COMPUTATIONAL TOPOLOGY ALGORITHMS FOR DISCRETE 2-MANIFOLDS

Thesis by

Zoë Justine Wood

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2003

(Defended May 16, 2003)

Dedicated to Barbara A. Barnett

1947–2003

Acknowledgements

I would like to thank my advisor, Peter Schröder for his guidance and support during my graduate career. He helped me grow into a deliberate researcher and provided enough flexibility for me to maintain my commitments to my family and to my academic career. This kindness and the opportunities that Peter continually provided to me are a gift I truly treasure.

My entire thesis committee: Peter Schröder, Mathieu Desbrun, Hugues Hoppe, Al Barr, and Pietro Perona provided me with valuable feedback and support to improve this thesis. I am indebted for all of the knowledge and advice they each shared with me.

I am also indebted to Mathieu Desbrun, Hugues Hoppe and Steven (Shlomo) Gortler for their academic guidance and personal support. Mathieu believed in me before I believed in myself and showed me how to climb steep hills a step at a time. Hugues has always treated me as if I could solve any problem he could solve. This faith helped me to see the path to my Ph.D. which had sometimes been clouded by my doubts in myself. Shlomo gave me hours of his time, without these conversations about loops, topology and cute babies, this thesis would be a sad creature. His kindness and wit helped remind me why computer graphics and the community that studies it are so cool. Without the collegial support and role-modeling of each of these mentors, this thesis might not have seen the light of day. I am very thankful for each of them.

To Eitan Grinspun, I am ever thankful that we started down this path of graduate school together. He provided support, companionship, editing, and dance partnership when I needed it most. Pedro Sander, provided collaboration, inspiration, and entertainment for our time working together. I would also like to thank all the various folks I have interacted with who have been a part of the Multi-Res Modeling and Graphics groups at Caltech and Microsoft Research. In particular, Nathan Litke, Igor Guskov and John Snyder have been helpful and kind colleagues.

In this field of few women, there are a few wonder women who helped me survive this process: Eve Schooler, mother, wife, Ph.D., thank you for being my officemate and reminding me so many days of my own talents; Thank you to Allison Klein for all the advice and support - meeting Allison allowed me to finally see myself reflected in my field, a gift I am so thankful for. The struggles of Jeroo Sinor, Jyl Boline, Julie Glass and Cricket Wood, as biology and math researchers are a constant inspiration to me of brainy scientist women existing in this world. And my owl-grandma's and my mother's love of mathematics always served as a talisman for my own explorations in mathematics. Another wonder woman I must thank is Jane Estes.

Without her council, I cannot imagine finding my way through this tangled path of the last six years.

Finally, my friends and family are the roots that feed my growth – all of them believed in me and were my cheer-leaders through many hard years and whiny phone calls. I cannot thank them all enough for their love, support, advise and friendship: Allison Klein, Julie Glass, Shlomo Gortler, Eitan Grinspun, Jyl Boline, Jeroo Sinor, Jamie Gross and Michelle Toth, Farnaz Fatemi and Paul Skenazy, Cricket Wood and Paul Muhl, Barbara and William Wood and my amazing husband, Demian Barnett. Jyl has given me a space to land in Pasadena and the care and friendship to make that space feel like a home. Jeroo has bridged the distance and made me feel cared for across all those miles. Eitan has given me singing lessons and honed my arguing skills, but most of all, he has inspired me to follow in his footsteps and try to live life as I am. Jamie's humor and friendship have made me laugh when I needed it most and her editing and design skills have helped me shine. Thank you to Farnaz, for being my travel companion and for wearing her lesbian avengers t-shirt the first time she came to Caltech and for loving me through my choices. Cricket's grounded advice, smile and support from her and Paul have helped me through so many hard times. In addition, I must thank the Muhlwood twins, my nephews, Kai and Tavi, for letting me cuddle them and soak in their baby warmth and calm when I have needed it. Thank you to my parents, Barbara and Willie, for their love, support and reminders to not let the turkeys get me down. I cannot count how many times I have escaped to the shelter of their home and love. Finally, thank you to Demian, whose love has sustained me, patience redeemed me, and our wacky dances have always helped me remember what really matters.

Abstract

This thesis presents computational topology algorithms for discrete 2-manifolds. Although it is straightforward to compute the genus of a discrete 2-manifold, this topological invariant does not tell us enough for most computer graphics applications where we would like to know: what does the topology look like? Genus is a scalar value with no associated geometric appearance. We can, however, isolate geometric regions of the surface that are topologically interesting. The simplest topologically interesting, and perhaps most intuitive, regions to consider are those with genus equal to one. By isolating and examining such regions we can compute measures to better describe the appearance of relevant surface topology. Thus, this work focuses on isolating *handles*, regions with genus equal to one, in discrete 2-manifolds.

In this thesis, we present novel algorithms guaranteed to identify and isolate handles for various discrete surface representations. Additionally, we present robust techniques to measure the geometric extent of handles by identifying two locally minimal-length non-separating cycles for each handle. We also present algorithms to retain or simplify the topology of a reconstructed surface as desired. Finally, the value of these algorithms is demonstrated through specific applications to computer graphics. For example, we demonstrate how geometric models can be greatly improved through topology simplification both for models represented by volume data or by triangle meshes.

Contributions The contributions of this work include:

- A robust and efficient method for identifying and isolating handles for discrete 2-manifolds.
- A method to robustly represent the topology of the surface with an *augmented Reeb graph*.
- A robust method to find two locally minimal-length non-separating cycles for each handle.
- A simple method to simplify the topology for volume data and triangle meshes which preserves the local geometry as much as possible.
- An out-of-core method for topology simplification for volume data.

Table of Contents

Acknowledgements **iv**

Abstract **vi**

List of Figures **x**

1 Introduction **1**

1.1 General setting 1

1.2 Framework and motivation 3

1.2.1 The interplay of geometry and topology 4

1.3 Introduction to the algorithms 6

1.3.1 Localizing and coding topology 6

1.3.2 Measuring feature size 7

1.3.3 Re-sampling topology: preserving genus and simplifying genus 7

1.4 Brief discussion of the applications 8

1.4.1 Isosurface topology simplification 8

1.4.2 Mesh topology simplification/Topological noise removal 9

1.4.3 Semi-regular mesh extraction from volume data 9

1.5 Contributions 10

1.6 Acquired Data 12

1.7 Summary 12

2 Background **13**

2.1 Surfaces 13

2.1.1 Simplicial complexes 13

2.1.2 Triangle meshes 14

2.1.3 Scalar volumes 15

2.1.4 Surfaces as graphs 16

2.2 Topology 16

2.2.1 Historical and mathematical context 17

2.2.2 Morse theory 18
2.2.3 Reeb graphs 21
2.2.4 Some useful definitions 21

3 Related Work **23**
3.1 Identifying topology 23
3.1.1 Morse theory 23
3.1.2 Reeb graphs 26
3.2 Cutting meshes open 28
3.2.1 Cut graphs 28
3.2.2 Polygonal schema 28
3.3 Topology simplification 29
3.3.1 Mesh-based topology simplification 29
3.3.2 Volume-based topology simplification 30
3.3.3 Model simplification 30
3.4 Computational topology for computer graphics 31

4 Algorithms **32**
4.1 Introduction 32
4.2 Localizing topology 33
4.2.1 Identifying topology 34
4.2.1.1 Morse functions 34
4.2.1.2 Height function and immersion 35
4.2.1.3 Face-by-face traversal 36
4.2.1.4 Constructing level sets 38
4.2.1.5 Intra-ribbon handles 39
4.2.2 Coding the topology 41
4.2.2.1 Reeb graph 42
4.2.2.2 Augmented Reeb graph 42
4.2.2.3 Contour construction 44
4.2.2.4 Ribbon construction 44
4.2.2.5 Asserting Reeb graph consistency 45
4.2.3 Detecting handles 47
4.2.4 Isolating handles 48
4.2.4.1 Combinatorial considerations 49
4.2.4.2 Pseudo-code for detection and isolation 50
4.3 Measuring feature size 50

4.3.1 Problem statement . . . 50
4.3.2 Loops . . . 51
4.3.3 Overview: two non-separating cycles . . . 52
4.3.3.1 General idea . . . 53
4.3.3.2 Shortest cycles . . . 53
4.3.3.3 Measure of handle sizes . . . 57
4.3.3.4 Measurement alternatives . . . 57
4.4 Resampling topology . . . 57
4.4.1 Handle removal . . . 58
4.4.1.1 Triangle meshes . . . 59
4.4.1.2 Volume . . . 59
4.4.1.3 Considerations . . . 60
4.4.2 Handle retention . . . 60
4.5 Conclusion . . . 63

5 Applications **64**
5.1 Setting . . . 64
5.2 Simplifying topology . . . 64
5.2.1 In volumes . . . 65
5.2.1.1 Our approach . . . 66
5.2.1.2 Locating handles . . . 67
5.2.1.3 Measuring topological handle size . . . 68
5.2.1.4 Removing handles . . . 69
5.2.1.5 Results and discussion . . . 69
5.2.1.6 Applications . . . 71
5.2.1.7 Discussion . . . 74
5.2.1.8 Summary . . . 77
5.2.2 For meshes . . . 77
5.2.2.1 Topological noise removal for meshes . . . 77
5.2.2.2 Overview of the algorithm . . . 77
5.2.2.3 Algorithm . . . 78
5.2.2.4 Region growing . . . 79
5.2.2.5 Cutting the mesh . . . 79
5.2.2.6 Global procedure and preprocessing . . . 81
5.2.2.7 Results . . . 81
5.2.2.8 Conclusions . . . 84

5.2.3 Mesh topology simplification revisited . . . 84
5.3 Hierarchical surface reconstruction . . . 86
5.3.1 In volumes . . . 86
5.3.1.1 Motivation . . . 86
5.3.1.2 Coarse mesh extraction . . . 87
5.3.1.3 Distance function propagation and Reeb graph . . . 87
5.3.1.4 Mesh construction from the Reeb graph . . . 88
5.3.1.5 Discussion . . . 89
5.4 Conclusion . . . 89

6 Conclusion **90**
6.1 Contributions . . . 90
6.2 Future explorations . . . 92

Bibliography **93**

List of Figures

1.1 Examples of geometric models used for various applications 2
1.2 Coffee cup and brain 3
1.3 Close-up view of an extraneous handle 4
1.4 Buddha with excess topology 5
1.5 Localizing a handle 7
1.6 Example of non-separating cycles 8
1.7 Progressive mesh of the David head 9
1.8 Overview of semi-regular mesh extraction 10

2.1 Scalar volume 15
2.2 Voxel 15
2.3 Critical points for a function of two variables 19
2.4 Illustration of height planes on a sphere and torus 20
2.5 Critical points and height planes 20
2.6 Torus and example Reeb graph 21

3.1 The star of two different vertices 25
3.2 A grounded simplex 26
3.3 Two different sized handles 30

4.1 Handle decomposition 34
4.2 Example saddle points 34
4.3 Example of immersion 36
4.4 Face-by-face operations 37
4.5 Example of an intra-ribbon handle 40
4.6 Ambiguous change in two level sets 41
4.7 Arbitrary sampling 43
4.8 Isosurface and Reeb graph 43
4.9 Example Reeb graphs 47
4.10 Isolating handles 48

4.11 Example of finding a Reeb cycle 49
4.12 Two holed torus 50
4.13 Canonical generator loops 51
4.14 Reeb loop 53
4.15 Example of loops on a handle 54
4.16 Possible tilings of a handle 55
4.17 Handle removal 58
4.18 Loops on the feline mesh 59
4.19 Distance function on the feline 61
4.20 Stitching 62
4.21 Examples of coarse meshes 63

5.1 Genus zero Buddha 70
5.2 Progressive meshes of a brain 71
5.3 A remesh of the genus one dragon 71
5.4 Geometry image of the genus one dragon 72
5.5 Labeled cortex models 72
5.6 Texture map comparison 73
5.7 Histogram 74
5.8 Scatterplot 75
5.9 Progressive meshes of dragon 76
5.10 Topological noise removal overview 77
5.11 Handle detection and removal 80
5.12 Cutting the mesh. 80
5.13 Smoothing result 83
5.14 Texture mapped ear 83
5.15 Example of a series of semi-regular meshes 86
5.16 Example of a surfel and following its active edges 88
5.17 Distance function propogation on a surfel 89

Chapter 1

Introduction

Imagine that we live in a world where everything is made of silly putty – very strong silly putty, that can be molded into different shapes, but that cannot be ripped apart and resealed together. In such a world, if you start with a round ball of silly putty – you could push and pull it to mold it into a shape that represented the folds and valleys of your brain. However, if you were asked to model a coffee cup, you could nicely form the bowl for the coffee to sit in, but you would not be able to model the handle without riping a hole in the silly putty (see Figure 1.2). In order for you to be able to build a model of the coffee cup, you would have needed to start with silly putty in a dough-nut shape not a round ball. This is the world of topology, where we ask "what is essential about the shape of an object?" This question has been considered for ages by mathematicians who have developed methods and language to address the issue of what can be said about the essential shape of an object. For this thesis, we re-consider this age-old question, but from the perspective of a computer scientist and more specifically, from the perspective of a computer graphics researcher.

1.1 General setting

One of the main goals of computer graphics is the digital representation of the real three dimensional world. Surface representation (or geometric modeling) is at the heart of this goal. Geometric models are used to represent real world objects in applications such as scientific visualization, topography, engineering, games, virtual environments and even art history, (see Figure 1.1). Digital surface representations can be created by artists but the bulk of geometric models come from acquired data – that is from scanning a real world object. For example using a laser range scanner is one method of acquiring data. This data is a discrete sampling of the geometry of a real world object. From this sampling, a surface is reconstructed and represented with a geometric model such as a triangle mesh. Computer graphics has traditionally focused on the geometry of a model, since the geometry closely determines the general appearance of the model. However, one of the aspects of geometric models that is commonly overlooked is the *topology* of the model and how the topology affects the representation and use of the model. This thesis presents computational algorithms to analyze the topology of geometric models. Building on mathematical tools from the continuous setting, we introduce algorithms, tailored for discrete surface representations to analyze the topology of geometric models. In general the discrete surface representation we work with are piece-wise linear discrete 2-manifolds. We present details about such surface representations in Chapter 2.

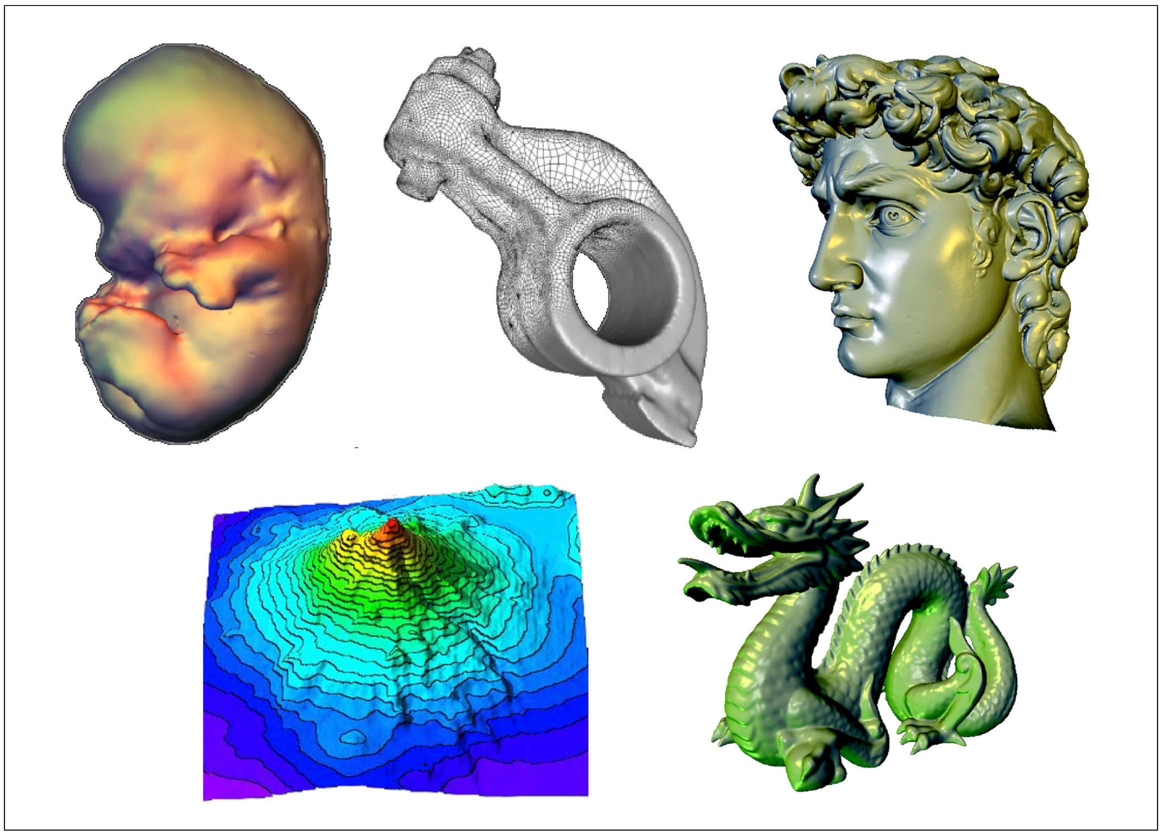


Figure 1.1: *Examples of geometric models used for various applications*
Examples of geometric models used for various applications – including: scientific visualization (isosurface of a mouse fetus), engineering (model of a mechanical part), art history (model of the head of Michelangelo's David statue), topography (model of Mount Shasta) and entertainment (model of a dragon). The model of the mouse fetus is from Caltech, the image of the mechanical part is from Desbrun [20], the model of the head of the David and dragon are from Stanford's model repository, and finally the image of the topographic model of Mount Shasta is from Reynolds et al. *[69]. See Section 1.6 for more information about the data used for this thesis.*

When talking about the topology of a model, one of the main aspects of the topology that we are concerned with is the *genus* of the object. *Genus* is, roughly speaking, the number of handles or "holes" that an object has. In the previous example of the coffee cup and the brain, the coffee cup is genus one and the brain is genus zero. Equivalently a *handle* is a toroidal region of the surface with genus $g = 1$ (various formal definitions exist, see for example, "Conway's ZIP proof" [33]). Talking about handles gives us a way to talk about the shape of an object. *Genus* can be more precisely defined as the largest number of non-intersecting simple closed curves that can be drawn on a surface without separating it, *i.e.*, when the surface is cut along such a curve it is still in one piece. Again, consider the brain and coffee cup example, every closed non-intersecting curve drawn on a brain will separate it into two different pieces of surface, while there are a whole family of closed curves on a coffee cup that would keep the surface as one piece. Consider the curve encircling the handle of the coffee cup. The coffee cup will remain a single connected component even if the surface is cut along such a curve. These curves are called *non-separating cycles*, as they leave the surface connected [5], (see Figure 1.2).

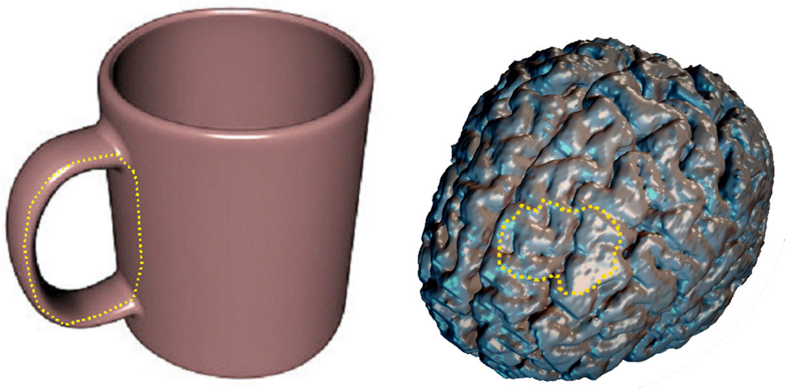


Figure 1.2: *Coffee cup and brain*
A coffee cup (genus one) and a brain (genus zero). Note that there are no non-separating cycles on the brain, while there are a whole family of such cycles on the coffee cup. One non-separating cycle is shown on the handle of the coffee cup (dashed yellow cycle). A separating *cycle is shown on the brain (dashed yellow cycle).*

1.2 Framework and motivation

Why should we care about the shape of an object? One fundamental reason is that to accurately represent our complex world, geometric models need to be able to represent arbitrary topology. We need to be able to represent coffee cups, brains and any other complex shape. Thus, within computer graphics, most geometric models are flexible and can represent arbitrary topology. However, few algorithms have been developed to analyze the topology of the models. There has been extensive focus on the geometry of surfaces and researchers go to great length to try to represent the geometry of a surface as accurately as possible. The digital Michelangelo project [59] acquired models with sub-millimeter geometric accuracy in order to correctly model Michelangelo's chisel strokes on his marble statues. However, a highly accurate geometric model is not necessarily an accurate surface representation. The same Michelangelo statue that had sub-millimeter accurate geometry had excess topology on the order of 957 extra little handles. See Figure 1.2 for an example of excess topology on a digital model.

These excess handles are not only a problem in terms of accuracy, but the topology of a surface affects the efficiency of its digital representation. Excess topology requires excess geometry to represent the handles,

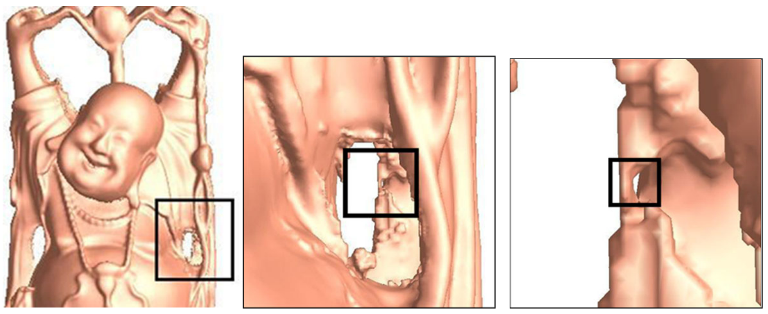


Figure 1.3: *Close-up view of an extraneous handle*
Sequence of progressively closer views revealing an extraneous handle in the Buddha mesh.

(see Figure 1.4). Beyond this, the topology of a surface affects the *parameterization* of a surface. *Parameterization* is an essential tool in computer graphics as it maps the surface of a three dimensional (3D) object into a two dimensional (2D) domain. Mathematical operations are easier to apply to a surface in a 2D domain, thus, parameterization is used regularly in computer graphics to apply texture maps, to remap surfaces to different representations, and even to apply physical simulations to the surface. Consider the task of flattening a model to a 2D domain. To accomplish this, a closed model requires: $2 \times g$ cuts, where g is the genus of the object. This means that for a model with high genus, it will have a large cut boundary which will affect the quality of the parameterization, (see Chapter 5 for more details). Excess topology is just one example of why we want algorithms to analyze the topology of a model. In general, creating accurate models with complex topology requires the ability to analyze the topology of the surface.

1.2.1 The interplay of geometry and topology

Although topology is the study of the "shape" of curves and surfaces, topology typically is not concerned with the embedding of that curve or surface. For example, topology is concerned with the fact that if you remove a point from a circle, it becomes a line segment. This is true whether the circle is an ellipse or whether the circle has knots in it. In computer graphics, we care about the embedding and geometry of a surface. If you were asked to create a digital representation of a coffee cup, no one would be happy if you returned a model that looked like a Krispy Kreme dough-nut. Even though you have returned an object with the correct topological shape, the geometrical shape is incorrect. For computer graphics, we typically are not concerned with purely topological aspects of a surface. Thus, the algorithms we introduce, are founded on the topology, but consider geometric aspects of the surface, for example, geometric measures of the extent of a handle.

This interplay of geometry and topology is inherent in the discrete nature of the surfaces used in computer graphics. Consider the statement above about removing a point from a circle. If we have a discretely sampled circle, removing a point from the circle could be interpreted a number of different ways in the reconstruction of a circle from the discrete points. Perhaps the missing point is just a change in the sampling rate, or perhaps it is break in the circle. When working in a discrete setting, we are given a finite sampling of the surface,

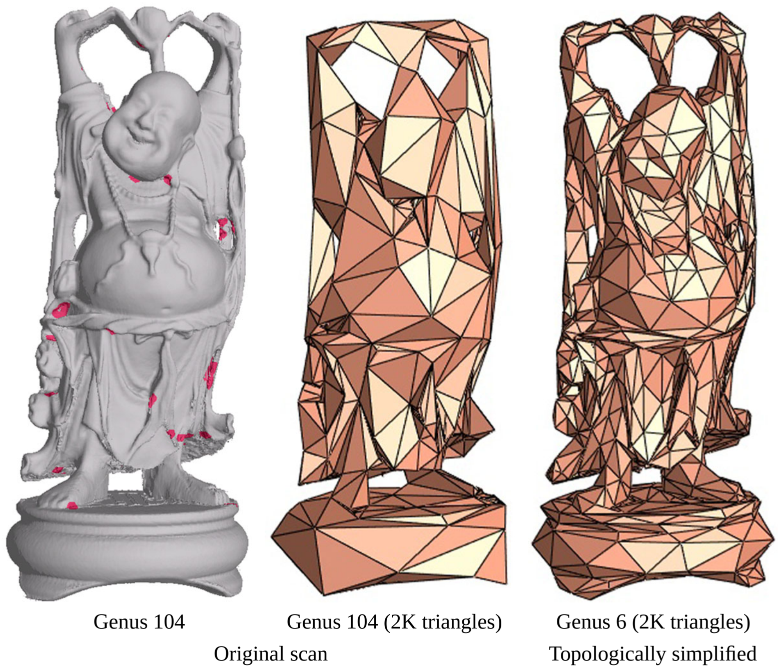


Figure 1.4: *Buddha with excess topology*
This scanned Buddha has genus 104 instead of the expected 6. On the left, regions with extraneous handles are highlighted in red. The two images on the right compare mesh simplification results before and after topology simplification The high genus mesh requires many triangles to needlessly represent topological artifacts, resulting in loss of overall geometrical quality for a given triangle budget.

for example either as points or as scalar grid values in a volume. We typically have rules for reconstructing a discrete surface from these discrete samples. During the reconstruction, geometric choices may be made that have topological consequences. Researchers have developed various tools for surface reconstruction to explore these topological and geometrical choices, such as alpha-shapes [28]. However, for the settings we consider the surface reconstruction is fixed. In such a setting the topology is fixed and we need methods to analyze the existing topology in a geometric context. For this end we must use some geometric measures in order to say something meaningful about the topology.

The genus of a surface in many ways determines the topological complexity of the surface. The fact that it takes $2 \times g$ cuts, (where g is the genus of the surface) to flatten a surface into a disk, is just one indicator of how genus affects operations to manipulate surfaces. Determining the genus of a discrete 2-manifold is a relatively simple task. By sweeping over the surface and counting the number of vertices (V), edges (E), and faces (F) for each individual surface component, we can compute the *Euler characteristic*, χ. Specifically, $\chi = |V| - |E| + |F|$, and the surface *genus* is $g = (2 - \chi)/2$ (for each of the the individual components of the surface). Genus is a global invariant for the surface and its scalar value gives us one measure of the complexity of the surface, (*e.g.*, a genus zero shape is much less complex then a surface with $g = 100$). However, this number on its own does not provide a complete picture of the topology of the surface. Genus is a global scalar value for the entire surface and does not give information about the location or geometric extent of a given topological feature of a surface.

For the field of computer graphics, great care is taken with the geometry of a surface, as the geometry plays such an important role in determining the appearance of a surface. Although a coffee cup is topologically equivalent to a dough-nut, geometrically the shapes differ. And the difference in their appearance matters greatly when the goal is accurately representing the appearance of real world objects. Thus, a great deal of work in computer graphics has focused on geometric aspects of a surface, including geometry acquisition [17, 45, 47, 59], geometry simplification [35, 43, 67], geometry smoothing [63] and geometry compression [51, 6, 20]. However, there is a direct relationship between the topology and the geometry of a surface that cannot be ignored. Geometry simplification, which preserves the manifold property of a surface, will only have a limited affect on a high genus model. See Figure 1.4 for an example of how geometry simplification is affected by the topology of a model. Alternatively, many mathematicians and computational topologist are concerned with studying purely topological properties of a surface. This thesis takes a combined approach and identifies and localizes topological features within a surface by mixing topological and geometrical approaches.

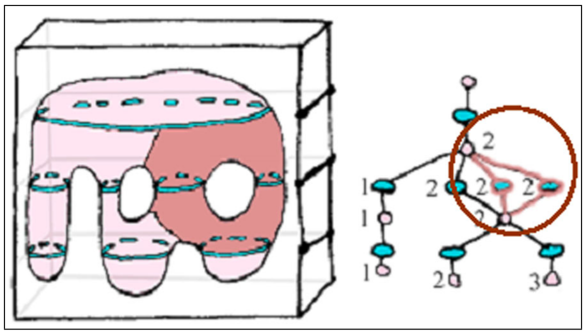
1.3 Introduction to the algorithms

This thesis presents computational topology algorithms that are designed for discrete surfaces and that simultaneously account for the geometry of a surface. Although there are easy methods to compute the genus of a surface, as stated above, the genus alone does not tell us enough. For most computer graphics applications we would like to know what the genus looks like. However, genus is just a scalar value with no associated geometric appearance. We can, however, isolate regions of the surface that are topologically interesting. The simplest topologically interesting, and perhaps most intuitive, regions to consider are regions with genus equal to one. Thus, instead of considering global topological invariants like the genus of a surface, we analyze the topology of the surface in terms of local handles, regions with genus equal to one. Handles give us a way to talk about the shape or topology of an object, while still considering the geometry and embedding of the surface. Toward this end, we introduce algorithms:

- to localize regions of topological interest, (regions with $g = 1$, *handles*) within a surface,
- to measure the geometric extent of those handles,
- to re-sample those handles and either retain or simplify the topology in the re-sampled surface.

1.3.1 Localizing and coding topology

It is feasible to compute the genus of the surface using the Euler characteristic. However, genus is a global property and we want to be able to isolate regions of topological interest within the surface. Thus, we present a robust and efficient algorithm to localize regions of genus one – or handles – within the surface. We prove that our method for traversing the surface to identify handles will identify *all* handles. We discuss our methods for identifying topology in Chapter 4. The handles of a surface are coded in an *augmented Reeb graph* which allows for easy identification and isolation within the surface. We discuss Reeb graphs in Chapter 2 and present an augmented Reeb graph in Chapter 4.



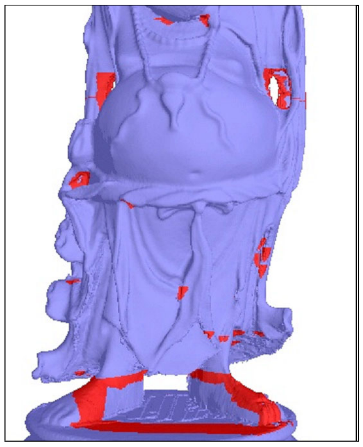


Figure 1.5: *Localizing a handle*
Illustration of localizing handles in a surface. On the left is a figure of an augmented Reeb graph used to identify handles and the corresponding handle on the surface. On the right is a close-up image of some of 104 handles on the Buddha statue highlighted in red.

1.3.2 Measuring feature size

Depending on the application, we may need to determine the geometric extent of a handle relative to the rest of the surface geometry. We present an algorithm to measure the handle size with the specific focus of considering the smallest extent of the handle. Our algorithm will compute two locally minimal-length transverse non-separating cycles for each handle. These cycles are provably, the discrete minimal-length cycles for each local handle. Combined, these cycles give a measure of the geometric extent of the handle, with particularly regard to considering minimal geometric changes to the surface to alter the topology. This measure is a very natural and intuitive measure for the size of a handle. We present the algorithm to compute locally minimal-length non-separating cycles in Chapter 4.

1.3.3 Re-sampling topology: preserving genus and simplifying genus

Since complex geometric models can simultaneously have topology that is inherent to the model and excess topology, we present algorithms to re-sample the topology of a surface. Handles which need to be preserved during remeshing, can be re-sampled to preserve the original topology of a model. The re-sampling can reconstruct the handle with fewer geometric samples if desired. In addition, since acquired data can easily have topological artifacts which need to be removed, we present algorithms to reduce the topological com-

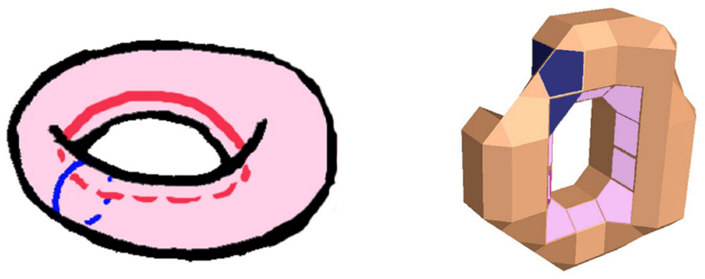


Figure 1.6: *Example of non-separating cycles*
Two examples of the cycles used to measure the topological extent of a handle. The figure on the left shows two possible non-separating cycles on a torus, while the figure on the right shows two actual discrete non-separating cycles found on a geometric model of a torus.

plexity of a surface. We have developed methods for topology simplification for both the mesh setting and the volume setting, each tuned for its respective setting.

These algorithms can be used together to build up applications which improve the usability and accuracy of geometric models. We discuss specific applications of these methods next.

1.4 Brief discussion of the applications

1.4.1 Isosurface topology simplification

Although many geometric models have been acquired at very high sampling rates with attention paid to geometric accuracy, small errors in the acquisition process lead to geometric noise, which in turn can lead to topological artifacts. Geometric models acquired from real world data, for example from laser range scanners or even MRI and CT scanners, commonly have topological artifacts. This high resolution volume data often cannot fit entirely into the main memory of most machines. We present an out-of-core method to remove topological artifacts directly from volume data. Rather than attempting to repair the defects on a mesh already extracted from the volume [40], our approach operates on the volume representation directly, as this offers advantages of efficiency and robustness. Our algorithm identifies topology in the volume through the application of techniques associated with Morse theory [64], discussed in detail in Chapter 2. Building on related approaches [9, 74, 78], the algorithm tracks changes in a wavefront induced by a discrete height function to identify the topology of the surface. The topology is coded in an augmented Reeb graph, where cycles in the Reeb graph correspond to handles. In order to measure the size of handles on the surface, we examine them one by one and consider cutting this region along a *non-separating cycle*. By subsequently pinching each of the two open boundaries of such a cycle to a point, the genus of the handle is reduced to that of a sphere ($g = 0$). For every handle of the surface, our approach computes a discrete approximation of the non-separating cycle with locally minimal-length (see Figure 1.6 for an example). Using the length of this cycle as a measure of the size of the handle, we choose either to retain the handle or remove it. Chapters 4 and 5 present the details of these algorithms and their applications.

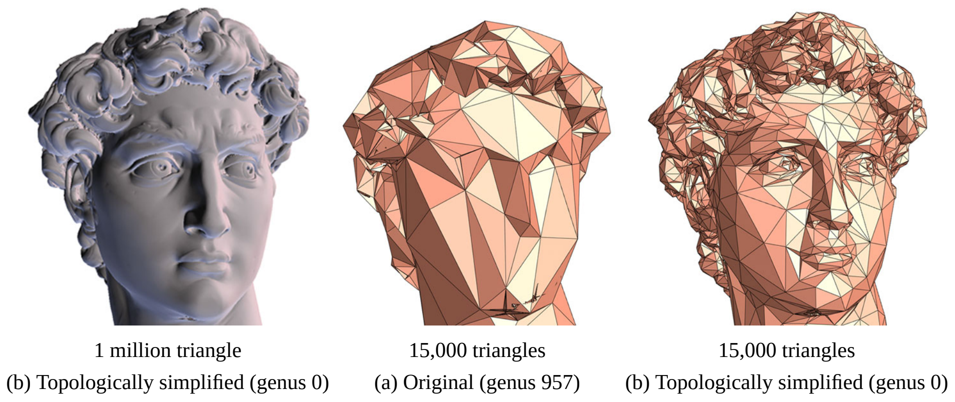

1 million triangle
(b) Topologically simplified (genus 0)
15,000 triangles
(a) Original (genus 957)
15,000 triangles
(b) Topologically simplified (genus 0)

Figure 1.7: *Progressive mesh of the David head*
Comparison of progressive meshes of the David model before and after topology simplification. In the center, many triangles are wasted representing invisible topological artifacts. The figure on the left shows that the fine detailed geometry of the model is not altered due to topology simplification.

1.4.2 Mesh topology simplification/Topological noise removal

Although many acquired surfaces are represented as volume data, some are stored as meshes which can also have topological artifacts. We present two algorithms to simplify the topology of triangle meshes. The first repeatedly searches over the surface in small local areas and removes any handles found within a small ϵ-radius. The second algorithm is a global algorithm with a tighter measure of the size of a handle. Both robustly remove topological artifacts from triangle meshes.

1.4.3 Semi-regular mesh extraction from volume data

The author's Masters research on the extraction of semi-regular meshes from volume data, is an example of an application which reconstructs a mesh with the same global topology as an original surface. This application is not a contribution to this thesis, but is included here due to its relevance as an application of computational topology algorithms to discrete 2-manifolds.

Isosurface extraction is a fundamental technique of geometric modeling and one of the most useful tools for visualizing volume data. The predominant algorithm for isosurface extraction, Marching Cubes (MC) [60], computes a local triangulation within each voxel of the volume containing the surface, resulting in a uniform resolution mesh. Often much smaller meshes adequately describe the surface since MC meshes tend to oversample the isosurface, encumbering downstream applications, (*e.g.*, rendering, denoising, finite element simulations, and network transmission). We present an alternative method for the *direct* extraction of an adaptively sampled multi-resolution isosurface mesh with good aspect ratio triangles. The multi-resolution

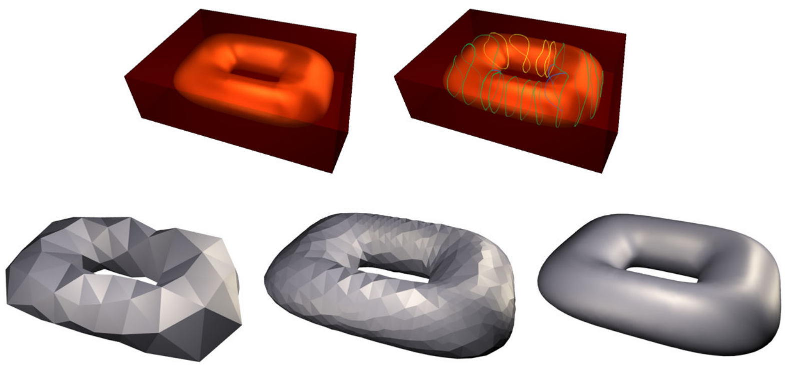


Figure 1.8: *Overview of semi-regular mesh extraction*
Overview of the semi-regular mesh extraction algorithm. Given a volume and a particular isovalue of interest (top-left), a set of topologically faithful rings is constructed (top right). Stitching them together creates the coarsest level mesh for the solver (bottom left). Adaptive refinement constructs a better and better fit with a mesh having semi-regular (subdivision) connectivity and an explicit multi-resolution structure (bottom).

structure is based on adaptive *semi-regular* meshes, well known from the subdivision setting [80]. In order to extract a semi-regular mesh, we need to start with a coarse base mesh with the same global topology as the desired isosurface. This requires methods to localize where the desired isosurface has handles in order to correctly reconstruct a surface with the same genus and similar geometric shape as the desired isosurface. After the initial coarse mesh is extracted, the algorithm uses an iterative solver to subdivide the coarse mesh and reposition the vertices to match the desired isosurface.

1.5 Contributions

The contributions of this thesis are the following:

A robust and efficient method to localize and isolate handles for discrete 2-manifolds. We propose a method where handles are efficiently identified through methods tuned to the discrete setting. The handle identification traversal of the surface is varied for efficiency while guaranteeing that all handles are located. We present a traversal method with a complexity of $O(n \log n)$, with proof that our traversal methods will detect *all* handles during the traversal. Handles can subsequently be efficiently identified during the traversal of the surface as cycles in the augmented Reeb graph as it is incrementally constructed (see next paragraph). Section 4.2.1.1 of Chapter 4 presents these methods in detail and the associated combinatorial proof.

A method to robustly represent the topology of the surface with an *augmented Reeb graph*. We present a method to construct an augmented Reeb graph which stores additional geometric information about the surface to facilitate isolating handles. We present a method to construct the augmented Reeb graph, which guarantees that for each interval of the traversal, each *ribbon* has genus equal to zero. In addition, we guarantee that for each interval, the number of cycles in the augmented Reeb graph matches the genus of the surface traversed thus far. Geometric properties of the surface are encoded in the augmented Reeb graph which allows geometrically succinct handles to be isolated within the original surface. Section 4.2.2.2 of Chapter 4 presents the augmented Reeb graph and the methods used to guarantee consistency between the number of handles and the number of cycles in the graph.

A method to find two locally minimal-length non-separating cycles for each handle. This thesis introduces a simple measure of handle size to be the length of two transverse non-separating cycles. The locally minimal-length non-separating cycles are detected efficiently for handles of the surface. We present a proof that we find two discrete locally minimal-length non-separating cycles with a complexity comparable to related approaches. See Section 4.3.2 in Chapter 4 for more details.

A simple method to simplify the topology for volume data and triangle meshes which preserves the local geometry as much as possible. Cutting the surface along the locally minimal-length non-separating cycle will reduce the genus of the model while retaining as much of the fine geometrical detail as possible. By using the smaller of the two non-separating cycles for each handle, the topology of the surface is only modified in a small local region. This targeted approach to modifying the topology preserves the fine geometry of the surface as much as possible. We propose a simple method to simplify the topology of triangle meshes and isosurfaces. Refer to Section 4.4.1 in Chapter 4 for more details. In particular, for isosurface topology simplification, to remove a handle, we alter the scalar values of the volume, thus indirectly modifying the isosurface. Since isosurfaces are always manifold, operating on the volume is robust. Also, by operating on the volume directly, we avoid computing an expensive triangle mesh and never compute or store floating point values to represent the geometric position of the vertices of the surface. Since our algorithm creates a topologically clean volume, this volume can then be used for surface extraction or other applications [48] that depend on a topologically accurate volumetric representation.

An out-of-core method for topology simplification for volume data. Complex 3D models are represented by large volumes that may not fit entirely in main memory. The model in Figure 1.7 is from a $885 \times 709 \times 736$ grid, and much larger models now exist [59]. The isosurface topology simplification algorithm is applied to such volumes using out-of-core methods. The algorithm employs a sweep method to read the volume in planar slices, so the data access pattern is highly regular. We encode surface topology as the sweep progresses using an augmented Reeb graph, requiring only a few slices in memory at any given time. For some large handles, previous slices may need to be reloaded to perform simplification. However, simplification can be performed on small segments of the volume one at a time, resulting in a purely out-of-core algorithm. The details of this out-of-core method are presented in Section 5.2.1 of Chapter 5.

1.6 Acquired Data

Throughout this thesis various representations of discrete 2-manifolds are used. We discuss the details of the particular surface representations we use in Chapter 2. In particular, we demonstrate the algorithms presented in this thesis on triangle meshes and isosurfaces represented in regular scalar volumes. All of this data was acquired from real world 3D objects using either Cyberware 3030MS optical triangulation scanner, Stanford Large Stature Scanner or an MRI or CT scanner. The models of the dragon, Buddha, and David head came from the Stanford Computer Graphics Laboratory Scanning Repository and Digital Michelangelo library [59, 18]. Ten MRI scans of the brain came from the Harvard Medical School (in particular from: Drs. Kikinis, M. Shenton, R. McCarley, and F. Jolesz) [52]. The feline, mouse fetus and torus volumes are from the Caltech Multi-Res Modeling group and the Caltech Graphics Groups. Any other data is cited appropriately in the figure captions.

1.7 Summary

The rest of this thesis is structured as follows:

- The **Background** chapter presents a short summary of material relevant to geometric modeling and computational topology. This chapter includes definitions of terms and some historical context to the study of topology.
- The **Related Work** chapter presents how other researchers have addressed similar issues of computational topology.
- The **Algorithms** chapter details the algorithms we introduce for locating, measuring and resampling handles for discrete surface representations.
- The **Applications** chapter presents in detail, three applications of computational topology algorithms to computer graphics.
- The **Conclusions** chapter summarizes the contributions of this thesis and discusses future extensions.

Chapter 2

Background

This thesis addresses the application of computational topology algorithms to discrete surfaces. The following background material is presented to provide context for this work. First, a discussion of surfaces and surface attributes is presented along with a discussion of the various representations of discrete 2-manifolds. Next, we present some historical and mathematical context for topology. We then present some useful methods from topology which can be applied to surfaces. Finally, some useful definitions for our specific setting are introduced.

2.1 Surfaces

Surfaces have been studied by mathematicians for centuries. Typically mathematicians conceive of surfaces as continuous, for example a surface may be defined as a continuous function of two variables. Each surface has a variety of attributes. For example, the surfaces typically used in computer graphics are *oriented 2-manifolds with or without boundary*. A *2-manifold* is a surface where the local area around every point on the surface is Euclidean, meaning, around each point the surface appears to be nearly "flat." The world we live on is an excellent example of a 2-manifold. Manifolds are a preferable surface representation because the surface can be divided into regions, called charts. Chartification allows 2-manifolds embedded in 3D to be flattened into a two dimensional domain (through *parameterization*). Surfaces used in computer graphics are typically *oriented*, this refers to the fact that the surface has two sides. For example, a sphere has two sides, while a Mobius strip has only one side. Another attribute of surfaces is whether the surface is *closed* or with *boundary*. This refers to the number of open boundary components of a surface. For example, an egg is closed but once it has been cracked open, it is has boundaries.

2.1.1 Simplicial complexes

Mathematically, surfaces are often conceived of as continuous and smooth, *i.e.*, one that has a sufficient number of partial derivatives. Smooth often refers to a surface with infinitely many partial derivatives but usually just two partials are sufficient. In computer graphics we operate in a discrete setting, where only a finite number of samples are used to represent a surface. These surfaces are often continuous, but are only piece-wise linear and are represented only by a discrete set of points which are connected together as triangles or polygons. Throughout this thesis, we refer to *discrete 2-manifolds*, all such surfaces are assumed to be piece-wise linear unless otherwise specified.

A common way to conceive of a discrete surface representation is as a simplicial complex. A simplicial complex is built up of simplices. Formally, a *simplex* is the convex hull of a set of affine independent points, such as, a vertex, an edge and a triangle. A vertex is said to be a simplex of dimension zero, while an edge is a simplex of dimension one and a triangle is a simplex of dimension two. A *face* is defined as a subset of a simplex, which is itself a simplex. For example, a vertex that is an endpoint of an edge is considered a face of that edge. Formally, a simplicial complex is a finite collection K of simplices, such that:

- Every face of a simplex K is in K
- The intersection of any simplices in K is a face of both simplices. For example, two triangles must meet exactly along their shared edge.

Simplicial complexes give us a formal way to talk about a discrete representation of a manifold. When talking about simplicial complexes, we may need to talk about specific neighborhoods on the complex. The *star* of a vertex v, $St(v)$ denotes the neighborhood around a vertex v. Specifically, $St(v)$ includes all of the simplices that include v as a vertex.

2.1.2 Triangle meshes

Triangle meshes are commonly used to represent discrete surfaces and are an example of a simplicial complex. A triangle mesh can be defined as $\mathcal{M} = (K, \mathbf{x})$ where $K = V \cup E \cup F$ is an abstract simplicial complex representing the connectivity of the mesh (V, E, and F are sets of vertices, edges, and faces, correspondingly), and $\mathbf{x} : \mathcal{V} \rightarrow \mathbb{R}^3$ is the coordinate function that gives the coordinates of every vertex of V. Triangle meshes are used so frequently to represent surfaces in the discrete domain that most computer graphics hardware is optimized to render triangles.

Multiresolution One particular triangle mesh representation deserves special mention: semi-regular meshes. Semi-regular meshes are a multi-resolution mesh representation which is well known from the subdivision setting [80]. A semi-regular mesh consists of a coarsest level triangle mesh which is recursively refined through quadrisection. The resulting meshes have regular (valence 6) vertices almost everywhere. Adaptivity is achieved through terminating the recursion appropriately and enforcing a restriction criterion (triangles sharing an edge must be off by no more than one level of refinement). Conforming edges are used to prevent T-vertices. Because of their special structure such meshes enjoy many benefits including efficient compression [51] and editing [81].

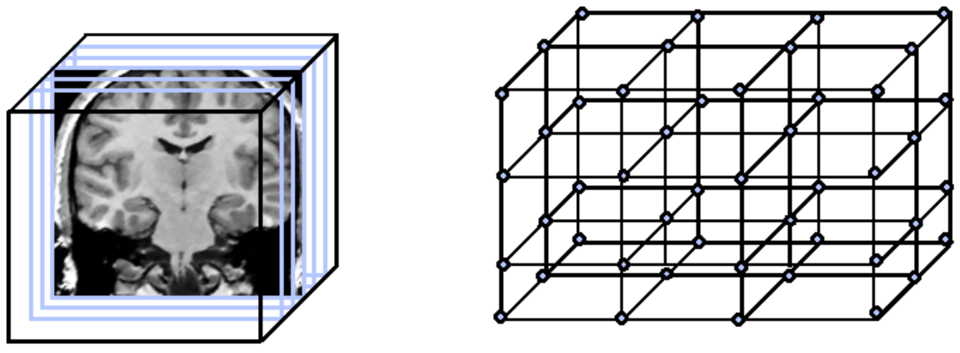


Figure 2.1: *Scalar volume*
Two representations of a scalar volume. On the left is a stack of images from an MRI scan. On the right is a grid of scalar values.

2.1.3 Scalar volumes

Another common discrete surface representation is an isosurface in a scalar volume. A scalar volume consist of a regularly sampled 3D grid of scalar values. A grid *cube*, also called a *voxel*, is bounded by 8 grid data points, (see Figure 2.1 and Figure 2.1.3). Scalar volumes are typically acquired from real world data from various sources, such as, magnetic resonance imaging (MRI) and computed Tomography (CT) imaging. These popular imaging techniques are used in a variety of medical and scientific applications to view and analyze three dimensional structures. These imaging systems typically create a stack of registered images, (see Figure 2.1). Each pixel of each image is a scalar node in the 3D grid. Scalar volumes can also be produced from simulations like computational fluid dynamics (CFD).

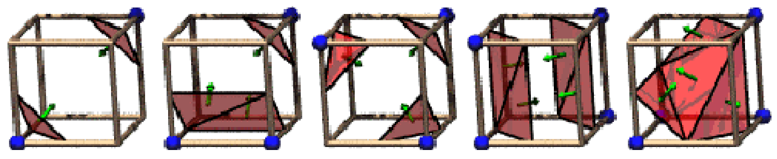


Figure 2.2: *Voxel*
Voxels of a volume with various surfel configurations.

Signed Distance Volumes Another type of volume data that is commonly used is signed distance volumes. A distance volume is a volume dataset that stores the shortest distance to the surface at the vertices of each voxel. Whether a vertex is inside or outside of the surface is encoded in the sign of the distance. Signed distance volumes are commonly the output of a surface reconstruction algorithm from acquired data. For example, laser range scanners achieve full coverage of complex objects by acquiring and merging multiple scans [17, 45, 59]. Thus, acquired models are frequently represented and stored as a signed distance volume.

Isosurfaces Although scalar volumes allow for the visualization of 3D structures, typically a user is interested in manipulating a surface instead of the entire volume of data. For example, a doctor may want to only visualize a specific layer of skin or tissue from a medical MRI scan. An *isosurface* is the surface defined by a specific scalar value. Specifically, consider an implied surface intersected by the volume grid. This intersection and the entire grid can be represented by tuples $(i, F(i))$, where i is a point in 3D space and $F(i)$ is the scalar value of the volume at that point in space. Without loss of generality we assume that the surface we are interested in is the zero isocontour of the volume. The surface will be pierced by the edges and faces of the grid, creating a collection of patches each of which we denote as a *surfel*, for surface element. Within each cube of the grid, an isosurface generation algorithm (*e.g.*, [54] or [60]) defines the set of surfels [78], (see Figure 2.1.3). Each cube may have up to 4 surfels. The surfels from all cubes together form a discrete representation of the isosurface. We use the connectivity rules of Lachaud [54] due to the fact that they produce a closed oriented surface without singularities or self-intersections [54]. Lachaud's table has proven properties by restricting data to well defined interior and exteriors, *i.e.*, for a scalar function $F(i)$, the interior is defined as $F(i) < 0$, while exterior is defined as $F(i) \geq 0$. This is similar to a standard general position argument, and creates a well defined isosurface, *i.e.*, the surface is perturbed away from the volume grid nodes. Isosurfaces are a fundamental data type for geometric modeling. Such a surface can either be extracted from the volume and manipulated as a triangle mesh, or the data structure of the volume itself can be manipulated, affecting the isosurface within.

2.1.4 Surfaces as graphs

Discrete surfaces, either triangle meshes or isosurfaces, can be viewed as a graph. A graph is a binary relation on a set of vertices, where the vertices are either connected or not. The primal graph of a surface is the graph with nodes for each vertex and graph edges that exactly follow the connectivity relationship of the vertices along the triangle or surfel edges. The *dual* graph is the graph constructed by replacing each face of the surface with a graph node and then connecting these nodes with graph edges for each edge between adjacent faces in the primal graph. Conceiving of the surface as a graph is a useful metaphor, especially as there are a variety of useful and applicable graph algorithms, such as Dijkstra's shortest path algorithm. This algorithm uses a breadth first traversal of the graph to compute the shortest path over the entire graph starting with any graph node, v. It is useful to apply Dijkstra's shortest path algorithm to surfaces to compute discrete distances between faces of the mesh. See, for example, [16] for details of the Dijkstra's algorithm.

2.2 Topology

As suggested by the above definition of a triangle mesh, surfaces are predominantly thought of in terms of their geometry, (*e.g.*, the 3D positions of the mesh vertices) and their connectivity, (*e.g.*, the simplicial complex that determines the relationship of the vertices). However, these are not the only qualities of a surface and in this thesis we focus on another aspect of surfaces, their topology.

In general, topology studies the *shape* of curves, surfaces and *spaces* in general. An example of a space is the real line (*i.e.*, the real numbers). Topology is the study of the properties of surfaces that are preserved under deformations other than tearing. In other words, topology considers the aspects of a curve or surface that do not depend on the geometry of the curve or surface. For example, a circle is topologically equivalent to an ellipse even though their geometric shape is different.

2.2.1 Historical and mathematical context

Topology has been studied by mathematicians for hundreds of years with many of the initial contributions made by Leonhard Euler in the 1700s. This initial work freed mathematicians to consider the properties of curves or surfaces that did not depend on a distance metric (*e.g.* properties independent of the measurement of geometry). The first publication about topology is considered to be Euler's publication about the Königsberg bridge problem. In summary this paper showed that 'a graph has a path traversing each edge exactly once if two vertices have odd degrees.' This work showed a property of graphs that is true regardless of the geometry of the graph. Soon after Euler proposed his formula for polyhedra: $V - E + F = 2$ (where V refers to the number of vertices of the polyhedra, E refers to the number of edges and F refers to the number of faces). Although this initial formula was incomplete, it set the stage for mathematicians to explore properties outside of distance measurements for curves and surfaces. Euler's formula was only later corrected (in 1813) by Simon Lhuilier to account for genus: $V - E + F = 2 - 2G$ (where G refers to the genus). Although this initial work seems in some ways simple, at the time it signaled a significant shift in mathematicians thinking. Specifically, this work presented the first work on invariant properties of graphs or polyhedra independent of the geometry.

Topology evolved quickly from its early beginnings and was explored from a variety of angles, resulting in a number of branches of topology. Much of the formal structure used in topology was formalized in 1895 by Jules Poincaré. Poincaré introduced the idea of homology groups and Betti numbers to quantify topological attributes of an object. Formally, the nth Betti number is the rank of the nth homology group. A more intuitive explanation of the Betti numbers is that, for example, β_0 is the number of components of a complex and β_1 is loosely the maximum number of cycles that do not separate a surface into two or more pieces, (*e.g.*, the two generator loops of a torus). This work forms the foundation of algebraic topology which uses algebraic structures like groups and rings to study hole structures in spaces. Differential topology take a slightly different approach and considers the non-metrical aspects of smooth manifolds. Topology has been studied from various mathematical angles and applied to various types of data ranging from point sets and curves, to manifolds. Regardless of the methods used to study topology the same fundamental thrust is always present which is the study of aspects of spaces, outside of geometric metrics. In other words, topology separates global shape properties from local geometric properties.

Much of the work done in topology has been applied to continuous smooth manifolds. Recently however, computational topology has been of growing interest due to the numerous applications of topology to the piece-wise linear discrete setting. Just as computational geometry has contributed to the usability of geometric models, computational topology offers a wealth of interesting problems to increase the usability of discrete models. Computational topology uses theory from topology to develop computational algorithms to analyze the shape of objects or spaces. This is a relatively new area of research, and only a few researchers have focused on piece-wise linear discrete surfaces.

2.2.2 Morse theory

One of the key tools used to study the topology of spaces is Morse theory. Morse theory is the study of the relationship between functions on a space and the shape of the space. Although Morse theory can be applied to spaces of infinite dimension, we are particularly interested in the application of Morse theory to 2-manifolds. Building on variational calculus, Morse theory draws a relationship between the critical points of a smooth function defined on a smooth manifold to the global topology of the manifold [64, 62].

To better understand Morse theory, first let us briefly consider the critical points of functions of one variable, $f(x) = y$. We can examine the maxima, minima, and inflection points of such a function by examining its critical points. Namely, the critical points of such a function satisfy: $f'(x_0) = 0$. A critical point is said to be *non-degenerate* if its second derivative is not equal to zero, ($f''(x_0) \neq 0$). Likewise for functions of two variables, such as $f(x, y) = z$, critical points are defined as the points where the gradient is equal to zero ($\nabla f = 0$, *i.e.*, the partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ are both equal to zero). A non-degenerate critical point for a function with two variables is one where the determinant of the Hessian of f is not zero. The Hessian is the matrix of second derivatives with the determinant of the Hessian being: $detHf(x, y) = \partial^2 f/\partial x^2 \times \partial^2 f/\partial y^2 - (\partial^2 f/\partial x \partial y)^2$.

Now, given a smooth manifold, M, a map, $f : M \rightarrow \mathbb{R}$, which assigns a real number to each point p of M, is a function f on M. Such a function is a *Morse function* if every critical point of the function $f : M \rightarrow \mathbb{R}$ on M is non-degenerate and isolated. The critical points are isolated if each critical point maps to unique real number.

Consider the case of a function of two variables, Morse theory shows that near a non-degenerate critical point the function can only look fairly simple. In fact, by choosing appropriate local coordinates, a function of two variable will have one of three possible shapes around a non-degenerate critical point:

- $f = x^2 + y^2 + c$
- $f = x^2 - y^2 + c$
- $f = -x^2 - y^2 + c$

where c is a constant. See Figure 2.3 for an illustration of these functions. By decomposing a smooth manifold into these “shapes”, the global shape and topology of the manifold is revealed. Morse theory also presents methods to classify which type of critical point a given point corresponds to. Specifically, by examining the number of negative eigenvalues of the Hessian, the critical point can be *indexed*. That is, a minimum has zero negative eigenvalues, a saddle point has one, and a maximum has two negative eigenvalues. This analysis corresponds to the fact that a minimum has no downhill sides, while an isolated saddle point has two downhill sides, one parallel to the direction of the eigenvector associated with the negative eigenvalue and one anti-parallel. A maximum has downhill sides associated with both directions of both eigenvectors.

A classic result from Morse theory is that given a closed surface M and a Morse function, $f : M \to \mathbb{R}$. If this function has only two non-degenerate critical points then M is topologically equivalent to a sphere. For example, a typical Morse function is a height function, and if we consider such a function and a sphere, we see that there are two critical points to the function, corresponding to the maximum and minimum at the north and south pole of the sphere, (see Figure 2.4). More precisely, the number of critical points is equivalent to the Euler characteristic of the surface: $\chi = \#minima - \#saddlepoints + \#maxima$. Another essential result from Morse theory shows that between the critical points the topology of the manifold is guaranteed not to change (called the *deformation lemma*).

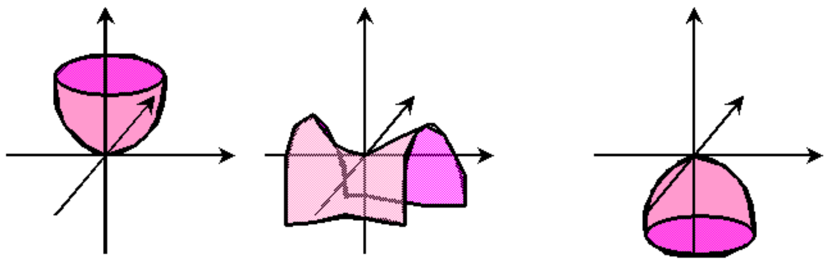


Figure 2.3: *Critical points for a function of two variables*
Representations of the possible shapes near critical points for a function of two variables. On the left is $f = x^2 + y^2 + c$, the middle figure represents $f = x^2 - y^2 + c$, and the figure on the right is $f = -x^2 - y^2 + c$.

To further illustrate the relationship of critical points and the global topology of a surface, consider the following geometric interpretation. Given a Morse function, $f : M \to \mathbb{R}$ which is a height function, that height function defines parallel planes. Now imagine a torus standing on its end, (see Figure 2.4). If we consider each of the tangent planes of the torus, the points with tangent planes that correspond exactly to the height planes will be the critical points of the Morse function. For the example of the torus, as we would expect there will be critical points corresponding to the maximum and minimum (at the north and south poles of the torus) and at the two saddle points of the handle.

There is further geometric interpretation of Morse theory for a 2-manifold correlating the tangent plane of the surface for each point and the planes defined by a height function. Specifically, we consider classifying critical points and trivial points (non-critical points). Similar to the method presented above, where we classify points based on their shape, let us consider analyzing the local shape of the surface, by looking at the relationship between a small circular neighborhood of each point on the surface and the height planes of

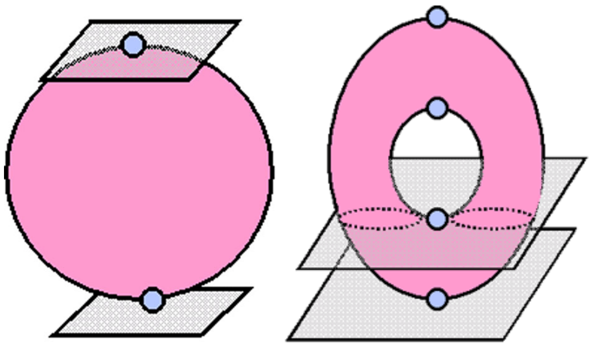


Figure 2.4: *Illustration of height planes on a sphere and torus*
On the left is a sphere and its two critical points, (and their associated tangent planes), at the north and south pole. On the right is a torus with showing height planes matching the tangent planes at critical points for a minimum and a saddle point.

a height function. For a regular (trivial) point q, the height plane is *not* the tangent plane. In fact the height plane divides the small circular neighborhood around q into two pieces. One piece lays above the height plane and the other below, (see Figure 2.5). Now, consider instead a maximum or minimum point and the small circular neighborhood around each of these points. The horizontal plane of the height function will not intersect either of these points nor their small circle neighborhood. Finally, consider a saddle point – for such a point, again consider the small circular neighborhood around the point and its relationship to the height plane. For a saddle point, its circle will be intersected in four places, dividing the region around the saddle point into four sections, (see Figure 2.5). These observation are important for understanding related methods in the discrete setting, discussed in Chapter 2.

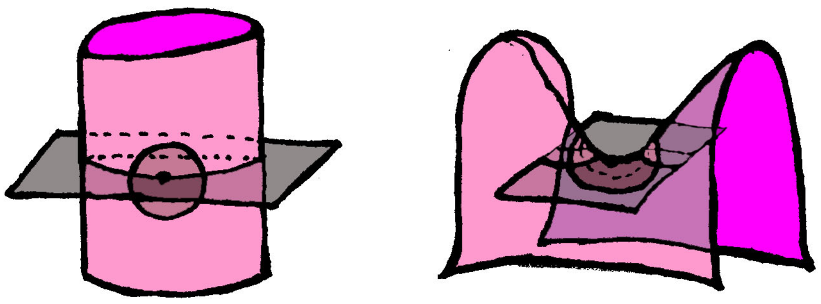


Figure 2.5: *Critical points and height planes*
On the left is a trivial point and a small circular neighborhood. Note that the height plane divides the circular neighborhood into two regions. On the right is a critical point, a saddle point, with a small circular neighborhood. For isolated saddle point the height plane divides the circular neighborhood into four regions.

Morse theory is a powerful tool for revealing the global topology of smooth 2-manifolds, however, this thesis is directed to the piece-wise linear discrete setting. We do not start with smooth manifolds, nor is it always easy to construct a smooth Morse function over the surface. There has been recent work on extending Morse theory for piece-wise linear functions on 2-manifolds. Work presented by Ulrike Axen [9] uses a wavefront traversal to generate a Morse function for triangulated meshes and work by Edelsbrunner *et al.* [26] extend Morse theory to piece-wise linear functions on 2-manifolds. This work is discussed in more detail in Section 3. The work presented in this thesis builds from Morse theory with some modifications to tune our methods to the discrete setting. We do not find exact critical points on a surface, but instead find *critical levels*. We discuss these methods in more detail in Section 4.

2.2.3 Reeb graphs

A Reeb graph is graphical representation of the connectivity of a surface between critical points. Reeb graphs were initially used to represent the skeleton of a manifold [68]. Initially, a Reeb graph would have a node for every critical point in the surface and edges (or arcs) between the nodes that represent the connected components of the surface between the critical points. However, more commonly Reeb graphs have been used to represent the relationship of the level sets for a surface [56, 75, 14, 53]. See Section 4.2.1.2 for a definition of level sets. Specifically, given a scalar function f defined on the surface, a Reeb graph tracks the connected components of the pre-image of the function. For instance, if the scalar function is a height function that returns discrete z intervals, its pre-image is the intersection of the surface with z planes, and the connected components consist of closed planar contours (see Figure 2.2.3). The Reeb graph tracks how these contours split and merge as z varies and is often used to analyze surface topology. Reeb graphs have been used to code surface topology by a number of researchers, which we will discuss in detail in Section 3. This thesis uses an *augmented Reeb Graph* to represent the topology of surfaces as discussed in Section 4.

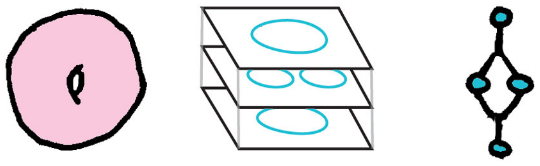


Figure 2.6: *Torus and example Reeb graph*
An example of a torus, planar contours defined by a height function and the associated Reeb graph.

2.2.4 Some useful definitions

In addition to the above background information, there are a few specific definitions that will be useful for understanding the work presented in this thesis.

Definitions

- A *simple cycle* is a closed loop that does not intersect itself.
- A simple *non-separating cycle* is a cycle that when removed from the surface, leaves only a single connected component.
- The *genus* of a surface is the largest number of disjoint non-separating cycles that do not intersect one another.
- A *handle* corresponds to a surface region with genus equal to one. The genus of a region with boundaries can be computed from the Euler characteristic, χ, of this region. Where $\chi = v - e + f$.

Surface topology The topology of a surface is characterized by its genus, its orientability, the number of its connected components, and the number of its boundary components [61]. The surfaces addressed in this thesis are oriented and closed. Specifically all, isosurfaces have the property that they are always orientable, and never have boundaries (if one pads all sides of the volume with “outside” scalar values). The algorithms presented in this thesis deal with multiple disconnected components by processing them independently.

Chapter 3

Related Work

Various researchers have addressed problems related to analyzing, measuring and altering the topology of discrete 2-manifolds. We consider relevant related work here.

3.1 Identifying topology

One of the algorithms presented in this thesis is a method to identify and isolate handles in a discrete 2-manifold. We first consider methods related to identifying the topology of a surface.

3.1.1 Morse theory

The algorithm presented in this thesis to identify the topology of a surface is related to Morse theory. As presented in Chapter 2, Morse theory examines the relationship of the critical points of a smooth function defined on a smooth manifold to the connectivity of the manifold [64]. The main challenges of applying Morse theory to discrete 2-manifolds is that it was developed for smooth manifolds and it produces many more critical points then handles. We propose an alternative approach which is tuned to the discrete setting which can efficiently isolate regions that correspond to the pairs of saddle points that correspond to a handle. Other researchers have addressed the challenges of applying Morse theory to the discrete setting differently. For example, Axen [10] and Edelsbrunner *et al.* [26] have addressed extending Morse theory and critical point analysis to a piece-wise linear discrete setting. We discuss how these methods relate to the goals and algorithms presented in this thesis.

Morse theory has been used for various applications to analyze the topology of discrete surfaces. Most notably, work by Axen [10, 9] and by Edelsbrunner *et al.* [26] address the challenge of finding exact critical points for piece-wise linear 2-manifolds. The work by Axen is closely related to the work presented in this thesis. Both methods examine the behavior of a wavefront traversal of the surface in order to detect the topology of the surface. The main difference between the work of Axen and the algorithm to identify topology presented in this thesis is that the work by Axen identifies critical points, while our method identifies critical levels. This difference has consequences for how the two methods deal with degenerate or non-isolated critical points in the piece-wise linear setting. In addition, our algorithm has advantages for later isolating handles in the surface, (discussed in Chapter 4).

In the work by Axen, a discrete distances is propagated to all vertices in a triangulated manifold starting from an arbitrary vertex, v. The distance between two vertices is defined as the smallest number of edges between those vertices, *i.e.*, a graph geodesic with all edges having weight equal to one. Furthermore, Axen defines that the distance of any simplex with dimension greater then zero, *i.e.*, an edge or triangle, is the minimum distance of any of the vertices of that simplex. Recall from Section 2.2.2 that the geometric interpretation of Morse theory for a 2-manifold analyzes critical points based on the correspondence of the local tangent planes of the surface with the planes defined by a Morse function. In general, the algorithm proposed by Axen examines the *star* of every vertex, v, $St(v)$. Refer to Section 2.1.1 for the definition of the star of a vertex. The algorithm identifies critical points on the surface by correlating the configuration of the distance function in the star of v, to the possible configurations of critical points. The analysis divides the simplices of the triangulated manifold into sets of wavefront simplices, $W(i)$, and the sets of simplices $S(i)$ that do not belong to $W(i)$. $W(i)$ includes any simplex α, where all vertices of α have distance i and α is a face of a simplex that has distance equal to $i-1$. For example, an edge with both vertices of distance i which is a face of a triangle which has a vertex of distance $i-1$ is in $W(i)$. All vertices of distance i are in the set $W(i)$, but an edge between a vertex of distance $i-1$ and i is in $S(i-1)$.

In this setting, a normal (*trivial*) vertex, $v \in W(i)$ is not a critical point if the star of v is divided into two pieces by $W(i)$, one piece with simplices in the set $S(i)$ and the other piece with simplices in the set $S(i-1)$, (see Figure 3.1). Likewise, an isolated saddle point for a vertex, $v \in W(i)$ will have a star which is divided into four regions alternating between regions of $S(i-1)$ and $S(i)$. Again, see Figure 3.1. There is only one local minimum for the wavefront traversal which is the start vertex v, where the star of v is comprised of simplices all belonging to $S(0)$. Maxima are any vertices, $v \in W(i)$ with a star of simplices only in $S(i-1)$.

The analysis presented in the initial algorithm by Axen [10] requires that the discrete distance function can be extended linearly to a continuous function. This requires that some faces of the surface be subdivided k times. This subdivision can cause the the data set to grow more than $((k+1)!)^k$ times. The subdivision is required as the analysis depends on extending the distance function to be continuous. For example the distance function can be extended across the simplices of distance $i-1$ to distance i. However, some simplices called *grounded simplices* do not permit such an extension. An example of a grounded simplex is the final face traversed by the wavefront, or an edge with both vertices of distance i which is not a face of a simplex of distance $i-1$. See Figure 3.2 for an example of one such grounded edge. By subdividing the faces of the triangle mesh these grounded simplices are resolved. Since subdivision is a costly solution, Axen later proposes a method to first identify critical points for each vertex on the normal wavefront [9]. After an initial pass over the surface to identify critical points which ignores grounded simplices, the modified algorithm performs a subsequent pass over the dual of the grounded simplices to detect critical points for these regions.

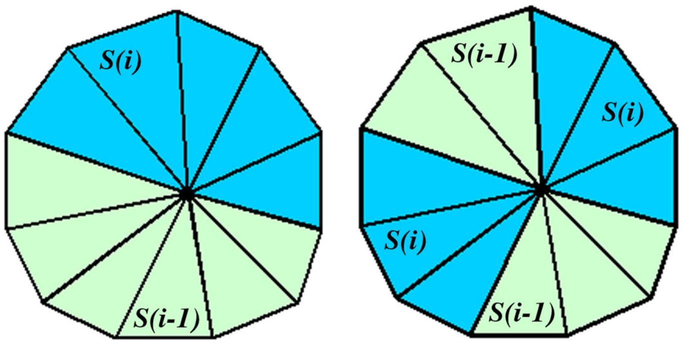


Figure 3.1: *The star of two different vertices*
Two different configurations of the star of two vertices $\in W(i)$. *On the left is a normal (trivial) vertex which is divided into two pieces. One piece is a subset of* $S(i)$ *and the other is a subset of* $S(i-1)$. *On the right is an isolated saddle point that has divided the star neighborhood into four pieces alternating between* $S(i)$ *and* $S(i-1)$.

The algorithm presented in this thesis to identify topology is related to this work by Axen [10, 9]. Our method relies on a wavefront traversal to detect the topology of the surface, however, the method does not analyze each vertex and its star, and instead analyzes the wavefront components to find *critical regions*. The *contours* identified using our algorithm are similar to the wavefronts identified by Axen. Likewise the *ribbons* are related to the intermediary subsets S. Our analysis correlates to methods used by Axen. For example, a trivial wavefront component (contour) is one which connects two ribbons, one above and one below. Every star of the vertices on this contour is divided into two pieces just like the trivial vertices in Axen's analysis. Our method is not affected by grounded simplices as we do not attempt to find exact critical points. In addition, grounded simplices are incorporated into the *ribbons* and correctly encoded in the augmented Reeb graph. See Chapter 4 for more details.

The work by Axen addresses non-isolated and degenerate critical points. For example, any saddle point that is not isolated will have a star neighborhood which is divided into more than four regions. In general degeneracies are handled by partitioning the degenerate sub-complexes into different wavefronts by perturbing the regular distance function with an ϵ offset of a local distance function. This is reminiscent of our approach for dealing with non-isolated degenerate critical points as presented in Section 4.2.1.5. Our approach locally modifies the distance function to guarantee that the critical points are isolated. The work by Axen has the same complexity as the algorithm proposed in this thesis. The prominent factor in the complexity of both algorithms is propagating distances over the surface in a breadth first traversal $O(n \log n)$, (where n is the number of vertices). Both algorithms use subsequent passes over the data on the order of $O(n)$ to compute relevant topological information.

In our work, we are only interested in identifying handles within a surface and we do not attempt to identify every critical point on the surface as Axen does. Instead, we are only interested in locating pairs of critical levels that correspond to a handle. Similar to the work of [9, 40, 78], we examine how the trace

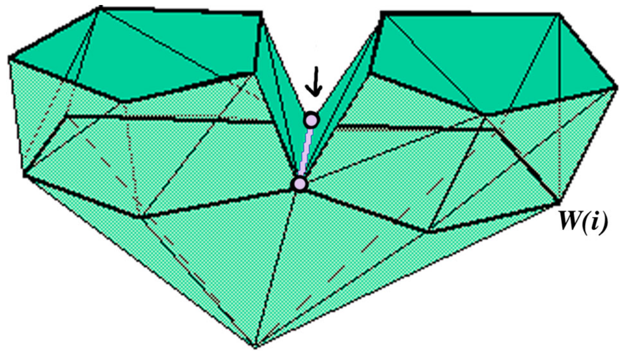


Figure 3.2: *A grounded simplex*
The purple edge e is a grounded simplex. Both of e's vertices are members of $W(i)$, however, e is not a face of a simplex of distance $i-1$, therefore e is not in $W(i)$. The edge e is $\in S(i)$, however it does not have one vertex of distance i and one of distance $i-1$ and therefore causes problems for extending the distance function to be continuous.

of a discrete wavefront, induced by a distance function or height function changes as it progresses over a 2-manifold. Regions where the wavefront splits and merges as it passes over the surface are related to the location of saddle points on the surface. By keeping track of the connected components of the surface and taking care that the traversal always captures critical levels of the surface we are able to identify handles with a discrete traversal and subsequently isolate handles in the surface. Chapter 4 gives details about our method for identifying and isolating handles.

Critical Points Critical point analysis has been used by alternative applications in computer graphics. For example, the work of Bajaj *et al.* [12] uses critical point analysis to constrain geometry simplification to be topology preserving. Work has been done by Stander *et al.* [76] on using critical points from Morse theory to guarantee the topology of the polygonization of an implicit surface. In addition, critical points have been used to analyze terrain data [77]. Our work is focused on identifying handles and does not isolate exact critical points in order to identify the relevant topology in the surface.

3.1.2 Reeb graphs

As discussed in Chapter 2, a Reeb graph can be used to represent the connectivity of a manifold between critical points. Reeb graphs were initially used to represent the skeleton of a manifold [68]. Initially, a Reeb graph would have a node for every critical point in the surface and edges (or arcs) between the nodes that represent the connected components of the surface between the critical points. However, more commonly Reeb graphs have been used to represent the relationship of the *level sets* for a surface [56, 75, 14, 53]. See Section 4.2.1.2 for a definition of level sets. Each node typically represents a contour of the level set and edges between the nodes represents the connected components of the surface between the contours. The level sets may be defined by a variety of functions, typically a height or distance function. Specifically, given a height function f, defined on the surface, a Reeb graph tracks the level sets of the surface, which are the connected components of the pre-image of the function. For instance, if a height function returns the z coordinate of a volume, its pre-image is the intersection of the surface with the z planes, and the connected components consist of closed planar contours (see Figure 4.8). See Section 4.2.1.2 for more details. The Reeb graph tracks how these contours split and merge as z varies.

Shinagawa *et al.* [75] use this framework for the reconstruction of surfaces from contours. In work done by Shinagawa [75], a Reeb graph is constructed from predetermined cross sectional contours only. In such a setting, a priori information about the topology of the initial shape is required to guarantee that the Reeb graph exactly matches the topology of the input shape. Typically Reeb graphs are constructed with a one to one correspondence between the nodes in the graph and the contours of the surface. A graph constructed with nodes for contours alone, can have degeneracies. We propose an *augmented* Reeb graph that has additional nodes to code the surface components between contours. See Section 4.2.1.1 for a more thorough discussion of this topic. The *augmented* Reeb graph can be constructed incrementally while sweeping across the surface guaranteeing that the topology of the graph always matches that of the surface processed thus far.

The work by Attene *et al.* [8] proposes an *Extended Reeb Graph* which is used to re-mesh a surface. This work builds on our initial work [78] and proposes a graph construction that can be down-sampled to re-mesh a surface. The method has interesting similarities to the algorithms proposed in this thesis as their topological analysis relies on identifying *critical regions* of the surface to construct their Extended Reeb graph. However, the work by Attene *et al.* uses a height function to construct contours for arbitrary polygonal meshes to identify the topology and reconstruct a re-mesh of the surface. This method does not build the Reeb graph directly from traversing the connectivity of the surface nor does it store ribbon information and thus encounters problems with degenerate critical points and with instabilities due to the sampling and the direction of the height function.

Reeb graphs are sometimes called contour trees and have alternatively been used to construct the medial axis of polyhedral objects [56], to compute seed sets for tracing isosurfaces [14, 53], to recognize shapes [44], and to re-mesh surfaces [78, 8]. For some of these applications, the cycles in the Reeb graph are not required to be in a one-to-one correspondence with handles as the applications use the Reeb graph as an *indicator* of the topological shape of an object.

Note that none of the related work discussed above subsequently isolates handles in the surface, once the topology has been identified. We have found that isolating handles is useful for measuring the geometric extent of topological features, and useful for simplifying topology with minimal changes to the global topology of a surface. In addition, isolating handles is useful for guaranteeing that a re-mesh correctly re-samples the original topology. We present more information about these methods in Chapter 4.

3.2 Cutting meshes open

Our approach shares common themes with work on cutting a surface into a single topological disk [55, 31, 37, 49, 15], as we use two locally shortest length non-separating cycles to measure and simplify each handle, if desired. Algorithms to cut a surface into a disk typically analyze the topology of the entire surface. Our problem is slightly different as we first localize handles in the surface using a Reeb graph and then measure each handle individually. We do not attempt to cut a genus g surface into a disk.

3.2.1 Cut graphs

Erickson and Har-Peled [31] address the task of optimally cutting a surface into a disk. They propose a greedy algorithm to compute a nearly minimal *cut-graph*. A cut-graph is a collection of edges that cut a surface into a disk. One step of their algorithm finds nearly-shortest essential loops which is related to our task of finding short non-separating cycles for each handle. However, their approach must analyze the entire surface, compute a cut-graph for the surface and then locally find nearly-shortest essential loops one at a time for each handle. Our approach never requires the computation of a cut-graph for the entire surface. Since our search for non-separating cycles is localized, our algorithm for volume data can operate out-of-core and generate fast locally shortest non-separating cycles used to simplify the topology.

Recent work by Hart [42] focuses on finding a bouquet of non-separating cycles to flatten a mesh into a planar domain. The approach uses Morse theory to find critical points in a mesh. For each saddle point on the mesh, two paths to the surface minimum are found. These paths together can be used to cut and flatten the mesh. These combined paths are not necessarily the shortest cycles. One of the contribution of this work is the introduction of a *fair* Morse function which smooths the geometry of the surface to reduce the extraneous critical points. The approach presents a method to produce a total number of critical points such that there is one maximum, one minimum and two saddle points for each handle. This is a valuable contribution as the excessive number of critical points is a limitation of critical point analysis.

3.2.2 Polygonal schema

The work of Lazarus *et al.* [55] finds a *canonical polygonal schema*, which is a set of $2 \times g$ loops on the surface, (where g is the genus), that share a single vertex in common and that are generators of the fundamental group of the surface. Such a sequence of loops will cut a surface into a disk, however, is more restricted then a cut-graph since every loop must pass through the same vertex. In our work, we examine each handle and measure two non-separating loops. The initial approach proposed by Lazarus [55] did not optimize the length of the cuts. In subsequent work by Colin de Verdière and Lazarus [15] they give an algorithm for the construction of an optimal system of loops equivalent to a given polygonal schema. They define an optimal system to be one where all the loops are of shortest length. An interesting result from this paper is that for a given loop, they can compute a shortest simple loop in the same equivalence class (a loop that can be deformed into the initial loop) in polynomial time. Given two initial non-separating loops on a torus, their method could be used to tighten the loops to shortest length. This algorithm could be used as an alternative to our proposed method to find shortest non-separating cycles, *given a starting pair of cycles*. Both methods make use of tiling the surface (also called a *universal covering*) in order to find the shortest path and both methods compute the loops in polynomial time. However, our method *generates* two shortest length non-separating cycles for each local handle while the method of Colin de Verdière and Lazarus could be used to tighten existing loops. See Section 4.3.3 for more details about our method for finding shortest non-separating cycles.

3.3 Topology simplification

One of the main applications of the algorithms presented in this thesis is topology simplification of both meshes and isosurfaces represented as a scalar volume.

3.3.1 Mesh-based topology simplification

Researchers have addressed topology simplification through a variety of methods. Using the concept of alpha hulls, El-Sana and Varshney [30] reduce surface genus by re-tessellating small handles in a model. Their algorithm creates candidate tessellation regions by heuristically detecting crease edges in mechanical CAD models. The approach lacks a well defined topological measure for evaluating the size of handles in the model. One difference with our approach is that we evaluate whether to retain or simplify a handle based on a topological measure defined on the handle itself. Their approach identifies removable tunnels by rolling a sphere of small radius over the object and filling the tunnels that are not accessible. The method performs well for mechanical CAD models. Another difference with our mesh based work is that it identifies tunnels by working within the surface, and thus can be applied to self-intersecting meshes as long as they are topologically 2-manifolds.

Edelsbrunner *et al.* [27] use alpha complexes to generate a filtration, a history of the evolution of complexes. A filtration allows for a combinatorial definition of topological feature size. Zomorodian expands this work in his thesis [79] and presents a practical algorithm to apply topology simplification to a variety of topological spaces. This work relies on constructing a Morse complex. Such a construction is presented in Edelsbrunner *et al.* [26] for piece-wise linear 2-manifolds. This work could be applied to filter small handles from volumetric data. To properly remove the handles from volume data requires the construction of a three dimensional Morse complex. Recent work addresses this issue [25].

Our initial work to remove excess topology was implemented for existing triangle meshes [40]. This work repeatedly grows ϵ-balls over the surface, and removes any handle enclosed within such a ball via mesh surgery. This approach is simple and effective, however it has some drawbacks. Namely, the definition of topological feature size fails to detect long thin handles, since they do not fit in a small ball (see Figure 3.3).

Finally, topological repair using mesh surgery can give rise to surface self-intersections. We present more information about this method in detail in Section 5. We also present an improved method for topology simplification for meshes and an algorithm to operate on the volume data, since an isosurface will always remain a manifold even after topological repair is applied to the volume data.

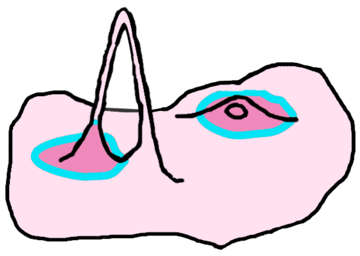


Figure 3.3: *Two different sized handles*
Two different types of handles. Methods that measure handles by only evaluating the surface within an ϵ-ball or sphere will fail to detect long thin handles (except when using large ϵ).

3.3.2 Volume-based topology simplification

Nooruddin and Turk [66] convert a polygonal model into a volumetric representation in order to repair its topology. They apply morphological operations (dilation and erosion) to the volume data, causing handles to close. However, the operators affect the entire volume, resulting in the smoothing of geometry and thus loss of fine detail. Extensions to this approach were recently presented by Bischoff *et al.* [13]. We prefer a more targeted approach that provides analysis of the sizes of the handles and exactly preserves geometrical detail in regions away from topological artifacts.

Shattuck and Leahy [72] address the specific problem of constructing a genus zero model of the human cortex from MRI scans, for use in cortical flattening and mapping. In contrast to the previous methods, they build Reeb graphs over the volume rather than the mesh. Specifically, they construct two Reeb graphs, encoding the connectivity of foreground and background voxels respectively. Their method removes all handles without regard to size, and always breaks handles along axis-aligned planes (Figure 4.18 shows an example where their strategy would fail to find a minimal way to break the handle). In contrast, our approach for volume data, performs one main sweep, constructs a single graph, has a tighter measure of handle sizes, and repairs the volume with a more general and spatially localized operation.

3.3.3 Model simplification

It is worth noting that there is another way to potentially "filter" or smooth the noisy topology of scanned data by smoothing/down-sampling the initial volume data or triangle mesh. Although this approach may remove many of the small tunnels present in the data, it will do so in an uncontrolled manner and will potentially wipe out other features of the model (thin tubes and connected components could be broken apart and the finer detailed geometry will disappear). Work by Gerstner and Pajarola [36] on topology preserving volume simplification is one potential solution to try to control the effect of the down-sampling, however, presently this work offers no method to distinguish important topology inherent to the model (such as a large handle) and small tunnels.

Likewise, several algorithms simplify topology as a byproduct of model simplification of a triangle mesh, *e.g.*, [35, 43, 67]. These methods can result in non-manifold structures which would hinder parametrization as much as the original topological artifacts. In addition, since these methods simultaneously simplify geometry and topology, removing topological artifacts invariably involves loss of geometrical detail. We focus on simplifying topology while preserving as much geometrical detail as possible.

3.4 Computational topology for computer graphics

A variety of researchers have relied on coding or matching the topology of a given mesh to a new configuration [7][3][4]. Most recently attention has been directed toward general simplicial complexes. Specifically, the problem of preserving the topology of simplicial complexes while applying edge contractions was considered by Dey *et al.* [23]. This work develops computational topology techniques for specific tasks for simplicial complexes such as edge contraction and is not closely related to the algorithms presented in this thesis. For a survey of some open problems in computational topology and other applications of computational topology to shape modeling and computer graphics see, for example, the work of Hart [41] and Dey *et al.* [22].

Chapter 4

Algorithms

4.1 Introduction

The genus of a surface in many ways determines the topological complexity of the surface. The fact that it takes $2 \times g$ cuts, (where g is the genus of the surface), to flatten a surface into a disk is just one indicator of how genus affects operations to manipulate surfaces. Determining the genus of a discrete 2-manifold is a relatively simple task. By sweeping over the surface and counting the number of vertices (V), edges (E), and faces (F) for each individual surface component, we can compute the *Euler characteristic*, χ. Specifically, $\chi = |V| - |E| + |F|$, and the surface *genus* is $g = (2 - \chi)/2$ (for each of the individual components of the surface). Genus is a global invariant for the surface and its scalar value gives us one measure of the complexity of the surface, (*e.g.*, a genus zero shape is much less complex then a surface with $g = 100$). However this number on its own does not provide a complete picture of the topology of the surface. Genus is a global scalar value for the entire surface and does not give information about the location or geometric extent of a given topological feature.

For the field of computer graphics, great care is taken with the geometry of a surface, as the geometry plays such an important role in determining the appearance of a surface. Although a coffee mug is topologically equivalent to a dough-nut, geometrically the shapes differ. The difference in their appearance matters greatly when the goal is accurately representing the appearance of real world objects. Thus, a great deal of work in computer graphics has focused on geometric aspects of a surface, including geometry acquisition [17, 45, 47, 59], geometry simplification [35, 43, 67], geometry smoothing [63] and geometry compression [51, 6, 20]. However, there is a direct relationship between the topology and the geometry of a surface that cannot be ignored. For example, geometry simplification which preserves the manifold property of a surface, will only have a limited affect on a high genus model. See Figure 1.4 for an example of how geometry simplification is affected by the topological complexity of a model. Alternatively, many mathematicians and computational topologist are concerned with studying purely topological invariants of a surface. This thesis takes a combined approach and identifies and localizes topological features within a surface by mixing topological and geometrical approaches.

The challenge of this research is to present computational topology algorithms that are designed for discrete surfaces and that simultaneously account for the geometry of a surface. For example, let us return to the issue of computing the genus of a discrete 2-manifold. Although it is straightforward to compute the genus of a surface, this does not tell us enough for most computer graphics applications where we would like to know what the genus *looks* like. Genus is a scalar value with no associated geometric appearance. We can however, isolate geometric regions of the surface that are topologically interesting. The simplest topologically interesting regions and perhaps most intuitive regions to consider are those with genus equal to one. By isolating and examining such regions we can compute a measure to better describe the appearance of relevant surface topology. Thus, this work focuses on isolating *handles*, regions with genus equal to one, in discrete 2-manifolds. Once these regions of interest have been isolated and measured, we may want to reconstruct these regions with a different geometric sampling then the original surface. During reconstruction, we want to be able to choose to retain or simplify topological features of the re-sampled surface. These are the tasks addressed in this thesis, thus we present algorithms:

- to localize regions of topological interest, (regions with $g = 1$, *handles*),
- to measure the geometric extent of those handles,
- to re-sample those handles and either retain or simplify the topology in the resampling.

4.2 Localizing topology

For the purposes of computer graphics we would like to locate geometric regions of topological interest on a surface. The simplest non-trivial regions are areas with genus equal to one. Such regions are *handles*. As mentioned in Chapter 2, handles correspond to the intuitive definition of this word. For example, a coffee cup with a handle has genus equal to one. Thus, the task of our first algorithm is to locate handles in a discrete 2-manifold. We can decompose a surface into balls and handles, where a ball is a region with genus equal to zero. However, there are many different ways to decompose a surface into separate handles and balls. For example, consider Figure 4.2, where even a two holed torus can be cut into different handle configurations. Thus, our task is two-fold:

1. *identify* regions of topological interest. That is identify where the topology changes on a surface
2. *isolate* geometrically succinct handles within the surface.

Inherent in this task is the need to represent the topology of the surface in a useful and simplified way. Thus, we also present a method to represent the topology of the surface. We first present algorithms for identifying regions of topological interest.

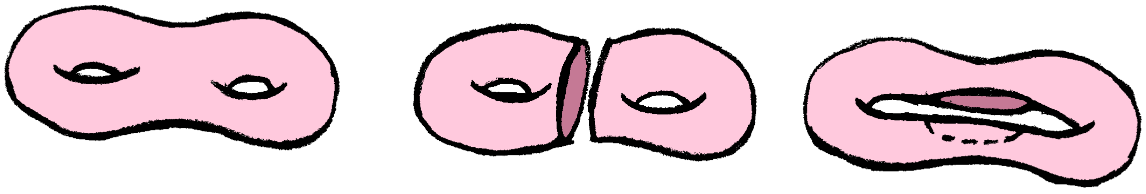


Figure 4.1: *Handle decomposition*
A two holed torus decomposed into different possible handle configurations.

4.2.1 Identifying topology

4.2.1.1 Morse functions

Morse theory, introduced in Chapter 2, is a powerful tool for revealing the global topology of smooth manifolds, however, there are a number of considerations for our setting that complicate the direct application of Morse theory to identify topology for discrete surfaces. Most notably, our setting is a discrete one with piece-wise linear 2-manifolds not smooth manifolds. Although there are recent methods to detect critical points for the piece-wise linear setting [10, 26], we consider methods tuned for the discrete setting that do not identify exact critical points. There are many more critical points than handles and and we are not interested in maxima, minima or lone saddle points. We are only interested in pairs of saddle points that correspond to a handle, (see Figure 4.2.1.1). For this purpose we have found that approaches related to Morse theory, tuned for the discrete setting and which identify *critical levels* of the surface are more appropriate.

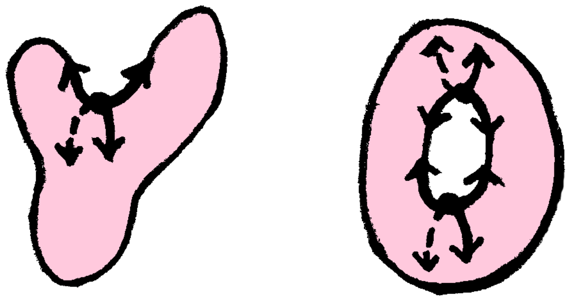


Figure 4.2: *Example saddle points*
A lone saddle point on a branching in the surface and pair of saddle points that make up a handle.

Building on Morse theory, we consider exploring the connectivity of a 2-manifold with a function defined on that manifold. We specifically work with discrete 2-manifolds and discrete functions. By observing the way the discrete function behaves we are able to make observations about the underlaying connectivity of the discrete 2-manifold. We make the observation that by tuning the sampling of our function to closely follow the discrete connectivity of the surface, we can choose appropriate sampling rates to capture *all* the topology of the surface.

4.2.1.2 Height function and immersion

Consider defining a height function for a surface. As mentioned in Chapter 2, height functions are often used as Morse functions. Consider the following observations about a height function defined on a closed 2-manifold. A height function is defined in a given direction in space by the vector **z**. The function returns the projection of the surface onto **z**. In other words, the function associates an elevation along **z** with each point on the surface, (see Figure 4.3). Formally, given a 2-manifold, M, defined by a simplicial complex, and an embedding of that manifold, $S : M \rightarrow \mathbb{R}^3$, for a point p of M, the height function H_z will return the dot product of **z** and the value of S for p, (*i.e.*, $H_z(p) = \mathbf{z} \cdot S(p)$). Given a height function for a 2-manifold and a specific height value, h, a *pre-image* of the function is a *level set* of the surface. The *pre-image* of the function H_z is the subset U of M, such that each element $u \in U$, satisfies $H_z(u) = h$. For our setting, a level set will be a set of closed *contours* created by intersecting the closed 2-manifold with a perpendicular plane at height, h. In the discrete setting, a contour is a closed polyline. We call the set of connected polygons between two successive contours a *ribbon*, (see Figure 4.3).

Observe that defining a height function for a closed 2-manifold and examining the pre-image of the function for various intervals along **z** will create a sequence of closed contours of the surface. This corresponds to placing the closed 2-manifold in a tank and slowly immersing it in liquid up to various heights by adding more and more water to the tank. The level set for a given height h will be the intersection of the surface with the top of the water, (see Figure 4.3). We observe that as the surface is immersed in the water, the topology of the level sets change, *i.e.*, the number of components of the level set changes for various heights. For example, imagine we are pouring water into a tank with the surface shown in Figure 4.3. As we first pour water in to level h_0, we do not intersect the surface and the pre-image of our height function will be empty (it will have no contours). When the water first touches the surface at level h_i, the topology of the level set changes and the pre-image now consists of a single contour. As we continue pouring water into the tank, the topology of the level sets will continue to change. For example, when the water level first reaches the "hole" of one of the handles, the topology of the level set will change from a single contour to two. Finally, consider when we pour in the last of the water and the level set changes such that we once again have no contours.

This analogy of immersing a surface in water is often used to describe the process of finding critical points in a surface. There is a direct correlation with the level sets of a surface changing and changes in the topology of the surface [9, 40, 78]. Consider for example the analysis used in the work of Axen [10] discussed in Chapter 3. Axen examines the star of a vertex to determine if the vertex is a normal vertex or a critical vertex. For example a vertex which is a minimum will be adjacent to a level set (or wavefront) which is entirely "above" the vertex, (*i.e.*, a wavefront with a larger distance value). Likewise a vertex that is a maximum will be adjacent to a level set which is entirely "below" the vertex, (*i.e.*, a wavefront with a smaller distance value). And finally a vertex which is a saddle point will be adjacent to both a "bottom" level set and a "top" level set, (*i.e.*, wavefront(s) with a larger distance value and wavefront(s) with smaller distance values). In fact, levels where the topology of the level set change directly correlate to levels where the topology of the surface

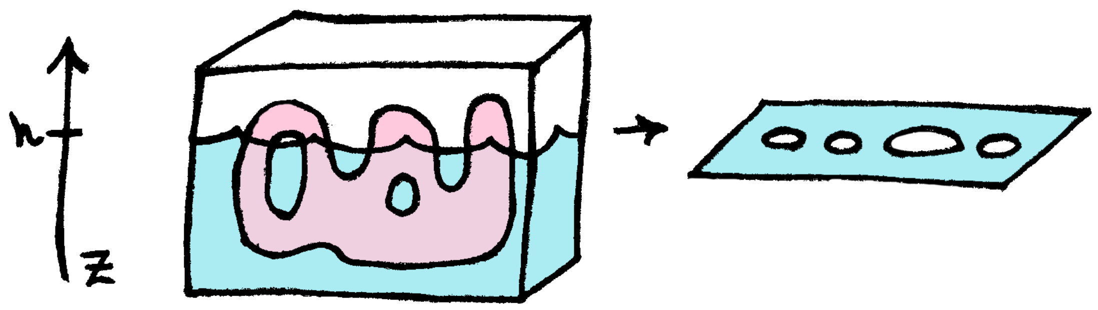


Figure 4.3: *Example of immersion*
On the left is an illustration of a height function with its associated direction vector **z**. *On the right is a surface being immersed in water and the associated level set for the surface at height h.*

change. Such levels are called *critical levels* and we analyze their correspondence to the surface topology in the next section.

4.2.1.3 Face-by-face traversal

Consider the relationship of the level set topology to the surface topology and how observing changes to level sets allows us to deduce the topology of the surface. We work in a discrete piece-wise linear setting where we can make some simple combinatorial observations. Namely, we observe that the discrete primitives that make up discrete 2-manifolds are all simple polygons or triangles and cannot individually represent complex topological features, (*e.g.*, a handle or even a cylinder). Only by combining these elements can we build up more complex topology. Now consider the level sets generated by a traversal of the surface one face at a time. For example, a traversal defined on the surface that starts from an initial face and immerses one additional face at a time. For such a *face-by-face* traversal, the level sets will be the boundary of the currently immersed faces.

Let us analyze the correlation between the face-by-face traversal and the topology of the surface, by examining the Euler characteristic of the surface for each step of the traversal. Recall that we can compute the genus of a surface from its Euler characteristic. The starting element for the traversal will be a single face. For the initial face, the starting contour of the first level set will be the ordered edges of the face. Algorithmically, we proceed by adding an adjacent face, to the current contour, ordered using Dijkstra's shortest path algorithm. After each face is added we compute a new level set which includes the contour modified by adding the new face. Consider the possible results of adding a face to the current contour and how those operations affect the genus of the current region. We enumerate the specifics of such a traversal with regards to a triangle mesh, however a similar analysis is applied for arbitrary polygons, *i.e.*surfels, for the volume setting. The possible changes to the contours of the level sets can be enumerated as one of three possible operations: *add-triangle*, *close-crack*, and *merge-edge*. We describe these operations in more detail.

Add-triangle We assume that the current contour and the new incoming triangle share at least one common edge. Then the *add-triangle* operation adds the triangle to the current contour by merging across a common edge. The resulting surface has one more face, two more edges, and one more vertex than the original one (see Figure 4.4(a)). For this trivial operation, the number of contours of the level set does not change. We observe that the genus of the corresponding surface region also does not change. Indeed,

$$\begin{aligned}\chi_{new} &= V_{new} - E_{new} + F_{new} + H_{new} \\ &= (V_{old}+1) - (E_{old}+2) + (F_{old}+1) + H_{old} \\ &= \chi_{old}.\end{aligned}$$

Since the genus of the region is $g = 1 - \chi/2$, and χ is unchanged, the genus of the current surface region is preserved during the *add-triangle* operation.

Close-crack Once the new triangle is added to the contour we need to resolve possible self-adjacencies along the contour. One local inconsistency is depicted in Figure 4.4(b). We fix the contour locally by eliminating the two redundant edges. The resulting surface has one less edge, and one less vertex than the original one. The number of faces and contours does not change. Again, the genus of the corresponding surface does not change. As,

$$\begin{aligned}\chi_{new} &= (V_{old}-1) - (E_{old}-1) + F_{old} + H_{old} \\ &= \chi_{old}.\end{aligned}$$

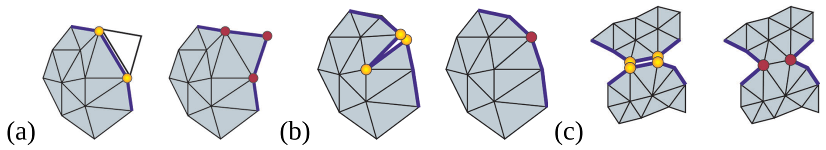


Figure 4.4: *Face-by-face operations*
(a) add-triangle *operation; (b)* close-crack *operation; (c)* merge-edge *operation. Current surface region shown in gray, the current contour is in blue.*

Merge-edge The last operation required to maintain a consistent level set is not local, in that it requires adjacency tests between different parts of the contour or even between different contours. Indeed, the *close-crack* operation cannot resolve situations such as the one shown in Figure 4.4(c). Here two edges lying on two separate pieces of the contour correspond to the same edge of the original surface. We fix this inconsistency by merging the current contour(s) across this edge. As a result the number of contours will either increase by one (when the merged edges belong to the same contour), or decrease by one (when two different contours become one). The *merge-edge* operation results in one less edge and two less vertices for the current region, and the number of faces does not change. Depending on the value of the change in the number of contours we will encounter two cases:

- *A contour splits.*

$$\begin{aligned}\chi_{new} &= (V_{old} - 2) - (E_{old} - 1) + F_{old} + (H_{old} + 1) \\ &= \chi_{old}.\end{aligned}$$

- *Contours merge.*

$$\begin{aligned}\chi_{new} &= (V_{old} - 2) - (E_{old} - 1) + F_{old} + (H_{old} - 1) \\ &= \chi_{old} - 2.\end{aligned}$$

In this last case the genus of the current region increases by one! This simple combinatorial argument illustrates the relationship between changes in the topology of the level sets and changes in the topology of the surface. We observe that if we constructed level sets using a face-by-face traversal we would be guaranteed to capture all the topological changes of the surface.

The problem with a face-by-face traversal of the surface is that such a construction is very redundant. For most of the surface traversal, we only need to use the *add-face* operation. We therefore revisit the question of level set construction but use what we have learned from the face-by-face traversal to guarantee that we construct contours that will encode the topology of the surface.

4.2.1.4 Constructing level sets

In the discrete setting, we would like to only construct level sets necessary to capture all the topology of the surface. Thus, let us consider a modification to the face-by-face traversal

Distance functions We start by considering another function that is commonly used as a Morse function, which is a distance function (also called a geodesic function, or brush-fire or wavefront traversal). A geodesic function computes the shortest path from a seed point on a surface to every other point on the surface. A *graph geodesic* is the shortest path between two graph vertices and can be computed using a breadth first traversal, for example, Dijkstra's algorithm to compute shortest paths. This kind of traversal is similar to the face-by-face traversal except that now the level sets are constructed only after all the faces of distance d from the seed face have been added to the current region. The level set of the geodesic function for any specific distance, d, will be one or more contours, similar to the contours computed for a given h of a height function. One interesting aspect of the contours of a geodesic function is that they will closely follow the geometry of the surface, (see Figure 4.19). As the function traverses the surface and the number of contours changes for various distances from the seed point, the topology of the surface changes.

Now let us consider the two settings for discrete 2-manifolds and which functions might be appropriate for traversing the surface to construct contours. We discuss methods to guarantee that these traversals are accurate and construct contours necessary to capture all the topology of the surface in the subsequent section.

Triangulations For triangle meshes, (as described in Chapter 2) the data is organized as a connected set of faces. In this setting, it is natural to consider traversing the mesh using a graph geodesic defined on the faces of the surface. Starting from a seed face, we use a discrete distance function to traverse the mesh and construct contours representing the topology of the surface. For the purpose of discovering the topology of the surface, we typically use an equal weighted integer distance function, where each face is distance one from all adjacent neighbor faces. When we traverse across the surface as the distance function progresses, the level set of the distance function will be a set of contours as depicted in Figure 4.19. Between the contours the surface may have several connected components. We call these components *ribbons*.

Scalar volumes Let us consider the volume setting, which were defined in Chapter 2. For this setting, the data is organized on a discrete grid with the connectivity of the surface defined for each grid voxel by a look-up table [54]. For such a setting it is very natural to consider sampling this grid using a height function at the predefined z intervals by the grid itself. In fact, such a sampling has advantages for out-of-core algorithms, as discussed in the Chapter 5. For the volume setting we can choose to use a height function to construct contours representing the topology of the surface. In such a setting we can use an axis-aligned *sweep* through the volume to visit the grid data along parallel data *planes*. As expected, the isosurface intersects each such plane along a set of contours as depicted on Figure 4.8 and 4.12. A *slice* of the volume is the set of grid cubes between two adjacent data planes. Within each slice, the surface may have several connected components. Once again, we call these connected components *ribbons*.

4.2.1.5 Intra-ribbon handles

Both of these sampling rates are very natural to their respective setting and allow for the traversal of the surface in incremental regions, or *intervals*. However, there are consequences of this sampling choice for both the volume setting and the mesh setting. Specifically, it is possible that a handle is entirely contained between the previous and next contour, *i.e.*, within a ribbon. For such a case, the handle would not be detected, as the computed level sets have identical topology. One such *intra-ribbon handle* for the volume setting is generated by the 6×6 cube grid shown in Figure 4.5. Although such cases are rare, we must address their presence.

One might propose a possible solution to dealing with such cases for a height function would be to locally increase the sampling of the level set construction. Unfortunately, just increasing the sampling is not a solution. For example the two saddle points that define a handle could appear exactly at the same height, thus no matter how fine the sampling, they could not be isolated from one another and the topology of the

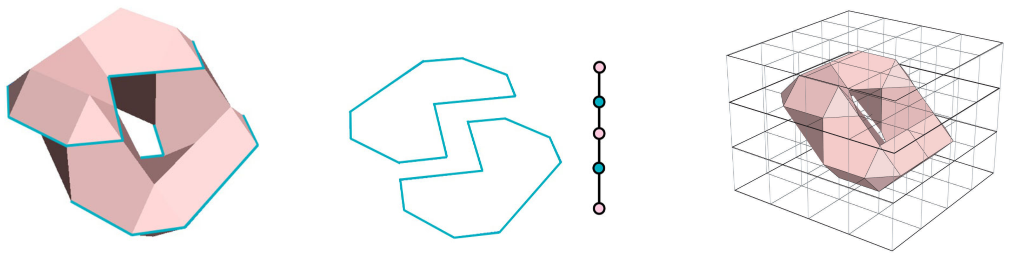


Figure 4.5: *Example of an intra-ribbon handle*
Example of an intra-ribbon handle. This torus tilted at an angle is formed by two "C" shaped contours. As shown in the figure in the center, the Reeb graph does not contain any cycle. On the right we see the volume grid overlaid on this isosurface.

level sets would not appear to change. Methods such as those used by Axen [9] seek to apply perturbations to the data to guarantee that every critical point is isolated at a unique height. We opt for a simpler solution by making an observation about the discrete setting and about the discrete primitives that make up our discrete 2-manifold. Each surface element is topologically simple and only by piecing them together can we construct a handle. Specifically, for the volume setting, observe that an isosurface cannot have a handle within a single cube, thus, by traversing the surface voxel by voxel when necessary, we can isolate critical levels and identify the topology. This is exactly the method presented in the discussion of a face-by-face traversal. This observation gives us the insight to solve the problem of intra-ribbon handles for both a height function or a geodesic function traversal.

We observe that an alternative distance function can be defined which guarantees that *all* the topology of the surface is discovered. As we presented earlier, a distance function defined on faces that orders each face uniquely, such that a contour is constructed after each face is added, guarantees an encoding of all the topological changes in a surface. Thus, for any intra-ribbon handles we modify the traversal to be a face-by-face traversal and construct a contour after adding each neighboring face one at a time. Using an interval traversal with a face-by-face traversal when necessary, we are guaranteed that the level sets for a handle will at some point break from one contour into two contours and then merge back to one, and we will identify all handles.

An intra-ribbon handle relates to a non-isolated critical level. Just as critical points must be isolated for a function to be a Morse function, in order to correctly encode the topology, we must detect isolated critical levels. Altering the algorithm to construct a contour after adding each face within the slice, will isolate distinct critical levels where the previous sampling could not. We limit this face-by-face traversal to only the current ribbon because such a traversal is redundant and is costly when intra-ribbon handles are typically only 1–10% of the input. A face-by-face traversal is costly because it constructs a contour for every face, thus the number of contours computed will be on the order of n for a surface with n elements. In contrast, an interval traversal using a height function or graph geodesic would construct contours on the order of $\sqrt{n}$. Thus, we opt for a mixed solution which uses predominantly intervals of a height function or graph geodesic for the surface traversal, but where any interval with an intra-ribbon handle is traversed using a face-by-face traversal. We discuss how to detect intra-ribbon handles in Section 4.2.2.5.

A note about traversal For a height function, the function may be defined for various directions (for example along the x, y, or z axis). Likewise a graph geodesic is defined from a given seed point. Although alternative directions and seed points will change some geometric information about individual ribbons and contours, the level set analysis will always compute the correct topology. This is due to the guarantees provided by the combined interval and face-by-face traversal discussed above. In addition, we have found that the sweep direction and seed point do not change the behavior of the algorithm to isolate handles within the surface.

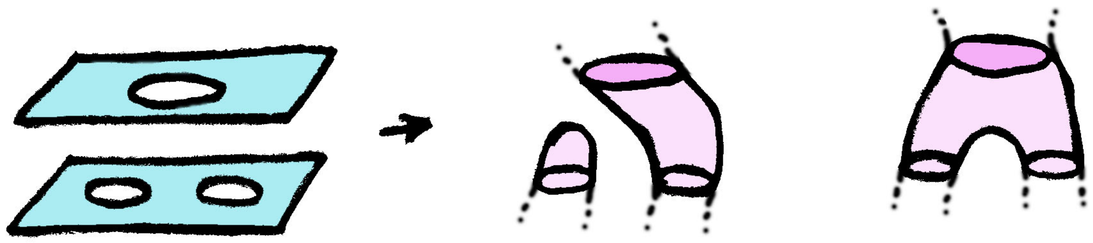


Figure 4.6: *Ambiguous change in two level sets*
An illustration of two level sets which transition from two level sets to one. Shown in the middle is a surface reconstruction corresponding to a maxima while on the right we see a surface reconstruction corresponding to a saddle point.

4.2.2 Coding the topology

Given that we know how to traverse the surface to construct contours in order to correctly detect changes in the topology of the surface, we need to address how to store and analyze this information. Although at first it may seem sufficient to count the number of contours for each level set and store any level where the number of contours changes, this is not enough information to quantify changes in the topology of the surface. For example, a level with two contours followed by a single contour could be a saddle point or a maximum, (see Figure 4.6). Instead we need to analyze how the connectivity of the contours change for varying level sets so that we can disambiguate when a contour disappears versus when two contours merge together. In order to do this, we must keep track of the connected components of the surface as they intersect the height or geodesic function.

Morse theory provides a more complex analysis of critical points of a surface that automatically differentiates the possible topology changes. Recall that critical points are identified and indexed by looking at the number of negative eigenvalues of the Hessian. However, we have chosen to not identify exact critical points and use critical level analysis as this gives us more flexibility to identify and isolate topology for discrete 2-manifolds. Instead, we keep track of surface components and level sets with the use of a *Reeb graph*. Coding the topology of a surface by examining the critical levels of a surface has been used by other researchers [75, 56, 14, 53]. Specifically, by coding the contours in a Reeb graph, researchers have attempted to encode the topology of the surface.

4.2.2.1 Reeb graph

As presented in Chapter 2, a Reeb graph can be used to represent the topology of a surface. Traditionally Reeb graphs were used to represent critical points, where the edges in the graph represented the connected components of the surface between critical points. However, many researchers have used them to code the connectivity of discrete level sets of a surface [75, 56, 14, 53]. Typically, each node of the Reeb graph represents a contour of a level set and the edges between the nodes represent the connected components of the surface between the level sets. Specifically, given a scalar function, f, a Reeb graph can be used to track the connected components of the pre-image of the function. For the example of using a height function, the Reeb graph would contain nodes for each of the contours for each level set generated by the height function. The graph would have edges for each of the surface components between the contours. See Figure 4.9 for examples of some Reeb graphs.

We require that the Reeb graph exactly represents the topology of the surface, *i.e.*, cycles in the Reeb graph correspond to handles in the surface. See Figure 4.9 (A) for an example. Arbitrarily choosing discrete level sets from the height function may miss some levels where the level set topology might have changed. See Figure 4.7 for an example of a problem with arbitrarily sampling the level sets. This is why in work done by Shinagawa [75], where the Reeb graph is constructed from predetermined cross sectional contours, a priori information about the topology of the initial shape is required. It is only with this additional information that the method is able to guarantee that the topology of the Reeb graph will exactly match the topology of the input shape. This is not an issue with our methods, as we have established in Section 4.2.1.1 that by traversing the exact connectivity of the surface we can construct level sets such that the contours will always expose the surface topology. Another short coming of a contour Reeb graph is that occasionally discrete contours alone do not provide enough information about how the topology of the surface is changing (see Figure 4.9 (B) and (C)).

4.2.2.2 Augmented Reeb graph

As presented in Section 4.2.1.1, we know that we can traverse the surface such that we will construct level sets that reflect the changes in the topology of the surface. However, Figure 4.9 (B) and (C) shows an example where constructing a Reeb graph from discrete contours with edges only between sequential contours can create a degenerate Reeb graph with no cycle. More importantly, for our purposes we would like to use the Reeb graph to not only identify handles, but also to isolate the handles within the surface. Thus, we propose an *augmented* Reeb graph, which has nodes not only for contours, but nodes for all the *ribbons* of a surface.

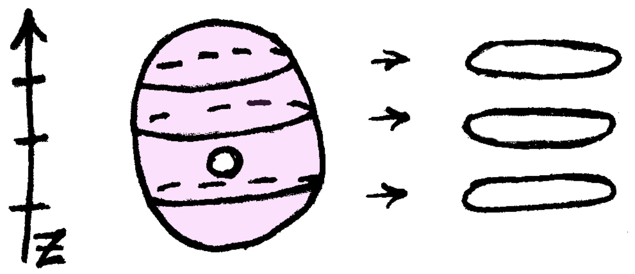


Figure 4.7: *Arbitrary sampling*
An illustration that an arbitrary discrete sampling of a height function which can miss critical levels of the height function.

This augmented Reeb graph contains more information so that we can disambiguate cases where contours alone do not provide enough information and we can isolate geometric regions of the surface corresponding to handles. Specifically, we augment the Reeb graph with additional nodes for each connected component of the surface, (see Figures 4.8 and 4.9). These ribbon nodes store more geometric information about the polygons that make up the topology of the surface. Thus, these additional nodes allow us to not only identify handles but reconstruct them from the information stored in the graph (see Figure 4.10). We discuss the methods we use to isolate handles in Section 4.2.4.

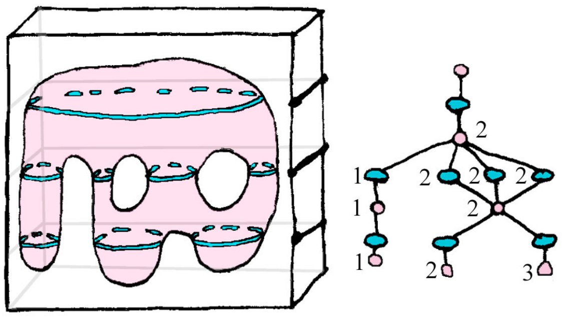


Figure 4.8: *Isosurface and Reeb graph*
An isosurface and its corresponding Reeb graph for a bottom to top sweep of the volume. In the graph, contour nodes are shown in blue, and ribbon nodes in pink. Also shown on the graph are component labels, here represented as numbers.

Augmented Reeb graph construction algorithm Let us formalize the construction of the augmented Reeb graph and examine the correspondence of cycles in this graph to handles in the surface. To construct a Reeb graph, for each z interval of the volume or each face interval of the geodesic function, we must construct the contours and associated ribbons.

4.2.2.3 Contour construction

Before we construct contours, height and distance values must be appropriately propagated throughout the current interval of the discrete 2-manifold. Both contour construction and ribbon construction can easily be done with a breadth first search starting from an appropriate starting element, *e.g.*a triangle or surfel. For example to construct contours for the volume case, for each plane of the volume, the grid is traversed and a contour is constructed starting with any edge of the isosurface. Contour construction proceeds in the plane from the starting edge to the next edge of the isosurface until the contour is closed. The following pseudo-code summarizes contour construction for the volume setting.

```
function Construct_Contours(int $z$)
    Foreach grid cell $\{i, j\}$ of the $z$ plane
        if the grid cell $\{i, j\}$ contains the isosurface
            Compute the appropriate edge, $e$, of the isosurface
            if $e$ has not been used by a previous contour
                Create a new contour $c$ with start edge $e$
                Store $c$ in Current_contours list
                Mark $e$ as used by $c$
                Compute next_edge in the $z$ plane
                While the next_edge is not equal to $e$
                    Add next_edge to $c$
                    Mark next_edge as used by $c$
                    Compute next_edge in the $z$ plane
```

A similar approach is used for a triangle mesh, except edges are no longer followed in the plane and instead the edges are traversed between triangles of distance $d - 1$ and distance d.

4.2.2.4 Ribbon construction

Ribbons are constructed by running a breadth first traversal starting with the polygons adjacent to one contour and ending with the polygons adjacent to the previous contour. We use sets of triangles or surfels to represent the ribbons of triangle meshes and volumes respectively. During ribbon construction, we keep track of adjacent contours in order to track the edges of the augmented Reeb graph. We create nodes in the Reeb graph corresponding to both ribbons *and* contours, and record their adjacency as graph edges, as illustrated in Figures 4.8 and 4.12. We represent adjacencies in the Reeb graph, *i.e.*, edges, as child and parent pointers stored in both the contours and ribbons. That is, each contour has one child ribbon and one parent ribbon, while each ribbon has a list of child contours and a list of parent contours. These pointers are updated to store the adjacency information of the Reeb graph edges. The following pseudo-code summarizes ribbon construction for the volume setting:

```
function Construct_Ribbons(Contourlist Current_contours, int $z$)
    Foreach Contour $c$ in the Current_contour list
        Create a new Ribbon $r1$
        Set child_ribbon of $c$ to $r1$
        Foreach edge of $c$ compute adjacent surfel $s$ of height $= z - 1$
            Add $s$ to queue of Cur_Surfels
        While Cur_Surfels is not empty
            $s$ = top of Cur_Surfels
            if $s$ is a member of any other Ribbon $r2$
                $r1$ = Union($r1$, $r2$)
                Update $r1$ and $r2$ parent_contours pointers to include $c$
                Foreach of the parent_contours of $r1$ and $r2$
                    Update child_ribbon pointers to $r1$
            Add $s$ to $r1$
            Add any unvisited neighbors of $s$ with height $= z - 1$ to Cur_Surfels
            Mark $s$ as visited
```

A similar approach is used to construct ribbons for a triangle mesh with the exception that sets of triangles are distinguished by distance value not height value. Note that during ribbon construction we automatically update the adjacency edges for the newly created contours by updating the child_ribbon pointers. After the ribbons are constructed, we need to compute the adjacencies between the newly constructed ribbons and their child contours. This is done by testing adjacencies in the surface, *e.g.*, each new ribbon adds all child_contours of height $z - 1$ which are adjacent along the edges of any surfels in that ribbon.

Note that any *end region*, like the bottom ribbon of the torus on its side in Figure 4.9 (B) will not have any child contours. In addition, to identify the ribbon associated with the end region corresponding to the *top* layer of the torus on its side in Figure 4.9 (B), we must perform a final pass. Specifically, all contours of height $z - 1$ must have a parent ribbon, if not, we must construct an end region ribbon. This is done with a method similar to the Construct_Ribbon function above. End region ribbons will have either no child contours or no parent contours. We assert throughout the construction algorithm that every contour must have valid child and parent ribbons. Such a construction will create an augmented Reeb graph with the addition of the following consistency check.

4.2.2.5 Asserting Reeb graph consistency

As proven in Section 4.2.1.1, we can traverse the surface such that we will construct level sets that reflect the changes in the topology of the surface. To guarantee that the Reeb graph accurately represents the topology of the surface, we keep track of the Euler characteristic of the surface during the traversal. Recall from Section 4.1 that we can compute the genus of a surface or region of surface using the Euler characteristic. During the Reeb graph construction, we use the Euler characteristic of the surface in order to detect intra-ribbon handles and to guarantee a correspondence between the cycles in the augmented Reeb graph and the handles of the surface.

During ribbon construction, we test that the Euler characteristic for each ribbon corresponds to a region with genus equal to zero. If a ribbon has non-zero genus, it obviously contains one or more handle(s). For such intra-ribbon handles, where the topology cannot be resolved by the normal interval sampling, we locally alter the traversal to be a face-by-face traversal within the current ribbon. As proven in Section 4.2.1.1 such a traversal is guaranteed to produce contours and ribbons that represent a handle. Constructing a Reeb graph from contours and ribbons defined by this traversal will guarantee that we encode every handle as a cycle in the Reeb graph. However, we only use a face-by-face traversal when needed, since constructing a Reeb graph from the contours created by a face-by-face traversal would create an overly dense graph.

To confirm the *overall* correct construction of the augmented Reeb graph for each interval, we use the Euler characteristic of the surface traversed thus far, to assert that the number of cycles in the Reeb graph matches the computed genus of the surface. We employ a step by step construction, where we guarantee that each ribbon interval has genus equal to zero and where after each interval the number of cycles in the Reeb graph matches the number of handles in the surface. Such a construction results in a Reeb graph where each cycle corresponds to a handle of the surface. We discuss how to detect handles and the combinatorial choices for isolating handles within the surface in the next section. The following pseudo-code summarizes the construction of the augmented Reeb graph.

```
function Augmented_Reeb_graph_construction
    Foreach interval $i$ of height or distance function
        Compute the Euler characteristic of the surface up to interval $i$  (See Section 4.1)
        Construct contours $c$ for interval $i$  (See Section 4.2.2.3)
        Foreach contour construct and merge ribbons  (See Section 4.2.2.4)
        Foreach new ribbon $r$
            Compute the Euler characteristic of ribbon $r$
            if the genus of $r$ is greater than zero
                Delete parent_contours
                Construct contours and ribbons with a face-by-face traversal  (See Section 4.2.1.3)
            Add_ribbon_to_Reeb_graph(ribbon $r$, Reeb Graph $G$)  (See Section 4.2.4.2)
        Compute number of cycles $n$ in the Reeb graph $G$
        Assert that $n =$ genus of the surface up to interval $i$
```

Note that the examples of the degenerate Reeb graphs from Figure 4.9 (B) and (C) are related to the

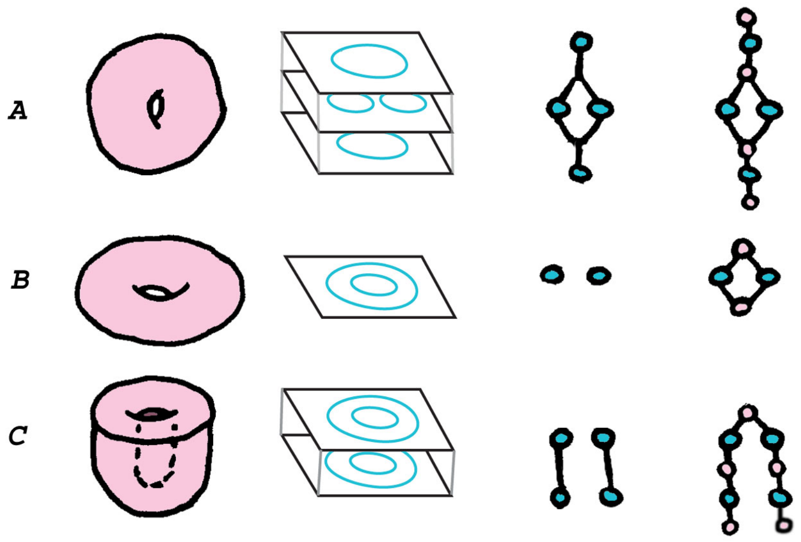


Figure 4.9: *Example Reeb graphs*
Example surfaces and their associated level sets, Reeb graphs and augmented Reeb graph. *The examples are: (A) an upright torus, (B) a torus on its side, and (C) a bowl-like surface. Note that (B) and (C) have degenerate discrete contour Reeb graphs which are corrected with an augmented Reeb graph representation.*

grounded simplices in the work of Axen [10]. Axen's initial solution to use subdivision would create more contours and resolve the degenerate Reeb graph. Likewise, our inclusion of the ribbon nodes is reminiscent of Axen's second solution to examine the dual of the wavefront to resolve grounded simplices. Automatically, by including ribbon nodes, we can resolve all end regions as these regions will be represented by ribbons nodes in the Reeb graph. Figure 4.9 illustrates how the degenerate Reeb graphs are resolved in an augmented Reeb graph.

4.2.3 Detecting handles

Finding cycles in the augmented Reeb graph Once an augmented Reeb graph is constructed, we need a method to detect where the graph has cycles so that we may detect where the surface has handles. We could use various techniques to detect the cycles but we are specifically interested in an algorithm that allows us to locally and incrementally check the Reeb graph for a cycle (*i.e.*, the entire graph need not be constructed first). For a given level set of the Reeb graph corresponding to a specific height h or distance d, we would like to detect if any of the newly added nodes in the Reeb graph correspond to a cycle. For a distance function, any ribbon with more then one child contour is adjacent to a cycle. Such ribbon nodes are easy to detect by counting the number of adjacent contour nodes in the previous interval.

For a height function, the algorithm needs to locally differentiate between a lone saddle point and a pair of saddle points that form a handle. Both of these events are encoded in the augmented Reeb graph by the merging of two contours to one ribbon. See Figure 4.8 for an example of both cases. To distinguish between these two cases we associate a label with both ribbons and contours, that identifies the connected component to which they belong. Such a labeling allows us to locally differentiate between the merging of (a) two previously *disconnected* components (*i.e.*, a lone saddle point) and (b) two previously *connected* components (*i.e.*, a handle forming from the second of a pair of saddle points). For a height function, the only way that a Reeb cycle can form is when two contour nodes in the previous interval are added to a single ribbon node in the augmented Reeb graph. When adding such graph edges, we test whether the two contour nodes have the same component label. If so, they belong to the same connected component and a Reeb cycle is formed. In any case, after the graph edge is added, we relabel the graph nodes to reflect the merging of connected components.

4.2.4 Isolating handles

Once we have detected a cycle in the augmented Reeb graph, we need to isolate the handle in the surface that corresponds to this cycle. The augmented Reeb graph contains all the information that we need to isolate the corresponding geometric region of the surface. Isolating groups of adjacent nodes in the Reeb graph corresponds to isolating adjacent contours and ribbons which form a geometrically succinct region of the surface. Recall that the ribbons and contours have been constructed by a function that traverses the surface in a spatially localized manner, either defined by regions of the surface within a height interval or within a distance interval. Isolating groups of adjacent nodes in the augmented Reeb graph corresponds to isolating a region with boundary on the surface, where the open boundaries correspond to the contour nodes, (see Figure 4.10).

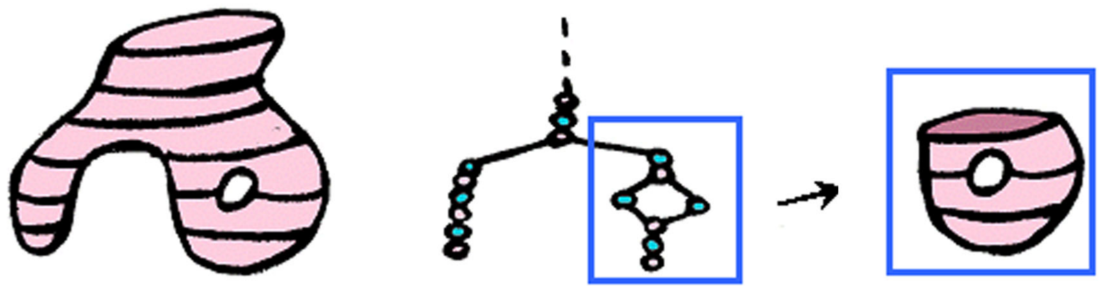


Figure 4.10: *Isolating handles*
Example of isolating nodes in the augmented Reeb graph and the corresponding geometric region on the surface.

4.2.4.1 Combinatorial considerations

As there are many ways to decompose a surface into handles, we must consider the criteria for making combinatorial choices for isolating handles. For the purpose of computer graphics, we consider handles to be toroidal regions that are geometrically localized in the surface. Each of the handles that we consider, correspond to a cycle in the augmented Reeb graph. Each of these cycles correspond to the height or geodesic function splitting into two components and then merging back together. Since the traversal function that generates the cycles is a spatially localized function, the cycles have spatial correspondence to geometrically concise regions. However, even within the Reeb graph, there are many possible cycles in a cyclic graph. Consider Figure 4.11. In order to find geometrically succinct regions, when we detect a cycle, we want to find a short graph cycle. Thus, when a Reeb cycle is detected, we perform a breadth first search through the graph to find the shortest cycle, starting from one of the like-labeled contour nodes, *e.g.*, $c1$ to the other $c2$. This Reeb cycle path consists of alternating ribbon and contour nodes and defines a *handle* with boundary.

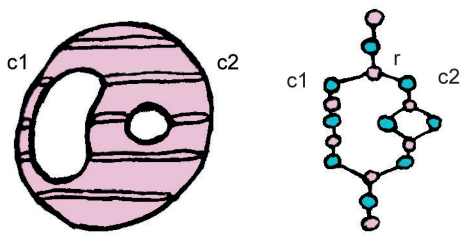


Figure 4.11: *Example of finding a Reeb cycle*
Example surface and its augmented Reeb graph with an adjacent handle. The ribbon r has the previous contours $C = \{c1, c2\}$. When we discover the Reeb cycle associated with contours $c1$ and $c2$, we construct the cycle path by finding the shortest path from $c1$ to $c2$.

More than one cycle can be formed simultaneously in the Reeb graph. Such a case is indicative of handles being adjacent to one another. For example consider the two holed torus and its associated augmented Reeb graph in Figure 4.12. In such a case, a given z interval has a ribbon with k child contours with the same component labels, with $k \geq 2$. Although the local genus of the surface is $k - 1$, we need to isolate a handle for all pairs of child contours. Figure 4.12 shows a two handled torus with a Reeb graph with three possible cycles. In order to accurately consider all of the possible ways to isolate a handle for this region we must consider all pairs of child contours. A single ribbon with $k \geq 2$ child contours can correspond to a region with a degenerate critical point, such as a multiple (or non-isolated) saddle. In such a case, our method of checking the possible combinations of child contours relates to splitting a multiple saddle into simple saddles (similar to [79]).

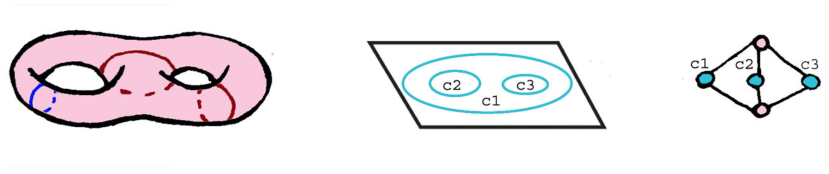


Figure 4.12: *Two holed torus*
*A two-holed torus, the associated planar cross section, and its associated augmented Reeb graph. To find the minimal-length Reeb loop (shown in blue), we must explore all pairs of child contours with the same component label (*i.e., $\{c1, c2\}$, $\{c1, c3\}$ *and* $\{c2, c3\}$ *where the minimal-length Reeb loop is associated with* $\{c1, c2\}$*).*

4.2.4.2 Pseudo-code for detection and isolation

The following pseudo-code summarizes the key parts of the detection and handle isolation algorithm (see Figure 4.11):

```
function Add_ribbon_to_Reeb_graph(ribbon $r$, Reeb Graph $G$)
    Add ribbon $r$ as node in Reeb graph $G$.
    label($r$) := unique_label().
    Identify all previous contours $C$ adjacent to $r$ on surface.
    Foreach (pair contours $c1, c2 \in C$)
        if label($c1$) = label($c2$) then
            path $P$ := shortest path from $c1$ to $c2$ in $G$.
            Report Reeb cycle as $(c2, r) + (r, c1) + P$.
    Foreach (contour $c \in C$)
        Add edge ($c$, $r$) to $G$.
        Unify labels of contour $c$ and ribbon $r$.
```

Once a handle has been isolated, we would like a method to measure the geometric extent of each handle. The measure proposed in this thesis is presented in the next section.

4.3 Measuring feature size

4.3.1 Problem statement

Once we have isolated a handle within the surface we would like to measure the geometric extent of the handle to analyze the importance of each particular topological feature. Specifically, we are interested in a measure that gives a reasonable idea of the geometric importance of a given handle, since it is possible that any acquired data may include some topology which is inherent to the model, (see Figure 1.4). While we could isolate handles one by one and display them for a user to inspect, we sought an automatic method to measure and evaluate the size of handles for a given model. To do this, we consider measuring two locally shortest-length transverse non-separating cycles.

4.3.2 Loops

When considering the extent of a handle there are a variety of possible measures for the geometric size of the handle. We have a chosen a measure based on finding two transverse non-separating loops. We present some information about loops to motivate some of the intuition behind our choice of considering two transverse non-separating cycles. It is well known that a torus can be cut open and flattened to a disk with two loops. In fact, a surface with genus g can be flattened to a disk with $2 \times g$ loops [65]. For a torus, two possible cycles are the canonical generator loops, (see Figure 4.13). Generator loops are basis elements for the torus and as basis elements the entire space of the torus can be reached through integer combinations of these loops. Observe that the generator loops on a torus are both non-separating cycles. Consider, starting with the set of loops that includes the canonical generator loops. Call them A and B. Refer to Figure 4.13. From these initial loops, we can build up an infinite number of equivalence classes of loops by considering integer combinations of loops, where an integer multiple of a given loop such as $2A$, intuitively, means to wrap around A twice. The integer combinations include the trivial loop or null loop which can be contracted to a disk, (*i.e.*, $0A$ or $0B$) and the loops in the opposite direction as A and B respectively, called $\bar{A}$ and $\bar{B}$, (*i.e.*, $-1A$ and $-1B$). Just as there are an infinite number of equivalence classes of loops for a torus there are an infinite number of generator loops. Any pair of loops will be a basis pair provided that the canonical generator loops can be formed from that pair. For example, $A + B, B$ is a valid basis, however, $A + B, 2A + 2B$ is not.

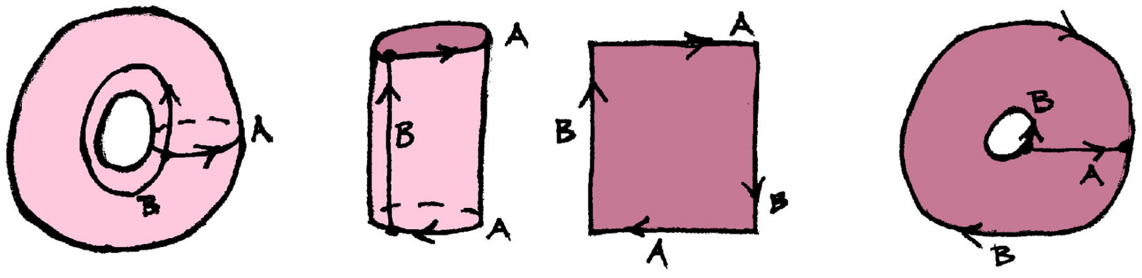


Figure 4.13: *Canonical generator loops*
A torus with the canonical generator loops. On the right we see possible ways to cut the torus open along each of these two loops, either into a cylinder, rectangle or annulus.

We consider finding two loops to measure the two dimensional extent of a torus. With just two loops we can describe the entire space of a torus and it takes two loops to decompose a torus into a rectangle. We do not require that the algorithm finds two generator loops, instead, we seek to find two non-separating cycles from different equivalence classes. The two shortest-length non-separating cycles give a reasonable measure of the minimal geometric extents of the torus.

4.3.3 Overview: two non-separating cycles

Given a handle, we wish to identify two non-separating loops which are transverse to one another. For the sake of discussion we distinguish and name these loops depending on the orientation of the Reeb graph:

- the *Reeb loop* is the nearly shortest loop around the Reeb cycle, and
- the *cross loop* is the nearly shortest loop “transversal” to the Reeb loop.

Note that depending on the direction of the height or distance function, a loop that may be called a Reeb loop would alternately be called a cross loop from an alternative direction. Thus, for an up-right torus, like the one shown in Figure 4.9 (A), the Reeb loop length measures its inner circumference, and the cross loop length measures its girth. For a horizontal torus, (Figure 4.9 (B)), the opposite would be true. See Figures 1.6 and 4.18 for an example of both loops. There are many possible non-separating cycles for a given handle. Since our goal is to measure the extent of the handle relative to rest of the surface geometry and relative to other handles, we find tight fitting loops of shortest length from two different equivalence classes. With the measure of the length of the two loops, we can define the handle size in various ways depending on the application. For example, we may define handle size to be the smaller of the Reeb loop length and the cross loop length or alternatively as a ratio between the two loops. Each of these measures gives information about the relative size of the handle with regards to the rest of the surface.

A first look at finding shortest cycles One approach to find nearly shortest-length non-separating cycles is to use the algorithm of Erickson and Har-Peled. This work creates a cut-graph for the entire surface, where key sub-paths of the cut-graph are shortest paths. Short non-separating cycles are formed by contracting each of these sub-paths to a single vertex, v, and then constructing a shortest non-separating cycle that passes through v (which can be done in O($n \log n$) time). This shortest non-separating cycle will be within a factor of two of the overall shortest non-separating cycle. Once a shortest loop has been found using this approach, a transverse loop could be found with some modifications to their algorithm. One drawback to this approach is that it requires constructing a cut-graph for the entire surface. We wish to simplify this problem to only look at local regions of the surface, thus we take a more localized approach to finding minimal-length non-separating cycles. Our approach is to isolate regions of genus one, *i.e.*handles or a torus with boundary, on the surface and find shortest-length cycles for each local region. Our method is competitive with that of Erickson and Har-Peled for the case of a torus. See the following section, for more details.

The following section addresses finding short loops on a torus. Our method breaks a surface into handles, which are equivalent to a torus with boundary, however, these methods still apply, as we can simulate a closed torus by closing all the boundaries on a handle with temporary disks.

4.3.3.1 General idea

We find the **Reeb loop** by constructing a non-separating cycle that matches the Reeb cycle. Observe that any one of the contours in the Reeb cycle are non-separating cycles, which can be used to cut the Reeb cycle. Intuitively, this corresponds to cutting along a contour of the handle to open it into an cylinder, which is open on both ends, see Figure 4.14. We construct a locally shortest-length non-separating cycle that follows the Reeb cycle by computing the shortest path from one side of such a contour to the other. On the cylinder this corresponds to computing the shortest path from the top of the cylinder to the bottom. We discuss the exact method we use to find the shortest path in the next section. We guarantee that the loop that we find matches a given Reeb cycle by restricting the area that we search for the loop to the isolated region corresponding to the cycle in the Reeb graph, as described in Section 4.2.4. We construct the **cross loop** in a similar manner. Now, starting from one side of the Reeb loop, we compute the shortest paths to the other side of the Reeb loop. Among all shortest paths forming cycles, the shortest is the cross loop.

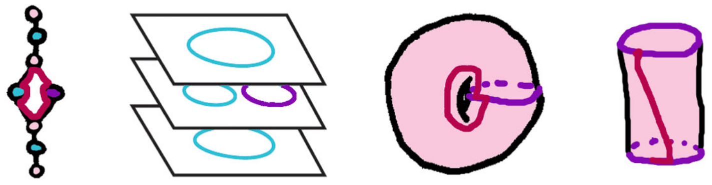


Figure 4.14: *Reeb loop*
Illustration of the process of identifying the Reeb loop for a torus. In this case, we are searching for a loop on the surface of the torus that corresponds to the red cycle in the Reeb graph shown on the left. The search is started from one of the contours in the Reeb cycle, shown in purple in the middle two images.

4.3.3.2 Shortest cycles

Let us consider the details of finding the minimal-length non-separating cycles on a torus. First we consider a simple approach. Given a toroidal region on the surface, we start with an arbitrary non-separating cycle (such a cycle is a contour form by either a height or geodesic function which is stored in the Reeb graph). From this arbitrary non-separating cycle, γ we run Dijkstra's shortest path algorithm from each vertex in γ from one side of the cycle to other (matching the start vertex v with itself). The shortest path of all paths found from one side of γ to the other will be the shortest non-separating cycle in this direction that crosses γ once. We will call this loop L, (see Figure 4.14). However, there may be a shorter loop that crosses γ more than once. Such a loop may wrap around the handle more than once or may simply *backtrack* across γ more then once before returning to the originating vertex v, (see Figure 4.15 (A)).

Although this simple algorithm gives us a short loop, we would like L to be the shortest overall loop and thus be composed of all shortest path segments. That is, we require that starting at any arbitrary vertex k on the loop L, the path to any other vertex, k' on L would be the shortest path between k and k'. However, this is

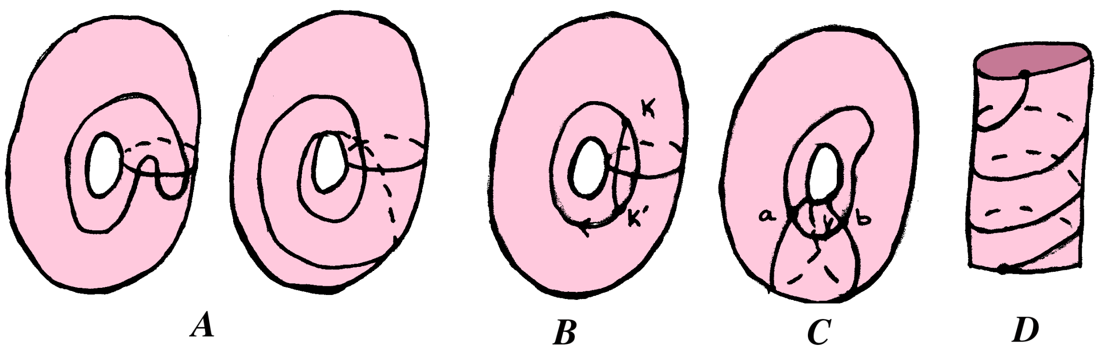


Figure 4.15: *Example of loops on a handle*
Examples of some loops on a handle. (A) On the left is a torus with a loop that backtracks across γ. On the right is a torus with a loop that wraps around and crosses γ twice. (B) An illustration of two possible paths from k to k'. (C) A loop that crosses L twice, once at a and again at b. (D) A loop that winds around a cylinder or handle.

not currently true for all segments of L. Although all sub-paths of a shortest path algorithm are also shortest sub-paths, we have run a constrained shortest path algorithm. Specifically, the constraint that the path start from some vertex v on γ and return back to v by only crossing γ once, means that we have not constructed a complete shortest path. We can see this, when we consider the following. The path L will first cross γ at some vertex u. Since we restrict backtracking, our algorithm must close the loop by connecting v to u by the shortest path *along* γ, (see Figure 4.14). Instead, we need to allow the shortest path L to cross over γ and return to v by the genuinely shortest path between u and v. Thus, we make the following modification to the algorithm. We start with the same arbitrary loop γ, however, this time when we cut along γ we tile duplicate copies of the cylinder one on top of the other, (see Figure 4.16). Each tiling is a duplicate copy of the handle (cylinder). We number each copy of the contour γ and all associated vertices sequentially, *i.e.*, γ_0, γ_1, γ_2, etc. The number of tilings we will need is bounded as discussed later. For now, assume there are an infinite tiling of cylinders. We now run the same shortest path algorithm for every vertex v_0 of γ_0. However, this time we may cross into any of the other cylinders as long as the path returns to some copy of v_i on γ_i. This is similar to what is done in Colin de Verdière and Lazarus [15]. By tiling the handles together, we now run the shortest path algorithm from v_0 to v_i allowing the path to cross over γ as many times as necessary.

Now that L is a shortest path, every sub-path along L is also a shortest path. To convince ourselves, consider, as a counter example two vertices along L, k and k', (see Figure 4.15 (B)). Assume for simplicity that the path from k to k' on L crosses γ once at a vertex v. We know that the segment of L from k' to k, that does not cross γ, is a shortest path because this path is just a sub-path of Dijkstra's shortest path algorithm. We call this sub-path K. Now assume that there is a shorter path from k to k' that crosses γ at vertex p. Call this path P. Such a path cannot exist because if it did, then the loop M comprised of $P + K$ would be the overall shortest path for this handle and would have been detected when measuring the path for the vertex p!

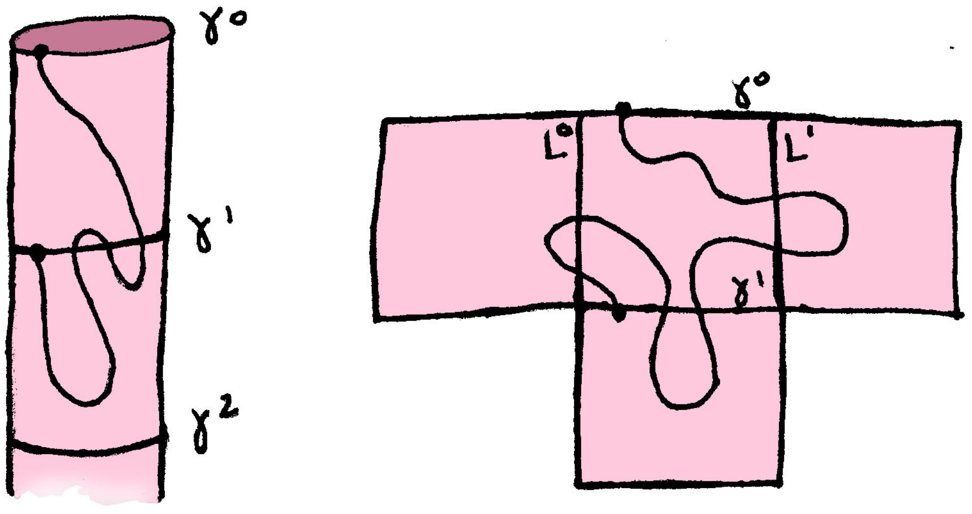


Figure 4.16: *Possible tilings of a handle*
On the left is a tiling of the handle that are decomposed into cylinders by cutting along γ. On the right is a tiling of the handle as rectangles.

We have shown by proof by contradiction that *all sub-paths along L are shortest sub-paths*.

Finding the transverse loop Now starting with L, we apply the same Dijkstra's algorithm for every vertex of L, from one side of L to the other to compute the next non-separating cycle (transverse to L). Call the transverse loop that we find γ'. Note two features of this second pass: we do not need to tile the handle and there cannot be a shorter cycle that crosses L more than once. We see this by the following argument (refer to Figure 4.15 (C)). Assume there is a cycle that crosses L twice, once at vertex a and a second time at vertex b. Call this double crossing path D. Call the path along L from a to b, δ. Now consider the path along D from a to b, call this α and call the path along D from b to a, β. Now we know that δ is the shortest path from a to b (see previous paragraph), thus we know that $\alpha \geq \delta$ and that $\beta \geq \delta$. Thus, we know that there are two paths that cross L just once that are shorter or of equal length to D (namely $\delta + \alpha$ or $\delta + \beta$). Thus there is no shorter path that crosses L twice. We also do not need to consider paths that backtrack across L because of a similar argument, *e.g.*, all sub-path of L are shortest paths. For example, if γ' starts at a vertex g on L and returns to L at a vertex l, we know that the shortest path from g to l will be *along L*.

After running these two passes of the shortest path algorithm we have found two transverse shortest-length non-separating cycles for the torus.

Number of Tilings We can prove that we only need a limited number of tilings of the handle to find the shortest path. Specifically, the number of tilings is limited by two factors. If γ has k vertices, where $k < n$ and n is the number of vertices in the torus, then we need at most k tilings. This is due to the fact that we will never construct a path that crosses through a vertex on γ more then once, nor will we construct a shortest path with a cycle, as all the edge weights are positive. Likewise, given some shortest path of distance d, where $d < n$, we would only need at most d tilings. The smaller of d or k constrains the number of tilings needed to find the shortest path. Note that on average k will be on the order of $\sqrt{n}$. In addition, either $k << n$ or $d << n$, as the n vertices cannot simultaneously be spread out to create long cycles and a long γ. Either way, the number of tilings are limited by the smaller of k or d.

Complexity considerations If we compare the torus case for our approach and the approach of Erickson and Har-Peled we see that our algorithm performs competitively. To run their proposed algorithm would take $O(n \log n)$ and would return a loop within a factor of two of the shortest loop. It takes $O(n \log n)$ time to construct a Reeb graph (as we use a breadth first search to construct the contours and ribbons). It then takes $O(k \times n \log n)$ time to find the shortest loop given an initial γ from the Reeb graph, where k is the number of vertices in the loop γ. In general, k will be on average $\sqrt{n}$, as γ is one dimensional subset of the n vertices in the torus. Thus, with an average factor of $\sqrt{n}$ more complexity, the algorithm proposed in this thesis will find the shortest non-separating loops on a torus.

Note that another option for finding the shortest non-separating loops, would be to run a modified Erickson and Har-Peled approach that also made use of γ from the Reeb graph. Specifically, a modified algorithm could run k shortest path searches for a non-separating loop using the method proposed by Erickson and Har-Peled, for each vertex in γ. Some additional work would be required to make sure that the correct non-separating cycle is found, (*i.e.*one that is not in the same equivalence class as γ itself). Such an approach would have the same complexity as the method proposed in this thesis, $O(k \times n \log n)$, and now both methods would return the shortest non-separating loops for the torus.

Considerations Note that the initial loop that we find, L, may wrap around the handle more then once. Although L is the shortest loop and is a non-separating loop, there are times when we may want to restrict L to not wrap around the handle more then once. Specifically, there are geometric considerations when using L to simplify the topology of a handle. See Section 4.4.1 for more information. It is interesting to note that we can easily constrain that L not wrap around the handle more then once in one direction and will still be within a bounded difference of the shortest loop. We can constrain that L only wrap around once but is allowed to backtrack, by using the same tiling method however, by restricting that L must return to the first copy of v_1 on γ_1, (see Figure 4.16). In this case, L will be within a factor of k of the shortest loop. This is due to the fact that L will take at most a path of length $k/2$ to return to v once it crosses γ at u.

One interesting fact about constraining L is the limitation, that we can prevent wrap around only in one direction. We cannot constrain L to not wind around the handle like a candy cane, (see Figure 4.15 (D)). The only reason that we can constrain L in the first place to not wrap around the handle more then once is because we start with a non-separating cycle that itself does not wrap around the handle (this is due to the planar nature of the level sets for height functions). If we had a similar guarantee for a loop in the same equivalence class as L that also did not wind around the handle, we could construct a rectangular tiling and likewise constrain the path to end within the original rectangle copy of the handle, (see Figure 4.16). It is conceivable that a geometric constraint could be constructed to prevent winding however, we leave this for future work. In general, we are content that the loops found by our algorithm are non-separating loops of minimal-length.

Note that the paths found by the algorithms proposed in this thesis are discrete approximations. First, the shortest loop is not computed as the geodesic over the continuous surface, but as the shortest path over the discrete connectivity graph. Second, we assign all edges in this graph a constant cost, motivated by the fact that the discrete 2-manifolds that we work with tend to have faces of equal sizes. For example, consider the fact that all cubes in the volume have uniform size. Euclidean distance costs could be computed, however, we found our discrete approximations work well in practice. It is possible to extend these results to arbitrary paths over a surface, which we leave for future exploration.

4.3.3.3 Measure of handle sizes

From the two non-separating cycles, we can now derive a measure of the handle. Generally, we use the smaller of the two cycles as the measure of handle size. If desired, we can provide additional criteria, for example, the ratio between the two loop lengths. From the orientation of any contour in the Reeb graph cycle, one can determine whether the ribbon cycle encloses a void or encloses material, which can be used to determine criteria about which loop to measure for topology simplification. Alternatively, instead of finding shortest loops we could find loops with the longest length for each handle if desired.

4.3.3.4 Measurement alternatives

Note that there are other simple ways to measure the extent of a topological feature. For example, we also experimented with a very simple measure of feature size, by restricting the geodesic function to a limited ϵ search radius. That is, from an arbitrary seed point on the surface, we would start a distance function and only traverse the surface out an ϵ distance. By using a very small ϵ value, if any topology was discovered within this ϵ radius, it would be considered *small*. By repeatedly searching over the surface from various seed points, we could discover if any of the handles in the surface covered roughly less than an $\pi\epsilon^2$ area, (see Figure 3.3). The advantage of this measure is that is is very simple. The drawback of such an approach is that it is not a very tight measure for the handle and it only gives a measure for topology smaller than ϵ, while we would like to measure the extent of all the handles in the surface.

4.4 Resampling topology

Once we have information about the location of handles and a measure to evaluate the extent of the handles, we may choose to re-sample the surface topology. We may want to re-sample a surface to eliminate certain handles. Many acquired models suffer from topological noise in the form of excess handles, which we may want to eliminate. Or we may want to reconstruct a new mesh with the same topology. *Remeshing* is common in computer graphics, as various mesh representations have various advantages. Given an initial irregular mesh representation (with arbitrary valence vertices), we may, for example, want to re-mesh the surface to be a semi-regular mesh. During the remeshing process we may want to retain the original topology of the mesh for large handles. This process will be a resampling of the existing topology, guaranteeing that the genus of the re-meshed surfaces matches that of the original.

4.4.1 Handle removal

Let us first consider resampling the topology of the surface in order to remove excess handles.

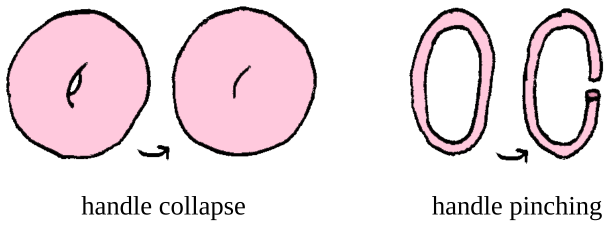


Figure 4.17: *Handle removal*
Two ways of removing a handle, illustrated on two tori. The "fat" torus is best repaired by collapsing the handle, and the "skinny" torus is best repaired by pinching the handle.

There are two natural ways to remove a handle (Figure 4.17). In order to remove either of the handles shown in Figure 4.17, we reduce the number of non-separating cycles for the surface. Recall that genus is defined as the maximum number of simple non-separating cycles that do not intersect. By collapsing a non-separating cycle, we conceptually either fill in the interior of a handle or we pinch open the handle (Figure 4.17), reducing the genus of the surface. Local surface geometry determines which method is more appropriate, as illustrated in Figure 4.17. Both methods are in fact the same operation applied to two different non-separating cycles. We call this operation *loop closure*. Topologically, the loop closure operation collapses the loop to a single point, removing the handle. In terms of geometry, loop closure removes the handle by removing a thin strip of surface about the loop, and closing the resulting two boundaries using two parallel "membranes" spanning the loop.

The ideal choice of which handles to remove is subjective, since some topology may be inherent to the model. While a system could be designed to locate handles and repeatedly ask the user for guidance, we sought an automatic solution. To make this problem computationally tractable, we use the smaller of the two non-separating cycles as a measure, and let the algorithm remove all handles whose measured size is smaller than a threshold ℓ. The locally minimum-length non-separating cycle is an appropriate measure for handle removal because we would like to modify the surface geometry in a minimal way. Thus, we are interested in

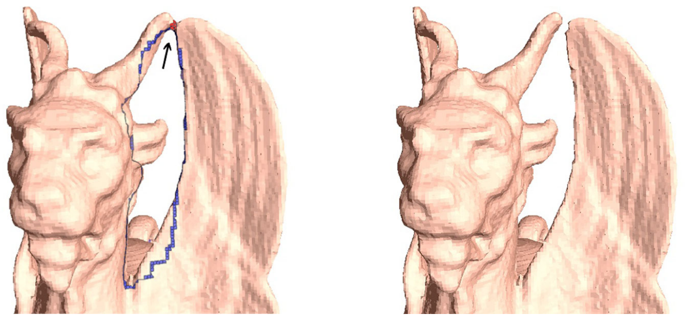


Figure 4.18: *Loops on the feline mesh*
Close-up of the feline mesh with the Reeb loop shown in blue, and cross loop shown in red. The right image shows the output of our algorithm after collapsing the cross loop of the handle.

applying loop closure to the smallest non-separating loops. Loop closure removes the handle using the same minimal-length loop found to measure handle size.

The algorithm to modify the surface topology depends on the surface representation.

4.4.1.1 Triangle meshes

For triangle meshes, loop closure is applied by simply locally modifying the surface connectivity to reflect the closure. First the mesh connectivity is cut along the smallest non-separating cycle. Since the loop is generally non-planar, one could construct some approximation to the minimal spanning surface. For efficiency, we simply use a triangle fan about the centroid of the loop. Next, the new boundaries are closed by adding a fan of triangles to close the boundary on each side of the open loop. Note that the resulting mesh will have connectivity that is guaranteed to be topologically accurate. However, depending on the orientation of the loop, the fan of triangles may intersect one another or intersect other regions of the surface. This region may be smoothed or displaced to remove the self-intersection, however, self-intersections due to loop closure in one of the challenges for the mesh setting.

4.4.1.2 Volume

To perform loop closure on an isosurface represented by a scalar volume, we scan-convert a surface spanning the loop into the volume grid data [50]. The scan-conversion writes either positive or negative scalar values in the grid, depending on the orientation of the loop. We can determine whether we need to add or subtract material based on the orientation of adjacent contours in the Reeb graph. This rasterization technique both collapses and pinches off handles through insertion of a thin wall. The modified isosurface is guaranteed to remain a manifold and to have no self-intersections.

4.4.1.3 Considerations

There are a few potential issues to consider with topology simplification. Our algorithm provides the essential information to always successfully reduce the genus of a handle, *i.e.*, the location of the locally minimal-length non-separating cycle for each handle. Topologically, the handle can always be closed along this loop, reducing the genus of the model. However, as mentioned above, depending upon how the surface is embedded in $\mathbb{R}^3$ the fan of triangles closing the non-planar loop could be self-intersecting, or could intersect other regions of the surface, for instance, if the handle were to contain another, nested handle. In practice, this is not an issue since we only typically simplify *small* handles. At worst, the loop closure could introduce additional handles. Note that for any loop that wraps around the handle more then once, self-intersections would be particularly bad, (see discussion in Section 4.3.3.2). Thus, for topology simplification we may want to constrain the loops to only wrap around the handle once.

One way to address the issue of handles that may be created due to obstructions is to locally rebuild the Reeb graph after a loop closure operation. If any new components or handles were introduced in the previous simplification step, they will appear in the reconstructed Reeb graph and be subsequently processed. In theory the introduction of new handles could cause halting problems if each collapse always created a new handle. We have chosen a relatively simple method for performing loop closure due to the fact that our target application is to remove small excess topology from models. In practice, the issue of obstructions has never caused the creation of new handles or halting problems. In addition, even for large loops that do not contain obstructions, our simple closure routine performs as expected (see Figure 1.4). However, for an alternative application that targeted closing large loops it would be important to add a criterion to check for obstructions and alter the closing routine accordingly.

One difference between the volume setting and the mesh setting is that although in either case loop closure can cause the surface to become self-intersecting, one advantage of the volume setting is the fact that the extracted isosurface will always be a non-self-intersecting manifold [54].

4.4.2 Handle retention

If the goal is to reconstruct handles to re-mesh a surface, this can be done using the information stored in the Reeb graph. Note that this work was done as a part of the author's Master thesis research, however, it is summarized again here as it is a relevant algorithm for applying computational topology algorithms to discrete 2-manifolds. The author's Masters research on the extraction of semi-regular meshes from volume data, required the construction of a base mesh with the same global topology as a desired isosurface. By building a Reeb graph for the desired isosurface, and appropriately selecting contours from the Reeb graph, the selected contours can be stitched together to construct a coarse approximation of the surface with the same genus as the desired isosurface.

Details of Coarse Mesh Construction The Reeb graph provides everything needed to build a coarse mesh from a discrete set of the level sets. Specifically, by carefully choosing which level sets to include in the reconstruction and then stitching the level sets together, a coarse sampling of the surface can be constructed. In order to have a good coarse sampling of the surface, we may wish to only include the smallest number of contours necessary: contours essential for coding topology are those at critical levels in the surface. As described above, critical levels correspond to cycles in the Reeb graph. For example, to reconstruct a torus, it is necessary to include a level set corresponding to the bottom of the torus, with a single contour; a level set when the number of contours changes to two and finally a level set with a single contour. Recall that the topology of the surface between these key contours does not change and thus the surface can be drastically down-sampled in these regions without changing the topology of the surface.

The desired coarseness of the mesh can be controlled by adding criteria for contour selection. For example, consider a requirement that the coarse mesh exhibit good aspect ratio triangles. This can be achieved by selecting contours at multiples of some integer distance w and changing the sampling density along the contours to also be of average distance w.

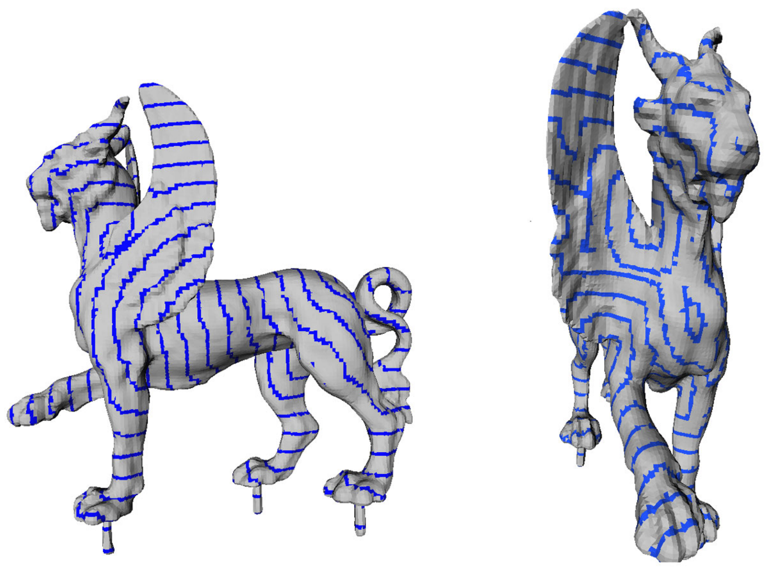


Figure 4.19: *Distance function on the feline*
Two views of some of the distance contours (at multiples of an integer distance) for the feline dataset. The seed for the distance function is near the tail of the feline.

Given the list of required contours for tiling a good coarse approximation of the final surface, we need to stitch them together. The final step of the algorithm is related to contour stitching [11, 34, 29]. However, since we work within the framework of an existing surface (isosurface or mesh) we do not face the traditional correspondence problems of contour stitching. Specifically, the volume data or mesh and the Reeb graph prevent ambiguities about inter-contour connections.

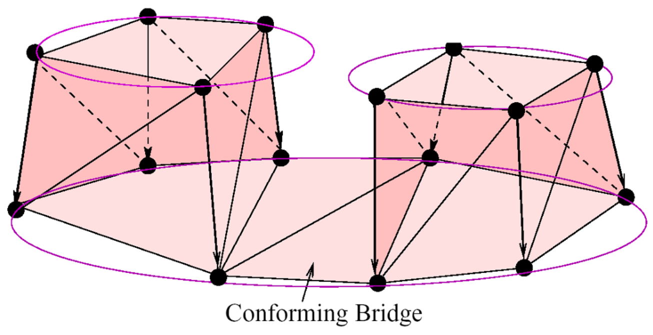


Figure 4.20: *Stitching*
Stitching example for a saddle.

Ribbon sub-sampling and shortest distance projection The general procedure is to subsample each contour along its length to convert it into a coarse contour of edges and vertices to be triangulated with adjacent contours. Adjacent contours are connected to one another by projecting contour samples to the next saved contour (see Figure 4.4.2). This projection step is done by simply following the shortest paths from one contour to the next. Recall that these paths are automatically constructed during the propagation of the geodesic function used to construct the Reeb graph. The projection may result in samples being too close or too far away from one another due to changes in the geometry of the isosurface. In this case we can adjust the number of samples to accommodate the density change by snapping close points together, or inserting a midpoint sample. The samples on both contours are enumerated in corresponding order to facilitate triangulation. The starting and ending contour(s) for a model are evenly sub-sampled and connected to a central point.

Stitching It is easy to tile two contours that have a one-to-one correspondence in their sample enumeration. The general approach of the algorithm is to *break* the contours into one-to-one correspondence and then use bridges between adjacent connected contours to correctly model the topology of the surface. For example, for a saddle where there is a transition from one contour to two, we “break” the saddle into two pairs of 1-to-1 contours with a conforming bridge between appropriate segments (Fig. 4.4.2). This is done by making a pass around the single contour to find if two neighboring samples have been projected from different predecessor contours, in which case they are stored to make the conforming bridge (Fig. 4.4.2).

Once all the contours are stitched together they form a resampling of the original surface that has the same global topology as the desired isosurface. See Figure 4.21 for examples of coarse meshes constructed using this method. Although this method has only been implemented for isosurfaces, the same method could be applied to meshes in order to re-sample the mesh topology and retain inherent topology.

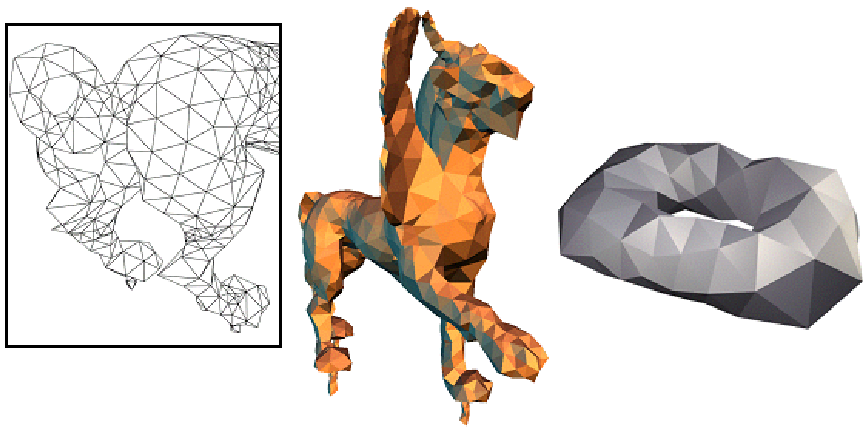


Figure 4.21: *Examples of coarse meshes*
On the right and middle are examples of two coarse meshes reconstructed from volume data. The feline is genus two and the torus is genus one. On the left is a close up of tail of the feline, showing a correct reconstruction of the two handles.

4.5 Conclusion

We have presented algorithms to detect, isolate, measure and re-sample topology for discrete surfaces. These algorithms focus on isolating handles within a surface. We present a robust method to identify handles by using a combined interval and face-by-face traversal of the surface. We have shown that the proposed traversal will detect *all* handles in a surface. This method is tuned for the discrete setting and can be effectively applied to triangle meshes and volume data. We represent the topology of the surface using an augmented Reeb graph which allows us to subsequently isolate handles within the surface. We construct the augmented Reeb graph such that the number of cycles in the Reeb graph is guaranteed to match the genus of the surface at each interval of the surface traversal. Once we have localized regions with interesting topology, we propose an algorithm to measure the minimum geometric extent of these regions. We present an effective method to compute two locally minimal-length non-separating cuts for each handle. Finally, we present methods to simplify or re-sample the topology of a surface if desired. In the next chapter we explore the various applications of these methods to specific settings and present results and implementation details.

Chapter 5

Applications

With the algorithms presented in Chapter 4 we are able to build a number of applications to improve the usability and accuracy of digital models for computer graphics applications. We consider these applications here.

5.1 Setting

Highly detailed geometric models are used in computer graphics to convey visually rich data. Highly accurate geometric models of physical objects are often acquired through discrete scanning techniques. For example, models are commonly obtained using laser range scanners, computed tomography (CT) or magnetic resonance imaging (MRI). Laser range scanners achieve full coverage of complex objects by acquiring and merging multiple scans. Once these scans have been acquired, surface reconstruction is applied and the model is represented as a discrete 2-manifold. Many surface reconstruction algorithms perform the merging of scanned data using a volumetric grid representation, in which the model is represented as the zero-contour of its sampled distance function, *i.e.,* as an *isosurface* [17, 45, 47, 59]. Similarly, CT or MRI produce data volumes from which isosurfaces are extracted [60]. Thus, much of the acquired surface models used in computer graphics are either represented in a scalar volume or as a triangulated manifold.

5.2 Simplifying topology

Consider that although acquired data is being captured with higher and higher resolution geometric accuracy, often these models may have higher genus than expected, due to the presence of extraneous topological handles. In fact, although the real Buddha statue is genus 6, the scanned Buddha surface has genus 104 because of nearly invisible artifacts like the one revealed in Figure 1.2. Similar artifacts also arise in models acquired from CT and MRI scans, and can result in incorrect connectivity of biological structures, such as a brain surface with non-zero genus. In general, topological defects are caused by a number of factors, including sampling density, sampling noise, misalignment of scans, and grid discretization.

While often invisible, extraneous handles create significant problems for subsequent geometry processing like model simplification, smoothing, and parametrization. As seen in Figure 1.4, traditional mesh simplification preserves all handles, resulting in inferior overall quality at coarse resolutions. Also, topological artifacts hinder any processing that must parametrize the surface, such as texture mapping and remeshing (see Section 5.2.1.5). Finally, correct topology can be essential for applications such as the fitting of organ templates to medical MRI data [72, 48].

Given the presence of this excess topology in acquired models, we present a method for removing topological defects in a surface. We consider acquired data that is represented as either an isosurface in a scalar volume or as a triangle mesh. First we address isosurface topology simplification.

5.2.1 In volumes

We present a method for removing topological defects in an isosurface. Rather than attempting to repair the defects on a mesh already extracted from the volume [40], our approach operates on the volume representation directly, as this offers advantages of efficiency and robustness. In terms of efficiency the advantage of the volume setting is the natural ordering of the data in the form of planar slices. This ordering allows us to develop *out-of-core* algorithms to process very large dataset. Operating directly on a volume is robust because even when we alter discrete grid samples in the volume to simplify the topology, the final isosurface *will remain a manifold* [54].

As presented in Chapter 4, our algorithm identifies topology in the volume through the application of techniques associated with Morse theory [64]. The topology is coded in a augmented Reeb graph [68], where cycles in the Reeb graph correspond to handles. In order to measure the size of handles on the surface, we examine them one by one and consider cutting this region along a *non-separating cycle*. By subsequently pinching each of the two open boundaries of such a cycle to a point, the genus of the handle is reduced ($g = 0$). See Figure 4.18 for an example. Using the length of this cycle as a measure of the size of the handle, we choose either to retain the handle or remove it. Our method sweeps through the volume grid to locate handles, compute their sizes, and selectively remove them, accessing only a small buffer of the volume at a time. The contributions of our method are the following:

Out-of-core execution Complex 3D models are represented by large volumes that may not fit entirely in memory. The model in Figure 1.7 is from a $885 \times 709 \times 736$ grid, and much larger models now exist [59]. The algorithm is applied to such volumes using out-of-core methods. The volume is processed using a sweep method, so the data access pattern is highly regular. We encode surface topology as the sweep progresses using an augmented Reeb graph, requiring only a few slices in memory at any time.

Fast identification of handles Handles are efficiently identified during the sweep, as cycles in the augmented Reeb graph as it is incrementally constructed. We detect *all* handles during the sweep.

Handle size estimation and local repair Some models have genus that should be preserved, such as the handles formed by the Buddha's arms. We use a simple measure of handle size to be the length of a non-separating cycle, and remove all handles with a size smaller than a user defined threshold. Cutting along such a cycle helps retain as much as possible of the fine geometrical detail of the model.

Volumetric modification To remove a handle, we alter the scalar values of the volume, thus indirectly modifying the isosurface. Since properly extracted isosurfaces are always manifold [54], operating on the volume is robust. In contrast, traditional "mesh surgery" must deal with issues of surface self-intersection and non-manifoldness. Also, by operating on the volume directly, we avoid computing an expensive triangle mesh and never compute or store floating point values to represent the geometric position of the vertices of the surface. Since our algorithm creates a topologically clean volume, this volume can then be used for surface extraction or other applications that depend on a topologically accurate volumetric representation, for example cortex labeling [48] or 3D morphing.

5.2.1.1 Our approach

Definitions and terminology Our input consists of a regularly sampled 3D grid of scalar values. As presented in Chapter 2, the surfels from all cubes of the scalar volume together form a discrete representation of the isosurface. For our algorithm, the important element is the connectivity of the surfels, as this connectivity defines the topology of the surface. Our algorithm never requires the construction or storage of a triangulation of the surface. We assume that the connectivity of the surfels is pre-determined, for example, by some table driven isosurface generation algorithm. We use the connectivity rules of Lachaud [54] due to the fact that they produce a closed oriented surface without singularities nor self-intersections [54]. Lachaud's table has proven properties by restricting data to have well defined interior and exteriors, *i.e.*, for a scalar function $F(x)$, the interior is defined as $F(x) < 0$, while exterior is defined as $F(x) \geq 0$. This is similar to a standard general position argument, and creates a well defined isosurface, *i.e.*, the surface is perturbed away from the volume grid nodes.

Problem statement The topology of a surface is characterized by its genus, its orientability, the number of its connected components, and the number of its boundary components [61]. Isosurfaces have the property that they are always orientable, and never have boundaries (if one pads all sides of the volume with "outside" scalar values). Our problem of topology simplification corresponds to reducing surface genus, *i.e.*, removing handles. Our algorithm deals with multiple disconnected components by concurrently simplifying them independently. Typically, for the final output, one discards all but the largest component. However, for completeness we simplify the topology of all the components in the volume.

Our goal is to locate handles in the isosurface and selectively remove them. We remove all handles whose measured size is smaller than a user-provided threshold ℓ. For the application of topology simplification, we define the appropriate measure of the size of a handle to be the minimal-length *non-separating cycle*. In order to support out-of-core processing of the data, our algorithm does not find globally minimal-length non-separating cycles. Instead our method finds locally short non-separating cycles for each handle. See Figure 4.18 for an illustration of such cycles.

Summary Our approach can be summarized as:

- Sweep through the volume to *locate* all handles.
- For each handle found, *measure* its size.
- If the size is sufficiently small, *remove* the handle.

We now present each of these steps in more detail.

5.2.1.2 Locating handles

Using the algorithm described in Chapter 4, we use a height function to sweep through the volume along the z axis, and construct an augmented Reeb graph to track the connected components of the surface as the sweep advances. We analyze the isosurface one slice or z-interval at a time. Within a slice, the surface is made up of ribbons, whose boundaries are contours in the two adjacent z planes. Both the ribbons and contours are identified using a breadth first search within the slice to find connected sets of surfels (in the slice) and edges (in the planes) respectively. Contours are constructed by searching from an arbitrary edge in the plane until the contour is closed. Ribbons are constructed by running a breadth first traversal in the slice starting with the surfels adjacent to one contour and ending with the surfels adjacent to the previous contour. Section 5.3.1.3 provides further implementation information on traversing surfels within the volume. We create nodes in the augmented Reeb graph corresponding to both ribbons *and* contours, and record their adjacency as graph edges, as illustrated in Figures 4.8 and 4.12. Given the reconstruction rules of Lachaud, the isosurface is well defined and likewise all the ribbons and contours are well defined. Ribbons, contours and surfels are all topological entities and are constructed based on their connectivity. We do not store floating point values representing the geometric position of vertices of the mesh.

As observed in Chapter 4 the sampling rate for a discrete height function will have consequences. We choose to sample the height function at the discrete z intervals of the volumetric grid as such planar slices are a natural choice for the volume setting as an ordered traversal through the slices allows for the out-of-core processing of the volume data. However, the consequence of this sampling choice is that when a handle is entirely contained within a ribbon, (*i.e.*, within a slice of the volume), it does not initially appear as a cycle in the augmented Reeb graph. In practice, these intra-ribbon handles occur for 1–10% of the total slices for a volume. As proposed in Chapter 4 we detect these intra-ribbon handles by computing the Euler characteristic of each surface ribbon and locally modify the height function to be a distance function defined on the faces within the slice as necessary. We confine face-by-face traversal to the slice because the breadth first traversal of the entire surface would cause irregular access to the volume data, making out-of-core computation impossible.

Finding cycles in the augmented Reeb graph Reeb cycles are detected incrementally as the sweep advances through the volume. This progressive detection allows for handle removal to occur concurrently during the sweep. Our approach is similar to the algorithm discussed in Chapter 4. We keep track of component labels using a Union-Find algorithm on a disjoint-set data structure [16], taking negligible time. When a Reeb cycle is detected, we isolate the associated handle using the method proposed in Chapter 4.

5.2.1.3 Measuring topological handle size

For each handle, we compute two transverse non-separating cycles. One which we call the *Reeb loop* because it is the locally shortest loop around the Reeb cycle and the other which we call the *cross loop* is the locally shortest loop transverse to the Reeb loop. There are many possible non-separating cycles for a given handle. Since our goal is to simplify the topology in a way that minimizes geometric changes to the volume we attempt to find tight fitting loops of short length.

We find the Reeb and cross loop as discussed in Chapter 4. That is we, cut the handle along one of the planar cross sections in the handle and compute the shortest path from one side of the cut to the other. As an implementation detail, for this application, we ran the constrained shortest path that required that the loop not backtrack or loop around more then once. This loop is only nearly minimal-length because we *close* the loop by only traversing along the contour. However, such a loop is within a bounded difference of the minimal-length non-separating cycle, e. As presented in Chapter 4, for a starting contour with k surfels on its boundary, the non-separating cycle we find is at most only $e + k/2$. We guarantee that the loop that we find matches a given Reeb cycle by restricting the area that we search for the loop. Specifically for this implementation we observe that the Reeb cycle contains at least one pair of contours in the same plane, thus we start the search from one such contour and only consider returning paths that have passed through the other contour. We represent paths over surfels instead of the mesh vertices and edges that a marching cubes extraction would produce, as the difference is not significant due to the regular sampling of the volume data.

We construct the **cross loop** in a similar manner. Starting from one side of the Reeb loop, we compute the shortest paths to the points on the other side of the Reeb loop. Among all shortest paths forming cycles, the shortest is the cross loop. Note that neither the cross loop nor Reeb loop is required to lie along a plane. They can cut diagonally through the volume, as shown in Figure 4.18.

Measure of handle sizes From these two non-separating cycles, we can derive a measure of the handle. Generally, we use the smaller of the two cycles as the measure of handle size. If desired, we can provide additional user-control. For example, if the user wants to avoid removing long skinny handles, we can preserve handles that have a large ratio between the two loop sizes. Also, the user can specify that material is to only be added or subtracted from the volume.

As a measure of loop size, we chose the perimeter length of the loop. This length corresponds to the extent of the cut along the surface necessary for loop closure. An alternative would be to measure the area of the loop, *e.g.*, the area of the spanning minimal surface. This area would correspond to the extent of the new surface necessary for loop closure. We have chosen loop length because on our examples this typically was a tighter measure than area. Consider a handle in the shape of a wide, thin-walled vertical tube. The minimal-length non-separating cycle is then a tall, thin rectangle. Even though the loop area may be quite

Model	Grid size	#Faces	Thresh. size ℓ	Genus before	Genus after	#Intra-ribbon	Loops collapsed #Reeb	Loops collapsed #cross	Timing (minutes)
David	$885 \times 736 \times 709$	15,244,302	166.5	1063	0	76	332	731	87.5
Buddha	$400 \times 400 \times 950$	4,736,292	9.5	106	6	26	42	58	6.5
Dragon	$500 \times 714 \times 324$	3,222,612	46.5	60	1	18	31	28	3.8
Brain1	$125 \times 255 \times 255$	688,248	32.5	366	0	6	320	46	2.8
Brain2	$125 \times 255 \times 255$	452,050	14.5	21	0	6	12	9	0.7
Brain3	$125 \times 255 \times 255$	529,012	10.5	41	0	4	25	15	0.6
Brain4	$125 \times 255 \times 255$	699,566	14.5	50	0	7	11	39	1.7
Feline	$332 \times 148 \times 316$	653,922	4.5	6	2	1	2	2	0.2

Table 5.1: *Results table*
Quantitative results: The handle threshold size ℓ is expressed in units of cube edge size. The number of removed handles (original genus minus simplified genus) is broken down into handles collapsed by Reeb or cross loop. Times are shown in CPU minutes for a 1 Ghz, Pentium 4. All values listed are for the entire volume, i.e.*, for the surface and any spurious disconnected components in the volume data.*

small, its perimeter is quite long, and it will therefore be preferable to identify this handle as a large feature. The user may specify an area metric instead of length if this is deemed more appropriate for a particular application, *e.g.*, if filling a long narrow opening in a wide surface is a desired result.

Handle size approximation Our method for computing the minimal-length loop size makes several approximations. First, the shortest loop is not computed as the geodesic over the continuous surface, but as the shortest path over the discrete surfel connectivity graph. Second, our current implementation assigns all edges in this graph a constant cost, motivated by the fact that all cubes in the volume have uniform size. Euclidean distance costs could be computed, but the resulting effect is too small in our examples to matter as the minimal-length cycles are very small to begin with. These approximations improve the speed of our algorithm. Our approach is tailored for the detection, measurement and removal of *small* excess topology.

5.2.1.4 Removing handles

The same minimal-length loop used to define handle size is also used to remove the handle through loop closure. We perform loop closure on the isosurface using the method presented in Chapter 4. This rasterization technique both collapses and pinches off handles through insertion of a thin wall. The modified isosurface, once extracted from the volume data, is guaranteed to remain a manifold and to have no self-intersections [54].

5.2.1.5 Results and discussion

We have run our topology simplification algorithm on a number of volumes, as shown in Table 5.1. The Buddha, dragon, feline and David models are from laser range scans at Stanford University. The brain models are from MRI scans from the Harvard Medical School [52].

We have demonstrated the robustness of our algorithm using convoluted geometry (Figure 5.2.1.5) and large volumes (Table 5.1). Our method is also able to simplify topology for large handle sizes. For example,

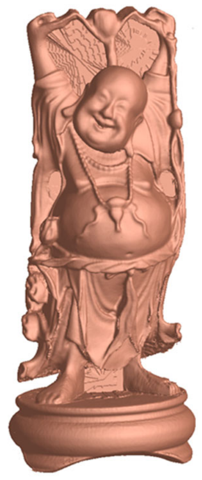


Figure 5.1: *Genus zero Buddha*
Setting the loop size threshold ℓ to infinity for topology simplification results in a genus zero Buddha.

setting ℓ to infinity produces a genus zero Buddha, where even the large handles (with lengths up to 246) are removed (Figure 5.2.1.4). Since our algorithm locally reconstructs the Reeb graph after every topological change, it guarantees that we are accurately identifying all of the topology of the surface, even as its topology evolves. In practice our method has always removed all handles with length less than ℓ.

The timing for our algorithm depends on the size of the volume and on the number of handles. It depends particularly on the number of handles that need to be simplified, since the Reeb graph must be locally rebuilt each time a handle is simplified. In general, our processing takes on the order of minutes, see Table 5.1.

During topology simplification, collapse and pinch operations appear with approximately equal frequency. Topological artifacts are generally small, in terms of both Reeb and loop sizes, and are oriented randomly throughout the volume, leading to equal likelihood of either the Reeb or cross loop having size $< \ell$.

The scatterplot in Figure 5.8 shows a typical distribution of handle sizes for an object with large-scale topology. Typically, extraneous handles in the isosurface are small with 90% having loop lengths of 4–8 (see Figure 5.7). However, there are some volumes containing handles with larger Reeb and cross loops. For laser range data, these larger loops are typically associated with spurious data, external to the intended surface. For example, whereas the surface of the dragon has predominantly small handles, one of its spurious external surface components has a handle of length 46.

The number of intra-ribbon handles varies strongly depending upon the data and the sweep direction. For example, one of the brain MRI volumes has 120 intra-ribbon handles in the original scan direction. This high number is due to the nature of the data. MRI data is often segmented by hand, and small misalignments between these segmented contours commonly give rise to intra-ribbon handles. Sweeping the brain MRI data along an orthogonal direction produces only 6 intra-ribbon handles. Given this observation, one could choose to reorder MRI volumes to sweep in a direction orthogonal to the original data orientation. For range scans, intra-ribbon handles are less frequent, and seem to be independent of sweep direction.

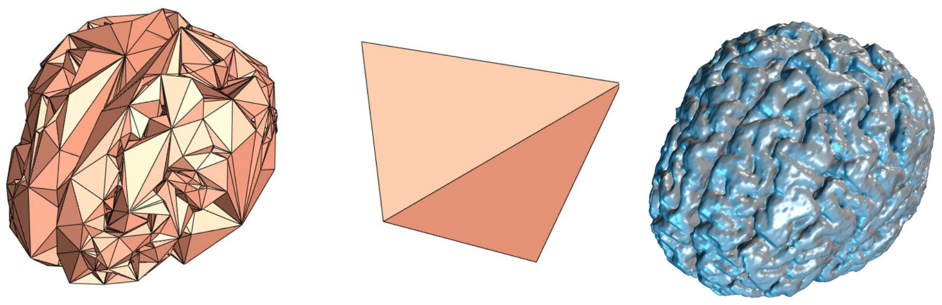

base mesh (5,464 triangles)
base mesh (4 triangles)
7k triangles
Original (genus 366)
Topologically simplified (genus zero)

Figure 5.2: *Progressive meshes of a brain*
Comparison of the base meshes of progressive meshes on a brain model (MRI).

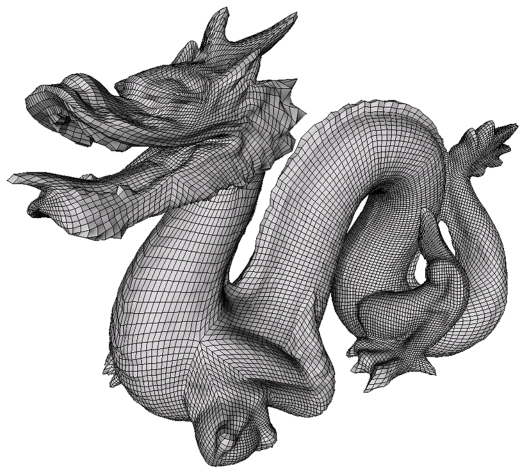


Figure 5.3: *A remesh of the genus one dragon*
A remesh of the genus one dragon. Given the difficulty of achieving a high quality parametrization for high genus models, remeshing the original dragon with genus 46 would be quite challenging and require numerous elements in the base domain.

5.2.1.6 Applications

Topology simplification facilitates many surface operations:

- Fewer triangles are wasted to encode topological defects during *mesh simplification*, as shown in Figures 1.4, 1.7 and 5.2.1.5 using the progressive mesh representation of Hoppe [46]. Consequently, coarser meshes can be created, and geometrical quality is improved at all levels of detail.

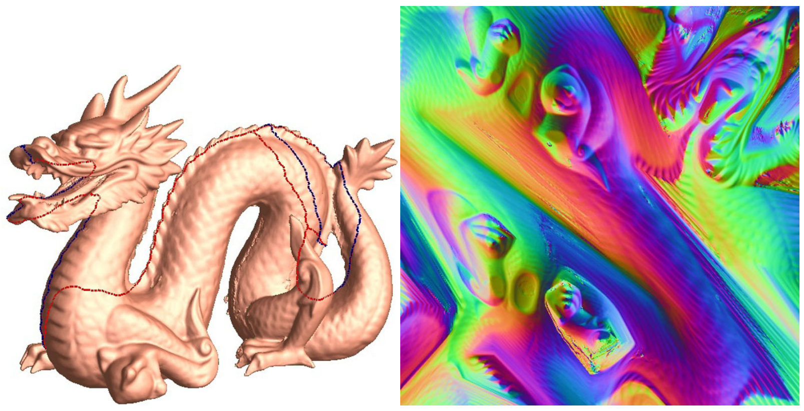


Figure 5.4: *Geometry image of the genus one dragon*
Geometry Image of the genus one dragon model, showing the cuts used to parametrize the entire model onto a single chart. Parameterizing the 500K face dragon onto a unit square would cause large distortion with the original genus 43 model.

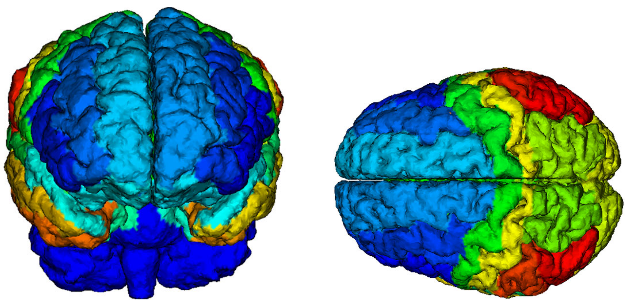
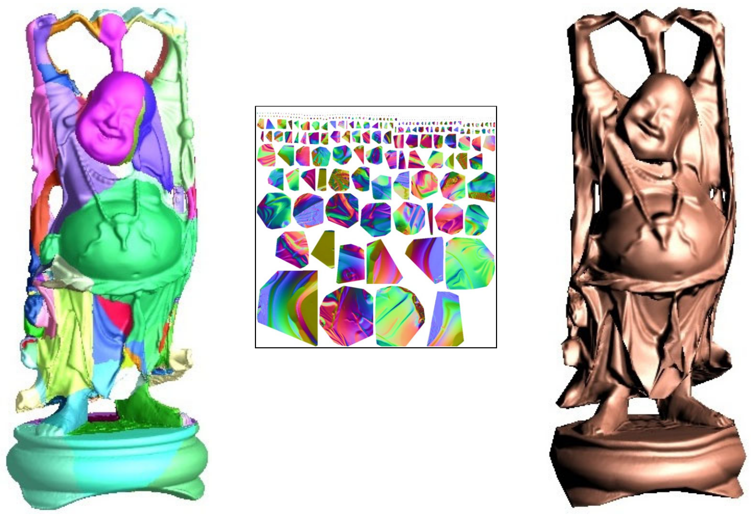


Figure 5.5: *Labeled cortex models*
Two different views of a brain model in which cortex labels have been propagated from one brain to the next through the method of Jaume et al..

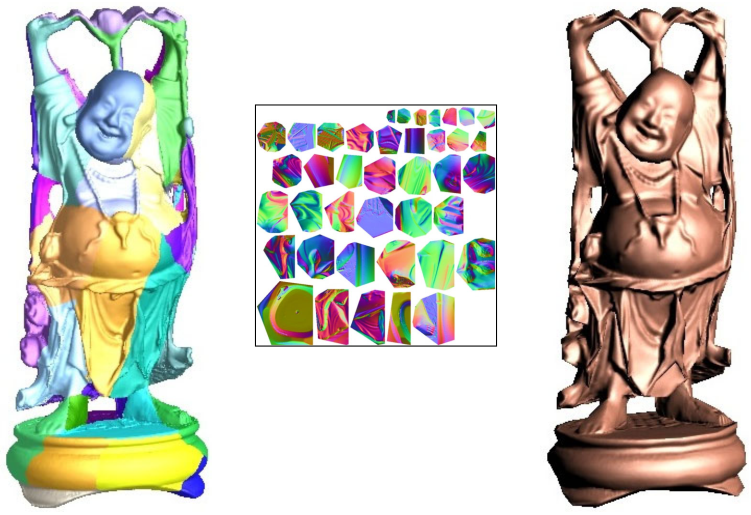
- Better surface parametrization improves *texture mapping*, as shown in Figure 5.6 using the method of Sander *et al.* [70]. Fewer charts are necessary to partition the surface, which results in a nicer parametric domain.
- Removal of topological defects greatly facilitates *remeshing*, as shown in Figure 5.3 using the method of Guskov *et al.* [38]. The remesh has nice regular face sizes and allows for efficient progressive geometry compression [51] as well as many other semi-regular geometry processing algorithms [71]. The topologically clean volumes can also be more readily used for semi-regular mesh extraction [78]. Applications such as geometry images [37], as shown in Figure 5.4, which remesh the entire surface to a completely regular structure by parametrizing the surface to a disk, would suffer from large distortion if applied to surfaces with many topological artifacts.
- Medical applications such a cortex labeling benefit from operating on topologically clean volumes. For

example, the approach of Jaume *et al.* [48], requires genus zero brain models and volumes to propagate cortex labels correctly. See Figure 5.2.1.5 for an illustration of excess topology in brain data. Using our method to obtain topologically clean volumes, Jaume *et al.* are able to propagate cortex labels from one labeled volume to others, (see Figure 5.5).



353 surface charts texture domain 3,700 triangles

(a) Original model (genus 101)



40 surface charts texture domain 2,000 triangles

(b) Topologically simplified model (genus 6)

Figure 5.6: *Texture map comparison*
Comparison of normal-mapping progressive meshes before and after topology simplification. Both models refer to 512x512 texture images. The topological complexity of the original model requires many more parametric charts, shown in pseudocolor. The resulting fragmentation of the parametric domain restricts simplification.

5.2.1.7 Discussion

Setting the handle size threshold For our examples, we first make an initial pass over the volume to gather statistics on handle sizes, and examine these using a scatterplot (Figure 5.8) or histogram (Figures 5.7). By looking at the relative sizes of handles, we select an appropriate ℓ. For most of the models, the excess topology has loop lengths in the range of 4–8. Thus, the setting of ℓ typically ranges from 10–20.

We observed that the initial statistics can change significantly as handles are simplified. Figure 5.8 shows a large handle with a small nested handle. For this configuration, the large handle has a large Reeb loop and small cross loop, and the small handle has an even smaller Reeb loop and shares the same cross loop. During topology simplification, the small handle is removed first leaving only the large handle which now has both large Reeb *and* cross loop. This phenomenon is also reflected in the scatterplots before and after topology simplification (Figure 5.8), where a data point near the ℓ line moves to the top right once the small handle is removed. Note that this effect would be present in any topology simplification that sequentially treats handles locally one by one, when handles are adjacent to one another as seen in Figure 5.8. As adjacent handles are simplified, the topology and measure of that topology changes. There is more than one way to simplify the topology of a shape. In order to support out-of-core operations on large data we have chosen to analyze topology locally in terms of handles. Since our intended application is the removal of topological artifacts, we have chosen an approach that automatically removes excess topology of a given size. An alternative application could integrate visualization of the handles and their non-separating cycles to allow the user more fine control over the order in which loops are collapsed if so desired. For our setting, we found that our method performed well and achieved the desired results.

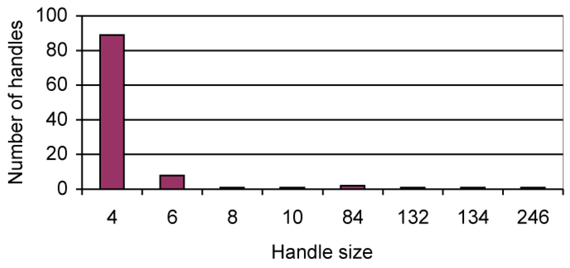
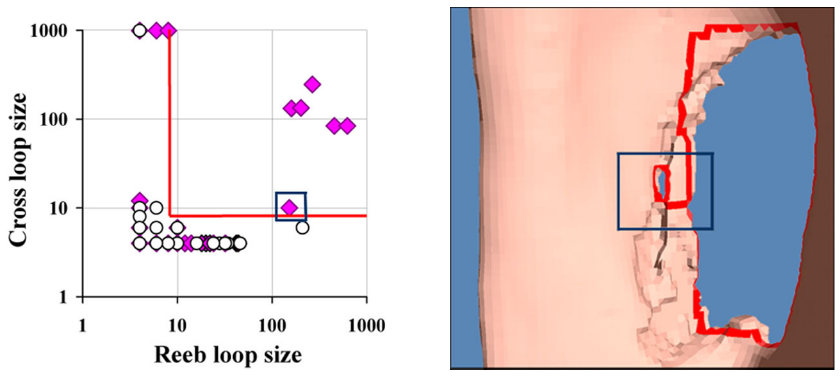


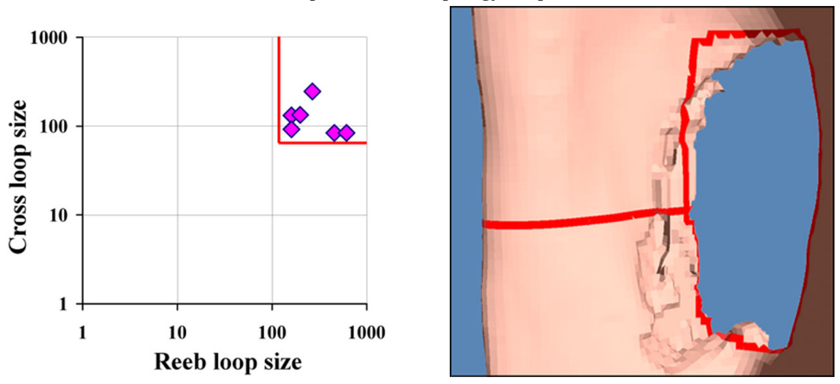
Figure 5.7: *Histogram*
Histogram of handle sizes for the original scanned Buddha model. Recall that handle size is the smaller of the Reeb and cross loop lengths.

Algorithm time complexity The overriding time complexity term for the algorithm is the traversal of the volume, which requires accessing $O(n^3)$ grid values, where n is the extent of the grid in each dimension. Typically the surface has only $O(n^2)$ surfels, and the augmented Reeb graph only $O(n)$ nodes and edges, so the processing steps related to the surface and augmented Reeb graph do not require significant time. However, there is processing time associated with each handle discovered and its subsequent measurement and possible removal. This processing time is dominated by the time complexity of the breadth first searches run to compute the length of the Reeb and cross loop. For each loop, the complexity is $O(d \log d)$, where d is the number of surfels in the local handle. This measure correspond to running Dijkstra's algorithm. In the worst case situation the complexity for a given handle could be $O(n^2)$, if the entire surface were composed of one large handle.

A final concern for time complexity is the fact that for every handle that is simplified, we must reconstruct the Reeb graph locally to account for the resulting changes. The cost of this reconstruction is on the order of $\ell \times n$ (where ℓ slices with n surfels need to be reconstructed after the topology changes).



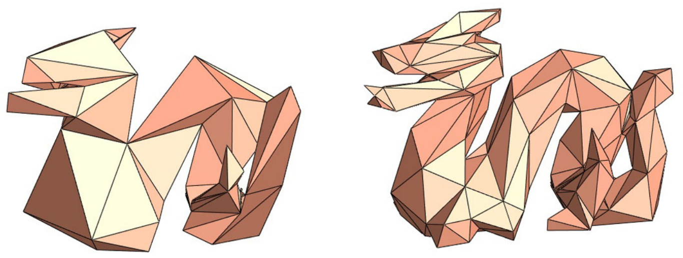
Buddha loop data before topology simplification



Buddha loop data after topology simplification

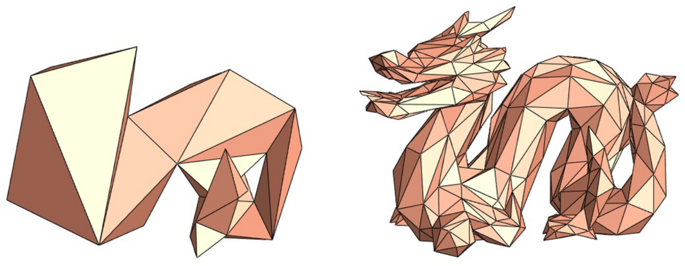
Figure 5.8: *Scatterplot*
Scatterplot of the Reeb loop and cross loop lengths of the handles of the Buddha, before and after topology simplification. Hollow circles identify handles whose minimal-length loop encloses a void. The red lines mark the range of ℓ that keeps exactly these 6 handles. On the right we see corresponding close up view of two adjacent handles on the Buddha model with a shared small cross loop. After topology simplification (bottom), the small handle is collapsed and the larger handle now has a larger cross loop.

Algorithm space complexity Perhaps more important are the space requirements. In practice, we only keep 50 slices of volume in memory at any time, making the algorithm viable even for low-end computers. The choice of buffer size is flexible and can change due to the size of the volume and the memory resources available. All of the operations have strong spatial coherence, and in practice, we found that we rarely reload



base mesh (752 triangles) simplified mesh (1,000 triangles)

(a) Original model (genus 43)



base mesh (52 triangles) simplified mesh (1,000 triangles)

(b) Topologically simplified model (genus one)

Figure 5.9: *Progressive meshes of dragon*
Comparison of progressive meshes with a given triangle budget on the dragon before and after topology simplification. See Figure 5.3 for a remesh *of the detailed genus one mesh.*

the same slice more than twice.

In addition to the buffer of volumes slices, the algorithm stores the augmented Reeb graph $O(n)$ for the surface, and some limited local number of surfels, $P < O(n^2)$. Typically we store $\approx 50 \times n$ surfels, since surfels below the bottom of the current buffer are erased to minimize memory use. These local surfels are stored to compute loops for any handle that is being processed, however, they can also quickly be recomputed using a table look-up [54]. In general computing the Reeb loop for a handle requires access to the surfels in all ribbons referred to in the Reeb cycle. Accurate computation of the cross loop requires additional slices above and below the cycle. The number of additional slices is determined according to ℓ, such that we are guaranteed to find a cross loop of length less than ℓ if one exists. The worst case situation is that of a very long handle with a thin cross-section somewhere along its length. In this worst case setting, the memory requirements may reach $O(n^2)$ to store all of the surfels, in practice this does not happen.

Since surfels below the *bottom* of the current buffer are erased to minimize memory use, Reeb and cross loop computation for handles with length > 50 may require reloading the buffer with previous slices of the volume that have already been flushed from memory. This reloading is only necessary to re-allocate surfels and all computations can be done in sequence, locally on the volume data. The algorithm will never require more memory than the allocated buffer of slices or $O(n^2)$, in the rare case when all the surfels must be stored. Reloading previous buffers is rare since few handles have large extents.

5.2.1.8 Summary

We have introduced an algorithm for automatically removing handles from isosurfaces through direct processing of the original volume data, and demonstrated its effectiveness on several complex models. We have also demonstrated that removing topological artifacts is important for many subsequent modeling operations.

5.2.2 For meshes

Acquired data is also often represented as triangle meshes, and the same excess topology present in volume data is present in these existing meshes. In order to address the issue of excess topology for existing meshes, we have looked at two applications to simplify the topology of triangle meshes. Our initial work uses a simple approach to locate and measure topology. Later we revisit mesh topology simplification using a global approach much like the isosurface topology simplification research presented in the previous Section 5.2.1.

We first present our initial work on triangle meshes which uses a simple local method to identify topological noise.

5.2.2.1 Topological noise removal for meshes

Our initial work on simplifying topology for triangle meshes introduced a simple criteria for identifying topological noise, and a fast algorithm that finds small handles in the data, and removes them one by one. The user could control criteria to help determine which handles are noise and which are inherent to the model. *Portions of the following text, which refer to 'Topological Noise Removal' are reprinted from [40] with permission from the Canadian Information Processing Society (CIPS).*

5.2.2.2 Overview of the algorithm

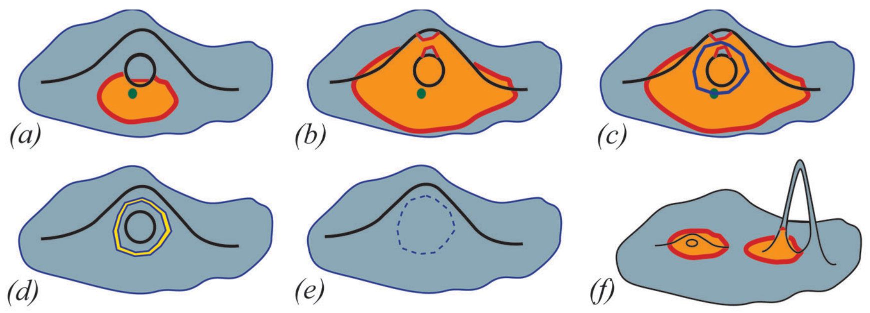


Figure 5.10: *Topological noise removal overview*
(a-e) Overview of the algorithm: (a) a small region is grown around a seed face; (b) the genus of the grown region becomes non-zero; (c) a non-separating cycle is found; (d) the mesh is cut; (e) both new holes are sealed. (f) The left handle is "fully inside" a ball of a small radius; the right handle is not. Note that both handles could be eliminated by short cuts. Our algorithm will only remove the left handle. (Formally, the left highlighted region is of genus one, while the right highlighted region is of genus zero.)

For this work we used an algorithm similar to what was presented in Chapter 4. That is we traversed the surface using a distance function starting from a seed face. However, for this initial work on meshes, we used a face-by-face traversal for the entire search. The algorithm would grow an open region by adding faces one by one, while explicitly keeping track of the current level set, *i.e.*, the boundaries of the open region. Every time a contour of the growing region touched itself along an edge, we would split this contour into two smaller contour fronts and continue the distance propagation. Whenever contours of two different components touch along an edge, we found a handle. For this work, we used two stopping criteria for the region growing procedure. We either exhaust all the faces that are closer than some given radius from the seed face, or we actually find a handle in which case the growing stops. When a handle is found, the mesh is cut along a non-separating cycle, which is later triangulated using methods described in [73][57], or commercial packages [1][2]. Similar to the loop closure operation, this reduces the genus of the surface. Figure 5.10 illustrates the entire process.

5.2.2.3 Algorithm

We consider a triangular mesh $\mathcal{M} = (\mathcal{K}, \mathbf{x})$ where $\mathcal{K} = \mathcal{V} \cup \mathcal{E} \cup \mathcal{F}$ is an abstract simplicial complex representing the connectivity of the mesh ($\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$ are sets of vertices, edges, and faces, correspondingly), and $\mathbf{x} : \mathcal{V} \to \mathbb{R}^3$ is the coordinate function that gives the coordinates of every vertex of $\mathcal{V}$. This work focused on meshes guaranteed to be orientable manifolds. Topology of such surfaces is easily characterized by their genus.

Small handles In order to define which handles should be removed from the mesh, we used a distance metric on the mesh (as described in Chapter 4). We considered any handle that was within an ϵ distance from the seed face to be small. Our distance function for this application was defined on the dual graph $(\mathcal{F}, \mathcal{E}')$ of the mesh $\mathcal{M}$ where a dual edge (t_1, t_2) between two faces of the mesh is in $\mathcal{E}'$ if t_1 and t_2 share a (primal) edge in the triangulation. If some non-negative weight function w is defined on $\mathcal{E}'$, we can now define the distance $d(s, t)$ between any two faces s and t as the minimal sum of weights over all the paths in the dual graph. One easy example is given by setting $w(e') = 1$ for every $e' \in \mathcal{E}'$. It is also possible to make weights that would approximate geodesic distances on a manifold. In this work we use $w \equiv 1$.

The general principle that we use to remove small handles was to create *ε-simple meshes*. Mesh $\mathcal{M}$ is ε-simple if for every face $t \in \mathcal{F}$ all level sets within an ϵ distance enclose a region of genus zero. We call these regions, ε-balls. This was done by finding non-separating cycles for any handles found within the corresponding ε-balls. Note, however, that short non-separating cycles can exist in meshes that are ε-simple for small ε, such as the ones containing long narrow handles, see Figure 5.10(f). However, for this initial work, we choose to not remove such long handles automatically. In this work, we only found cycles corresponding to handles that were completely contained in small regions of the mesh.

The size for ε varies depending on the input data and the relative size of handles to be simplified. For example, in practice we found that values ranging from four to twelve were appropriate for input models ranging from 184K to 4,000K faces.

5.2.2.4 Region growing

We first describe the algorithm that looks for handles in the neighborhood of a seed face. Later, in Section 5.2.2.6 we explain a global search for handles that will use this local procedure as an elementary operation.

The local procedure starts with a seed face $t_{seed} \in \mathcal{F}$. The faces from the ε-ball around t_{seed} are considered one by one in the order produced by using Dijkstra's algorithm on the dual graph, as described in Chapter 4. The process starts with one triangle which is obviously of genus zero. We then proceed either until all the faces of the ε-ball are exhausted, or until we find that after the current triangle is added, the genus of the active region has grown. If the latter happens, the region growing stops and a non-separating cycle is found inside the active region. We then cut the mesh (possibly locally subdividing it), seal the two resulting holes, and start with the current seed face again. Thus, the small handles in the mesh are extinguished one by one.

In order to find small handles, we keep track of the level sets within the ε-ball. If a contour from one level set later splits apart into two contours and then merges back together, we have found a handle. The level sets are constructed after adding each face, exactly as described in Chapter 4. That is, we consider changes to the level sets based on three operations: *add-triangle*, *close-crack*, and *merge-edge*. For this work, we do not explicitly construct a Reeb graph and instead just keep track of the current contours within an ε-ball. Only local information is stored, such as, path information from the add-triangle operation, which is used to find the non-separating cycle later. Specifically, each face stores a pointer to the face to which it was added. To set up the notation, let t be the new face and $t' \in A$ be a face from the active region that shared a common edge with t. We call t' the *parent of* t, or $t' = parent(t)$.

5.2.2.5 Cutting the mesh

In this section, we describe how a non-separating cycle is found inside the active region after a merge-edge operation has merged two contours. Suppose that the two contours merged along the edge $e_M = \{v^{(1)}, v^{(2)}\} = t^{(1)} \cap t^{(2)}$. We build two sequences of faces, $p^{(1)}$ and $p^{(2)}$, defined as $p^{(j)} = (t_1^{(j)}, \ldots, t_{K^j}^{(j)})$, where $t_{k+1}^{(j)} = parent(t_k^{(j)}), j = 1, 2$. Note that both of these face paths end at the original seed face which has no parent. After excluding a common tail of these two paths we have a closed path in the dual graph of the active region. It is then possible to subdivide the faces on this closed path so that there is a closed cycle along the edges of this locally subdivided mesh which does not intersect itself, see Figure 5.12. Note that this path is *completely inside the interior of the current active mesh region*.

This is a different method for computing a non-separating cycle than was described in Chapter 4. Let us briefly prove that this cycle is non-separating, that is, it leaves the active mesh region (and hence the mesh

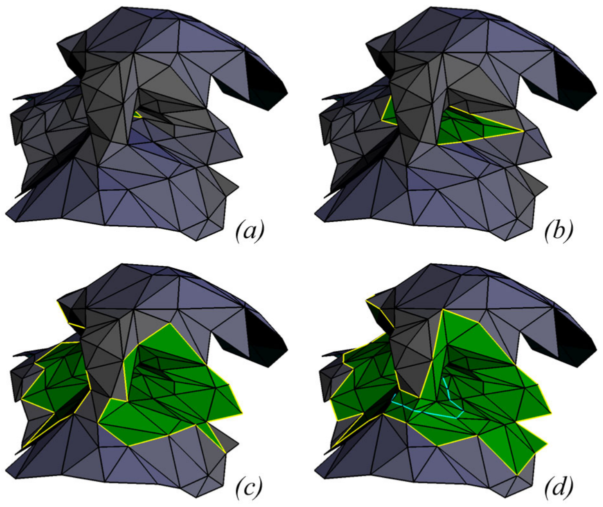


Figure 5.11: *Handle detection and removal*
Running the algorithm locally: (a) the active region is seeded with a single face; (b) propagation has started; (c) the active region has two contours; (d) two contours have merged and a non-separating cycle is found in a locally subdivided mesh.

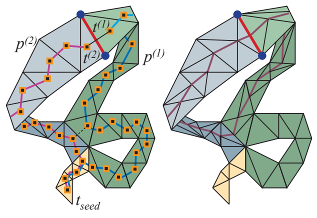


Figure 5.12: *Cutting the mesh.*
Cutting the mesh Left: two paths in the dual graph from the faces $t^{(1)}$ and $t^{(2)}$ to the seed face are found by following the parent links. Note that the closed face path in the dual graph can be reduced as shown by the black dashed line (two adjacent faces allow a shorter connection rather than taking the longer path through the first common face of the paths $p^{(1)}$ and $p^{(2)}$). Right: the non-separating cycle with the corresponding local mesh refinement.

itself) connected. In order to prove that we simply notice that the two vertices $v^{(1)}$ and $v^{(2)}$ lie on the different sides of the cycle locally but we can reach $v^{(2)}$ from $v^{(1)}$ by following the contour of the current active region (we can do that because the cycle is fully inside the active region and thus does not touch the boundary). We also further reduce the length of the cycle, by using reductions similar to the one shown in above figure. During these reductions we do not allow faces $t^{(1)}$ and $t^{(2)}$ to disappear, therefore the argument above still holds. We then seal these two new gaps in the mesh, and thus remove the handle. Figure 5.11 illustrates the process on a fragment of a real mesh.

The subdivision performed during the cycle computation changes distances in the dual graph. We fix this problem by assigning zero weights to the new edges introduced during subdivision (of course, the dual edges corresponding to the edges in the cycle itself simply disappear from the dual graph of the modified mesh.)

5.2.2.6 Global procedure and preprocessing

In the previous section we described a procedure that grows a mesh region of some radius $\varepsilon > 0$ centered at a seed face and removes all the handles that are discovered inside this mesh region one by one. We can run this procedure starting from all the faces in the original mesh. This will produce a mesh that is ε-simple. However, as ε grows the running times of this naive algorithm become unacceptable. We propose a preprocessing step that excludes large portions of seed faces from the consideration. We rely on the following fact which is true in a metric space. Let $B_R(t_0)$ be the closed ball of radius R centered at t_0 (note that we measure the distances on the surface, so in our case, a ball is a surface region.) Then for any $t' \in B_{R-\varepsilon}(t_0)$ the ball centered at t' of radius ε is contained in $B_R(t_0)$, in fact, $B_\varepsilon(t') \subset B_R(t_0)$. Therefore, in the preprocessing step we will be growing balls until their genus changes, without any restriction on their radius. Suppose that we have grown a mesh region A that includes the ball $B_R(t_0)$ for some $R > \varepsilon$, and the genus of A is zero. Then we can be assured that any subset of A will also be of genus zero, and since the balls of radius ε centered inside the smaller region $B_{R-\varepsilon}(t_0)$ are subsets of A, we can exclude them from the potential seed set. These large regions are seeded in the preprocessing step at randomly chosen faces of the original mesh (in practice, taking one percent of the original number of faces produces good results). This procedure greatly reduces the potential seed set for a given ε. For example, without preprocessing, the algorithm takes 1147 seconds to perform simplification with radius 3 on the David's head model; while the improved procedure takes only 136 seconds. More performance numbers can be found in Table 5.2.2.6.

5.2.2.7 Results

To demonstrate our approach we have applied our technique to a variety of the Stanford laser range finder datasets. For example, we consider the triangle mesh dataset of the David's head from Stanford's Michelangelo project [58]. The original irregular mesh has genus 340. Obviously, none of these 340 tiny handles are actually present in the original sculpture, therefore all these handles can be removed to facilitate further processing tasks. An irregular mesh of David's head containing more than a million triangles is processed by

Dataset	Radius	Removed handles	Time
David's head I	8	241	35m 34s
4000K faces	10	264	1h 24m 43s
genus 340	12	283	3h 13m 30s
David's head II	8	291	12m 53s
1173K faces	10	301	27m 37s
genus 340	12	313	56m 52s
David's head III	8	323	4m 27s
184K faces	10	326	9m 36s
genus 340	12	330	19m 6s
David (complete statue)	8	12	34m 4s
8254K faces	10	13	45m 11s
genus 20	12	14	57m 43s
Buddha	8	71	10m 23s
1087K faces	10	82	34m 24s
genus 104	12	85	2h 43m 9s
Dragon	8	21	6m 4s
870K faces	10	32	16m 59s
genus 46	12	35	53m 3s
St.Matthew	6	3	21m 19s
3382K faces, genus 5	12	4	29m 37s

Table 5.2: *Results Table*
Timings given for Pentium III Xeon 550 MHz.

our algorithm in one hour, removing 313 (92%) of the handles automatically. We have found that most of the models reconstructed using Curless and Levoy's VRIP method [17] have topological artifacts. We noticed that higher resolution models and meshes that were more convoluted in shape typically have more topological noise. We have run our algorithm on models of different resolution with different threshold radius settings and recorded the number of handles removed and the algorithm's running time. These results are illustrated in Table 5.2.2.6. Note that this automatic technique fails to generate a genus zero surface when there are handles larger than the given ε. For this initial work, we used a very simple measure. We later improved our results for triangle meshes with subsequent work discussed in the next Section 5.2.3.

We have applied various mesh processing techniques to meshes that have been topologically simplified using our algorithm with encouraging results. In particular, we were able to apply the multi-resolution remesher of Guskov et al. [39] to the simplified genus zero mesh of David's head. The base mesh for this remesh contained 262 triangles. It would be impossible to achieve such a small number of patches without first applying a topology simplification operation to the original data (recall that the original mesh had 340 handles).

Similarly, parameterization of mesh regions is a fundamental part of many remeshing, texturing, and other mesh processing algorithms. Figure 5.2.2.7 shows the parameterized mesh region of the David's ear. The texture coordinates are assigned with the (u, v)-coordinates computed with the shape-preserving parameterization of Floater [32]. The original region of this mesh contained twelve handles. Our algorithm removed all

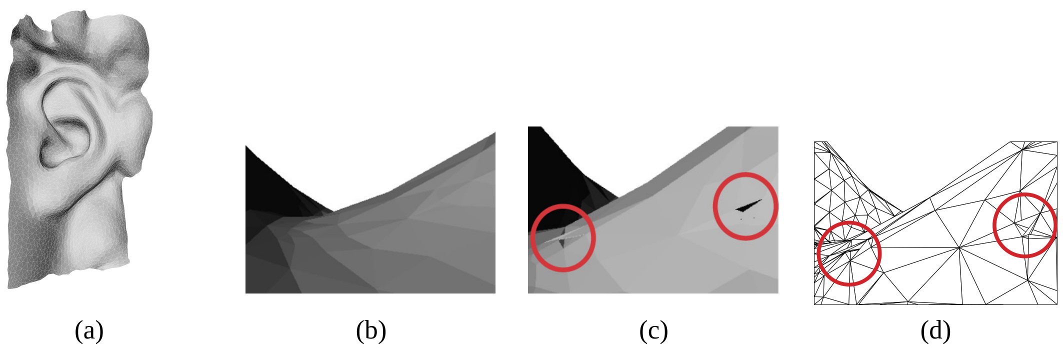


Figure 5.13: *Smoothing result*
Smoothed version of David's ear (a) and close up view of the smoothed ear after topology simplification (b) and a close up of the artifacts that occur without simplification (flipped triangles) (c) and a detailed view of the handles causing the artifacts (d).

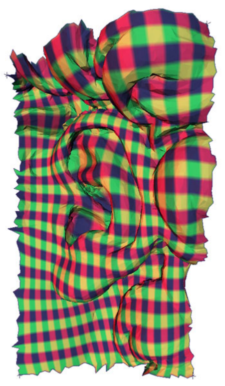


Figure 5.14: *Texture mapped ear*
An example of texturing mapping after successfully parameterizing the topologically simplified David ear.

of these handles in fifteen seconds, and produced a mesh that is homeomorphic to a square, allowing it to be properly parameterized.

Additionally, acquired meshes often contain geometric noise, and have to be filtered with various mesh smoothing/noise removal techniques. In particular, we used the method described in Desbrun et al. [21]. If the original mesh contains unnecessary non-trivial topological artifact, the smoothing procedure typically results in a mesh with artifacts that foil its appearance (such as flipped triangles), as shown in Figure 5.13. This is due to the fact that smoothing operators cannot modify the topology of the mesh, and the presence of these small handles impairs the smoothing process by limiting its effects. Attempts to smooth the region around small handles can potentially result in collapsing the handle, creating undesirable degeneracies. Thus first removing the topological noise greatly improves the performance of geometric noise removal procedures, as illustrated by Figure 5.13.

5.2.2.8 Conclusions

Excess topology is a serious problem for many scanned models. This incorrect topology results in visible artifacts when these meshes are smoothed, encumbers parameterization and hinders the performance of many multi-resolution techniques. We have presented a simple criteria for identifying excess topology and a computational procedure that removes these topological artifacts. However, after this initial work we considered enhancements from the lessons we learned working with volume data (presented previously in Section 5.2.1).

5.2.3 Mesh topology simplification revisited

We learned valuable lessons from our research on simplifying the topology of isosurfaces within volume data and choose to extend this new knowledge back to the mesh setting. Specifically, we looked at replacing the repeated local ε-ball searches with a global augmented Reeb graph construction. Such a global construction requires that we visit each face of the mesh only once to build up connectivity information for the entire surface. This allows us to guarantee that we encode all of the topology of the mesh without needing to repeatedly propagate ε-ball searches over the surface. In addition, we wanted a measure that would measure *all* handles, including ones that did not “fit” into an ϵ radius. See Figure 3.3 for an example of a handle that would not be measured using our initial mesh based approach. Finally, we wanted to apply the tighter handle size measure used in the volume setting. That is, we wanted to find two locally minimal-length non-separating cycles to evaluate the size of a handle. Thus, we revisited triangle mesh simplification, using a global augmented Reeb graph and a tighter handle measure.

Specifically, we designed an algorithm that uses a distance function defined over mesh faces, as described in Chapter 4. Starting from an arbitrary seed face, we propagate a discrete distance function to construct contours and ribbons. Contours and ribbons are constructed on the fly after the current distance d is propagated. We use an augmented Reeb graph and contours and ribbons are constructed using the methods described in

Section 4.2.2.2. Contours are constructed starting with the first unused edge between a face of distance d and a face of distance $d-1$. Contour construction is completed by following the orientation of the triangle faces to find the next unused edge. Ribbons are constructed by gathering together all faces of distance d that are edge adjacent to one another. After ribbon and contour construction, the adjacency of the contours and ribbons in the surface are used to determine the correct edges to add to the augmented Reeb graph.

As described in Chapter 4 we must account for intra-ribbon handles and adjust the traversal function to be face-by-face when necessary to guarantee that the number of cycles in the augmented Reeb graph always matches the Euler characteristic of the surface traversed thus far. Note that unlike the case of a height function, we do not need to keep component information with a distance function traversal from a single seed point. This is due to the fact that no new components will be created during the traversal, each new contour is connected to a previous ribbon. In this setting *any* ribbon node in the augmented Reeb graph that has more than one child node will be a part of cycle and thus a critical level. After processing the contours of distance d and adding all the appropriate edges to the augmented Reeb graph, we check for critical levels of the distance function. Anytime a ribbon node is adjacent to more then one child contour, we know that it is adjacent to a cycle. We then identify the possible cycle(s) and isolate the associated handle(s). This is done using the methods discussed in Chapter 4. Note that just as a ribbon node defined by a height function may have more then two child nodes the same case may occur with a distance function. For such a case, each of the possible cycles must be isolated and measured. For the newly isolated handle(s), we construct the two locally minimal-length non-separating loops and evaluate if the handle should be simplified, as described previously in Chapter 4.

Just as was done for the volume setting, we first gather statistics for a mesh about handle sizes and then simplify all handles with a non-separating cycle of length less than the user defined threshold. Handles are simplified using the methods presented in Chapter 4. We locally re-build the Reeb graph to account for the changes to the topology. For a distance function the augmented Reeb graph must be rebuilt only for the distance levels adjacent to the isolated handle.

This algorithm improved the accuracy of our mesh topology simplification, as we are now able to reduce the genus 100% for the David head, as opposed to the 92% by our initial algorithm. The global traversal and augmented Reeb graph guarantee that we will find *all* the topology present in a mesh and the locally minimal-length non-separating cycles give a more accurate measure of the handle sizes. This allows the topology simplification algorithm to be more precise and change the geometry of the surface in a locally minimal way. Similar to the results already presented, the topologically simplified meshes can now be parameterized for subsequent texture mapping and remeshing as desired.

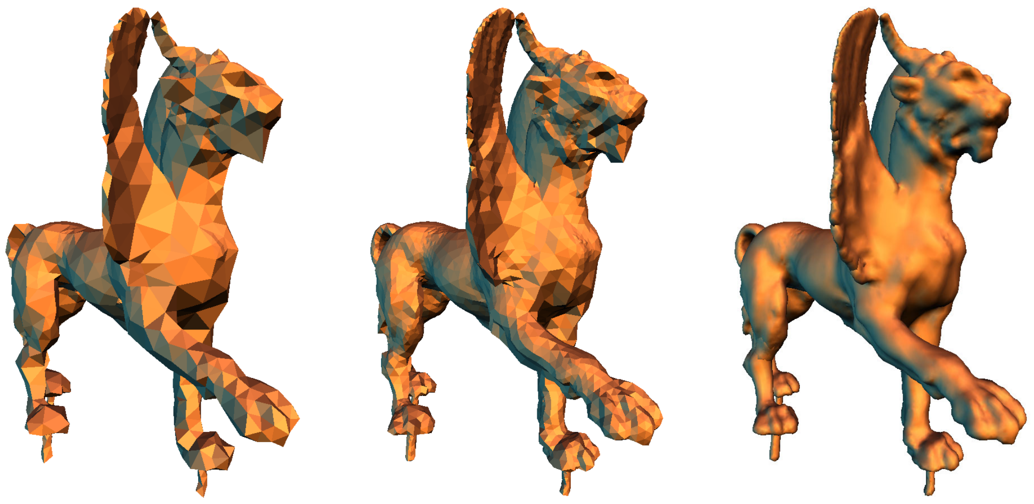


Figure 5.15: *Example of a series of semi-regular meshes*
Example of various levels of an extraction of adaptive semi-regular meshes from a volume using our algorithm. On the left is a coarse resolution version of the surface, followed in the middle by an intermediate version. Finally the finest resolution surface is on the right.

5.3 Hierarchical surface reconstruction

Our final application concerns *remeshing* a surface from one discrete surface representation to another. Note that this work was done as a part of the author's Master thesis research, however, it is summarized again here as it is an relevant application of computational topology algorithms to discrete 2-manifolds. *Portions of the following text, which refer to 'Semi-Regular Mesh Extraction from Volumes', are reprinted from [78] with the permission of IEEE.*

5.3.1 In volumes

5.3.1.1 Motivation

We are interested in remeshing an irregular mesh to a multi-resolution mesh representation. Such representations have advantages for editing, compression and level of detail rendering. In particular we are interested in looking at improving isosurfaces extracted using Marching Cubes (MC). Marching Cubes (MC) [60], is the predominant algorithm for isosurface extraction. As described in Chapter 2, this method computes a local triangulation within each voxel of the volume containing the surface. This results in a uniform resolution mesh. Often much smaller meshes adequately describe the surface since MC meshes tend to oversample the isosurface, encumbering downstream applications, e.g., rendering, denoising, finite element simulations, and network transmission.

There are a variety of methods to remesh existing meshes. Many techniques start with the existing mesh and apply topology preserving geometry simplification to compute an initial coarse mesh. This coarse mesh is subsequently processed to create a multi-resolution representation [24, 57, 39]. Such a process is guaranteed to create a coarse mesh with the correct topology. We tackle a slightly different problem, which is given volume data directly extract a multi-resolution mesh without first extracting a full resolution isosurface. We present a method for the *direct* extraction of an adaptively sampled multi-resolution isosurface mesh with good aspect ratio triangles. The multi-resolution structure is based on adaptive *semi-regular* meshes, well known from the subdivision setting [80]. Figure 5.15 shows an example of a multi-resolution semi-regular mesh extracted from a distance volume with our algorithm.

5.3.1.2 Coarse mesh extraction

In order to extract a mesh with semi-regular connectivity from volume data, we need to first evaluate the volume data itself to determine the topology of the desired isosurface. In turn, this evaluation process facilitates the construction of a coarse mesh with the correct topology. The approach has three main features: guaranteed topology, low memory requirements and adjustable complexity of the initial mesh.

Since volume data can potentially be very large and fill main memory, the task of extracting an isosurface can be time consuming and memory intensive due to the need to compute and maintain the local triangulation per voxel. We want to avoid this costly triangulation step and only store and use a small amount of data to construct the coarsest mesh. An alternate approach for dealing with large volume data is to down-sample the volume through a smoothed pyramid construction and then extract a coarse mesh. The problem with this approach is that it cannot guarantee the topology, e.g., small handles will disappear in the smoothing step, causing a change in the topology of the initial mesh. Instead we work with the original sampling of the volume and leverage the connectivity information inherently represented by voxel adjacency. This allows us to minimize the amount of extra data we need to store for constructing a topologically accurate coarse mesh.

Essentially, we treat the voxel grid as a data structure already representing the surface in an implicit way and we traverse this data structure to extract an accurate coarse mesh. While doing this, we do not compute or store the local triangulation per voxel and instead store a Reeb graph to represent the topological structures of the desired isosurface. Our general approach is based on constructing and traversing a Reeb graph, in order to subsample the volume data and extract a coarse mesh. When extracting a coarse mesh, the user may define the discretization rate of the initial mesh. Alternatively, the algorithm can automatically generate a coarse mesh with the minimal discretization that maintains the topology. This is done by guaranteeing that we include any critical levels of the surface.

5.3.1.3 Distance function propagation and Reeb graph

As presented in Chapter 4, we can build a Reeb graph from the contours associated with a distance function on the surface. For coarse mesh construction, we use a distance function starting from an arbitrary seed point. We use a distance function and not a height function because the contours for the distance function will be closely aligned with the geometry of the underlying surface and thus provided a good sub-sampling of the surface geometry. As our setting is a scalar volume as described in Chapter 2, we define the distance function on surfels. Our notion of distance is very much like Chamfer distance in image processing (also called "chess-board" distance): two surfels are a unit distance apart if they share at least one vertex. We can follow the connectivity of surfels in the volume by using a table look-up to determine edge adjacency with neighboring voxels. We use a priority queue to walk from surfel to surfel sequentially (Fig. 5.17, left). Our algorithm is equivalent to running Dijkstra's algorithm to discover all paths from the source surfel to all other unvisited surfels.

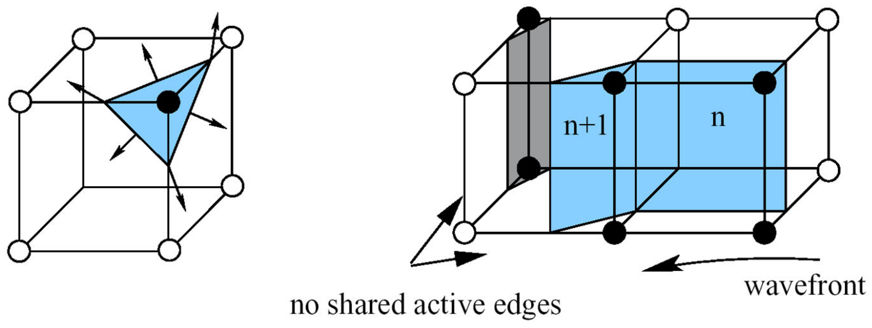


Figure 5.16: *Example of a surfel and following its active edges*
Arrows indicate how to follow active edges from a given surfel (left). On the right we see that the surfel with distance n will propagate the distance $n+1$ across its active edges to the connected surfels. Note that the other surfel in this voxel will only receive a distance when the wave front reaches it.

In order to come up with a consistent ordering within the contours, we use the orientation of the surfels to follow from one surfel of distance d to the next. For a given level d, of the distance function, after a single contour is constructed, we check to make sure that all the valid surfels of level d are part of a contour. If not, we start the contour construction again with one of the unprocessed surfels at level d. As each contour is constructed the appropriate edges of the Reeb graph are added to the previous contours. Note that again, we do not need to keep component information with a distance function traversal from a single seed point. This is due to the fact that no new components will be created during the traversal. In this setting *any* contour node in the Reeb graph that has more than one child node will be a part of cycle and thus a critical level.

5.3.1.4 Mesh construction from the Reeb graph

The Reeb graph provides all the information needed to build the coarse mesh. We use the coarse mesh construction algorithm described in Chapter 4. Contour construction *and* critical level detection can be performed in a sweep algorithm. Once the contours at level d are constructed, we check for a critical level by checking the relative number of children at level $d-1$. Once a critical level is detected it is sub-sampled and stitched together with the previous contour to create a coarse mesh.

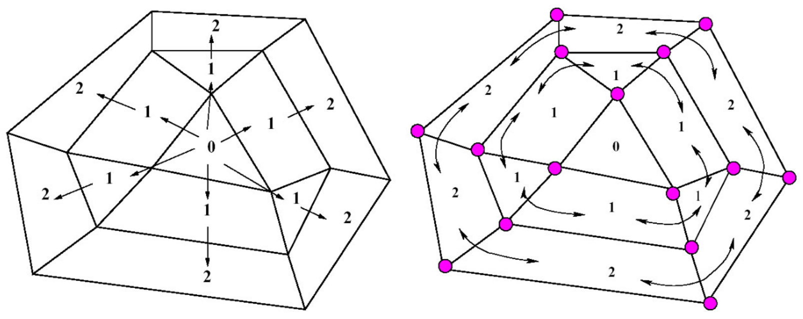


Figure 5.17: *Distance function propagation on a surfel*
Illustration of the distance function overlaid on some surfels (left). Same portion with adjacencies of the Reeb graph (contours) added (right).

5.3.1.5 Discussion

One of the benefits of this approach is the low memory overhead for the Reeb graph representation. In the case of an $O(n^3)$ volume the storage requirement for propagating the distance function is $O(n^2)$, as it depends on the size of the surface. Once again the computation time for the Reeb graph will be $O(n \log n)$ as the Reeb graph construction is equivalent to running a Dijkstra's algorithm over the mesh to propagate distances and then an $O(n)$ time step to collect surfels into contours. The only other data that we need to store for generation of the coarse mesh is dependent on the contours in the Reeb graph and is $O(n)$. Memory overhead for contours is minimized by keeping only, (i) the contours selected to be part of the coarse mesh; (ii) the last contour constructed and (iii) the current contour, which is being evaluated for possible selection.

Conclusion Once we have a coarse mesh, we use quadrisection to create a semi-regular mesh. We use a force based solver to position the vertices to best match the desired isosurface. This work presents a novel algorithm to extract isosurfaces in the form of hierarchical, adaptive semi-regular meshes. It relies on a novel approach to construct a coarsest mesh with the same global topology as the isosurface. The resulting meshes have a natural multi-resolution structure since they are semi-regular, making them suitable for a variety of powerful digital geometry processing algorithms.

5.4 Conclusion

We have presented a variety of applications which demonstrate the usefulness of computational topology algorithms for computer graphics applications. By using these applications we can create more useful geometric models which accurately represent the topology of acquired data and can more readily be used for texture mapping, remeshing, smoothing or other applications which rely on a parameterization of the surface.

Chapter 6

Conclusion

This thesis has presented computational topology algorithm for acquired 2-manifolds used in computer graphics. These algorithms are tailored for the discrete representation of 2-manifolds as triangle meshes or scalar volumes. We have presented algorithms for the identifying and isolating handles within a surface, algorithms for measuring the length of two locally minimal-length non-separating loops for each handle and methods to re-sample the geometry of the surface to retain or simplify handles. We have demonstrated the use of these algorithms through applications which simplify the topology of triangle meshes and scalar volumes. In these applications, we introduced methods for automatically removing handles from isosurfaces and triangle meshes through direct processing of the original volume data or mesh. We have demonstrated the effectiveness of these methods on several complex models. We have also demonstrated that removing topological artifacts is important for many subsequent modeling operations.

6.1 Contributions

The contributions of this thesis are the following:

A robust and efficient method to localize and isolate handles for discrete 2-manifolds. We propose a method where handles are efficiently identified through methods tuned to the discrete setting. The handle identification traversal of the surface is varied for efficiency while guaranteeing that all handles are located. We present a traversal method with a complexity of $O(n \log n)$, with proof that our traversal methods will detect *all* handles during the traversal. Handles can subsequently be efficiently identified during the traversal of the surface as cycles in the augmented Reeb graph as it is incrementally constructed (see next paragraph). Section 4.2.1.1 of Chapter 4 presented these methods in detail and the associated combinatorial proof.

A method to robustly represent the topology of the surface with an *augmented Reeb graph*. We present a method to construct an augmented Reeb graph which stores additional geometric information about the surface to facilitate isolating handles. We present a method to construct the augmented Reeb graph, which guarantees that for each interval of the traversal, each *ribbon* has genus equal to zero. In addition, we guarantee that for each interval, the number of cycles in the augmented Reeb graph matches the genus of the surface traversed thus far. Geometric properties of the surface are encoded in the augmented Reeb graph which allows geometrically succinct handles to be isolated within the original surface. Section 4.2.2.2 of Chapter 4 introduced the augmented Reeb graph and the methods used to guarantee consistency between the number of handles and the number of cycles in the graph.

A method to find two locally minimal-length non-separating cycles for each handle. This thesis introduces a simple measure of handle size to be the length of two transverse non-separating cycles. The locally minimal-length non-separating cycles are detected efficiently for handles of the surface. We present a proof that we find two discrete locally minimal-length non-separating cycles with a complexity comparable to related approaches. See Section 4.3.2 in Chapter 4 for more details.

A simple method to simplify the topology for volume data and triangle meshes which preserves the local geometry as much as possible. Cutting the surface along the locally minimal-length non-separating cycle will reduce the genus of the model while retaining as much of the fine geometrical detail as possible. By using the smaller of the two non-separating cycles for each handle, the topology of the surface is only modified in a small local region. This targeted approach to modifying the topology preserves the fine geometry of the surface as much as possible. We propose a simple method to simplify the topology of triangle meshes and isosurfaces. Refer to Section 4.4.1 in Chapter 4 for more details. In particular, for isosurface topology simplification, to remove a handle, we alter the scalar values of the volume, thus indirectly modifying the isosurface. Since isosurfaces are always manifold, operating on the volume is robust. Also, by operating on the volume directly, we avoid computing an expensive triangle mesh and never compute or store floating point values to represent the geometric position of the vertices of the surface. Since our algorithm creates a topologically clean volume, this volume can then be used for surface extraction or other applications [48] that depend on a topologically accurate volumetric representation.

An out-of-core method for topology simplification for volume data. Complex 3D models are represented by large volumes that may not fit entirely in main memory. The model in Figure 1.7 is from a $885 \times 709 \times 736$ grid, and much larger models now exist [59]. The isosurface topology simplification algorithm is applied to such volumes using out-of-core methods. The algorithm employs a sweep method to read the volume in planar slices, so the data access pattern is highly regular. We encode surface topology as the sweep progresses using an augmented Reeb graph, requiring only a few slices in memory at any given time. For some large handles, previous slices may need to be reloaded to perform simplification. However, simplification can be performed on small segments of the volume one at a time, resulting in a purely out-of-core algorithm. The details of this out-of-core method are presented in Section 5.2.1 of Chapter 5.

6.2 Future explorations

When simplifying the topology of a surface topological obstructions can inhibit the effectiveness of our simple methods. Our work on simplifying the topology of triangle meshes in particular can create surfaces that are self-intersecting after the loop closure operation. It would be worth exploring loop closure operations that avoid self-intersections. In addition, the mesh setting poses challenges for out-of-core approaches and it would be interesting to explore ordering the data such that it facilitates out-of-core processing.

For our purposes we have found that our combinatorial choices for isolating handles are sufficient. However, it is possible to check other pairings of previous components in order to check other combinatorial choices for cycles in the Reeb graph and their corresponding handles. For example, by defining distances on the Reeb graph such that every node is distance one from all adjacent neighbors, another combinatorial choice would be all pairings in the Reeb graph within a user defined ϵ distance. Exploring these kinds of combinatorial choices, especially in light of finding globally minimum-length non-separating cycles, are an interesting avenue for future work. In addition, as mentioned in Chapter 4 it may be interesting to explore constructing shortest-length non-separating cycles that do not wrap around a handle at all. It is conceivable that a geometric constraint could be constructed to prevent winding for the initial construction of the loop L. Finding shortest paths to be arbitrary paths over a surface is another interesting extension to consider for this work.

Also of interest is the task of incorporating algorithms for analyzing the topology earlier on in the surface acquisition pipe-line. For example, during the segmentation of MRI data, it would be useful to create a tool that only allows a doctor to segment the data such that a surface with the correct topology is generated.

Another area of future work is to improve the local surface geometry after handle removal. The handles removed in our test examples were so small as to be nearly invisible, so we did not consider smoothing to be important. However, it is conceivable that some settings and applications could require the removal of topological features of a more substantial size. Since we have information about the local region affected by the loop closure, we could smooth the newly inserted surface. For example, this smoothing would improve the visual appearance of the regions bounded by the arms in the genus zero Buddha (Figure 5.2.1.4). In range data reconstruction algorithms, it is already common to smooth the unscanned, filled-in regions of the surface [17, 19]. For larger handles, it may be desirable to use more accurate approximations of true geodesic non-separating cycles rather than the discrete graph approximation. More generally, we are interested in exploring alternative methods for measuring handle size. We have experimented with one such measure with success, however other criteria could be explored for varying applications.

To explore data such as MRI, some systems allow the isosurface value to be varied interactively. Efficiently removing handles in the changing isosurface is an interesting problem. Perhaps it is possible to pre-process the volume to remove topological artifacts for a range of isosurface values. For such a setting, the work of Zomorodian [79] is promising.

Bibliography

[1] Geomagic studio 3.0, 2000. Raindrop Geomagic.

[2] Paraform 2.0, 2000. Paraform Inc.

[3] Ergun Akleman and Jianer Chen. Guaranteeing 2-manifold property for meshes. In *Proceedings of the International Conference on Shape Modeling and Applications*, 1998.

[4] Ergun Akleman, Jianer Chen, and Vinod Srinivasan. A new paradigm for changing topology of 2-manifold polygonal meshes. In *Pacific Graphics'2000*, 2000.

[5] P. Aleksandrov. *Combinatorial Topology*, volume 1. Graylock Press, 1956.

[6] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. *Proceedings of SIGGRAPH*, pages 195–202, 2001.

[7] Nina Amenta, Sunghee Choi, Tamal Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. *ACM Symposium on Computational Geometry*, pages 213–222, 2000.

[8] M Attene, S. Biasotti, and M. Spagnuolo. Re-meshing techniques for topological analysis. In *International Conference of Shape Modeling and Applications*, pages 142–151, Genova, Italy, May 2001. IEEE Computer Society.

[9] U. Axen. Computing morse functions on triangulated manifolds. In *Proceedings of the SIAM Symposium on Discrete Algorithms (SODA)*, 1999.

[10] U. Axen and H. Edelsbrunner. Auditory morse analysis of triangulated manifolds. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 223–236. Springer-Verlag, Berlin, Germany, 1998.

[11] C. Bajaj, E. Coyle, and K. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Models and Image Processing*, 58(6):524–543, 1996.

[12] C. Bajaj and D. Schikore. Topology preserving data simplification with error bounds. In *Computer and Graphics*, pages 3–12, 1998.

[13] Stephan Bischoff and Leif Kobbelt. Isosurface reconstruction with topology control. In *Proceedings of Pacific Graphics*, pages 246–255, 2002.

[14] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *Symposium on Discrete Algorithms*, pages 918–926, 2000.

[15] Éric Colin de Verdière and Francis Lazarus. Optimal system of loops on an orientable surface. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 627–636, Vancouver, Canada, November 2002.

[16] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[17] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH*, pages 303–312, 1996.

[18] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH 96*, pages 303–312, 1996.

[19] James Davis, Stephen R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing, Visualization, and Transmission*, 2002.

[20] Mathieu Desbrun, editor. *A Tutorial on Mesh Compression*. Course Notes. Institute for Mathematics and Applications (IMA, UMN), Minneapolis, May 2001.

[21] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proceedings of SIGGRAPH 99*, pages 317–324, 1999.

[22] T. K. Dey, H. Edelsbrunner, and S. Guha. Computational topology. *Advances in Discrete and Computational Geometry*, 223:109–143, 1999.

[23] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S.)*, 66:23–45, 1999.

[24] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH 95*, pages 173–182, 1995.

[25] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. ACM Sympos. Comput. Geom.*, pages 98–101, 2003.

[26] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse complexes for piecewise linear 2-manifolds. In *Proc. 17th Ann. ACM Sympos. Comput. Geom.*, pages 70–79, 2001.

[27] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Proc. 41st IEEE Sympos. Found. Comput. Sci.*, pages 454–463, 2000.

[28] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[29] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, 1991.

[30] J. El-Sana and A. Varshney. Controlled simplification of genus for polygonal models. *Proceedings of IEEE Visualization*, pages 403–412, 1997.

[31] Jeff Erickson and Sariel Har-Peled. Optimally cutting a surface into a disk. In *ACM SOCG*, pages 244–253, 2002.

[32] Michael S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.

[33] G.K. Francis and J.R. Weeks. Conway's zip proof. *Amer. Math. Monthly*, 1999.

[34] H. Fuchs, Z. Kedmen, and S. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, 1977.

[35] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH*, pages 209–216, 1997.

[36] Thomas Gerstner and Renato Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. *Visualization* '00 Proceedings , 2000.

[37] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *Proceedings of SIGGRAPH*, pages 355–361, 2002.

[38] Igor Guskov, Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Hybrid meshes. In *ACM SOCG*, 2002.

[39] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. *Proceedings of SIGGRAPH 00*, 2000.

[40] Igor Guskov and Zoë Wood. Topological noise removal. *Graphics Interface*, pages 19–26, June 2001.

[41] John C. Hart. Computational topology for shape modeling. In *Shape Modeling International*, pages 36–45, 1999.

[42] John C. Hart. Fair morse functions for extracting the topological structure of a surface mesh. Technical Report 2318, Univ. of Illinois Urbana-Champaign, 2003.

[43] Taosong He, Lichan Hong, Amitabh Varshney, and Sidney W. Wang. Controlled Topology Simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, 1996.

[44] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. *Proceedings of SIGGRAPH*, pages 203–212, 2001.

[45] A. Hilton, A. J. Toddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. *Fourth European Conference on Computer Vision*, I:117–126, 1996.

[46] Hugues Hoppe. Progressive meshes. *Proceedings of SIGGRAPH*, pages 99–108, 1996.

[47] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):71–78, 1992.

[48] Sylvain Jaume, Benoit Macq, and Simon K. Warfield. Labeling the brain surface using a deformable multiresolution mesh. In *MICCAI*, 2002.

[49] Elena Kartasheva. The algorithm for automatic cutting of three-dimensional polyhedron of h-genus. In *International Conference of Shape Modeling and Applications*, pages 26–33, Aizu, Japan, March 1999. IEEE Computer Society.

[50] A. Kaufman. Scan-conversion of polygons. *Proceedings of Eurographics*, pages 197–208, 1987.

[51] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. *Proceedings of SIGGRAPH*, pages 271–278, 2000.

[52] Ron Kikinis, Martha E. Shenton, Dan V. Iosifescu, Robert W. McCarley, Pai Saiviroonporn, Hiroto H. Hokama, Andre Robatino, David Metcalf, Cynthia G. Wible, Chiara M. Portas, Robert Donnino, and Ferenc A. Jolesz. A digital brain atlas for surgical planning, model driven segmentation and teaching. *IEEE Transactions on Visualization and Computer Graphics*, 1996.

[53] M. Van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *13th ACM Symposium on Computational Geometry*, pages 212–219, 1997.

[54] J.-O. Lachaud. Topologically Defined Iso-surfaces. In *Proc. 6th Discrete Geometry for Computer Imagery (DGCI)*, volume 1176 of *Lecture Notes in Computer Science*, pages 245–256. Springer-Verlag, 1996.

[55] Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *ACM SOCG*, pages 80–89, 2001.

[56] Francis Lazarus and Anne Verroust. Level set diagrams of polyhedral objects. In *ACM Solid Modeling'99*, Ann Arbor, Michigan, USA, June 1999.

[57] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, 1998.

[58] Marc Levoy. The digital michelangelo project. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, Ottawa, October 1999.

[59] Marc Levoy and others. The digital michelangelo project: 3d scanning of large statues. *Proceedings of SIGGRAPH*, pages 131–144, 2000.

[60] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Proceedings of SIGGRAPH*, 21(4):163–169, 1987.

[61] W.S. Massey. *Algebraic Topology: An Introduction*. Harcourt, Brace & World, Inc., 1967.

[62] Yukio Matsumoto. *An Introduction to Morse Theory*. American Mathematical Society, 1997.

[63] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In Hans-Christian Hege and Konrad Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

[64] J. Milnor. *Morse Theory*. Princeton University Press, 1963.

[65] J. Munkres. *Topology*. Prentice Hall, 2000.

[66] F.S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. Research Report 99-37, Georgia Tech, 1999.

[67] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. *Proceedings of SIGGRAPH*, pages 217–224, 1997.

[68] G. Reeb. Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus Acad. Sciences Paris*, pages 847–849, 1946.

[69] Stephen Reynolds and Julia Johnson. web page: http://geology.asu.edu/˜reynolds/topo_gallery/topo_gallery.htm, 2003.

[70] P. Sander, J. Snyder, S. Gortler, and H. Hoppe. Texture mapping progressive meshes. *Proceedings of SIGGRAPH*, pages 409–416, 2001.

[71] Peter Schröder and Wim Sweldens, editors. *Digital Geometry Processing*. Course Notes. ACM Siggraph, 2001.

[72] David W. Shattuck and Richard M. Leahy. Automated graph based analysis and correction of cortical volume topology. *IEEE Transaction on Medical Imaging*, 2001.

[73] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on Applied Computational Geometry (Philadelphia, Pennsylvania)*, pages 124–133. Association for Computing Machinery, May 1996.

[74] Yoshihisa Shinagawa, Yannick L. Kergosien, and Tosiyasu L. Kunii. Surface coding based on morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, 1991.

[75] Yoshihisa Shinagawa and Tosiyasu L. Kunii. Constructing a reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11(6):44–51, 1991.

[76] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Proceedings of SIGGRAPH '97*, pages 279–286, 1997.

[77] S. Takahashi, T. lkeda, Y. Shinagawa, T.L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. In *Eurographics'95*, pages 181–192, 1995.

[78] Zoë Wood, Mathieu Desbrun, Peter Schröder, and David Breen. Semi-regular mesh extraction from volumes. In *Proceedings of IEEE Visualization*, pages 275–282, 2000.

[79] Afra Zomorodian. *Computing and Comprehending Topology: Persistence and Hierarchical Morse Complexes*. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2001.

[80] Denis Zorin and Peter Schröder, editors. *Subdivision for Modeling and Animation*. Course Notes. ACM SIGGRAPH, 1999.

[81] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. *Proceedings of SIGGRAPH 97*, pages 259–268, 1997.