Can constrain the optimal contour to only pass through original edges in the mesh

- Can refine the mesh to obtain smoother optimal contours
- Optimization approach allows contour to avoid local minima when possible
- Energy definition relies on the minima rule and Gestalt laws

1.5 Document organization

The rest of this work is organized as follows. In Chapter 2, we survey snake approaches which optimize contours for images or surface meshes, as well as previous work using minimum ratio techniques. Chapter 3 details the construction of the ambient graphs from the given input mesh. Either a regular ambient graph can be constructed (Section 3.1), which constrains the contour to only pass through mesh edges), or a refined ambient graph can be utilized (Section 3.2) to allow the contour to cut across mesh faces, improving the smoothness quality of the final contour. Section 3.3 describes the reasoning and principles that are incorporated into the energy definition, while Section 3.4 details the explicit energy form and its computation.

Chapter 4 describes the construction of an acyclic graph, using a subset of the nodes and arcs from either the regular or refined ambient graphs. Section 4.1 provides an overview of the algorithm, while Section 4.2 contains the explicit details to obtain the minimum ratio contour within a given search band. Section 4.4, Section 4.5, and Section 4.6 discuss the incorporation of hard constraints, soft constraints, and open curves into the algorithm, respectively.

Experimental results are provided in Chapter 5, and Chapter 6 concludes and outlines possible extensions to improve the existing algorithm, such as introducing an additional length bias into the energy definition (Section 6.1.1), or the inclusion of area or region information into the energy definition (Section 6.1.2).

Chapter 2

Previous Work

This chapter surveys research literature which is relevant to the work presented in later chapters. Image and mesh snakes are reviewed in Section 2.1, and their drawbacks are established. Section 2.2 presents a general, ratio energy definition of a contour, which addresses the shortcomings of the snake energy. Lastly, Section 2.3 describes other relevant research to the topic at hand.

2.1 Snakes

One of the most well-known boundary-based image feature extraction and segmentation methods uses the active contour model, first introduced by Kass et al. [27]. In this approach, a snake is represented by a curve. Given a parameterization of a snake $v(s) = (x(s), y(s))$ the evolution of the snake over time is controlled by minimizing its associated energy

$$\int int(v) + ext(v)\, ds. \tag{2.1}$$

The energy is a combination of the snake's internal energy term $int(v)$, and external energy term $ext(v)$. Minimizing the internal energy tends to shorten the snake while keeping it smooth (reduces bending). External energy serves to attract the snake toward features (local minima occur over feature regions), and its definition is application dependent.

If the snake is discretized, we refer to each of the n points or line segments that comprise the snake as a *snaxel*. In the case of images, snaxels are typically pixels (though sub-pixel resolution is possible), while for meshes snaxels can be points or line segments on the surface

of the mesh. The energy of a discretized snake is defined as the sum of the energies at each snaxel in the snake.

Additional energy terms can be incorporated into the energy definition of a snake. For instance, Amini et al. [2] shows how to optimize image snakes in the presence of hard constraints. This is achieved by adding a constraint energy term, and then making use of dynamic programming techniques. In this case, every snaxel on the snake can move to at most m different positions, and the optimal snake is found in $O(nm^3)$ time (where n is the number of snaxels in the snake). Williams and Shah [56] in turn propose a greedy algorithm, which may not find the locally optimal snake, but runs in $O(nm)$ time.

Cohen [10] proposes an additional energy term to facilitate inflation of the snake, so that the user only needs to specify a small initial snake, and it grows outwards until its progress is halted when it is attracted to feature regions with low external energy.

There are two general approaches for implementing snakes. These are governed by whether an *implicit* or *explicit* representation of the snake is used. Explicit models [2, 3, 4, 22] store the exact locations (snaxels) that the snake passes through, while implicit methods [8, 9] consider the snake positions to lie in the zero level set of a function defined over an ambient space. Implicit models need not track topological changes explicitly, but do not allow intuitive user control of the snake. This is in contrast to explicit models, where the addition of constraints (snaxels that the snake must pass through) is straightforward, but extra techniques are needed to detect self-intersections of the snake and to update their topology appropriately [39].

Snakes have been successfully applied to feature extraction and segmentation for image analysis, but only recently have they been extended to work with surface meshes [4, 35, 22]. An early work on mesh snaking is due to Milroy et al. [41], which uses a greedy algorithm to segment a wrap-around model. Jung et al. [22] adapt the fast greedy approach of Williams and Shah [56] in a straightforward manner to triangle meshes. Lee and Lee [35] take a different approach, by first locally parameterizing the snake from the surface of a 3D mesh onto a 2D domain, then evolving the snake in the plane using techniques developed for images, and finally mapping the 2D snake back onto the 3D mesh. The parameterization and image snake movements are carried out patch by patch, which overlap each other to ensure smoothness of the overall mesh snake across patch boundaries. To avoid difficulties due to parameterization artifacts, Bischoff and Kobbelt [3] implement an explicit, parameterization-free mesh snaking algorithm, where snaxels may lie on the lines

of a uniform grid, so that topology control and self-intersections of the snake are handled in an efficient and controlled manner. This technique has later been extended to triangle meshes [4]. However, as with all active contour models, the final snake is only locally optimal.

A scissoring technique that uses a snake approach is given by Funkenhouser et al. [17]. A graph based optimization is applied to find an optimal contour within a given search region. The search region is usually a small neighbourhood surrounding an initial segment of a contour given by a user, and the optimal contour is found using Dijkstra's algorithm. Fundamentally, their method is similar to a snaking approach, since the cost function that they are minimizing is a *total energy*. This differs from our approach, since we advocate the use of a *ratio energy*, to remove bias towards shorter contours.

The advantages of snakes are that implementations are typically very fast, and explicit models provide direct control over the snake (in the form of constraints and user-intervention). However, snakes suffer in a number of ways: they are local in nature, and are prone to find local minima (as some researchers [17] performing mesh processing have found), and are thus highly dependent on the initial contour. The user is required to select a number of parameters (usually in an ad hoc manner), and most troubling and rather importantly, the global minimizer of the classical snake energy is in general, *not* the desired solution.

It is well known that methods to optimize snakes using gradient descent are inherently local, as it is not guaranteed that the global minimum will be found as a solution. For example, consider finding an optimal snake (contour) in the non-trivial region of the mesh shown in Figure 2.1. Here, edges of the mesh are assigned weights that correspond to external feature energy. Lower external energy values indicate features, and snakes passing through such edges are more desirable. The snake traveling from edge S to edge T with the lowest ratio of external energy to length is sought. For the sake of simplicity, internal bending energy weights are ignored, and the length of a snake is taken to be the number of edges that it is composed of.

In Figure 2.1, three (open) snakes are shown starting from edge S, and ending at edge T, coloured red, blue, and green. If, starting from the initial snake coloured blue, gradient descent is used, the locally optimal snake in green is obtained. This contrasts with the graph optimization approach used in this work to find a minimum ratio contour (or snake). With this approach, the globally optimal snake (within a pre-defined search space) will be

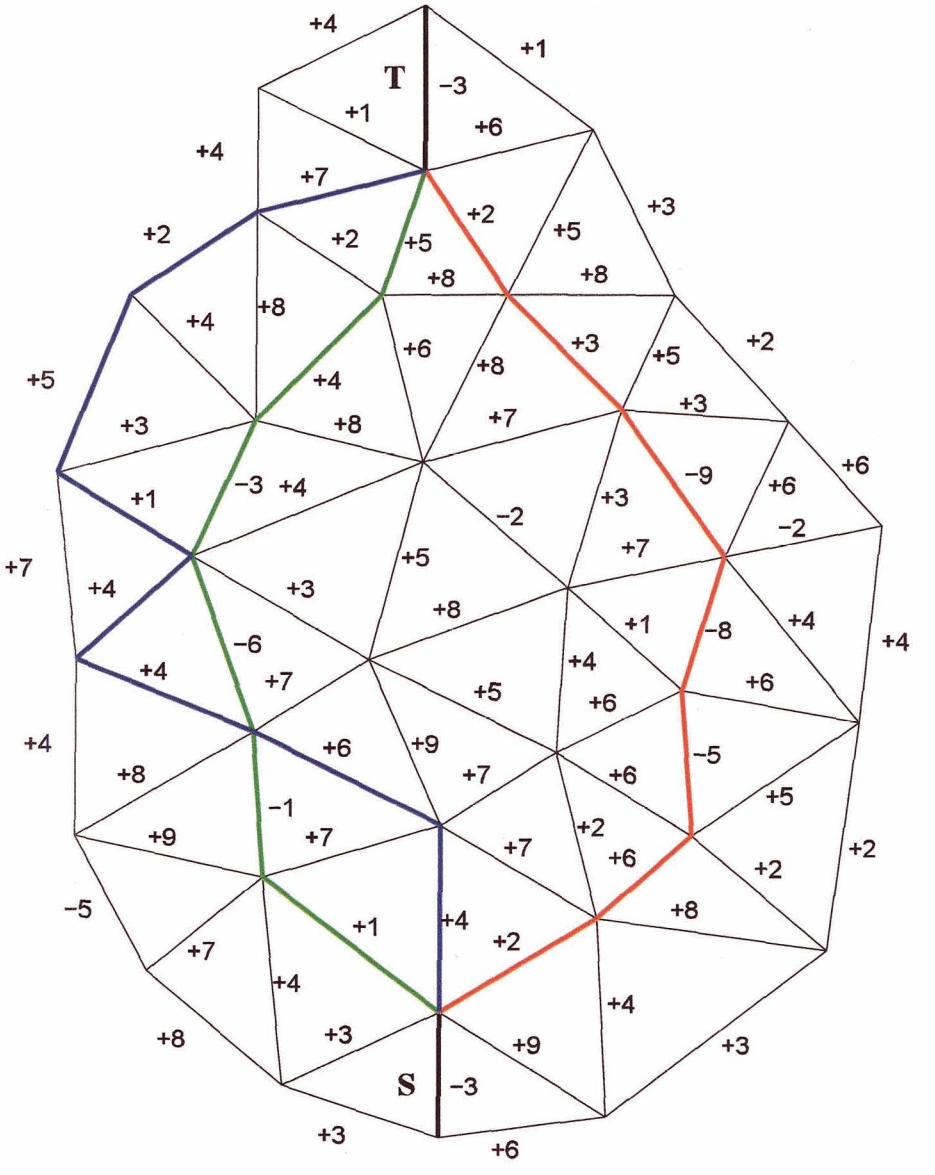


Figure 2.1: A region of a mesh. Three open snakes are shown starting from edge S, and ending at edge T. The energy ratio of the three snakes are: blue= $27/10 = 2.7$, green= $-6/8 = -0.75$, red= $-15/9 = -1.66$. These energy ratios are obtained by dividing the sum of external energy weights along the snake by the number of edges in the snake. Therefore, as explained in the accompanying text, the globally optimal snake is the lowest ratio snake, the red one. If, starting from the initial snake coloured blue, gradient descent is used, the locally optimal snake in green will be found. However, the graph optimization approach used in this work will find the globally optimal snake with a given search region.

returned irrespective of the initial contour. In this case, the locally optimal snake obtained using gradient descent differs from the globally optimal snake, coloured in red.

Attempts to overcome the local nature of snakes, and to find a global minimum of the energy have serious drawbacks. If the classical non-negative snake energy in equation 2.1 is used, finding a global minimum will result in finding a trivial snake, of length zero and of zero energy. If negative energies are allowed, it is possible that a negative weight contour exists; any other contour appended to this negative contour would have energy unbounded below, creating an ambiguous problem. Even if a negative cycle does not exist, it is possible that the minimizing contour can self-intersect or self-overlap. As Jermyn and Ishiwaka [21] have pointed out, attempts could be made to minimize over simple (non-intersecting boundaries), but this constraint creates an NP-hard problem since solving this problem in polynomial time would enable a polynomial time solution to the Hamiltonian cycle problem.

Therefore, implementations of snakes are successfull *precisely* because they find local minima. Attempts to find the global minima would produce an undesired result (trivial solution), produce an ill-posed problem, or have no known polynomial time solution.

2.2 Minimum ratio techniques

To overcome the limitation of the linear form of the snake energy, several authors [21, 52] have proposed using a ratio energy. In this framework, the optimal contour sought is the one that has the lowest ratio of feature and bending energy to length. This is accomplished by utilizing a graph theoretic search to obtain the minimum ratio cycle in a graph. The formulation of the ratio energy for a contour is designed to avoid the trivial solutions that are present when using a linear sum of energies, as snakes do (see equation 2.1). The general form of the ratio energy is given by

$$\frac{\int int(v) + ext(v)\, ds}{arclength(v)}. \tag{2.2}$$

It is designed to remove the bias towards short, trivial contours. The optimal contour no longer depends on its length directly, but on its ratio of numerator energy to length. As with the snake energy in equation 2.1, the numerator energy models external feature energy and bending of a contour. If we disregard the internal bending energy momentarily and consider

two contours, one shorter than the other, the longer contour will have a lower energy *ratio* if it passes through more features per unit length than the shorter contour.

The minimum ratio methods are attractive since a global solution to the problem can be obtained in polynomial time. Research into these methods has been carried out in the field of image analysis, but so far has not been applied in any form to triangular meshes.

Cox et al. [14] first modeled the image segmentation problem as a *global* minimum ratio problem, to solve for a directed cycle over the image grid that minimizes a ratio of the cost of the perimeter of the segmented region to the benefit assigned to its enclosed interior. The graph over which the optimization is performed must be planar, and the algorithm to compute it is rather unwieldy. Their approach does have the advantage that it can force the minimum ratio cycle to pass through a given pixel (node) in their graph.

Of more interest is a similar approach due to Jermyn and Ishikawa [21], which finds an optimal region by minimizing the ratio of an energy flow across the boundary of the region over the internal boundary energy (usually Euclidean boundary length). Green's Theorem can be used to show that information about the interior region (for example, its homogeneity), can also be incorporated into the minimization. Their algorithms run in $O(nm)$ time, where n is the number of nodes (pixels in an image), and m is the total number of edges between all nodes.

Wang et al. [53, 52] use a different, bottom up approach for finding globally optimal contours in an image. First, edge fragments are found using Canny edge detection [6]. These fragments are preprocessed, and become nodes in their graph. Arcs between nodes (fragments) are inserted, and the external feature energy of an arc depends on the distance between fragments, as well as the smoothness of a spline which joins the edge fragments. The global minimum ratio cycle on their undirected fragment graph is then found. This corresponds to the contour which possesses small gaps and little bending between adjacent fragments. In contrast to the minimum ratio cycle problem solved by Jermyn and Ishiwaka [21], their optimization occurs over an undirected graph, and as such, the standard minimum ratio cycle algorithms do not apply. Therefore, several reductions to the problem are applied, until an equivalent problem of finding a minimum weight perfect matching is achieved. In general, computing a minimum weight perfect matching requires $O(n(m+n\log n))$ time (see [12] for a survey of such algorithms). The resulting contour found is however not guaranteed to be intersection free.

2.3 Other related work

A minimum ratio algorithm has been used by Veksler [49, 50] to solve another, different segmentation problem. Windows (connected subregions) of an image are found, for the purpose of stereo matching. The graph constructed from images is planar, and this enables a more efficient, $O(n\sqrt{n})$, negative cycle detection algorithm, which reduces the cost of finding a minimum ratio cycle.

The minimum ratio techniques described in the previous section share similarities with the well known normalized cut algorithm for image segmentation [45]. In this approach, a graph cut is sought that minimizes the normalized cut criterion. The optimization is performed by solving a generalized eigenvalue problem. This is a global approach, and requires expensive computations in order to acquire eigenvectors.

Another technique for extracting closed contours from images is that of Elder and Zucker [16]. Given a set of edge fragments, a prominence is assigned to pairs of fragments, which models the likelyhood that a contour would travel between fragments. The salience of a contour is defined as the product of the prominences along the contour, and a shortest path algorithm is used to find the optimal contour. In order to solve the problem efficiently, the search space must be heavily pruned.

In geometry processing, a well known task is to find approximate shortest paths between all pairs of vertices or faces on a mesh, for example to cluster mesh elements in mesh segmentation. This can be accomplished by using a dual graph [28, 38], where faces of the mesh are nodes, and nodes are connected if their faces are adjacent. Dijkstra's algorithm can then be used to find approximate shortest paths between vertices in the mesh. Such dual graphs have also been used for the purpose of mesh compression [31]. We briefly mention that other graph structures have been proposed for geometry processing, such as constructing a Riemannian Graph [20] for the purpose of surface reconstruction,

The graphs used in this work differ from previous graph constructions, in that directed edges of the mesh are represented by nodes, and arcs of the graph correspond to consecutive directed edges in the mesh. Such a graph enables the weight of an arc between two nodes (edges in the mesh) to depend on the curvature (bending) between edges in the mesh. Such information cannot be encoded into the weights of a graph where the nodes are vertices of a mesh. This approach could be useful in other applications where discrete paths which attempt to maintain straightness are desired. A possible example where straightness can be

incorporated is to find the best path continuation between two non-adjacent mesh edges. The resulting path should exhibit shortness while minimizing bending.

Chapter 3

Ambient Graph Construction

This chapter outlines the ambient graphs that are used later for deriving an acyclic search graph in Chapter 4. Section 3.1 describes the construction of a regular ambient graph from a mesh. In order to overcome the limitation that a contour may only traverse along the edges of a mesh in the regular case, Section 3.2 describes the construction of an ambient graph from a refined mesh. A refinement scheme similar to that of Lanthier et al. [33] is used to permit segments of a contour to cut across faces so as to avoid connectivity artifacts, improving the resolution and smoothness quality of the resulting contours.

Section 3.3 describes and explains the Gestalt laws [5] and the minima rule [18]. These are the perceptual considerations and theories that are taken into account when assigning numerator weights to arcs in the constructed ambient graphs. Section 3.4 details the explicit form of the numerator energy and its computation.

3.1 Regular ambient graph construction

Unlike mesh snakes which model continuous movements of a contour [35, 4], we discretize the process. A contour is represented by a circular sequence of line segments, where each line segment must lie on the surface of a mesh. For the regular optimization approach, we restrict contours to pass through only vertices and edges of the mesh. We now seek to construct the ambient graph that will be used in the construction of the acyclic graph in Chapter 4.

Given a mesh M, we use the graph $H = (V, E)$ to represent the connectivity of M. Here, V represents the set of vertices in M, and E represents the set of directed edges in M.

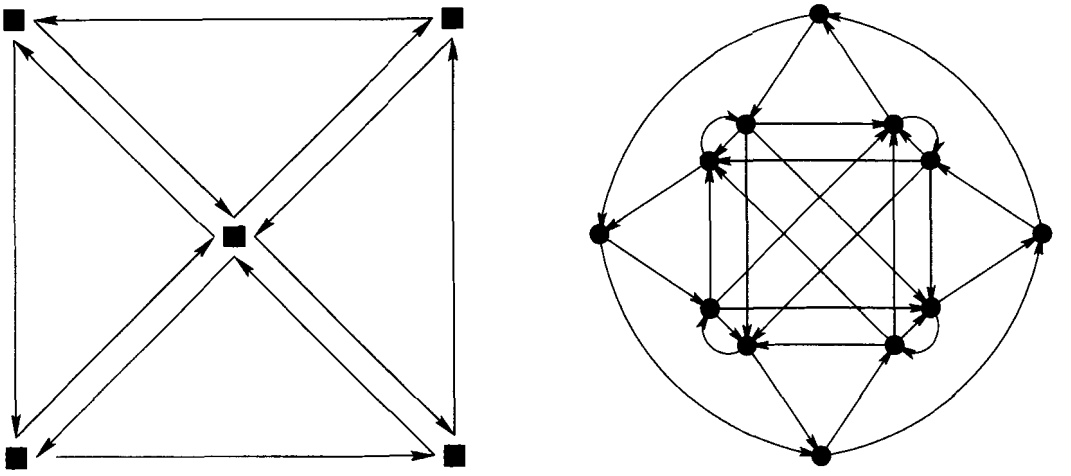


Figure 3.1: (a) A portion of a mesh. Vertices are represented by squares, and directed edges are represented by line segments whose arrows indicate their orientation. (b) Corresponding portion of the ambient graph constructed from the mesh in (a). Nodes are represented by circles, and are joined by directed arcs. The nodes are located at the midpoints of their corresponding directed edges shown in (a).

For our purposes, E consists of directed edges, and contains twice the number of undirected edges in the mesh. We now construct a regular graph $G = (N, A)$, where N is the set of nodes in G, and A is the set of directed arcs in G. The terminology *nodes* and *arcs* is used to avoid confusion with the vertices and edges of H. Each node in N is associated with one directed edge in E. That is, the nodes in our graph correspond to edges in the mesh. Nodes n_1 and n_2 in G, with associated edges e_1 and e_2 in E, are connected if the terminating vertex of e_1 is the starting vertex of e_2 — see Figure 3.1.

It is interesting to note that G is the *directed line graph* of H, that is $G = L(H)$. The key property in a line graph [51] is that arcs in the line graph correspond to pairs of adjacent arcs in the base graph H. This enables a scalar value to be attached to each arc of G, which can take into account the smoothness (bending) of the angle between the arc's two corresponding edges in H. Also observe that the nodes of a directed cycle in G corresponds to a cycle of directed edges in H, and thus to a contour in the mesh M.

3.2 Refined ambient graph construction

For certain applications, it may be desirable to restrict contours to only pass through vertices and edges of the mesh. This could for instance, ensure that no sliver triangles are created. Except in exceptional circumstances, a typical tessellation of a mesh yields contours which are bumpy or jagged, and so do not appear smooth. In order to overcome this limitation, the following refinement scheme [33] is used.

Specifically, we insert k equally spaced Steiner points along each edge of the original mesh. For each triangle T, we define a *directed chord*, a term chosen to distinguish it from directed edges and directed arcs, to be an ordered pair (p, q) for which one of following is true:

- p and q are adjacent points, either a vertex of T or a Steiner point, along the same edge;
- p is a vertex of T and q is a Steiner point on the triangle edge opposite to p;
- p is a Steiner point and q is another Steiner point that lies on a different edge of T.

Nodes are inserted into the refined ambient graph for every directed chord, and each refined contour is composed of a directed sequence of chords. The larger the value of k, the smoother a refined contour can be. Experimentally (see Chapter 5), a small k (such as $k = 2$), is quite sufficient to produce high-quality contours. Figure 3.2(a) shows the set of chords (undirected) in a triangle, with $k = 2$. In contrast, refinement using subdivision, as shown in Figure 3.2(b), may not result in sufficiently smooth contours. Note that our setting is a bit different from that of Lanthier et al. [33], since we work on a directed chord graph.

If the limit is taken as k approaches infinity, we find that all geodesic paths between any two vertices are present on the refined mesh. This is due to the fact that geodesics on discrete triangle meshes are characterized by a sequence of straight line segments, and are the straightest (with least bending) paths between two points; see Polthier and Schmies [43] for more details. Increasing k has the effect of providing better approximate geodesic paths between vertices, and this was the original purpose for the refinement scheme used by Lanthier et al.[33].

It is not hard to verify that there are $6k^2 + 12k + 6$ directed chords per triangle. If we join two chords as long as they are connected, then the size of the search graph may be quite large, even for small k. Although the acyclic graph construction presented in Chapter 4

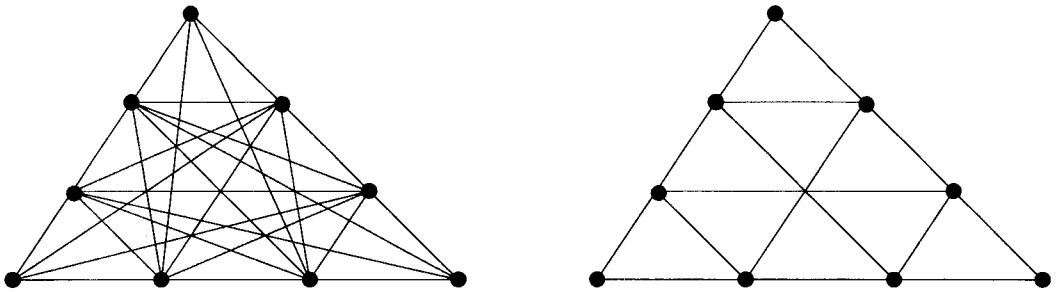


Figure 3.2: (a) Refinement with $k = 2$. Each line segment shown is an undirected chord. (b) Subdivision with the same number of Steiner points does not result in sufficient smoothness.

can reduce that complexity dramatically, we adopt two simple strategies first to prune the refined ambient graph. Specifically, we remove any arc that

1. connects two chords interior to the same triangle, or
2. connects two chords that sustain an angle exceeding a certain threshold; when the bending is too great. We have chosen to remove arcs from the refined graph when the angle between successive directed chords exceeds $\pi/2$ radians.

As in the regular unrefined case, the refined ambient graph G is constructed as the directed line graph of the refined chordal graph. Each node in the refined ambient graph is associated with one directed chord. Two nodes n_1 and n_2 in G, with associated chords c_1 and c_2, are connected if the terminating point of c_1 is the starting point of c_2.

3.3 Energy motivation

This section provides motivation for the principles that will be encoded into the energy definition. Given a constructed ambient graph G, of either the regular or refined type, the numerator energy (in equation 2.2) needs to be defined for every arc in G. In order to do so, a precise mathematical model is needed to express which regions on a mesh are features, and which are not.

This work does not seek to propose new insights into what humans consider a feature. Rather, it seeks to incorporate previous, known models of human perception, and attempt to find visually significant contours which these models dictate. One such theory is the

minima rule [18]. The minima rule states that when dealing with surfaces, humans perceive segmentation boundaries which consist of surface points at the negative minima of principal curvatures.

Also taken into consideration is Gestalt theory[5, 23, 29, 55], which deals with perceptual organization in humans. Perceptual organization is the process that organizes or partitions *low-level features* in a scene into groups. Here, low-level features could refer to specific points or edge fragments in an image or mesh. For the purpose of image segmentation, many researchers have attempted to incorporate such qualities into their algorithms, for example in [46, 52, 58].

A number of Gestalt laws have been proposed, and three of the most relevant ones that govern the salience of contours are as follows:

- **Law of Closure:** Contours, or curves that are closed, are more salient than curves which are open.
- **Law of Proximity:** The gaps between features should be as small as possible. In the context of contours, gaps correspond to the length of the curve joining the two features.
- **Law of Continuity:** The completion of a curve between two features should be as smooth as possible. The smoother the curve, the more visually salient it is.

The law of closure and its importance in human perception has been confirmed experimentally in a psychological study by Kovacs and Julesz [30]. In this study, human subjects were found to perceive closed curves with relatively greater gaps between edge fragments than for open curves.

While the above are not the only laws of Gestalt psychology that have been proposed, for instance Boring [5] lists 14 laws, the school of thought of Gestalt psychology has faded over the years. However, the simple insights into perceptual organization listed above have not.

3.4 Energy definition

While the minima rule and the Gestalt laws give insights into what curves humans perceive to be perceptually salient, the main obstacle in implementing these theories in practice

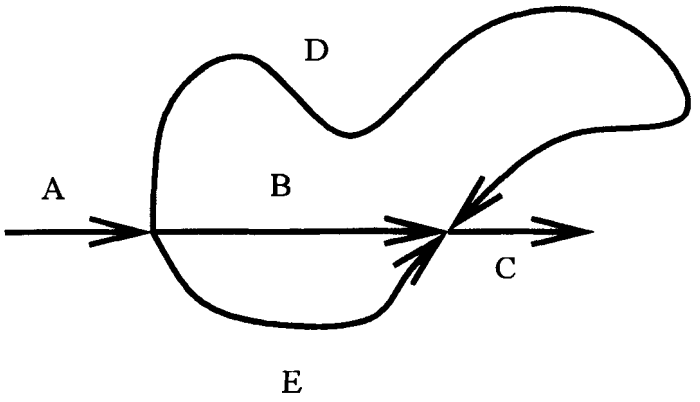


Figure 3.3: The effect of proximity on ratio energy. Two edge fragments, A and C, with uniform negative energy, are denoted in black. Three possible curve completions, B, D, and E, between the two edge fragments are shown in blue. The curves joining the two edge fragments have uniform positive energy. It is easily calculated that the curve with the least ratio energy passing through fragments A and C is the shortest one, ABC.

is that they are descriptive theories, and do not provide precise mathematical details for computation. This section explicitely defines the energy of contours, taking into account the principles and theories discussed in the previous section.

Consider an arc $r = (e_1, e_2)$ in either a regular or refined ambient graph G. This arc corresponds to an ordered pair of directed mesh edges in the regular case, or a pair of directed chords in the refined case. In either case, we refer to the two segments on the surface mesh as $e_1 = (u, v)$ and $e_2 = (v, w)$. The denominator weight of r models length and is simply defined as

$$g(r) = \frac{1}{2}(|e_1| + |e_2|) \tag{3.1}$$

where $|\cdot|$ is the Euclidean norm. The numerator weight should serve to attract e_1 and e_2 towards the desired features and to minimize the bending between e_1 and e_2; this is discussed below. Based on the motivation and principles discussed in the previous section, the following are incorporated into our energy definition:

- **Feature adaptation:** For segmentation using the minima rule [18], which states that segmentation boundaries should consist of surface points at *negative minima of principal curvatures*, the feature energy at a vertex is related to its minimum principal curvature κ_{min}. More specifically, the negative curvature minima are used to attract

the contour[1] and no preference is given to points with $\kappa_{min} \geq 0$.

- **Closure:** Closure is not a local property of a curve, but by design closure is enforced through the use of the minimum ratio contour algorithm developed in Chapter 4.

- **Proximity:** The length of a curve joining two feature points should be minimized. This can be directly attributed to the definition of the energy as a ratio, where feature edges have negative energy and non-feature have positive energy. Consider Figure 3.3, where two feature edges A and C are shown. Suppose that these feature edges have uniform negative energy, while the non-feature curves joining them have uniform positive energy. Then the shortest completing curve, B, yields the total curve with the lowest ratio energy.

  The above is a simplification of the continuous energy weights, but demonstrates that if all other factors are considered equal, the shortest contour between feature edges will have the lowest ratio energy.

- **Contour Steering:**

  To respect the law of continuity, we would like to maximize smoothness. This can be ensured by minimizing the bending of a contour, but we would also prefer to steer contours into a direction that is consistent for feature adaption. Therefore continuity is modeled as the deviation of a contour from some pre-described flow, and is accomplished by steering each edge of the contour, bringing it into alignment with a consistent flow direction. If the flow direction is continuous across the surface of the mesh, then smooth contours are ensured.

  In the context of mesh segmentation, we have chosen the flow direction to be the directions of maximal principal curvature. These directions are continuous everywhere on the surface of the mesh, except at *umbilical* points. Umbilical points are singularities that occur when the minimum and maximum principle curvatures at a point are equal. The principle curvature directions are smoothed to reduce the effect of noisy mesh data, and to remove spurious singularities — this is described later.

  In contrast to smoothness energies previously defined for mesh snakes [35, 22], which model second-order derivatives in the 3D ambient space, we wish to consider only

[1] Obviously, when ridge lines are to be detected, positive curvature maxima would become relevant.

bending "inside" the surface. This notion corresponds to the idea of "straightness" of Polthier and Schmies [43]. Instead of using left and right curve angles as defined in [43] to model straightness, we project e_1 and e_2 onto the tangent plane at v and measure smoothness in the tangent plane, see Figure 3.4.

Combining the above considerations, we define the numerator weight of the arc $r = (e_1, e_2)$ to be

$$f(r) = \begin{cases} \frac{1}{2}\kappa_{min}(v) \cdot \Big[|e_1| \cdot \cos\theta_1 + |e_2| \cdot \cos\theta_2\Big], \text{if } \kappa_{min}(v) < 0; \\ \frac{1}{2}\kappa_0 \cdot \Big[|e_1| \cdot (1 - \cos\theta_1) + |e_2| \cdot (1 - \cos\theta_2)\Big], \text{otherwise}, \end{cases} \tag{3.2}$$

where θ_1 (resp. θ_2) is the angle between the projection of the vector e_1 (resp. e_2) in the tangent plane and the maximum principal curvature direction $\vec{p}_{max}(v)$, as shown in Figure 3.4. Note that the tangent plane at v is spanned by $\vec{p}_{max}(v)$ and $\vec{p}_{min}(v)$. As we do not consider an elliptical region, where $\kappa_{min} > 0$, to be a feature region for segmentation, we replace κ_{min} by $\kappa_0 > 0$, which is a large constant, in that case.

As we can see, bending is automatically incorporated since minimizing $(|e_1| \cdot \cos\theta_1 + |e_2| \cdot \cos\theta_2)$ has the effect of reducing the angle between e_1 and e_2, in the tangent plane. Also, it is possible to introduce free parameters as exponents in (3.2) for a trade-off between the two factors.

Finally, for a contour or path π, formed by a progression of arcs, its energy ratio is $\tau(\pi) = \frac{\chi(\pi)}{l(\pi)}$, where

$$l(\pi) = \sum_{r \in \pi} g(r) \quad \text{and} \quad \chi(\pi) = \sum_{r \in \pi} f(r). \tag{3.3}$$

We note in passing that this energy of a contour is invariant to a cyclic permutation of its edges, as it should be.

For determination of the energy value for an arc, robust measures of the principle curvatures and directions are needed at points on the surface of the mesh. This is due to the nature of triangle meshes: they are often noisy or contain small bumps in the geometry. The method of Cohen-Steiner and Morvan [11] is used to compute discrete approximations of the principle curvatures and directions at vertices on the mesh. To estimate the curvature tensor at a vertex v, the curvature tensors of edges lying within a geodesic disk B (centered at v) are averaged according to the following formula

$$\mathscr{T}(v) = \frac{1}{|B|} \sum_{e \in E} \beta(e) \; |e \cap B| \; \bar{e}\,\bar{e}^t. \tag{3.4}$$

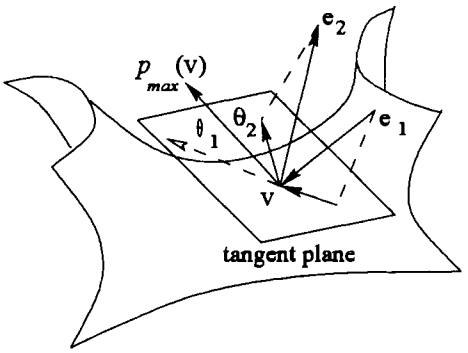


Figure 3.4: Numerator weight is computed by projecting edges onto the tangent plane.

Here, $|B|$ is the surface area of the geodesic disk, $\beta(e)$ is the signed angle between the normals of the two faces incident on e (positive if convex, negative if concave), $|e \cap B|$ is the length of the edge e that lies within B, and $\bar{e}$ is a unit vector whose direction is aligned with either orientation of the mesh edge e.

To improve robustness, the principle curvature tensors are smoothed term by term using a Gaussian filter, as described by Alliez et al. [1]. This provides more reliable estimates in the presence of noise, and helps remove spurious singularities present in the principle curvature field.

In the refined case, it is necessary to compute the principal curvatures and directions at Steiner points along mesh edges. This is accomplished by linearly interpolating the values at adjacent mesh vertices. For example, to obtain the value of the maximal curvature at a given Steiner point p_s, let the endpoints of the edge on which the Steiner point lies be p_1 and p_2, and let their corresponding maximal curvatures be w_1 and w_2. Let

$$t = \frac{|p_s - p_1|}{|p_1 - p_2|}, \tag{3.5}$$

where $|\cdot|$ is the Euclidean norm. Then the curvature value at the Steiner point p_s is given by $w_s = (1-t) \cdot w_1 + t \cdot w_2$. The same procedure can be used to obtain the principle curvature directions and minimum curvature value at p_s.

Chapter 4

Minimum Ratio Contour On Meshes

One of main the challenges in applying the minimum ratio idea for contour optimization on an irregular-grid surface mesh is to come up with an appropriate search space. Important criteria are that the optimal contour should be simple (no self-intersections), and also be efficient to compute. These properties are ensured with the construction of an acyclic graph. Contours in the search space are represented by path's in the acyclic graph, and the acyclicity of the graph also guarantees that the contours are simple. An efficient algorithm operating on the acyclic graph is developed which yields the minimum ratio contour within the search space.

Previously, Sections 3.1 and 3.2 have described the construction of regular and refined ambient graphs G. In this chapter, a local search region of the mesh is selected, and within this region an acyclic graph is constructed which only contains a subset of the nodes and arcs of G. Once constructed, a number of *minimum ratio path* (MRP) problems (Section 4.2.5) in the acyclic graph are solved to determine the minimum ratio contour within the restricted search space.

Section 4.1 gives an overview of our MRC algorithm. Further details of the algorithm are presented in Section 4.2, and Section 4.2.5 presents a simple and efficient linear time procedure (solving a minimum ratio path problem in an acyclic graph), for determining the minimum ratio contour. Modifications necessary for contour optimization in the refined case are discussed in Section 4.3. If desired, hard and soft constraints may be enforced within our

optimization framework, which are described in Sections 4.4 and 4.5 respectively. Finally, minor modifications to the algorithm to handle open curves are given in Section 4.6.

4.1 Overview of algorithm

In Figure 4.1, we illustrate the major steps of our algorithm for optimizing a contour that only passes through mesh edges. These steps are elaborated below.

1. **Search space — Figure 4.1(a) and 4.1(b):** Given an initial contour on a mesh, a search space consisting of a band, with a user-specified width, surrounding the initial contour is first constructed, as shown in Figure 4.1(a). The search band is computed via face dilation, starting from the initial contour, as shown in Figure 4.1(b). Also obtained are the "strip boundaries", shown in light gray, which are separated by single strips of triangles and used to refine the search space when constructing the search graph.

2. **Edge cut — Figure 4.1(c):** Next, a minimal edge cut of the search band is found, as shown in Figure 4.1(c). Each mesh edge along the cut is duplicated, resulting in pairs of corresponding source and destination edges for the subsequent search. This is to ensure that the resulting contour wraps around the band as the initial contour does; in general, a minimum ratio contour over the search space may not possess that property.

3. **Acyclic search graph — Figure 4.1(d) and 4.1(e):** Within the search space, we build a weighted acyclic graph whose nodes are directed mesh edges, as shown in Figure 4.1(e). The nodes and arcs of the acyclic graph are taken as a subset of the nodes and arcs of the regular ambient graph constructed in Section 3.1. The source and destination nodes in the acyclic graph correspond to the duplicated edges in the edge cut.

   An intermediate step in the construction of the acyclic graph is the computation of *gate* segments which connect adjacent strip boundaries, as shown in 4.1(d). Intuitively, the gates are located at constrictions between adjacent strip boundaries and they help define a linear ordering of mesh edges along the strip of triangles sandwiched between adjacent strip boundaries.

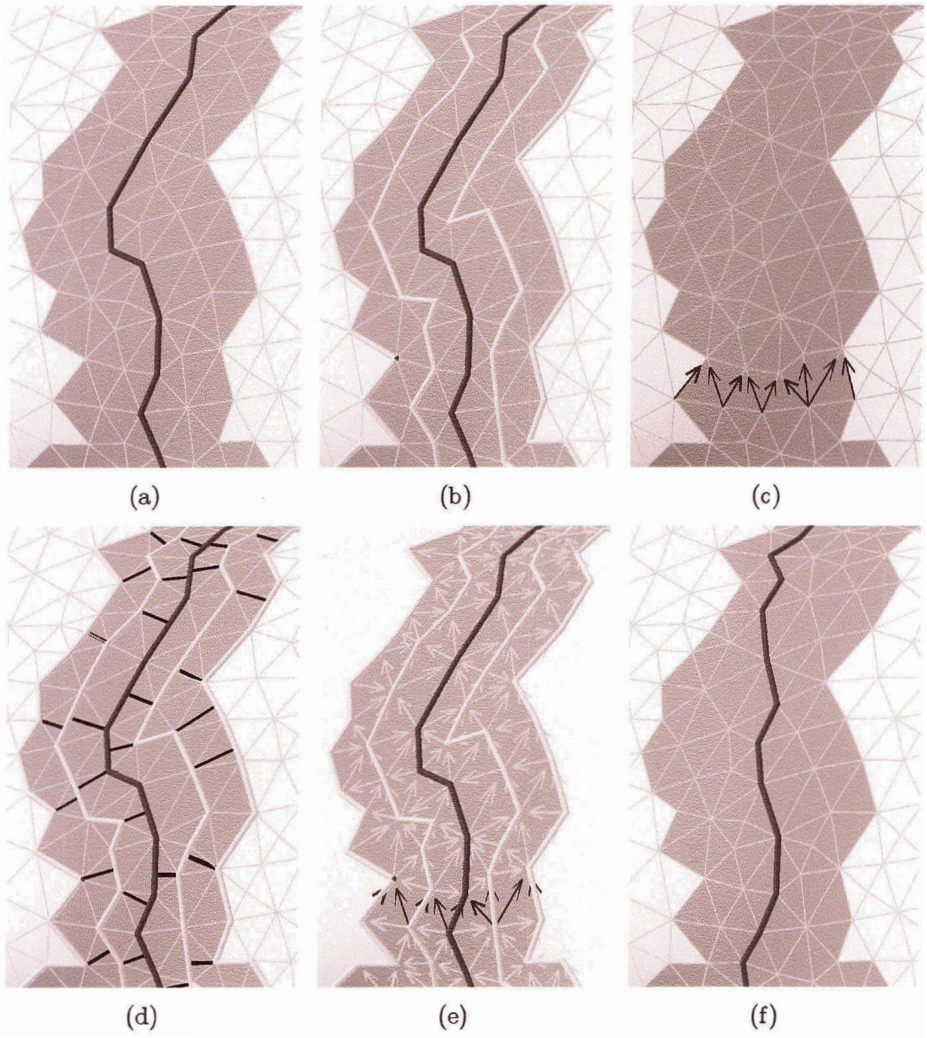


Figure 4.1: Major steps of our algorithm. (a) Given an initial contour (dark gray), a search band around it with a user-defined width d (here $d = 2$) is found. The search space is shown in medium gray throughout figures (a)-(f). (b) The 2-ring search region is found by dilation from the initial contour, resulting in a set of strip boundaries, drawn in light gray. (c) An edge cut through the search space is obtained. (d) Gate segments are inserted between adjacent strip boundaries; they aid in the determination of which directed mesh edges to include in the acyclic graph. (e) The acyclic edge graph is constructed. Each edge in the edge cut is duplicated and serves as source and target nodes in the acyclic graph. (f) Optimal ratio contour found by running the minimum ratio path algorithm once for each directed edge in the edge cut.

4. **Optimization — Figure 4.1(f):** A *minimum ratio path* (MRP) problem is solved for each pair of corresponding source and destination nodes s and t in the acyclic graph. The path with the minimum ratio corresponds to the contour in the search graph that has the minimum ratio energy.

Restricting a contour to be along mesh edges has its advantages and disadvantages. On one hand, the smoothness of the contour is limited by the mesh resolution and may be compromised by possible connectivity artifacts. On the other hand, cutting through mesh faces may add sliver triangles to the segmented mesh and also increase computation time. If desired, the mesh can be refined, as explained in Section 3.2, to produce an optimal contour with increased resolution and smoothness.

Additionally, the optimization approach allows

- **Incorporation of hard constraints:** It is easy to force an optimal solution to pass through a given constraint edge, such as a salient feature edge of the mesh. To enforce such a constraint, we first build an edge cut which contains the given constraint edge. Then instead of solving a MRP problem for each edge in the cut, only one MRP problem needs to be solved — corresponding to the constraint edge in the edge cut.
- **Incorporation of soft constraints:** Proximity to soft constraint edges can be included in the definition of numerator energy of arcs in the ambient graphs.
- **Open curves:** As the algorithm essentially reduces a minimum ratio contour problem to a number of MRP problems, open curves are straightforward to handle.

4.2 Minimum ratio contour algorithm

In this section, further details of each stage of the minimum ratio contour algorithm are given. Refer to Figure 4.1 for an illustration of the major steps of the algorithm.

4.2.1 Finding a search space

Given an initial contour C_0, we construct a search space, which is a band to *approximate* a d-ring neighborhood about C_0. This is accomplished by using the concept of *dilation*, which is well known in the context of mathematical morphology [44]. Specifically, we successively perform a form of face dilation, starting from C_0, to obtain a series of contours $C_1, \ldots, C_d$.

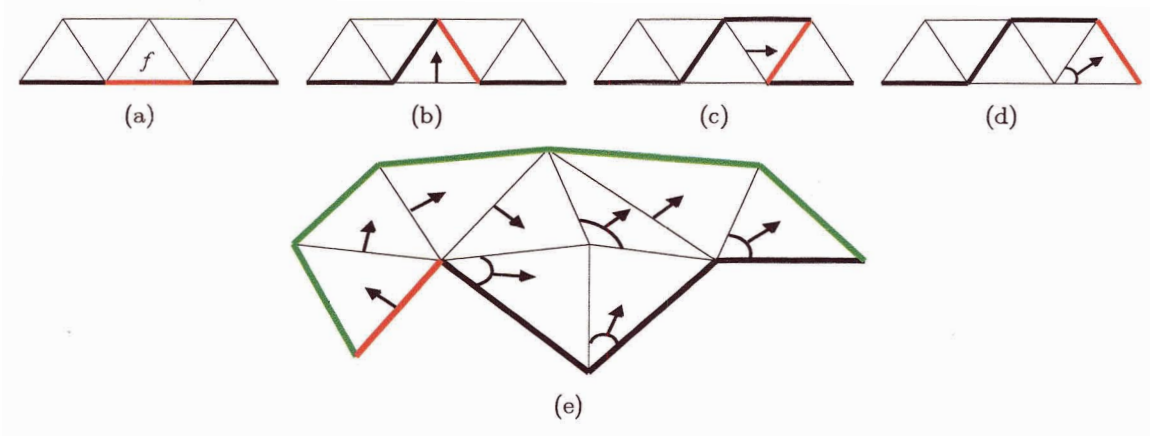


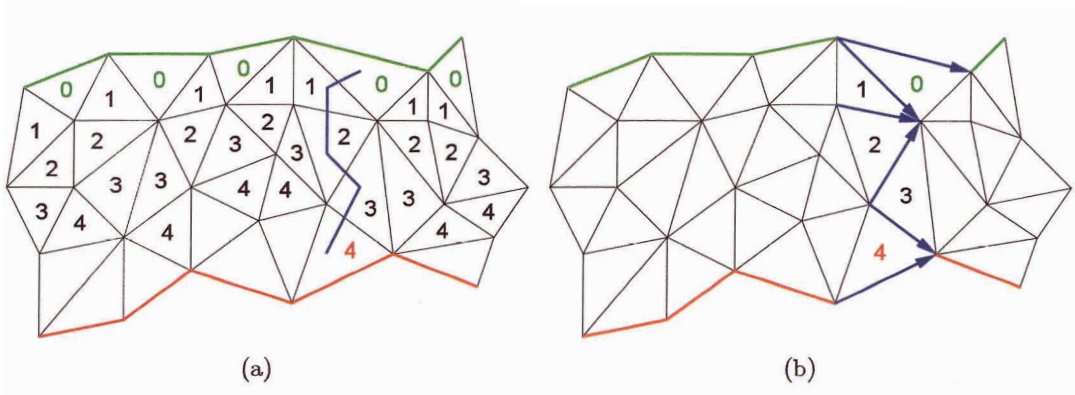
Figure 4.2: Face dilation. (a) An initial contour (thick edges). The active edge is coloured red. (b) and (c) Spike operations. (d) A merge operation. (e) A non-trivial sequence of spike and merge operations. The initial contour consists of the thick black edges. The resulting dilated contour is displayed in green. The initial active edge is displayed in red. Spike operations are indicated with an arrow, and merge operations are indicated with an arc at the base of an arrow.

Note that for each C_i there are two contours to either side of C_0. Each round of face dilation transforms the current contour C_i into contour C_{i+1}. After each round, the vertices on the newly transformed boundary C_{i+1} will lie on the set of 1-ring vertex neighbours of C_i.

To carry out a single round of face dilation, a sequence of *spike* and *merge* operations are performed on the current contour C_i to transform it into the dilated contour C_{i+1}. An *active edge* is defined as an edge on which the current spike and merge operations will be performed. An arbitrary edge on the current contour is chosen as the initial active edge. Let f be the face on the outside of the initial contour C_i that is incident on the active edge. The case where f is on the inside of the initial contour C_i is treated in a similar fashion. Then two types of operations can be executed on the active edge. They are

1. **A spike operation** — This step is executed if the active edge is the only edge of the contour incident on f. The active edge is removed from the current boundary, and the other two edges of f are spliced into the current boundary, as shown in Figure 4.2(b). The active edge is then updated to be the next counter-clockwise edge in f from the current active edge.

2. **A merge operation** — This step is executed if there are two edges of the current contour incident on f. Both edges incident on f are removed from the current



(a) (b)

Figure 4.3: Finding an edge cut via BFS. (a) A minimal sequence of faces is found. (b) Corresponding edge cut.

boundary, and the third edge of f is inserted into the current boundary, as shown in Figure 4.2(d). The active edge is then set to the newly inserted edge.

Care must be taken to ensure that the face dilation terminates at the proper step. Also, if the active boundary of the dilation would intersect itself, we disallow addition of that face, in order to ensure that the search space is homotopic to a solid torus.

4.2.2 Finding an edge cut

A contour which traverses around the search band must pass through an edge in an edge cut of the band. Instead of implementing a full-fledged graph min-cut algorithm, we perform a breadth-first-search (BFS) on mesh faces within the search space due to the special banded shape of the search space. Specifically, we label each face having an edge on the inner boundary of the search band 0, and then use BFS with face adjacency to label the remaining faces in the search band, stopping when a face on the outer boundary is encountered. Now a minimal sequence of connected faces from the inner to the outer boundary is obtained by backtracking. The corresponding edge cut consists of edges which are adjacent to two faces in the minimal face sequence, plus one boundary edge, as shown in Figure 4.3.

4.2.3 Constructing the acyclic graph

To construct an acyclic search graph A in the search region, we wish to establish a *flow direction*, which mimics the general direction of the initial contour. We then orient each undirected mesh edge within the search space; the orientation chosen is the one which is

most aligned with the flow direction. Care must be taken to ensure that cycles (circular flow patterns) do not occur within the search space.

Strip boundaries: To facilitate construction of the vortex-free flow, we refine the search space into a series of *strip boundaries*. Each strip boundary is a progression of mesh edges, starting at an edge in the edge cut and ending at an edge in the edge cut. Specifically, the initial contour C_0, broken up at the edge cut, induces one of the strip boundaries. The set of all strip boundaries provide a complete cover of the mesh vertices within the search space. No two strip boundaries cross each other although they may overlap. The exact construction of the strip boundaries is described later.

Distance function along strip boundaries: We define a *monotonic distance function* $\mathcal{F}$, in the range [0,1], along each strip boundary. It measures some form of a distance between a given point along a strip boundary to the starting point of the strip boundary. The flow direction is dictated by the distance functions.

Edge orientation and graph construction: By our definition of strip boundaries, every mesh edge within the search space has its two vertices either on the same strip boundary or on two adjacent strip boundaries. To orient each undirected mesh edge (a, b), we consider the distance or $\mathcal{F}$ values at its endpoints. If $\mathcal{F}(a) < \mathcal{F}(b)$, the edge is oriented from a to b and added to the graph A as a *node*. If $\mathcal{F}(b) < \mathcal{F}(a)$, the edge is oriented from b to a and added to A. Otherwise, the edge lies along an "iso-level", and it is not added to A. To reduce the number of arcs in the acyclic graph, pruning is performed to remove undesirable arcs (also discussed in Section 3.2). An arc, connecting two adjacent nodes (a, b) and (b, c) in A, is added to A if and only if the angle between the directed mesh edges (a, b) and (b, c) does not exceed a threshold of $\pi/2$. By construction, the resulting directed graph A must be acyclic, and each strip boundary induces a directed path in the search graph A. Note that all nodes and arcs of A are a subset of the nodes and arcs of either the regular or refined ambient graph.

Construction of strip boundaries: We construct strip boundaries by levels. At level 0, we have the strip boundary corresponding to the initial contour. The i-th level strip boundaries (there are two of them, one to each side of the initial contour) are taken to be the boundaries C_i found when performing face dilation in Section 4.2.1. Note that not all vertices in the search space may lie on a strip boundary and it may be necessary to add additional, refined strip boundaries. This is achieved by successively inserting intermediate

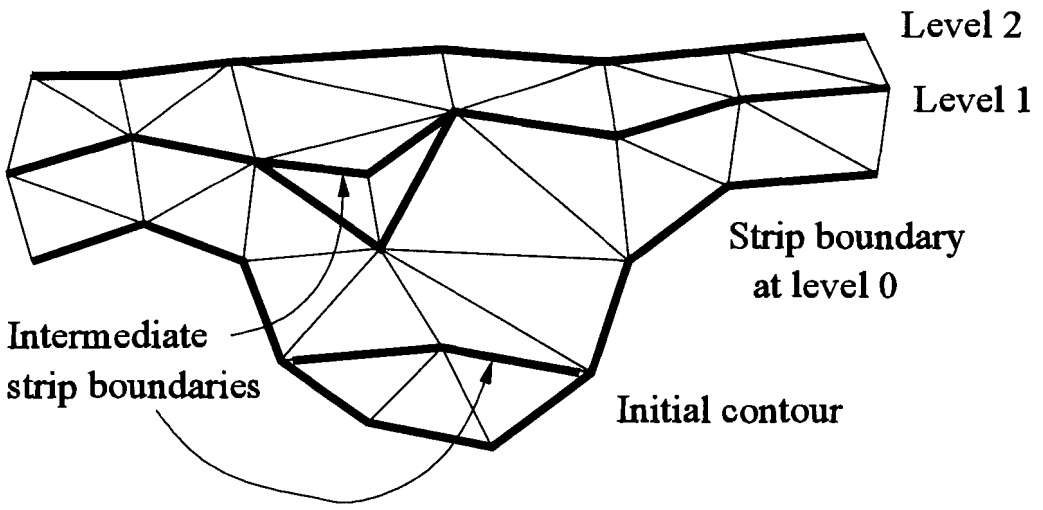


Figure 4.4: An initial contour and the strip boundaries, on one side, obtained.

strip boundaries until all vertices are covered by the strip boundaries. In Figure 4.4, we show the set of strip boundaries obtained in an artificial yet non-trivial configuration.

Distance function based on arc lengths: The distance function used could be based on arc lengths, where the $\mathcal{F}$ value at a point p along a strip boundary C would be the distance from the starting point of C to p, divided by the total length of C. This simple distance function may encounter problems if the length of adjacent strip boundaries differ greatly, which can result in flow directions which do not mimic the general flow of the initial contour.

Distance function based on gates: A better approach is to enforce a set of constraints between the distance functions of adjacent strip boundaries. A *gate* is a line segment which connects two adjacent strip boundaries. The distance function at the endpoints of a gate segment (p, q), where p is on strip boundary C_1 and q is on strip boundary C_2, are required to be equal; that is $\mathcal{F}_{C_1}(p) = \mathcal{F}_{C_2}(q)$. As a gate between adjacent strip boundaries C_1 and C_2, we have chosen pairs of points (p, q), $p \in C_1$ and $q \in C_2$, such that the closest point on C_2 to p is q, and the closest point on C_1 to q is p; see Figure 4.1(d). The motivation for choosing such gate segments is that such pairs of points (p, q) correspond to narrows between the strip boundaries. The narrows are natural places to insert gates. In practice it has been observed that the number and locations of the gates are sufficient in establishing adequate flow directions.

4.2.4 Optimization

Finally, nodes corresponding to edges in the edge cut are duplicated. Refer to one set of duplicated nodes as *source*, and the other as *dest*, for destination nodes. We only allow outgoing arcs from nodes in *source* and incoming arcs to nodes in *dest*. Now a progression of nodes in A starting at a node in *source* and ending at its duplicate node in *dest* corresponds to a contour (or cycle) on the mesh.

After finishing the construction of the acyclic graph, we run the MRP algorithm, as described in Section 4.2.5, once for each pair of duplicated nodes, one from *source* and the other from *dest*. We take the minimum ratio contour as the one obtained from the minimum of the set of MRP's, with the source and destination nodes merged.

4.2.5 Solving the minimum ratio path problem

This section first describes how to find a minimum ratio cycle in a general graph. Then modifications are presented for finding a more efficient, linear time algorithm for obtaining a minimum ratio path in an acyclic graph.

Given a directed search graph G by means of regular construction (Section 3.1) or refined construction (Section 3.2), associate two functions f, g to the arcs of G. In our optimization approach, $f : E \to \mathbb{R}$, $g : E \to \mathbb{R}^+$, and these functions are the numerator energy and length functions described in Section 3.4.

Consider a sequence π of consecutive arcs in G (which corresponds to a sequence of directed mesh edges in M), which may or may not form a cycle, and define the *numerator weight* of π to be

$$\chi(\pi) = \sum_{e \in C} f(e) \tag{4.1}$$

and the *denominator weight* to be

$$l(\pi) = \sum_{e \in C} g(e). \tag{4.2}$$

Let $\mathscr{C}$ be the set of all directed cycles in G. The *minimum ratio cycle* problem is one that seeks a cycle $\pi^* \in \mathscr{C}$ which minimizes the ratio $\tau(\pi) = \chi(\pi)/l(\pi)$. That is,

$$\pi^* = \text{argmin}_{\pi \in \mathscr{C}} \tau(\pi) = \text{argmin}_{\pi \in \mathscr{C}} \frac{\chi(\pi)}{l(\pi)}. \tag{4.3}$$

The related *minimum mean cycle* problem [26] replaces the denominator $l(\pi)$ by the arc count $\#(\pi)$. Both problems can also be defined for the set of paths from a source vertex s

to a destination vertex t; we refer to these as the minimum ratio path (MRP) and minimum mean path (MMC), respectively.

We first consider the problem of finding a minimum ratio cycle in G. The approach given below was first described by Lawler [34], and later generalized by Meggido [40]. Observe that for any graph G, the ratio of any cycle is reduced by $\lambda \in \mathbb{R}$ when a new set of numerator weights $f'(e)$, defined by

$$f'(e) = f(e) - \lambda \cdot g(e), \quad \forall e \in E(G)$$

are used to replace f. This is straightforward, as

$$\frac{\sum_{e\in\pi} f'(e)}{\sum_{e\in\pi} g(e)} = \frac{\sum_{e\in\pi} \left[f(e) - \lambda \cdot g(e)\right]}{\sum_{e\in\pi} g(e)}$$

$$= \frac{\sum_{e\in\pi} f(e)}{\sum_{e\in\pi} g(e)} - \lambda.$$

Since the ratio of each cycle in G is lowered by λ, the minimum ratio cycle using weights f' remains unchanged from the minimum ratio cycle in G using the original weights f.

To solve the minimum ratio cycle problem in G, we can then search for the smallest λ^* such that the minimum ratio cycle in G, weighted by $f^* = f - \lambda^* \cdot g$, has a ratio value of zero. This is in turn equivalent to determining whether G weighted by f^* has a zero-weight cycle and no negative-weight cycles. Thus, the problem of finding a minimum ratio cycle reduces to finding a λ such that there exists no negative cycles in G, and that there also exists a zero weight cycle in G.

Determining whether a negative weight cycle in G exists can be accomplished by running the Bellman-Ford algorithm on G [13]. If it exists, a negative weight or zero weight cycle in G can also be extracted using the Bellman-Ford algorithm. Thus to find λ^*, either a binary or linear search over λ can be performed. If the denominator and numerator weights are integral, the time complexity of binary search is logarithmic [32], which is lower than that of linear search (a pseudo-polynomial time bound can be obtained for linear search [21]). In practice linear search is found to be more efficient [21, 50]. As confirmed by both our experiments and related work on image contour extraction [21, 50], the number of linear search steps is typically a small constant, no more than six in all of our experiments (Chapter 5 provides statistics).

The Bellman-Ford algorithm is needed above to test whether G has a negative weight cycle for a given λ. In general, this would take $O(|V| \cdot |E|)$ time [13]. This leads overall to

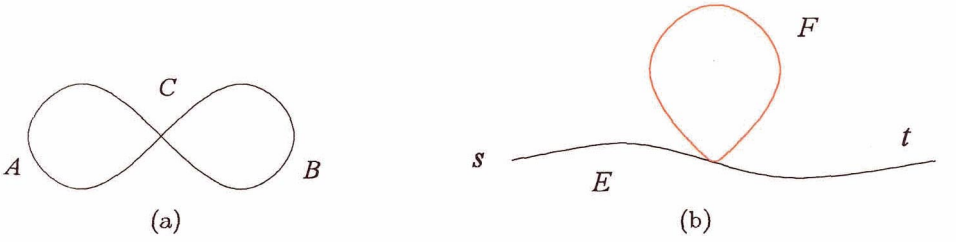


Figure 4.5: (a) The cycle C is self-intersecting. Unless the ratio of the left cycle A equals that of the right cycle B, then one of A or B will have a lower ratio than C. (b) The path E is a non-intersecting from s to t. If the cycle F has a lower ratio than E, then an intersecting path with a lower ratio than E can be formed by appending the cycle F to E.

at least an $O(|V| \cdot |E|)$ algorithm for determining the minimum ratio cycle in G, since the Bellman-Ford algorithm must be run for each λ query.

Now consider the problem of finding a MRP between two nodes s and t in G. The above analysis for finding a minimum ratio cycle in G is still applicable for finding a MRP in G, with one caveat: the MRP in G may self-intersect, whereas this is not possible with a minimum ratio cycle. To illustrate this, consider Figure 4.5. In Figure 4.5(a) an intersecting cycle C is shown, which consists of two sub-cycles A and B. It is not possible that the cycle C is the minimizer, since one of the sub-cycles (A or B) must have a lower ratio than the entire cycle, unless all sub-cycles have the same ratio. If the cycle with the minimum ratio is not unique, then the problem is degenerate, and all minimizers can be returned. Figure 4.5(b) shows a non-intersecting MRP, labelled E, from s to t. If the cycle F has a lower ratio than E, then an intersecting path with a lower ratio than E can be formed by appending the cycle F to E.

This undesirable behaviour can be removed by restricting the graph G to be acyclic, which trivially ensures that any MRP is not self-intersecting. Using an acyclic graph has another advantage: negative cycle detection can be achieved in $O(|V| + |E|)$ time. This follows, since an acyclic graph induces a topological ordering of its nodes, which in turn enables the Bellman-Ford algorithm (for finding shortest paths) to run in linear time (see [13] for details).

In related work, Wang et al. [52] consider the undirected version of the minimum ratio cycle problem and solve it by first transforming G into an augmented graph G' and then reducing the problem of finding a negative cycle in G' to that of finding a minimum weight perfect matching (MWPM) in G'. We have not considered this approach since in general

the cost of obtaining a MWPM is $O(|V|(|E| + |V| \log |V|))$ [12], which is much higher than the linear-time algorithm we have been able to obtain for our specialized graph model.

4.3 Optimization of refined contours

The optimization procedure for the refined case follows the same general structure as for the regular case, with only slight modifications. After identifying the search band, we consider the refined chord graph after pruning (described in Section 3.2), to find a refined edge cut.

Let γ be a path, in the original mesh graph, originating at a vertex situated on the inner boundary, and terminating at a vertex on the outer boundary of the search band. We take as an edge cut in the refined chord graph all chords which originate on one side of γ and terminate at some point, a mesh vertex or a Steiner point, along γ. In practice, we take γ to be a shortest (in graph distance) path from the inner to outer boundary of the search band. Note that although the number of edges in the resulting cut may not be minimal, it is nevertheless a close approximation.

The construction of the acyclic graph in the refined case proceeds in the same manner as in the regular case, except that directed chords in the search space need to be oriented instead of mesh edges. To find the principal curvatures and directions, as well as distance values for Steiner points, we linearly interpolate along a mesh edge those values that have already been estimated at the mesh vertices. This process has previously been described in Section 3.4.

4.4 Hard constraints

At times, it may be desirable to constrain the optimization, such that the optimal contour found must pass through a given vertex or edge of the initial contour. We refer to such constraints as hard constraints. We first consider the case where the optimal contour is restricted to pass through an edge of the initial contour; such an edge is referred to as a *constraint* edge. The natural method is to require that the constraint edge must be a member of the edge cut of the search space (see Section 4.2.2). If this is the case, then only one MRP problem needs to be solved, instead of solving a MRP problem for every edge in the cut.

To find such a constrained edge cut, the algorithm for finding an edge cut is modified as

follows. The initial contour divides the search space into (approximately) two halves, and a sub-edge cut is found for each half of the search space. To find a sub-cut, a breadth-first-search (BFS) is performed on mesh faces. We label the face adjacent to the constrained edge 0, and then use a BFS with face adjacency to label the remaining faces in the search band, stopping when a face on the outer boundary is encountered. Now a minimal sequence of connected faces from the initial contour to the outer boundary is obtained by backtracking. This process is repeated for the other half of the search space, to obtain a second sub-cut. The two sub-cuts are then combined to provide a complete edge cut across the whole search space, one member of which is the constrained edge of the initial contour.

After obtaining the constrained edge cut, the acyclic graph construction proceeds as in Section 4.2.3. To find the optimal contour which passes through the given constraint edge, instead of solving a MRP problem for every edge in the cut, only one MRP problem is solved. This corresponds to the path (contour) which starts and terminates at the contrained edge.

If we wish to enforce a vertex constraint, where the optimal contour must pass through a given vertex of the mesh, slight modifications of the above algorithm are required. First, the edge of the initial contour which terminates at the given vertex is located. This edge is treated as a constraint edge, and a constrained edge cut is found as described above. To find the optimal contour, the MRP problem is solved once for every edge in the cut which terminates at the constraint vertex. The lowest ratio path is selected as the the optimal contour which passes through the given constraint vertex.

If more than one hard constraint is required, the above procedure is repeated to generate multiple cuts throughout the search region, such that there is one edge cut for every constraint. Instead of solving MRP path problems between adjacent edge cuts, a single edge cut is selected, and edges in the remaining edge cuts which are not allowed are removed from the acyclic graph. This ensures that a path in the acyclic graph departing from a constraint edge in the selected cut satisfies all the hard constraints.

4.5 Soft constraints

Unlike hard constraints, soft constraints in general do not require the optimal contour to pass directly through a given region of the mesh, but would effectively attract the optimal contour to such a given region. In order to implement soft constraints, the external energies of arcs in the ambient graph are lowered in a given region of the mesh (the internal bending

energy is left unchanged). This reduces the ratios of contours passing through the given region, and increases the chances that the optimal contour will pass through the given region of the mesh. Soft constraints are implemented with the following steps:

1. A vertex v on the mesh is selected.
2. A node set N_r consisting of nodes from an ambient graph G is found. A node is inserted into N_r if its corresponding edges (or in the refined case, directed chords) of the mesh are within a given radius r of v.
3. The weights of arcs between adjacent nodes in N_r are lowered.

The new weights depend on the distance of the edges to the initial vertex v, and are controlled with a Gaussian weighting. The vertex v, the radius r, and the Gaussian parameters must be chosen by the user.

The following limitations of soft constraints should be noted. First, if the region in which the soft constraint is added is not within the search space during optimization, the soft constraint will have no effect on the outcome of the optimization. Second, if the soft constraint does not attract the optimal contour into the given region, the optimal contour may appear anywhere else in the search space.

4.6 Open curves

Open curves are implemented by holding the beginning and ending edges of the curve fixed. The optimal path within a search space with a given width is then found. Notice that when dilating to find the strip boundaries, the strip boundaries wrap around the initial curve, as shown in Figure 4.6(a). An edge cut is needed at each end of the curve to determine where the strip boundaries start and end. Figure 4.6(a) shows part of an the initial curve and the strip boundaries after dilation. Since the strip boundaries wrap around at the ends of the open curve, the strip boundaries need to be cut. To accomplish this, two edge cuts are found through the strip boundaries (one at each end of the open curve), in order to assign beginning and ending points to the strip boundaries.

In order to obtain these edge cuts, each edge cut is composed of two sub-cuts. The sub-cuts are found in a similar manner to the constrained edge cuts when optimizing in the presence of hard constraints (see Section 4.4). However, the algorithm for finding a

constrained cut cannot be used here, since the search space is homotopic to a disk instead of a torus — the initial open curve does not divide the search space into halves.

To find a sub-cut, all faces incident to one side of the initial curve are labeled as outside (-1 in Figure 4.6(b)). Next, the unlabeled face adjacent to the initial edge of the open curve is labeled 0. Then a BFS is performed, stopping when a face adjacent to the outside boundary is encountered. The sub-cut is recovered by backtracking through the search space. The other sub-cut is found by repeating the above procedure, but initializing the faces on the opposite side of the open curve. These two sub-cuts combined form an entire cut which passes through the first edge of the initial curve, shown in Figure 4.6(c).

The second edge cut through the last edge of the initial open curve is found in a similar manner, and then the search space is truncated, as shown in Figure 4.6(d). This discards extraneous faces at the ends of the search space. All strip boundaries within the search space are now open curves, and thus have proper beginning and ending points.

The construction of the acyclic graph then proceeds as in Section 4.2. After the graph is constructed, instead of running the MRP algorithm once for each edge in the cut, it is only run once, which corresponds to the MRP between the source and destination edges of the open curve.

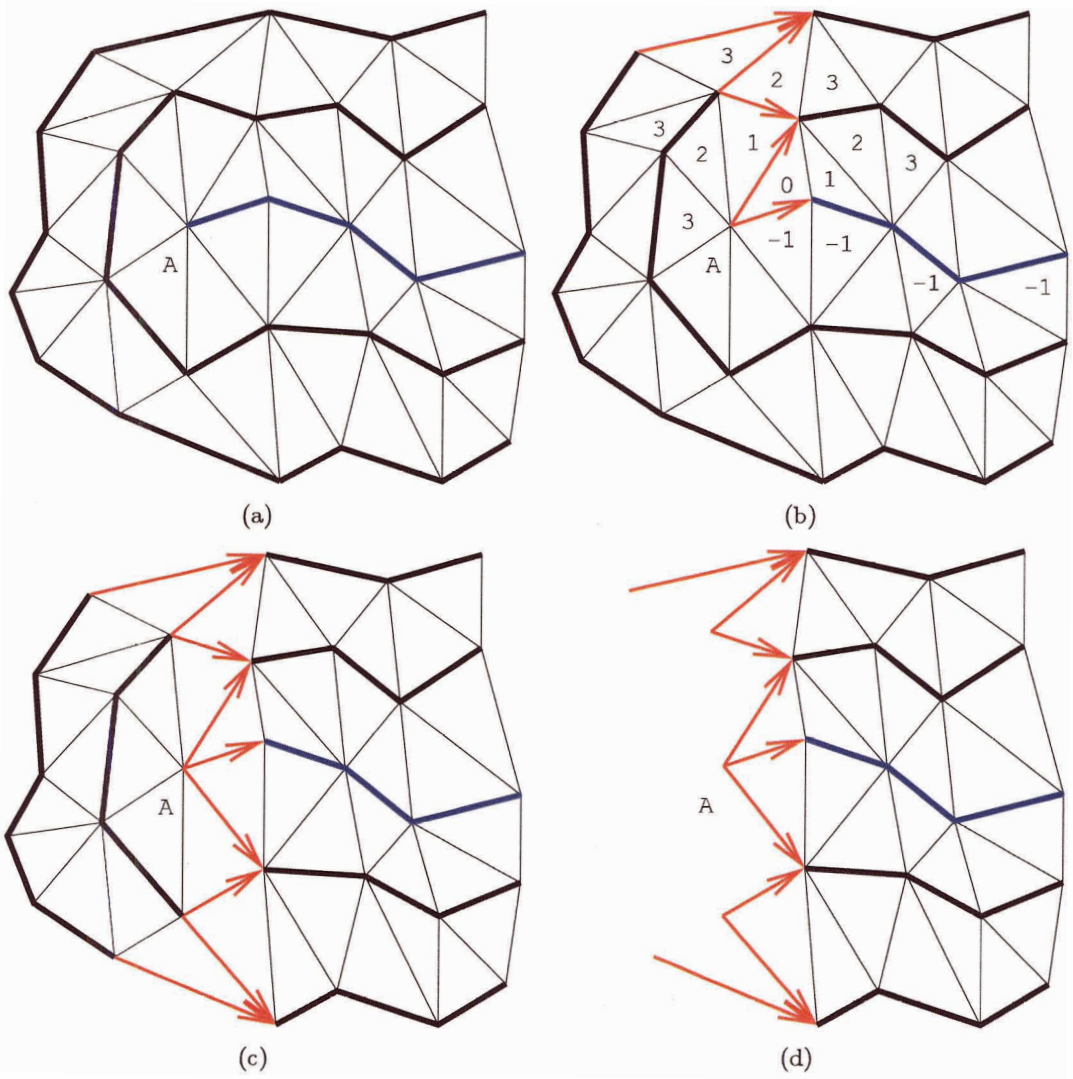


Figure 4.6: (a) An initial open curve (in blue) starting at vertex *A*. The initial strip boundaries (thick black) surrounding the open curve are also shown. (b) A sub-cut, which is an edge cut from the first edge of the open curve to the boundary, is found via BFS. Another sub-cut (not shown) is found on the opposite side of the open curve. (c) The entire edge cut found that passes through the first edge of the open curve. It is composed of the two sub-cuts passing through the first edge of the open curve. (d) The truncated search space — extraneous faces are discarded.

Chapter 5

Results

Results obtained from the minimum ratio contour algorithm are shown throughout this chapter. All experiments conducted were performed on a commodity desktop computer (described in detail in Section 5.6), and for each experiment, the user is required to select a search space width. The significance and effects of differing widths are explained in the subsequent sections. An initial contour is also required to run the algorithm. Most initial contours are acquired as output from an independent, fully automatic segmentation algorithm. These initial contours may not be high quality, and therefore the minimum ratio contour algorithm is run in an attempt to find a more perceptually salient contour. Alternatively, for some examples, the initial contours may have been created by hand by the user.

We distinguish between regular optimization, in which case the search graph is constructed directly from the original input mesh (as in Section 3.1), and the refined optimization in which case the search graph is derived from a refined mesh (as in Section 3.2). Section 5.1 compares the effect and differences between regular optimization and refined optimization. Due to the larger refined search space, refined optimization is shown to yield smoother contours.

In Section 5.2, experiments are provided which demonstrate under what conditions the minimum ratio contour algorithm can avoid local minima. Section 5.3 discusses the effects and possible benefits of iterating the minimum ratio contour algorithm. Section 5.4 shows an example where hard constraints are needed to constrain the optimal contour to pass through given vertices of the mesh, and Section 5.5 gives an example of optimizing an open curve. Section 5.6 provides and discusses statistics about the search graphs and timings for

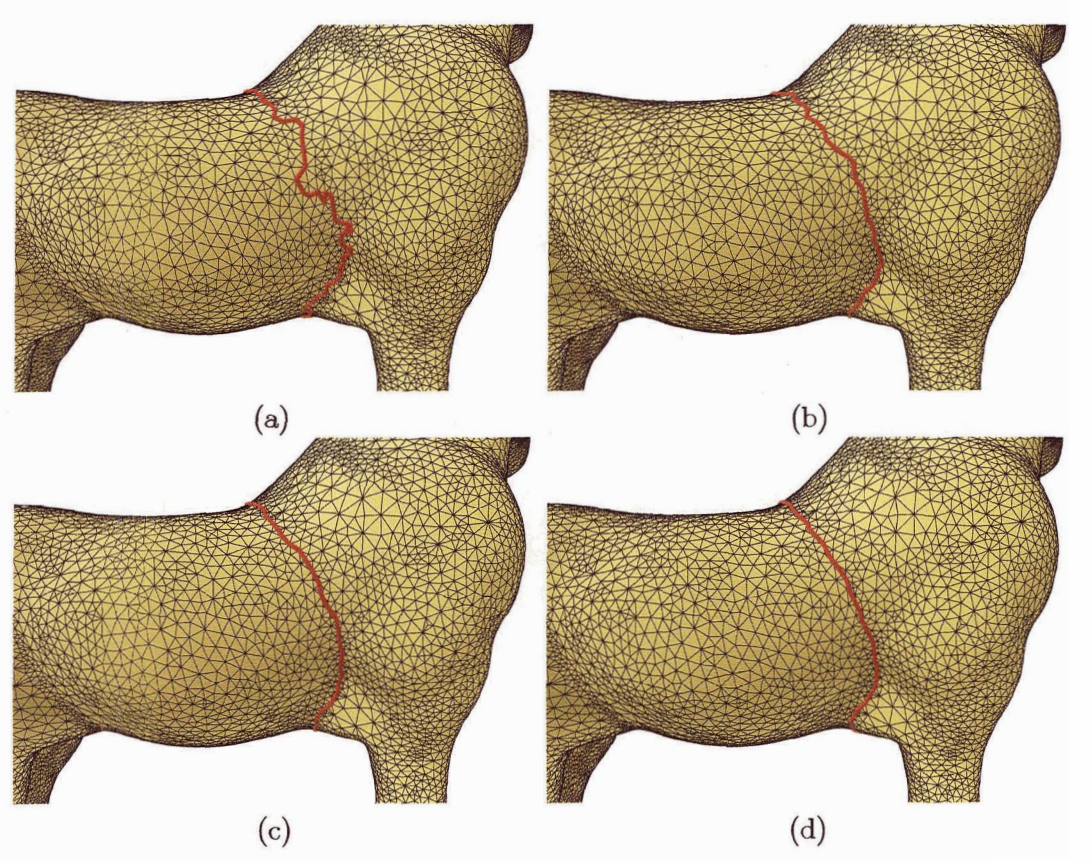
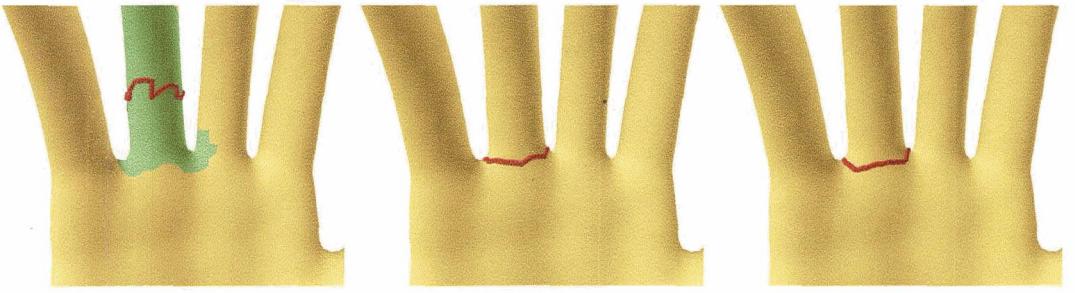


Figure 5.1: (a) An initial contour. (b) Optimal contour found with regular optimization using a search band width of 6. Due to the connectivity of the mesh, the contour is not smooth. (c) After optimization on refined graph, with $k = 1$. (d) After optimization on refined graph, with $k = 2$. Evidently, higher levels of refinement result in smoother contours. These results also demonstrate the stability of our MRC algorithm.

experiments conducted in this chapter.

5.1 Regular vs. refined optimization

When optimizing contours, it can be seen that connectivity artifacts are common in the regular, unrefined case, see Figures 5.1(b) and 5.2(b). By refining the mesh to allow contours to cut across mesh faces, the resulting contours are rendered sufficiently smooth, but at an increased computation cost. The level of refinement is controlled by the parameter k, which is described in more detail in Section 3.2. The proper choice of k to ensure smoothness depends on the resolution of the given mesh. Typically, the lower the mesh resolution, the



(a) (b) (c)

Figure 5.2: (a) Initial contour. A large search band of width 9 (shown in green) is used in order to make the contour jump to an optimal position. (b) Result after regular optimization; there is some roughness in the final contour as a connectivity artifact. (c) After refined optimization, resulting contour has increased smoothness.

larger the k value needs to be selected. Another factor to be considered is the visualization of the mesh. The size of the mesh triangles in screen space, and the distance of mesh model to the viewpoint will affect how smooth the resulting contours are perceived. All experiments were conducted with a screen resolution of 1280x1024 pixels, and sufficiently smooth contours for all mesh models were obtained with a k value that did not exceed 2.

Figure 5.1 compares the effects of refinement on the horse model by running the algorithm using the same initial contour under three different optimization conditions: regular, refined with $k = 1$, and refined with $k = 2$. Figure 5.1(a) displays the initial contour, and Figure 5.1(b) shows the result of regular optimization, where the optimal contour returned is in the desired vicinity, but lacks smoothness. The refined meshes allow for smoother final contours, which are demonstrated in Figures 5.1(c) and 5.1(d). The smoothness of the resulting contour can be controlled by choosing the refinement level, k. The higher the value of k, the more densely the mesh is refined, and the smoother the resulting contours.

Experimentally, when using the same width search space, for any given level of refinement, the algorithm converges to approximately the same final contour. The resulting contours differ in their degrees of smoothness; this is only restricted by the given search graph. This supports the claim that the minimum ratio contour algorithm is stable against the optimization conditions.

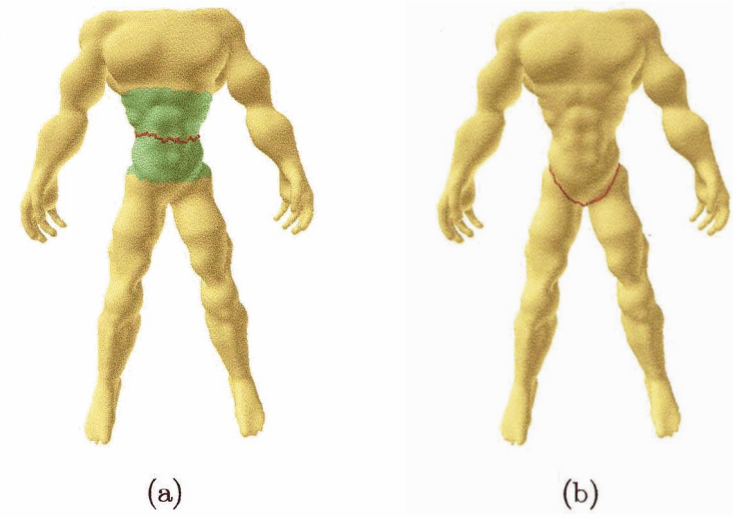
(a) (b) (c)

Figure 5.3: (a) Initial contour. Optimization with a neighbourhood size of 4 (shown in green) results in the locally optimal contour shown in (b). (b) By increasing the size of the band to 8 (shown in green), the contour jumps over the concave region present at the base of the thumb, indicated by an arrow. (c) After the contour has settled using regular optimization, refined optimization with $k = 2$ is used to produce the results in (b) and (c).

5.2 Escaping local minima

Figures 5.2, 5.3, and 5.4 give good examples to show possible differences between the minimum ratio contour algorithm and mesh snakes. Consider first the case of the ring finger, shown in Figure 5.2. Note that the initial contour is far away from the desired feature, the final contour shown in Figure 5.2(b). Geometrically, the finger is smooth with its diameter continuously narrowing down to the finger tip. Thus a mesh snake would converge to a trivial solution, unless it is specifically instructed to inflate [10]. This is not the case however with the minimum ratio contour algorithm. Since the energy is defined as a ratio, the trivial solution is not a minimizer, as described in Section 2.1. By increasing the search space neighbourhood to include a *portion* at the base of the finger, the algorithm is able to find a low ratio contour within that region. The ability of the minimum ratio contour algorithm to adapt to mesh features allows it to locate the desired feature, provided that at least part of the desired feature lies in the initial search space.

Figure 5.3 highlights the effect of the search space size on the resulting contour. With a search band of width 4, the resulting contour converges to that shown in Figure 5.3(b). This contour is the minimum ratio contour within the given search region. However, if the search band width is increased to 8, a lower ratio contour occurs in the search region, and the resulting contour jumps to the one shown in Figure 5.3(c). This is unlike a mesh snake,



(a) (b)

Figure 5.4: (a) Initial contour. Search space of width 12 is shown in green. (b) Several iterations with a large neighborhood size of width 12 allows the initial contour to realize a better position. Note that the contour has jumped across many local minima to find a better minima.

which would not be able to cross the large concave region present at the base of the thumb (indicated by an arrow in Figure 5.3(c)). The algorithm only requires that the minimum ratio contour be within the search space, and it will find it. Increasing the search space size makes it possible that another better, lower ratio contour will be found, as the above example illustrates.

Again, Figure 5.4 demonstrates how the minimum ratio contour algorithm can avoid or bypass local minimums that a mesh snake would not. The initial contour lies around the torso of the headless model, part of which lies amidst a series of grooves present in the abdomen. By using a large neighborhood width of 12, the algorithm is able to avoid the locally optimal contours passing through the grooves in the abdomen, and locate the much more significant final contour, shown in Figure 5.4(b).

5.3 Effects of iterating the MRC algorithm

It should be noted that if part of the desired contour does not lie within the search space, the resulting optimized contour will not be the desired one. In this case, iterating the minimum ratio contour algorithm will not help, unless an intermediate contour is found which has

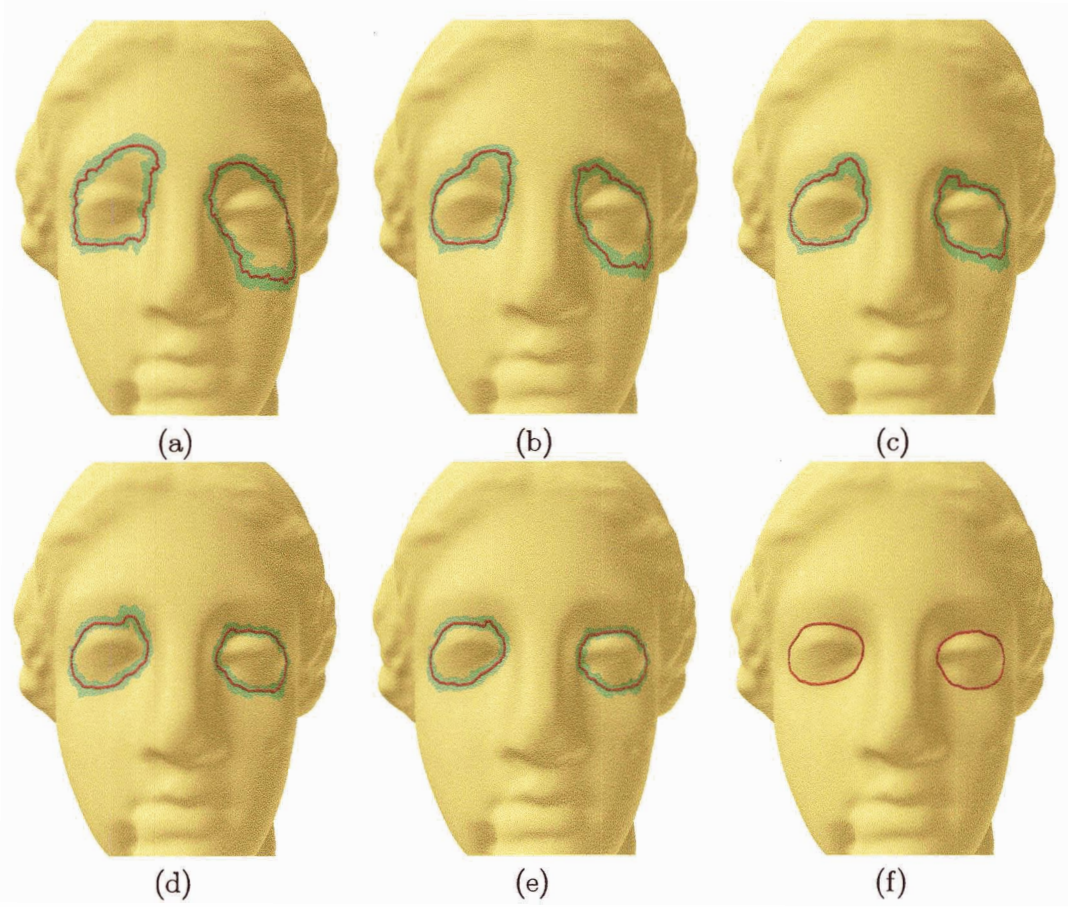


Figure 5.5: (a) Initial contours. (b)-(e) Contours after 1-4 iterations of the MRC algorithm. (c) After five iterations of the regular MRC algorithm, and one subsequent iteration using refinement with $k = 1$. Search spaces spaces of width 2 surrounding the contours are shown in green in (a)-(e).

a lower energy than the initial contour, but a greater energy than the desired contour. There is no guarantee that such a contour will always exist. If however, part of the desired final contour intersects the search space, then it is likely that such intermediate contours will exist. Iterating the minimum ratio contour algorithm will then yield a sequence of successively better contours, until the final desired contour is obtained.

To illustrate an example where repeated iterations are useful, consider the venus model in Figure 5.5(a), where there are a pair of contours whose initial positions are far from their desired final positions (segmenting the eyes). Note that parts of the initial contours lie at low energy, concave regions near the tips of the eyes. These regions are included in the

search space if using a small width of 2, and serve as anchors when iterating the minimum ratio contour algorithm. The result after one iteration is shown in Figure 5.5(b), and it takes five iterations to converge to the desired contour, shown in Figure 5.5(c).

5.4 Constraints

In certain cases, the minimum ratio contour algorithm may not return the desired result. Consider Figure 5.6(a), which displays an initial contour surrounding a face. If optimized, the undesired result in Figure 5.6(d) is returned. The problem that arises, is that the desired contour (segmenting the face), is not the lowest ratio contour in the search space. This is due to the large, and very prominent (low energy) ridge running down the neck. The optimal contour is attracted to this ridge, which lowers its ratio. To overcome such an undesired result, hard constraints can be added, as described in Section 4.4, to force the final contour to pass through some user specified vertices. Three hard constraints are added to the initial contour, and are displayed as spheres in Figure 5.6(a). After optimization with these constraints, the desired contour is obtained, which correctly segments the face.

5.5 Open Curves

An example of optimizing an open curve, as detailed in Section 4.6, is shown on the smiling face model in Figure 5.7. Here, we seek to segment the hair on the head from the face, by finding a contour that runs along the hairline. The curve delimiting the hairline does not permit a natural closure, therefore an open curve is used to locate a portion of the hairline. The initial open curve, shown in Figure 5.7(a), has its endpoints located on the hairline of the model. The search space surrounding the open curve is shown in Figure 5.7(b), and regular optimization with a search space width of 5 is used to obtain the optimal curve displayed in Figure 5.7(c). Notice that the optimal curve is quite smooth — there are no connectivity artifacts present along the hairline of the model. Note that when optimizing open curves, the endpoints of the initial contour must remain fixed.

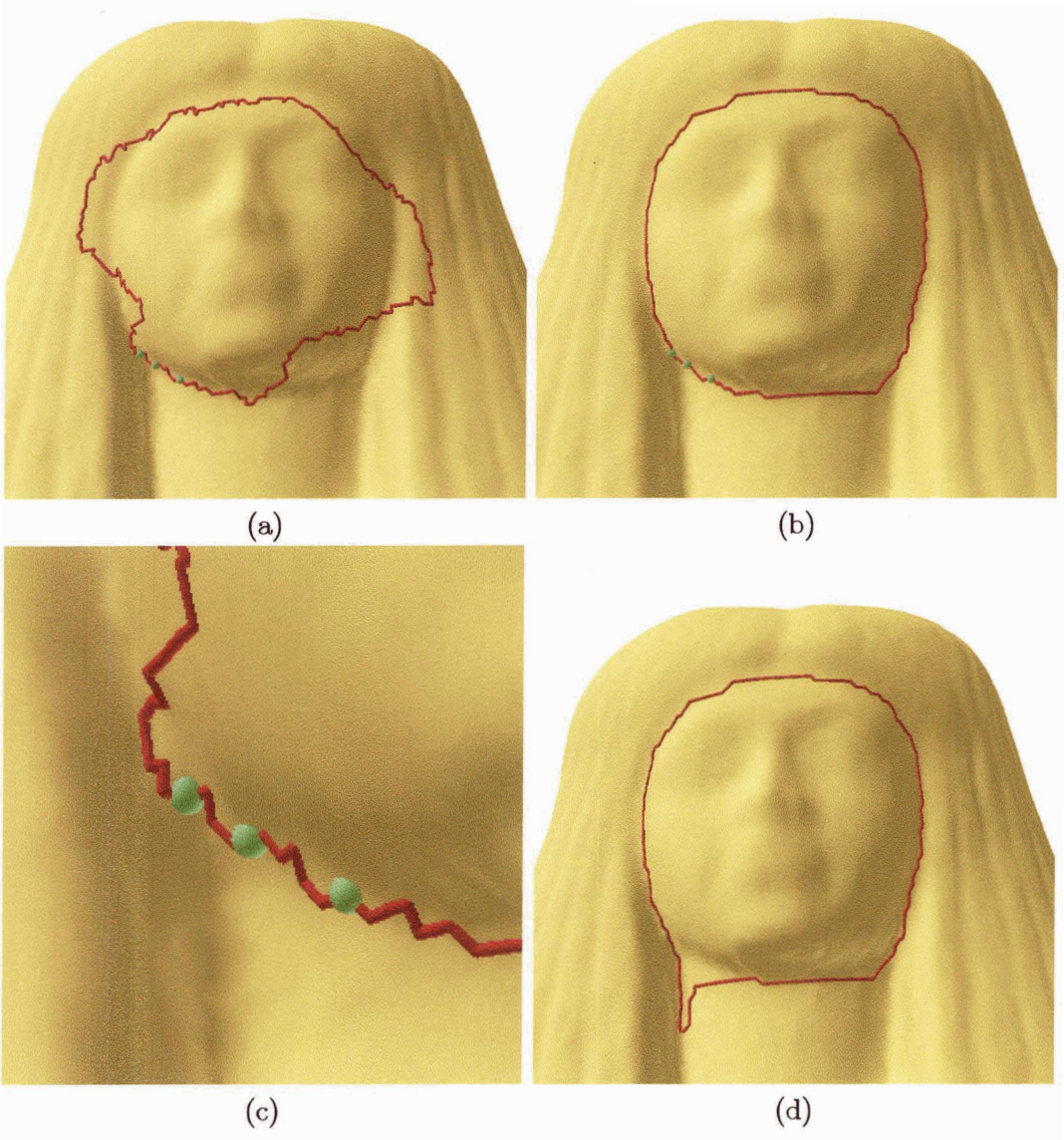


Figure 5.6: (a) Initial contour surrounding face. Green spheres are drawn at the vertices which are hard constraints. (b) Optimal contour found using regular optimization with constraints. The contour correctly segments the face. The optimization used a search space of width 4, and took 3 iterations to complete. (c) A close up of the hard constraints shown in (a). (d) Regular optimization without using hard constraints. The face is not correctly segmented.

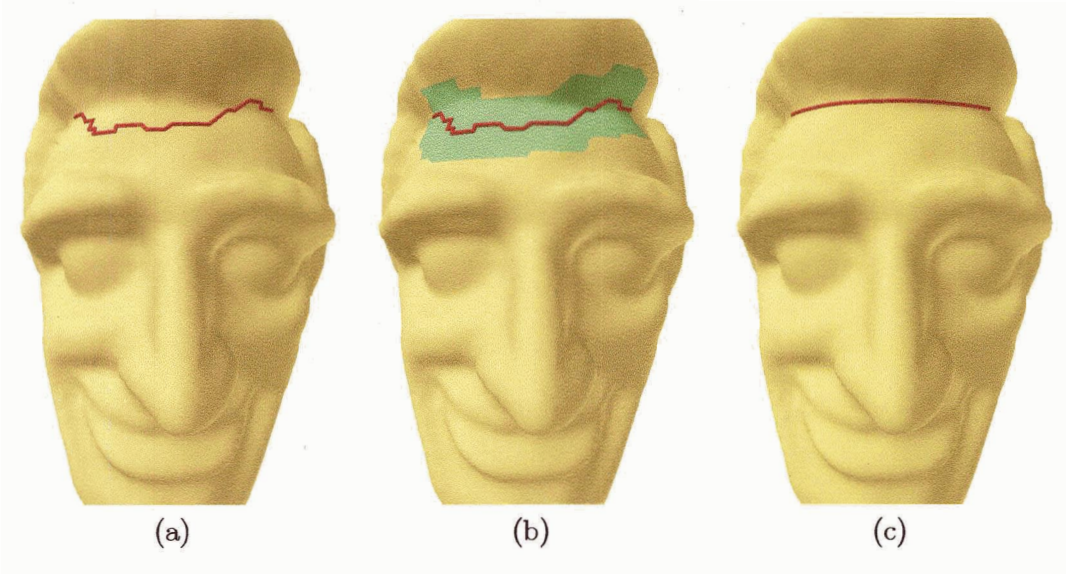


Figure 5.7: (a) Initial open curve. (b) The truncated search space surrounding the open curve of width 5 is shown in green. (c) After two iterations of the regular MRC algorithm, the contour converges to the final position.

5.6 Statistics and timings

All experiments were performed on a commodity computer with a 2.80 GHz processor and 2 GB of RAM. The timing statistics for all experiments in this chapter can be found in Table 5.2. Corresponding statistics regarding the search space and the constructed acyclic graphs can be found in Table 5.1.

The running times of the minimum ratio contour algorithm using the regular search graph are seen to be very fast, even with large neighbourhoods. Refinement yields much slower running times. This is not only attributed to the larger search graph (although it does play a factor), but to the large increase in the number of edges in a refined cut. Since the MRP algorithm must be run once for every edge in the cut, the sheer number of MRP iterations increases the running time significantly. For this reason, it is more efficient to run the algorithm with a large neighbourhood size in the regular case, and once the contour has settled, run the refined optimization to produce a final, smoother contour.

Of special interest in Table 5.2 is the recorded maximum number of linear search steps required to solve each MRP problem (one per pair of source and destination nodes), as discussed in Section 4.2.5. In all of our experiments, this number is at most 6, attesting to the efficiency of the linear search. This is in agreement with the results of other researchers

[21, 50] running minimum ratio cycle algorithms on graphs constructed from images.

Mesh	Graph Resolution	Search Band Width	# Faces	Size of Cut	# Nodes	# Arcs
Horse	Regular	2	979	9	1356	3588
Horse	Refined: $k = 1$	2	979	34	8535	35833
Horse	Refined: $k = 2$	2	979	78	21332	125258
Hand-finger	Regular	9	971	39	1482	4075
Hand-thumb	Regular	4	941	17	1414	3785
Hand-thumb	Regular	8	1934	33	2915	8167
Headless	Regular	12	3563	49	5386	15342
Venus-left eye	Regular	2	644	9	964	2476
Venus-right eye	Regular	2	740	9	1118	2853

Table 5.1: Statistics for the construction of search spaces and acyclic graphs. The number of faces is collected within the search space, on the original unrefined mesh. The number of nodes and arcs are collected on the constructed acyclic graph, which, given a fixed search space, would grow as the number of refinement levels k increases. These statistics are for the first iteration of optimization only.

Mesh	Max. # Linear Steps	Acyclic Time	Min. Ratio	Time/Iter	# Iters.	Total
Horse: reg	4	0.10	0.01	0.11	1	0.11
Horse: $k = 1$	5	0.14	0.33	0.47	1	0.47
Horse: $k = 2$	6	0.22	2.47	2.69	1	2.69
Hand-finger	5	0.02	0.01	0.03	1	0.03
Hand-thumb	6	0.05	0.01	0.06	1	0.06
Hand-thumb	6	0.09	0.06	0.15	1	0.15
Headless	6	0.21	0.18	0.39	2	0.78
Venus-left eye	4	0.00	0.04	0.04	5	0.20
Venus-right eye	5	0.00	0.06	0.06	5	0.30

Table 5.2: Timing statistics for our minimum ratio algorithm. Statistics for the construction of search spaces and acyclic graphs of the models are given in Table 1. Second column shows the maximum number of linear search steps needed per MRP problem. Third column shows time required for the construction of the acyclic graph, including finding the search space and edge cut. Fourth column shows time required to compute the MRC. Fifth column shows total time per iteration. Sixth column shows the total number of graph searches (iterations) performed to locate the final optimal contour. The last column shows the total time required. All timing statistics are averages if more than one iteration is required (except the last column), and are measured in seconds.

Chapter 6

Conclusion

This thesis has presented a minimum ratio contour algorithm for optimizing an energy ratio within a prescribed search band surrounding a given initial contour. The energy to be minimized is a ratio of a numerator energy to contour length. The numerator energy takes into account the Gestalt laws of closure, proximity, and continuity [29], as well as the minima rule [18] from psychology. It is also feature conforming and does not have any bias toward either short or long contours, nor any tendency to produce a large enclosed region, as with normalized cuts [45]. Our minimum ratio contour algorithm is efficient and has demonstrated promising results for post-processing of segmented meshes. This is made possible with a novel construction of an acyclic search graph and its associated edge weights.

6.1 Future work

For possible future work, we would like to investigate whether it is possible to introduce a bias towards short contours in the minimum ratio framework, so as to add more flexibility to our approach. This is covered in Section 6.1.1.

Furthermore, we would like to extend the search space to over the whole mesh surface. An interesting global approach would be to consider incorporating region information into the energy definition, this is discussed in Section 6.1.2. To be able to achieve reasonable speed for such a global optimization, a multiresolution approach may become necessary.

6.1.1 Length bias

The energy definition used in minimum ratio techniques does not have an inherent bias towards shorter contours. This contrasts with snakes, which do include a length energy term (as part of the internal energy term) that strives to keep a snake (contour) short. The lack of length bias is considered an advantage of the ratio energy, since the numerator energy of a contour is normalized by its length, and this ensures that trivial solutions are not minimizers of the ratio energy. However, it could potentially be desirable to combine the benefits of the ratio energy definition with the inclusion of a length term. Here, desired contours have a low energy ratio, but should also be short.

This could be accomplished by seeking a contour which minimizes a weighted sum of ratio energy and length as follows

$$(1-\alpha)\cdot\frac{numerator(\pi)}{arclength(\pi)}+\alpha\cdot len(\pi). \tag{6.1}$$

Here $numerator(\pi)$ gives the numerator energy of a contour π (measuring the internal and external energy of π), and $len(\pi)$ is some form of length of π. The parameter α determines the importance of minimizing length energy vs. ratio energy. Appendix A describes an algorithm for finding the minimum mean path with length in an acyclic graph. The algorithm is derived from Karp's algorithm [26] for computing the minimum mean cycle in a directed graph. This algorithm could be interchanged with the present algorithm in Section 4.2.5 for computing a minimum ratio path in an acyclic graph. If this algorithm is used, then an extra parameter α must be chosen by the user. This parameter may be hard to choose in general, since the contour length depends on the mesh resolution.

The algorithm in the Appendix A only handles the case where the length function $len(\pi)$ in equation 6.1 is the number of edges in π. This would be adequate if all edges in the mesh are nearly equal in length. However, it would be desired to expand this, so that the arclength of a contour could be taken into account. Another drawback of the algorithm in Appendix A is that it requires $O(nm)$ time and space (where n is the number of nodes in the graph, and m the number of edges). This limits its practicality for use with large graphs, especially refined graphs. Future work would therefore also involve attempting to obtain a more efficient algorithm for finding the minimum mean path with length.

6.1.2 Region information

This thesis has considered finding an optimal ratio contour on the surface of a mesh. We have incorporated perceptual considerations into the ratio energy definition, in an attempt to return perceptually salient contours. In this framework, the salience of a contour which delimits a feature is based solely on the points on the surface of the mesh that this contour passes through. No region information, or information about the feature part that a contour segments is included in the energy definition. This is adequate for the current application of post-processing segmented mesh boundaries. However, for the task of defining and locating a globally optimal contour on a mesh, such region information is highly relevant. For instance, Hoffman and Singh's [19] theory of *part salience* uses the area of segmented parts (among other criteria) in an attempt to establish the relative saliencies among differing contours.

Interestingly, Jermyn and Ishiwaka [21] have shown how to include region information for usage with a minimum ratio cycle algorithm operating on images. This is achieved through the use of Green's Theorem [24], which relates the integral of a vector field over the boundary of a region to an integral of the divergence of the vector field over the interior. Thus, the energy of a contour can include such information as the area or texture homogeneity inside the region.

To correlate such region information with contours on meshes, the more general Stoke's Theorem is required to deal with regions on surfaces in three dimensions. This has not been attempted before, and could form a substantial research problem to pursue.

Appendix A

Minimum Mean Path With Length

This appendix describes the minimum mean cycle problem, and a solution that is due to Karp [26]. It then describes modifications to solve a minimum mean path with length problem.

A.1 Karp's minimum mean cycle algorithm

This section describes the minimum mean cycle problem, and the algorithm given by Karp to solve it. The reader is referred to Karp's original paper [26] for further details and proofs.

We start by considering a general graph $G = (V, E)$. Here V is a set of vertices, and E a set of directed edges (these vertices and edges are not in any way related to the vertices and edges of a mesh). We define a function $f : E \rightarrow \mathbb{R}$, which associates a weight with every edge in E. The term *edge progression* is used to represent a sequence of edges $\pi = e_1, e_2, \ldots, e_k$ in E. The weight of an edge progression π is defined to be $w(\pi) = \sum_{i=1}^{k} f(e_i)$, and the number of edges in π is denoted by $\#(\pi)$. Let $\mathscr{C}$ be the set of all directed cycles in G. Then the minimum mean cycle problem is to find the cycle $\pi^* \in \mathscr{C}$ which minimizes the ratio $\tau(\pi) = w(\pi)/\#(\pi)$. That is,

$$\pi^* = \text{argmin}_{\pi \in \mathscr{C}} \tau(\pi) = \text{argmin}_{\pi \in \mathscr{C}} \frac{w(\pi)}{\#(\pi)}. \tag{A.1}$$

To solve for π^*, Karp introduces the function $F_k(v)$. Given an arbitrary source vertex s, and a non-negative integer k, $F_k(v)$ is defined as the minimum weight of an edge progression of exactly k edges from from s to v. Note that such an edge progression may self-intersect or self-overlap. If no edge progression from s to v of exactly k edges exists, then $F_k(v) = \infty$.

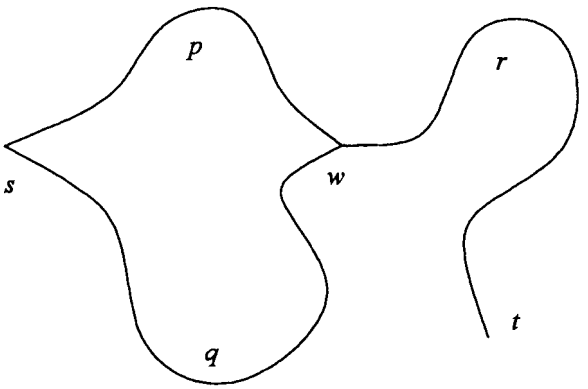


Figure A.1: Two edge progressions p and q, are shown from vertex s to vertex w. A third, completing edge progression r is shown from w to t.

Let n be the number of vertices in V. Then the ratio λ^* of the minimizing cycle π^* is given by

$$\lambda^* = \min_{v \in V} \max_{0 \le k \le n-1} \left(\frac{F_n(v) - F_k(v)}{n-k} \right). \tag{A.2}$$

The values of $F_k(v)$ can be computed using the recurrence relations

$$\begin{aligned} F_k(v) &= \min_{(u,v) \in E} \Big(F_{k-1}(u) + f((u,v)) \Big), \quad \text{for } k = 1, 2, \ldots, n \\ F_0(s) &= 0 \\ F_0(v) &= \infty, \quad \forall\, v \neq s \end{aligned} \tag{A.3}$$

Computation of $F_k(v)$ for all v and k can be accomplished in a recursive fashion using dynamic programming. The idea is to perform a sequence of decisions, each of which only depends on the last decision made. We compute $F_k(v)$ from the immediately previous $F_{k-1}(v)$ values of its neighbouring vertices. For every $F_k(v)$ computed, we store its minimizing vertex u in a predecessor table. This table is needed for extracting the minimum ratio cycle.

Equation A.2 yields the ratio of the minimum mean cycle, but does not provide the actual minimizing cycle. To find the minimizing cycle, we first find the minimizing v and maximizing k in equation A.2. A minimum weight edge progression σ of length n from s to v is found via back-tracking (this can be accomplished by using the predecessor table). The minimizing cycle is a cycle of length $n-k$ which occurs within σ.

The entire algorithm requires $O(nm)$ time, where n and m are the number of vertices and edges in G, respectively.

The correctness of the recurrence relation above to compute the $F_k(v)$ values depends on a minimality principle. The value of $F_k(v)$ at stage k should only depend on the values of $F_k(v)$ at previous stages. The idea is as follows: consider any two distinct edge progressions p, q, both from a vertex s to a vertex w, such that $\#(p) = \#(q)$, see Figure A.1. We use the notation $a \cdot b$ to refer to the concatenation of two edge progressions. If $w(p) < w(q)$, then for any possible edge progression r from w to a third vertex t, we have

$$\begin{aligned} w(p) + w(r) &< w(q) + w(r) \\ w(p) + w(r) = w(p \cdot r) &< w(q \cdot r) = w(q) + w(r) \end{aligned} \tag{A.4}$$

No matter what completing edge progression r is used, the minimum weight (and hence also the mean) of the completion of the edge progression p to r will always be less than that of the completion q to r. Therefore, we only need to keep track of one (the minimum) $F_k(w)$ value for every k and vertex w.

In essence, the $F_k(v)$'s are computing the shortest path of k edges from s to every other vertex, for all k. Every vertex holds k bins, each to store the weight of the shortest path of k edges to this vertex.

A.2 Combining ratio and length

A possible generalization is to include a length bias with the ratio of equation A.1. We could then consider the problem of finding the minimum mean cycle with length. The problem is to find the path (edge progression) π which minimizes the ratio

$$\phi(\pi) = (1 - \alpha) \cdot \frac{w(\pi)}{\#(\pi)} + \alpha \cdot \#(\pi). \tag{A.5}$$

That is,

$$\pi^* = \text{argmin}_{\pi \in \mathscr{C}} \phi(\pi) = \text{argmin}_{\pi \in \mathscr{C}} \Big((1 - \alpha) \cdot \frac{w(\pi)}{\#(\pi)} + \alpha \cdot \#(\pi) \Big). \tag{A.6}$$

where $\alpha \in [0, 1]$. The parameter α controls the tradeoff between minimizing the ratio $w(\pi)/\#(\pi)$ and the length $\#(\pi)$.

As previously discussed in Section 4.2.5, if the minimum mean cycle in a graph is unique, then it is a simple, non-intersecting cycle. However, the minimum mean path between two vertices in a general graph is not guaranteed to be simple. In the case that G is restricted to be an acyclic graph, the minimum mean path from vertex s to vertex v is obviously simple

(if a path exists from s to v), and is easily solved. The ratio λ^* of the minimizing path is given by

$$\lambda^* = \min_{k=1,\ldots,n} \frac{F_k(v)}{k}. \tag{A.7}$$

The minimizing path is also easily obtained by back-tracking using the predecessor table.

We now consider the problem of solving the minimum mean path with length problem in an acyclic graph, from an arbitrary source vertex s to a destination vertex v. If $\mathscr{P}_k$ is the set of length k edge progressions that start from s, define the function $\widehat{F}_k(v)$ to be

$$\widehat{F}_k = \min_{p \in \mathscr{P}_k} \Big((1-\alpha) \cdot F_k(v) + \alpha \cdot k^2 \Big). \tag{A.8}$$

The ratio of the minimizing path is then given by

$$\lambda^* = \min_{k=1,\ldots,n} \frac{\widehat{F}_k(v)}{k}. \tag{A.9}$$

The $\widehat{F}_k(v)$'s are computed using the following recurrence relation

$$\begin{aligned}
&\widehat{F}_k(v) = \min_{(u,v)\in E} \Big(\widehat{F}_{k-1}(u) + (1-\alpha) \cdot f((u,v)) + \alpha(k^2 - (k-1)^2) \Big), \quad \text{for } k = 1, 2, \ldots, n \\
&\widehat{F}_0(s) = 0 \\
&\widehat{F}_0(v) = \infty, \quad \forall\, v \neq s
\end{aligned} \tag{A.10}$$

It remains to show that the minimality principle still holds for the computed $\widehat{F}_k(v)$'s. As before, consider two distinct edge progressions p, q, both from a vertex s to a vertex w, such that $\#(p) = \#(q)$, see Figure A.1. We wish to show that if

$$(1-\alpha) \cdot w(p) + \alpha \cdot \#(r) < (1-\alpha) \cdot w(q) + \alpha \cdot \#(r), \tag{A.11}$$

then for any possible edge progression r from w to a third vertex t, we have

$$(1-\alpha) \cdot \frac{w(p \cdot r)}{\#(p \cdot r)} + \alpha \cdot \#(p \cdot r) < (1-\alpha) \cdot \frac{w(q \cdot r)}{\#(q \cdot r)} + \alpha \cdot \#(q \cdot r) \tag{A.12}$$

This is shown by

$$\begin{gathered}
(1-\alpha) \cdot w(p) + \alpha \cdot \#(r) < (1-\alpha) \cdot w(q) + \alpha \cdot \#(r), \\
(1-\alpha) \cdot (w(p) + w(r)) + \alpha \cdot (\#(p) + \#(r))^2 < (1-\alpha) \cdot (w(q) + w(r)) + \alpha \cdot (\#(p) + \#(r))^2 \\
(1-\alpha) \cdot \frac{w(p) + w(r)}{\#(p) + \#(r)} + \alpha \cdot (\#(p) + \#(r)) < (1-\alpha) \cdot \frac{w(q) + w(r)}{\#(p) + \#(r)} + \alpha \cdot (\#(p) + \#(r))
\end{gathered}$$

Now, since $\#(p) = \#(q)$, we have

$$(1-\alpha)\cdot\frac{w(p)+w(r)}{\#(p)+\#(r)}+\alpha\cdot(\#(p)+\#(r)) < (1-\alpha)\cdot\frac{w(q)+w(r)}{\#(q)+\#(r)}+\alpha\cdot(\#(q)+\#(r))$$

$$(1-\alpha)\cdot\frac{w(p\cdot r)}{\#(p\cdot r)}+\alpha\cdot\#(p\cdot r) < (1-\alpha)\cdot\frac{w(q\cdot r)}{\#(q\cdot r)}+\alpha\cdot\#(q\cdot r)$$

as desired.

Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003.

[2] A. Amini, S. Tehrani, and T. Weymouth. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints. In *IEEE Second Int. Conf. on Comp. Vision*, pages 95–99, 1988.

[3] S. Bischoff and L. Kobbelt. Parameterization-free active contour models. *The Visual Computer*, 20:217–228, 2004.

[4] S. Bischoff, T. Weyand, and L. Kobbelt. Snakes on triangle meshes. *Bildverarbeitung für die Medizin*, pages 208–212, 2005.

[5] E.G. Boring. *Sensation and perception in the history of experimental psychology.* New York: Appleton, 1942.

[6] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[7] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 586–593, New York, NY, USA, 2002. ACM Press.

[8] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *Int. Journal on Computer Vision*, 22(1):61–79, 1997.

[9] L.T. Cheng, P. Burchard, B. Merriman, and S. Osher. Motion of curves constrained on surfaces using a levet-set approach. *Journal of Computational Physics*, 175:604–644, 2002.

[10] L. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, 53(2):211–218, 1991.

[11] D. Cohen-Steiner and J.M. Morvan. Restricted delaunay triangulations and normal cycle. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 312–321, New York, NY, USA, 2003. ACM Press.

[12] W. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2):138–148, 1999.

[13] T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1997.

[14] I. J. Cox, S. B. Rao, and Y. Zhong. "ratio regions": A technique for image segmentation. In *IEEE Int. Conf. on Pattern Recognition*, pages 557–564, 1996.

[15] S. Dong, P-T. Bremer, M. Garland, V. Pascucci, and J.C. Hart. Spectral surface quadrangulation. In *SIGGRAPH '06: Proceedings of the 33rd annual conference on Computer graphics and interactive techniques.* ACM Press, 2006. To appear.

[16] J. Elder and S. Zucker. Computing contour closure. In *Proc. European Conf. Computer Vision*, pages 399–412, 1996.

[17] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. on Graphics*, 23(3):652–663, 2004.

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

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

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1992. ACM Press.

[21] I. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio weight cycles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(10):1075–1088, 2001.

[22] M. R. Jung and H. K. Kim. Snaking across 3d meshes. In *Pacific Graphics*, 2004.

[23] G. Kaniza. *Organization in Vision: Essays on Gestalt Perception.* New York: Praeger, 1979.

[24] W. Kaplan. *Advanced Calculus.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[25] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[26] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[27] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. Journal on Computer Vision*, 1(4):321–331, 1987.

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

[29] K. Koffka. *Principles of Gestalt Psychology.* Harcourt Brace and Company, New York, 1935.

[30] I. Kovacs and B. Julesz. A closed curve is much more than an incomplete one: Effect of closure in figure-ground segmentation. In *Proc. Nat'l Academy of Science USA*, volume 90, pages 7495–7497, 1993.

[31] B. Kronrod and C. Gotsman. Optimized compression of triangle mesh geometry using prediction trees. In *Proceedings of the International Symposium on 3D Data Processing Visualization and Transmission*, pages 602–608, 2002.

[32] S. Kwek and K. Mehlhorn. Optimal search for rationals. *Inf. Process. Lett.*, 86(1):23–26, 2003.

[33] M. Lanthier, A. Maheshwari, and J-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13-th Annual Symposium of Computational Geometry*, 1997.

[34] E. Lawler. Optimal cycles in doubly weighted directed linear graphs. In *Proc. Int'l Symp. Theory of Graphs*, pages 209–232. Gordon and Breach, 1966.

[35] Y. Lee and S. Lee. Geometric snakes for triangular meshes. *Computer Graphics Forum*, 21:229–238, 2002.

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

[37] X. Li, T.W. Toon, and Z. Huang. Decomposing polygon meshes for interactive applications. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 35–42, New York, NY, USA, 2001. ACM Press.

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

[39] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *IEEE Int. Conf. on Computer Vision*, pages 840–845, 1995.

[40] N. Meggido. Combinatorial optimization with rational objective functions. *Math. Operations Research*, 4:414–424, 1979.

[41] M. J. Milroy, C. Bradley, and G. W. Vickers. Segmentation of a wrap-around model using an active contour. *Computer Aided Design*, 29(4), 1997.

[42] R. Ohbuchi, S. Takahashi, T. Miyazawa, and A. Mukaiyama. Watermarking 3d polygonal meshes in the mesh spectral domain. In *GRIN'01: Proceedings of Graphics Interface 2001*, pages 9–17, Toronto, Ont., Canada,, 2001. Canadian Information Processing Society.

[43] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *Mathematical Visualization*, pages 391–410, 1998.

[44] J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, London, England, 1982.

[45] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Int. Conf. on Computer Vision*, pages 731–737, 1997.

[46] P. Soundararajan and S. Sarkar. An in-depth study of graph partitioning measures for perceptual organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):642–660, 2003.

[47] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 130, Washington, DC, USA, 2003. IEEE Computer Society.

[48] A. Tal and E. Zuckerberger. Mesh retrieval by components. In *International Conference on Computer Graphics Theory and Applications*, pages 142–149, 2006.

[49] Olga Veksler. Stereo matching by compact windows via minimum ratio cycle. In *ICCV*, pages 540–547, 2001.

[50] Olga Veksler. Stereo correspondence with compact windows via minimum ratio cycle. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1654–1660, 2002.

[51] J. L. Villar. The underlying graph of a line digraph. *Discrete Applied Mathematics*, 37/38:525–538, 1992.

[52] S. Wang, T. Kubota, J. M. Siskind, and J. Wang. Salient closed boundary extraction with ratio contours. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(4):546–561, 2005.

[53] S. Wang and J. M. Siskind. Image segmentation with minimum mean cut. In *IEEE Int. Conf. on Computer Vision*, pages 517–524, 2001.

[54] O. Watson, editor. *Longman Modern English Dictionary.* The Chaucer Press, 1976.

[55] M. Wertheimer. Gestalt theory. In W. Ellis, editor, *A Source Book for Gestalt Psychology.* Harcourt Brace and Company, 1938.

[56] D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Understanding*, 55(1):14–26, 1992.

[57] Y. Zhao, H.Y. Ong, T.S. Tan, and Y. Xiao. Interactive control of component-based morphing. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 339–348, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[58] S.C. Zhu. Embedding gestalt laws in markov random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(11):1170–1187, 1999.