SEMI-REGULAR MESH EXTRACTION FROM VOLUMES

Zoë Justine Wood

Technical Report CaltechCSTR 2000.006

for the
Computer Science

California Institute of Technology
Pasadena, California

2000

Abstract

We present a novel method to extract iso-surfaces from distance volumes. It generates high quality semi-regular multiresolution meshes of arbitrary topology. Our technique proceeds in two stages. First, a very coarse mesh with guaranteed topology is extracted. Subsequently an iterative multi-scale force-based solver refines the initial mesh into a semi-regular mesh with geometrically adaptive sampling rate and good aspect ratio triangles. The coarse mesh extraction is performed using a new approach we call *surface wavefront propagation*. Given a source voxel of the iso-surface, a set of discrete iso-distance rings are rapidly built and connected while respecting the topology of the iso-surface implied by the data. Subsequent multi-scale refinement is driven by a simple force-based solver designed to combine good iso-surface fit and high quality sampling through reparameterization. In contrast to the Marching Cubes technique our output meshes adapt gracefully to the iso-surface geometry, have a natural multiresolution structure and good aspect ratio triangles, as demonstrated with a number of examples.

Table of Contents

List of Figures

Chapter 1

Introduction

Medical visualization gives doctors the power to see into the human body and identify problems ranging from tumors to torn ligaments. As medical imaging hardware improves, the resolution of acquired data grows at rapid rates. Unfortunately many existing algorithms for visualizing and understanding this data are not scalable. This is particularly evident with *volume data*,[1] the typical output of imaging equipment such as MRI and CT scanners. Typical data from these scanners consists of scalar density regularly sampled over a volume. The user is often interested in viewing the boundary of a specific tissue, hence a specific surface within the volume data. This can be achieved by extracting an *iso-surface*: the locus of points in the volume that map to a given iso-value. For example, the user may be interested in viewing the exterior boundary of the liver. This surface can be extracted from the volume data by finding a particular iso-surface.

The predominant algorithm for iso-surface extraction, Marching Cubes [40], extracts a surface in the form of a triangle mesh. The algorithm computes a local triangulation within each voxel of the volume that contains the surface, resulting in a uniform resolution mesh. Uniform sampling generates large meshes that are problematic in many applications. Datasets of several gigabytes are found in medicine, and the size of Marching Cubes meshes may reach several million polygons. Often much smaller meshes adequately describe the surface since the uniform sampling approach results in oversampling portions of the iso-surface. To better represent the surface, the sampling rate of the extracted surface should vary over the surface, such that details are captured while the representation remains compact. Large Marching Cubes meshes are costly to render on even the most powerful graphics platforms, and often interactive applications are intractable. The excessively large meshes encumber other downstream applications as well, e.g., denoising, finite element simulations, and network transmission. One avenue to deal with scalability of large meshes is a multiresolution approach.

The multiresolution paradigm encompasses a class of surface representations that scale well under increasing geometric complexity and arbitrary topology [12]. As their name implies, multiresolution representations organize data into different levels of reso-

[1] Volume data is defined here as a discrete three dimensional field; it encodes samples of a function over a volume. A typical example is density data: the function is scalar-valued, and the samples are on a regular Cartesian grid.

lution.[2] Multiresolution approaches employ a hierarchy that encodes a coarse representation at the root and aggregate details at subsequent levels. Ideally, at any depth of the hierarchy, the accumulated geometry is the best possible representation of the surface given the available number of samples at that level.

Multiresolution meshes have many benefits:

Scalability A hierarchical mesh representation can be examined and manipulated at the appropriate level of detail. Level of detail can be chosen based on memory and time budgets, or user specification.

Transmission Progressive transmission over a network is very desirable and possible with multiresolution. The presence of a hierarchy defines a breadth-first traversal strategy, in which the coarse mesh is sent first, and details follow.

Compression For each level of a multiresolution surface, parameterization and connectivity can be inferred from previous levels. This fact gives rise to superior compression algorithms.

Modification A multiresolution surface can be edited at different levels to achieve different scales of effect on the surface. In particular, editing the surface at a coarse level has a more global effect while fine level editing will only affect a very small local area of the surface.

We observe that these representations are ideal for extracted iso-surfaces, particularly as volumetric datasets grow in size.

One approach to obtaining a multiresolution surface is repeated decimation of an initial fine mesh. This approach could follow the application of Marching Cubes, to address the oversampling problem [23]. Unfortunately, common mesh simplification algorithms have large memory footprints [24, 16], and are impractical for decimating meshes with millions of polygons (see [38, 37] for an approach to tackle this situation). Alternatively one may apply classic iso-surface extraction techniques such as Marching Cubes to hierarchical (filtered and down-sampled) volumetric representations [56, 2]. Unfortunately, it is difficult to guarantee the topology of the mesh extracted from the simplified volume.

An alternative to building a hierarchy through decimation is *remeshing* [12, 32, 36, 31, 21]. Although previous work is compelling, it is inefficient and inelegant in the setting of iso-surface extraction from volumes: such approaches first extract a poor, oversampled mesh, and then repair the problem through remeshing. In contrast, we opt for the *direct* extraction of an adaptively sampled hierarchical iso-surface mesh. Our approach adaptively samples the iso-surface and builds a mesh with good aspect ratio triangles within a multiresolution structure. This is achieved through a coarse-to-fine generation procedure which produces an adaptive *semi-regular* mesh.

Semi-regular meshes are well known from the subdivision setting [60]. A semi-regular mesh consists of a coarsest level triangle mesh which is recursively refined by quadrisecting each triangle. The resulting type of meshes have a *semi-regular* structure, i.e, all the

[2] Multiresolution representations are intimately connected to wavelets and their application to representing mathematical objects at different granularities. For example, wavelet techniques can be applied to images in order to represent them at different levels of resolution.

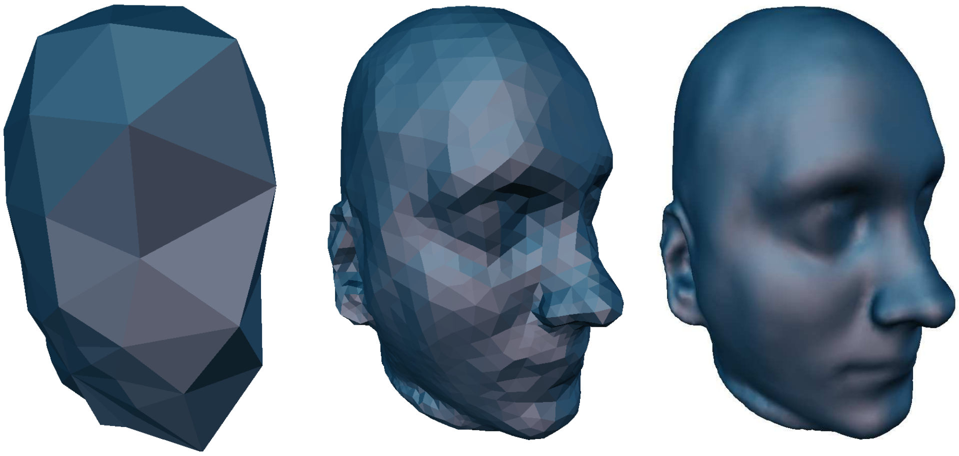

Figure 1.1: *Example of various levels of an extraction of adaptive semi-regular meshes from a volume using our algorithm. On the left is a coarse resolution version of the surface, followed in the middle by an intermediate version. Finally the finest resolution surface is on the right.*

vertices introduced via quadrisection have valence six. Since this surface representation has an inherent multiresolution structure, it enjoys all the benefits of multiresolution described above. Specifically, semi-regular meshes have theoretical, applied, and implementation benefits:

Theory *Specific* connectivity structure improves compression efficiency, data structure compactness, and analytic error estimates in various algorithms [28, 21, 61, 5].

Applications Many applications that are "downstream" with respect to iso-surface extraction are more efficient when using semi-regular meshes (these include editing [61], finite element simulations [5], and progressive transmission [28] among many others). Thus semi-regular meshes are a desirable format for representing iso-surfaces.

Implementation The mesh hierarchy is represented with a quad-tree data structure, thus implementations are simple, elegant, and efficient.

This thesis presents an algorithm for the extraction of semi-regular multiresolution iso-surface meshes directly from volume data. Figure 1.1 shows examples of a multiresolution mesh extracted from a distance volume with our algorithm. Typically the meshes produced by our algorithm are more compact. This has significant practical relevance: for example it is easier to design systems for visualizing medical data online. However, application of our algorithm is by no means confined to medical applications. For example, 3D scanners often combine scans from multiple viewpoints into one volume data set. Engineers, designers, artists, and entertainment specialists require rapid and efficient manipulation of detailed surfaces acquired from such scanned data. Our algorithm provides these users with a compact and versatile representation, opening the door for them to use a variety of down-stream applications with ease.

Chapter 2

Related Work and Algorithm Overview

This thesis addresses the application of multiresolution representations to the iso-surface extraction problem. Iso-surface extraction is a fundamental problem in scientific visualization and computer graphics. It has been investigated extensively, especially in the last 13 years since the publication of the very successful Marching Cubes extraction algorithm. Marching Cubes has been successful partially due to its simplicity. It is a very reliable algorithm and generates an accurate [1] representation of the expected surface. However, Marching Cubes meshes are not the most efficient representation of the desired surface.

As volume data has grown in size, the resulting storage concerns have been addressed through volume compression, streaming extraction techniques, and hierarchical structures [59, 39, 4]. However, these techniques do not solve the problem that the resulting mesh will still be excessively large and inefficiently represented due to uniform sampling. While some algorithms have focused on fixing the meshes produced by Marching Cubes using mesh decimation and multiresolution remeshing, others have focused on improving the meshes by replacing the Marching Cubes extraction method by other techniques, for example deformable models. Both of these approaches have problematic aspects. The former is a costly post-process while the latter techniques are usually limited in the type of data they can extract. In particular, extraction algorithms that use deformable models require user input to extract surfaces with handles. Instead, our approach leverages knowledge from both of these areas and improves both the final representation of the resulting mesh *and* the extraction algorithm.

[1] Marching Cubes produces an accurate surface, assuming no apriori knowledge regarding how the sampling of the volume was generated. For example, if the sampling was generated from a higher order function from a numerical simulation. In this case, the linear basis functions used for interpolation during a Marching Cubes mesh construction will not be sufficient to accurately represent the surface.

2.1 Traditional Methods and Multiresolution Models

Our approach is motivated by the desire to directly extract a multiresolution representation, as described in Chapter 1. Multiresolution comes in two fundamental flavors: (a) coarse to fine: constructions which are based on classical notions of multiresolution as they appear in wavelets and subdivision; and (b) fine to coarse: constructions based on mesh simplification. The former comes with a rich mathematical structure which can be leveraged for many applications [52, 53, 47]. The latter is applied when a very fine mesh has already been constructed. Such meshes arise from 3D scanning or Marching Cubes iso-surface extractions. As mentioned in the Introduction to this thesis, post-process decimation of large meshes is not an elegant solution for converting them to a multiresolution representation. In addition, most decimation algorithms create highly irregular meshes and miss out on some of the advantages of output constructed using a coarse to fine approach. For example, in contrast to semi-regular meshes, there is little correspondence between the connectivity and parameter information of the original arbitrary connectivity mesh and a decimated version, making them more difficult to compress [28]. Hence we prefer a coarse to fine approach to construct semi-regular—or subdivision connectivity—meshes. Our goal is the direct construction of such meshes for iso-surface extraction. We focus on direct extraction so that we avoid the unnecessary step of extracting a oversampled mesh and then decimating it in order to fix it.

2.2 Deformable Models

Since our approach is focused on directly extracting a good mesh, we have replaced the Marching Cubes extraction algorithm. Our approach is closely related to surface extraction based on deformable models [43, 26, 42, 46, 31]. In these approaches a potential function is defined and the surface is found by formulating a finite element problem whose minimum energy solution is the desired surface. Aside from the interpolation constraints the energy minimization typically employs additional terms to guarantee uniqueness and ensure a smooth solution. Our approach is similar to finding a minimum energy solution in a signed distance function potential. However, instead of deriving our forces from an energy field, we tailor the forces such that they lead to a good reconstruction subject to smoothness criteria. The main differences lie in the control we exert over the connectivity of the resulting mesh and in the minimization process we use. Instead of using a thin-plate functional we use a balloon force [6] approach coupled with a novel *reparameterization* force.

The largest advantage of our algorithm compared to other deformable model approaches is our ability to extract a surface of arbitrary topology. Almost all previous approaches assume that the global topology of the iso-surface is known apriori since the initial mesh for the finite element solver must have the correct topology [43, 46, 31, 42]. These previous approaches rely on user input to determine the appropriate global topology for the initial mesh. In practice this has meant that most such algorithms only dealt with the extraction of objects homeomorphic to a sphere. In contrast, our approach automatically extracts a surface with the correct global topology without depending on user input to determine the topology. Designing a solver which does not require the correct topology initially and instead topologically modifies the mesh as the algorithm proceeds is possible, but rather delicate [34]. Instead we opt for a robust algorithm which initially extracts a topologically correct coarse mesh from the volume data. Subsequent refinement is always performed through quadrisection, giving us the desired multiresolution structure.

2.3 Topology

Our algorithm extracts a coarse mesh with the same topology of the desired iso-surface. We examined a variety of approaches used to code the topology of a surface, however, none of the existing techniques accomplished what we needed. For example, one possible method to determine the topology of a surface is to construct the original triangulation of the surface and then compute the Euler characteristic of this triangulated surface (see Appendix A for more information about the Euler characteristic). By using the Euler characteristic, a coarse surface with the same genus could be selected as the initial coarse mesh (for example a tetrahedron for any surface that is homeomorphic to a sphere). However, this approach suffers from the same shortcomings as mesh decimation (mentioned in section 2.1). It is a time and memory intensive post-process. We require an approach that does not construct a full triangulation of the surface and does not depend on apriori knowledge of the topology of the surface. We have derived our own algorithm that meets our criteria and overcomes the shortcomings of the related approaches.

2.3.1 Digital geometry, Morse Theory and Reeb Graphs

Our coarse extraction algorithm makes use of the volume data structure in order to avoid a costly extraction of a uniform triangulation. We use volumes that are sampled on a Cartesian grid and we rely on the connectivity relationship of the grid elements to extract a topologically accurate mesh (see Chapter 4). Using the adjacency relationships of the grid we traverse the surface and store a representation of the surface's connectivity in a *topological graph*. This traversal and graph construction is related to work done by Lachaud [33] on topologically defined iso-surfaces.

Lachaud proposes an alternative to the Marching Cubes extraction: construction of the iso-surface from a graph. He constructs this graph using digital geometry definitions of adjacencies and connectedness in the volume. His graph elements are called loops which are small oriented pieces of the surface in a voxel. He proves the topological equivalence of the triangulation that is created using the loop graph and the marching cubes mesh for the same volume. We rely on this proof in our work, since the building blocks of our topological graph, *Surfels* (see 3.3), are equivalent to Lachaud's loops. However, the problem with Lachaud's approach is he triangulates every loop to generate his final surface. Instead we only use a subset of the Surfel connectivity graph in order to extract a coarse mesh. However, our assertions that our graph represents the same topology as a Marching Cubes mesh rely on the equivalence of Surfels and Lachaud's loops.

Other works concerned with coding the topology of a surface are Morse Theory and Reeb graphs [51, 49, 50]. Morse Theory is used to describe the minima of functionals on an infinite dimensional space of paths. By applying this theory to differentiable manifolds, the minima of a functional can be used to characterize topological features. In brief, Morse theory deals with defining critical points and their relationship to the topology of surfaces. Critical points are defined as the regions of the surface where the gradient is zero. However, the critical points alone do not uniquely identify the embedding of the manifold in space. This means that the same set of critical points can be interpreted as having different topology. The Reeb graph addresses some of these problems, as it encodes the connectivity of the critical points of a surface. Shinagawa has done work on constructing the Reeb graph from cross sections [50] which is reminiscent of our approach. However, the Reeb graph alone cannot completely capture the topology of a surface (in particular, it has degenerate cases see Fig. 3.1). Our approach is similar to the Reeb graph in terms of using contours to determine the topology of the surface. However, the topological graph we construct from contours *uniquely* determines the topology of the surface. We discuss this further in Chapter 3.

2.3.2 Distance Iso-contours

Our coarse mesh extraction approach was also inspired by work on computing level sets on manifolds, specifically polygonal meshes [29, 48]. Particularly of interest is the computation of the geodesic graph used to extract skeletal curves [35, 55]. Skeletal curves are another method to encode a surfaces topology, however, they suffer from similar degeneracy problems as the Reeb graph. We use a discrete distance computation related to these ideas, however, we apply these ideas to iso-surfaces defined only implicitly. Most importantly, we expand these ideas to the volume setting. Our algorithm propagates a discrete distance without constructing a triangulation of the surface. Instead, we use the connectivity relationship of voxels in the volume to build a graph representing the surface. Distances are then propagated on this graph, creating a discrete distance graph, similar to the geodesic graph. This graph is later used to create iso-contours of our surface that correctly encode the topology of the surface (see Chapter 3 for more information).

2.4 Signed Distance Volumes

Signed distance volumes are utilized in a variety of applications [8, 7, 17, 45, 57]. 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. While our coarsest mesh extraction algorithm works with arbitrary scalar volume datasets, our solver explicitly requires a distance volume. Our strategy for computing distance volumes involves calculating the exact shortest Euclidean distance within a narrow band around the surface. The information in the narrow band is then swept out to the remaining voxels using a Fast Marching Method [48]. Distance volumes for MRI and CT data are generated by fitting a level set model to the desired iso-surface creating a smooth segmentation of the input data [41, 58].

Due to the fact that iso-surface extraction is such a fundamental problem, it has been actively worked on and improved over the years. However, none of the existing techniques can directly extract a multiresolution representation of the isosurface. Building on ideas

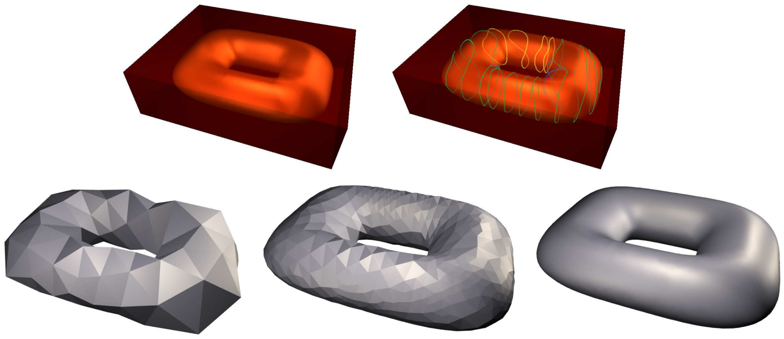

Figure 2.1: *Overview of our algorithm. Given a volume and a particular iso-value of interest (top-left), a set of topologically faithful rings is constructed (top right). Stitching them together creates the coarsest level mesh for the solver (bottom left). Adaptive refinement constructs a better and better fit with a mesh having semi-regular (subdivision) connectivity and an explicit multiresolution structure (bottom).*

from these related works we now present an overview of our semi-regular mesh extraction algorithm.

2.5 Algorithm Overview

Since our algorithm directly extracts a semi-regular mesh, we will start with the volume data and extract a coarse representation of the surface without first extracting a fine mesh. In addition, our algorithm handles arbitrary topology, therefore our initial irregular connectivity mesh must have the same global topology as the iso-surface we wish to extract (Fig. 2.1, left). This first stage of our algorithm works for arbitrary scalar volumes with well defined iso-surfaces. Our method has very low memory overhead, enabling us to handle very large datasets.

In the next step of our algorithm the mesh is refined and its geometry optimized (Fig. 2.1, lower right). In addition to the regular and hierarchical structure of a semi-regular mesh, the output of our algorithm should have good aspect ratio triangles, a larger number of samples where the surface has more detail and a smoothly varying number of samples across the surface of the mesh. In this stage of the algorithm, aspect ratios and sizes of triangles are controlled through adaptive quadrisection and additional *reparameterization force* terms. Since our algorithm proceeds from coarser to finer resolutions simple multi-scale methods are easily used. In particular we solve successively for the best fitting mesh at increasing resolutions using an upsampling of a coarser solution as the starting guess for an iterative solver at the next finer level. For this optimization stage of the algorithm we require a distance volume for the desired iso-surface.

In summary, novel aspects of our algorithm include:

- direct extraction of semi-regular meshes from volume data;
- a new and fast method to extract a topologically accurate coarse mesh with low memory requirements, suitable for large datasets;

- an improved force-based approach to quickly converge to a refined mesh that adaptively fits the data with good aspect ratio triangles.

Since our method does not change the global topology of the mesh during refinement it is more suitable for iso-surfaces whose topological complexity is significantly lower than their geometric complexity. For example, the surface of the brain satisfies this criterion, while intricate vessel networks do not.

Chapter 3

Coarse Extraction

3.1 Coarse Mesh Extraction

One of the goals of this thesis is to present an algorithm to extract surfaces of arbitrary topology. We accomplish this by first evaluating the volume data itself to determine the topology of the desired iso-surface. In turn, this evaluation process facilitates the construction of a coarse mesh with the correct topology. This approach has three main features: guaranteed topology, low memory requirement, and adjustable complexity of the initial mesh.

3.2 Problem Statement

Our first task is the extraction of a topologically accurate coarse mesh. Since volume data can potentially be very large and fill main memory, the task of extracting an iso-surface 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.

Since the volume data is represented as a regular grid, the location of each corner of a voxel in 3 dimensional space is represented by an x, y, z triple. This triple is also used to index the scalar value associated with that grid sample in space. Neighborhood relationships between voxels are easily traversed as a voxel's neighbors are reached by simply increasing or decreasing any of the index triple's values. Essentially, we treat the voxel grid as a data structure already representing our 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 small amount

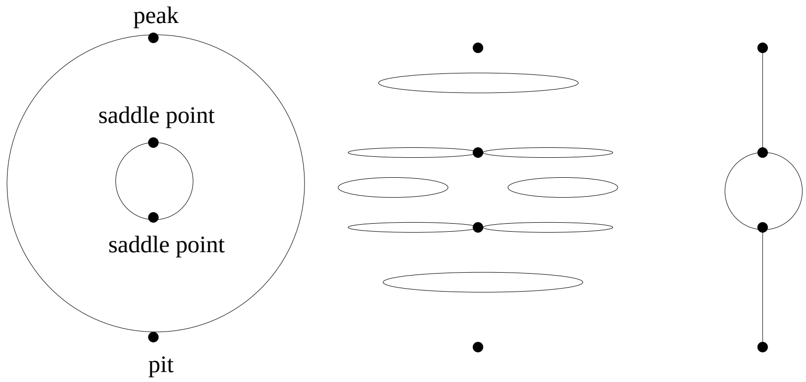

Figure 3.1: *The critical points of a torus (left). Cross sections and critical points of the same torus (middle) and the Reeb graph of the torus (right).*

of additional information in order to represent the topological structures of the desired iso-surface.

Our general approach is based on constructing and traversing a topological graph of the surface, 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 maintain the Euler Characteristic of the original surface (see appendix A for more information). Our approach is explained in the remainder of this section.

3.3 General Approach

In order to construct a topologically accurate coarse representation of a surface, one could imagine intelligently slicing the surface at specific locations and then tiling these slices together. This concept is similar to representing a surface with a Reeb graph, where the critical points of a surface along with planar cross sections of the surface are stored to represent the topology of the surfaces (Fig. 3.1).

However, Reeb graphs have limitations and can have degeneracies due to the way cross sections are acquired. For example, consider the planar cross sections of the torus in Fig. 3.1. Now consider if the torus was laying on its side, like a donut sitting on a tilted table. If we take cross sections of this torus at constant heights, we would derive a degenerate Reeb graph, see Fig. 3.2. Additional coding information is required in order to remedy this and reconstruct the correct surface. Specifically, Shinagawa's [49, 50] approach requires apriori information about the number of handles. In contrast, our approach automatically reconstructs the correct surface without prior knowledge about the topology.

The biggest difference between the Reeb graph and our approach is that the contours we use to represent the topology are not defined by a constant height function. Instead our contours are defined by a distance function defined *on* the surface. This means that

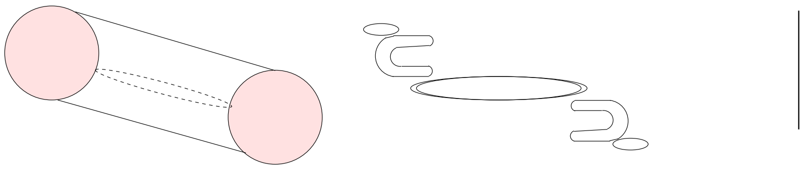

Figure 3.2: *A torus on its side and tilted(left). The cross sections of this torus (middle). The degenerate Reeb graph (right)*

our contours will always correspond to the geometry of the surface. More importantly, this means that our contours can represent the Euler characteristic of specific regions of our surface. By examining the way these geometric contours are connected to one another we can always uniquely encode a topological graph of our surface. Section 3.5 will discuss this process in more detail.

Our technique to acquire contours is related to computing the geodesics of a surface. The key to this approach is intelligent selection of slices to accurately capture the topology of the surface, while minimizing the computational complexity. This can be done by constructing an accurate topological graph of the surface that ignores redundant cross-sections. Our algorithm uses such an approach and a formal proof of this method can be found in appendix A.

Consider a surface intersected by a Cartesian 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 distance volume at that point in space. The surface is defined as the zero iso-contour of the volume, all values in the volume where $F(i) = 0$. The surface will be pierced by the edges of the Cartesian grid, creating a collection of patches which we denote *Surfels*, for surface elements (Fig. 3.3, left). The edges which pierce the surface are denoted *active edges*. They have the property that their endpoints lie on opposite sides of the surface (the endpoint vertices have scalar values in opposite binary sets). Edge endpoints are considered either outside the surface if $F(i) \geq 0$, or inside the surface if $F(I) < 0$. Since "outside" is not defined with a strict inequality an edge endpoint cannot degenerately lie *on* the surface. This definition of inside and outside is equivalent to using an epsilon perturbation to move the surface so that it only intersects the Cartesian grid on the grid's edges. The active edges intersect the surface at points called *nodes*. For the case of an iso-surface embedded in volume data, the resulting graph will be regular in the sense that all nodes are valence four, since a piercing edge of the Cartesian grid is shared by four Surfels.

Given this setting we return to the original goal of generating slices to subsample the surface while retaining the original topology. To establish the topology of the surface we code the Euler characteristic of important regions of our surface. In order to code the Euler characteristic we need to traverse our surface and establish connectivity relationships between all the regions of the surface. Connectivity information is already implicitly represented by voxel adjacency in the volume. By organizing and traversing these connectivity relationships we construct a topological graph of our surface. The construction of this graph has two parts. First we construct a topological distance tree, similar to propagating a wave front across a surface in the geodesic setting. Second, we augment this tree to be a topological graph by establishing connectivity between Surfels

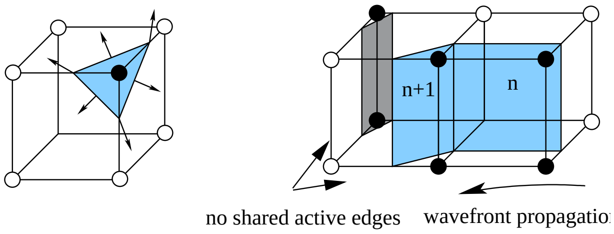

Figure 3.3: *Arrows indicate how to follow active edges from a given Surfel (left). On the right we see that the Surfel with distance n will propagate the distance $n + 1$ across its active edges to the connected Surfels. Note that the other Surfel in this voxel will only receive a distance when the wave front reaches it.*

of the same distance, similar to constructing iso-contours for geodesics on the underlying iso-surface.

3.4 Wavefront Propagation and Distance Tree

The first step in our approach is to construct our topological distance tree by enumerating the Surfels through a wavefront-like propagation of Surfel distance. Our notion of distance is very much like Chamfer distance in image processing (also called "chessboard" distance): two Surfels are a unit distance apart if they share at least one node, (*1-node adjacency*, see Fig. 3.4). Thus, a topological distance tree is an organization of all the Surfels into a tree hierarchy, where:

- Each Surfel is 1-node adjacent to its parent in the tree;
- The shortest distance from a Surfel to the root is the depth of the Surfel in the tree hierarchy.

3.4.1 Surface Wavefront Propagation

We start by identifying a source voxel. Any voxel that the surface passes through is sufficient and will serve as the root Surfel of our distance tree. The source voxel can be chosen trivially, e.g., first encountered. From there, we construct the distance tree by enumerating the Surfels in a breadth-first traversal. This propagation between adjacent Surfels can be done efficiently using active edges of the initial Cartesian grid containing the data. Active edges represent the transition of the iso-surface from one voxel to another, always connecting four Surfels. We use a priority queue to walk from Surfel to Surfel sequentially (Fig. 3.5, left) to construct the distance tree. Our algorithm is equivalent to running Dijkstra's algorithm to discover all paths from the source Surfel to all other unvisited Surfels. Since Dijkstra's algorithm is defined on the edges of a graph, our algorithm is equivalent to running Dijkstra's on the dual of the Surfel graph with edge weights all equal to one.

There are cases when more than one Surfel is associated with a single voxel. However, this is of no consequence to the algorithm since we propagate the wave front only across active edges (Fig. 3.3). The corresponding Surfels will be traversed in an ordered manner

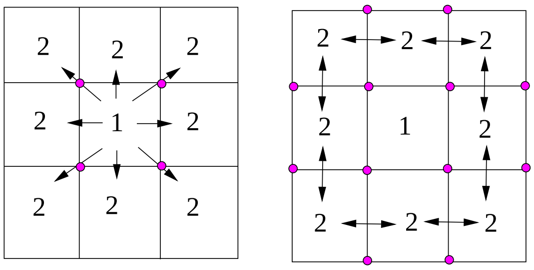

Figure 3.4: *On the left is an example of 1-node adjacency: the Surfel labeled 1 is 1-node adjacent to all the Surfels labeled 2 since it shares at least one node (colored pink) with each of them. On the right is an example of 2-node adjacency. Specifically this is an example of 2-node adjacency only between Surfels of the same distance as required in ring construction. Each of the Surfels labeled 2 is 2-node adjacent to any neighbor, also labeled 2, sharing an edge (i.e., 2 nodes).*

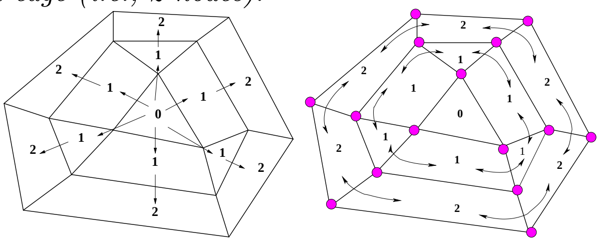

Figure 3.5: *Small portion of the distance tree overlayed on some Surfels (left). Same portion with adjacencies of the topological graph (rings) added (right).*

(at worst four for a single voxel). Ambiguities can arise when using only the eight corners of a voxel to determine an ordering of the active edges. We use the same solution as J. Bloomenthal in Graphics Gems IV [22] and avoid this problem by selecting one consistent solution in ambiguous cases.

Our distance tree requires only a compact data structure and facilitates later creation of the topologically correct coarse mesh. The distance tree is represented by storing an additional integer value for each voxel that the surface passes through and a pointer to the parent Surfel as indicated by Figure 3.5(right). In addition, we temporarily store the pointers to all the Surfels of a given distance in a bin structure (the distance bins) to later facilitate ring construction. See section 3.7 for more information about storage.

3.4.2 On-the-fly Construction of Topological Graph and Rings

The next step in our algorithm constructs a topological graph using the distance tree. This is done by collecting Surfels of the same distance into continuous rings, representing a "cross-section" of the surface topology. A topological graph is a representation of all the Surfels such that:

- All of the properties of a distance tree are true;
- Additionally, each Surfel is 1-node adjacent to its child;

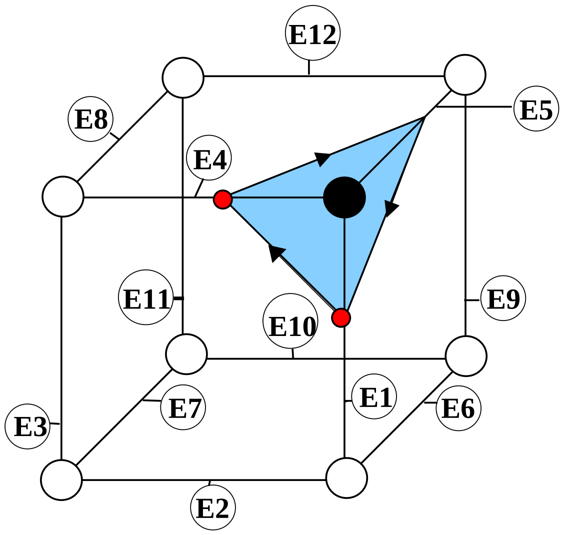

Figure 3.6: *A Surfel and its ordered edges. Despite the appearance of this figure, we never explicitly calculate the intersection of a Surfel and the active edges. Instead, every Surfel is defined by an ordered list of the "names" of the active edges. For example, in this case, this Surfel would be identified as: E1, E4, E5. During ring construction for the distance d, if we crossed into this Surfel by crossing the active edge pair {E1, E4}, we would first check the next active edge pair {E4, E5} to see if the neighboring Surfel incident on this edge pair is the same distance. If it was not, we would move on to check the next pair {E5, E1}. By definition one of these pairs must be the same distance. One can trivially check this by considering the nodes of the active edge pair {E1, E4} (nodes are shown in red). Since we know that this Surfel has distance d, we know that it received that distance from one of these two nodes (by definition of how we propagate distance), say the node on E1. Thus the node on E1 must also have propagated its distance to the Surfel incident on E1, E5 (again by definition of how we propagate distance).*

- Every Surfel has 2-node adjacency (see Fig. 3.4) with exactly two other Surfels of the graph that are of the same depth.

A Surfel is 2-node adjacent with another Surfel if they share two nodes (i.e., an edge). In essence, we collect the iso-distance rings and put them on separate lists to represent their inter-connectivity. The process of linking rings for the topological graph creation requires that we start with a given Surfel of distance n, traverse pairs of active edges, i.e., *faces* of the voxel bounding the given Surfel, in an ordered manner until we find another adjacent Surfel of the same distance n. As the ring is traversed, we enumerate an in-ring ordering for all the Surfels of the present ring to assist in the creation of triangles for the coarse mesh.

In order to come up with a consistent ordering within the rings, we use an idea very similar to work done on encoding a digital region boundary [14] and digital surface tracking [18]. Since we want to traverse the ring in an ordered manner, we need to pick a consistent orientation in space and an ordering for traversing that orientation. Luckily, the edges of each Surfel are ordered (see Fig. 3.6). This ordering is consistent, since the Surfel is oriented, which allows us to traverse around the iso-distance contour and construct a connected ring. The ring elements will only have 2-node adjacency with exactly two other Surfels of the graph that have the same depth. This definition of 2-node adjacency is once again very similar to rings of iso-Chamfer-distance on a rectangular grid [3].

For a given level of our distance tree, after a single ring is constructed, we check the distance bins to make sure that all the valid Surfels of level n are part of a ring. If not, we start the ring construction again with one of the unprocessed Surfels at level n. This process continues until all Surfels are incorporated in the topological graph structure. Additionally, each distinct ring at a given level will be assigned a distinct branch name. Thus if there is more than one ring at level n, each will have a unique branch name, derived from its parent, or sequentially assigned if it is a completely new branch. The following C++ like pseudo-code illustrates this process:

```
//for each iso-distance, try to generate rings
for(i = 1; i < max_distance; i++) {
  //for all the Surfels of distance i
  for (it = distance_bin[i].begin; it != distance_bin[i].end; it++) {
    //if this Surfel is not already a part of a ring
    if (!ElementofRing(it, i)) {
      //either use my parent's branch name if unused or new one
      if (UsedBranch[it->parent->branch])
        branchname = max_branch_name+1;
      else
        branchname = it->parent->branch;
      //construct a ring for distance i with the appropriate branch name
      ConstructRing(it, i, branchname);
      Usedbranch[branchname] = 1;
    }
  }
}
```

3.4.3 Cleanup of Rings

If distance is propagated naïvely, rings could have tails (Fig. 3.7). Tails are large or small dead-ends of the wave front. A dead-end of a wave front occurs when the wave front runs into itself. Tails do not provide additional topological information (see Appendix A for a proof) and can confuse the ring construction algorithm. Specifically, at a dead-end it may appear that a Surfel is 2-node adjacent to more than two other Surfels. Tails are eliminated from ring construction by pruning them from the distance tree during distance propagation. During this stage of the algorithm if a voxel cannot propagate its distance forward (because all of its neighbors are already visited), we prune this voxel from the distance tree. It is clear that this procedure exactly prunes dead ends, since a dead-end is defined as when the wave front cannot proceed.

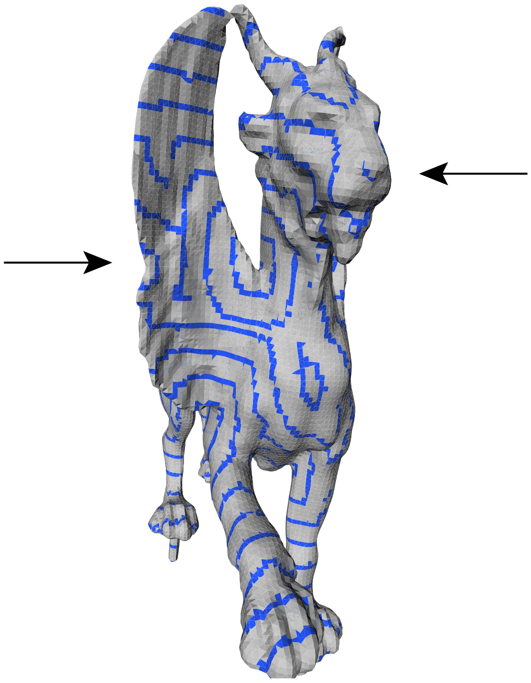

Figure 3.7: *Unmodified distance rings for the feline dataset. The source cell for the distance tree is near the feline's tail. Note that there are two visible tails - one on the left wing where two wave fronts run into each other and another on the nose where the wave front completely dead-ends.*

3.5 Mesh Construction from Topological Graph

The topological graph provides everything needed to build the coarse mesh. In order to have a good coarse sampling of the surface, we only include the smallest number of rings necessary. Rings essential for coding topology are those inducing topological *events*. A ring represents a topological event based on its adjacency relationships in the topological graph.

3.5.1 Ring Classification by Topological-Event Search

From the rings, we create a coarse mesh which respects the initial topology. Indeed, there are only three types of important ring adjacencies:

- Endcap: the root Surfel or a leaf ring;
- Split: any two Surfels of a single ring at level n which have at least two different child Surfels belonging to two or more different disjoint rings at level $n + 1$ (Figs. 3.8(a) and 3.9(a));
- Merge: any two Surfels of a single ring at level n having two or more different parents belonging to two or more different disjoint rings at level $n - 1$ (Figs. 3.8(a) and 3.9(b)).

For example, in a torus there would be one Split where the graph traversal first encounters the hole of the torus and one Merge where the hole ends. Both of these events need to be captured in order to construct the correct topology of the torus. In contrast, an "unimportant" adjacency is when rings of the same branch number are stacked on top of one another with no change in branch number between any of the rings (for example on the torus between the Endcap and the Split). These rings can be discarded without changing the topology of the surface.

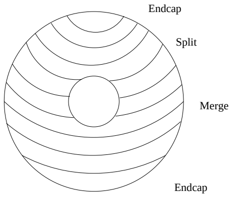

Figure 3.8: *Topological Events on a torus*

In order to capture the topology of a given surface, we are specifically interested in discovering the number of handles of that surface. A handle occurs when a ring *splits* into n distinct branches (where $n \geq 2$), and then subsequently, these same branches or some subset of these branches *merge* together. Thus, we only store specific pairs of rings that have split and then subsequently merged together. See Appendix A for a proof of how this construction guarantees the correct topology of the coarse mesh.

Since these adjacency relationships are completely determined by a ring's parent and child, ring construction *and* event detection can be performed in a sweep algorithm. Once the rings at level n are constructed, event detection is performed by walking along the parent rings at level $n-1$ to see if an event ring is encountered. For each of the Surfels in rings at level $n-1$, we check that their children have the same branch number as their parent ring. If not, a split has been found.

While doing this traversal, we also keep track of the branch numbers of the children already visited. For a Merge, for example, the initial parent ring of branch 1 will register that it has seen a child of branch 1. When the other parent ring, branch 2, checks the branch name of its children, it will find that all its children are a different branch name and have already been seen. Thus a Merge is detected. Finally, if a ring cannot construct a valid child ring, it is entered into the Ring Master as an Endcap. The first ring constructed is also entered into the Ring Master as an Endcap.

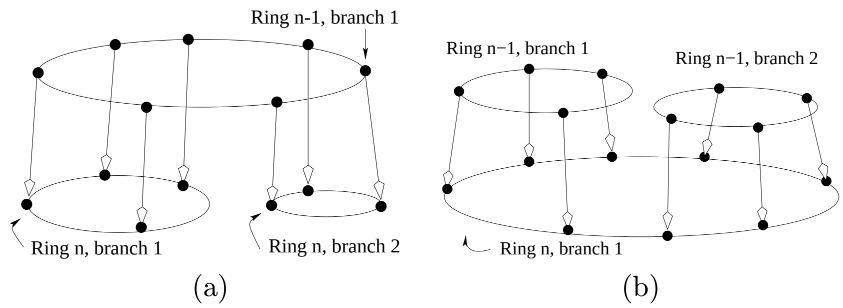

Figure 3.9: *(a) Split detection; (b) Merge detection.*

The desired coarseness of the mesh can be controlled by adding criteria for ring selection. For example, consider a requirement that the initial mesh exhibit good aspect ratio triangles. This can be achieved by selecting rings at multiples of some integer distance w and changing the sampling density within the rings to also be of distance w. Another useful feature of ring selection is that the rings coarsely approximate distance on the surface (assuming each Surfel has approximately the same size). Thus, ring placement corresponds to the underlying geometric complexity of the surface.

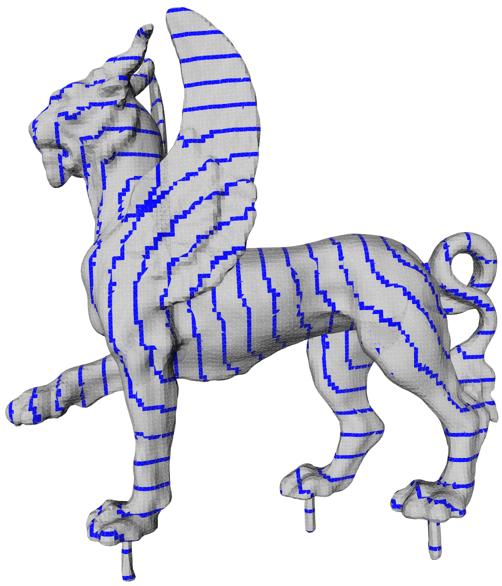

Figure 3.10: *The distance rings used to extract the coarse mesh for the feline dataset. The source cell for the distance tree is near the feline's tail.*

3.5.2 Connecting Rings: The Ring Master

By tiling together all the relevant rings that are either a paired Split and Merge (forming a handle) or an Endcap, we can construct a coarse mesh that is topologically equivalent to the surface represented in the volume. To do so, we add some temporary information to what we call the Ring Master. When a ring is designated as a topological event during selection that information is stored for the tiling step. Specifically, we keep track of how many child rings a given ring has as well as their branch numbers. For each child we keep track of how many parents it has and their respective branch numbers.

3.6 Coarse Mesh Construction

At this point, we have a list of all cross sections of the surface which are required for tiling a good coarse approximation of the final surface. This final step is related to contour stitching (see [1, 15, 13]). However, since we work within the framework of the volume data with the additional information stored in the Ring Master, we do not face

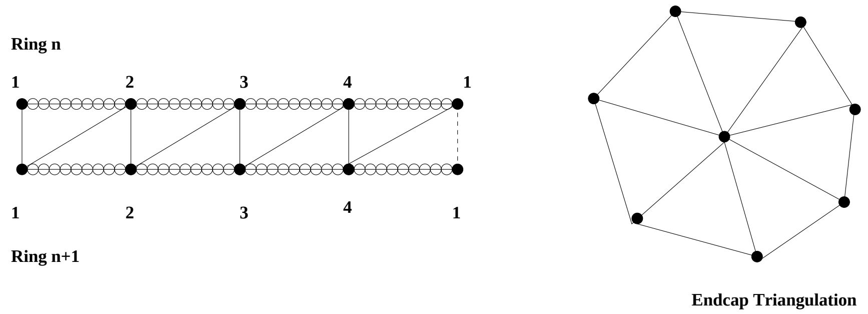

Figure 3.11: *On the left is a triangulation between two unrolled rings. The circular nodes represent the Surfels used to construct the ring. The dark circles represent where the ring has been subsampled. We now use the dual of the Surfel ring and just treat the samples as vertices and Surfels connecting these samples as an edge. The ordering of the samples is used to trivially determine their connectivity. We make a triangle by inserting an edge between every in-ring sample in order and between all like numbered samples between adjacent rings. Finally, we just use a consistent rule that the final edge is from the lower ring's vertex* n *to the upper ring's vertex* $n + 1$ *(modulo the total number of vertices). On the right is an example of an endcap triangulation.*

the traditional correspondence problems of contour stitching. Specifically, the volume data combined with the topological graph can be used to resolve any ambiguities about inter-contour connections.

3.6.1 Ring Subsampling and Shortest Distance Projection

The general procedure is to subsample a ring along its length followed by projection to its child ring. The topological graph is used for this projection as indicated in Figure 3.9. The samples on both rings are enumerated in corresponding order within their ring to facilitate triangulation (see Fig. 3.11). This process is repeated for all corresponding layers of rings. If a ring is an Endcap we evenly subsample it based on desired triangle size criteria and connect all these samples to a central point (see Fig. 3.11).

The projection step may result in samples being too close or too far away from one another due to changes in the geometry of the iso-surface. In this case we can adjust the number of samples to accommodate the density change. Specifically, we either snap close points together, or insert a midpoint sample.

3.6.2 Stitching

It is easy to tile two contours that have a one-to-one correspondence in their sample enumeration. The general approach of our algorithm is to use the information stored in the topological graph and in the Ring Master, to *break* each connection into a one-to-one connection. By breaking the rings into one-to-one correspondence and then using bridges between adjacent connected rings, we correctly model the topology of the surface. Additionally, it allows us to maximize the number of vertices of valence six.

Thus Splits and Merges are handled by "breaking" the larger ring into the appropriate number of smaller ring segments by relating which subsamples have projected from/to

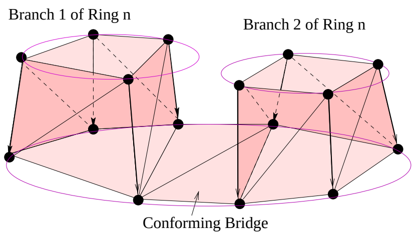

Figure 3.12: *Stitching a Merge (Splits are handled similarly).*

each distinct smaller ring. Using the information stored in the Ring Master it is known whether a given ring has more than one child or parent. During the projection step we separate parent and child Surfels into appropriate *subrings* based on the branch number of the child (respectively parent) to which the Surfel projects (Fig. 3.12). In a second pass around the larger ring, branch names are compared along the projection. If two neighboring samples have come from different parent rings, the samples are stored in an edge list and later paired with their matching edge to make the conforming bridge between the two subrings (Fig. 3.12). With this the rings can be triangulated as usual.

It is worth noting that there is a case equivalent to a Merge immediately followed by a Split. Due to the discrete nature of the samples this can appear as a *double Split*. This case is easily identified and tagged in the event detection: two child rings will have more than one parent in common. In such a case we follow the same routine, but a little more care needs to be taken with inserting the conforming bridge.

3.7 Discussion

One of the benefits of this approach is the low memory overhead for the topological graph representation. In the case of an $O(n^3)$ volume the storage requirement for the distance tree is $O(n^2)$, as it depends on the size of the surface. The only other data that we need to store for generation of the coarse mesh is dependent on the rings of the topological graph and is $O(n)$. Memory overhead for rings is minimized by keeping only, (i) the rings selected to be part of the coarse mesh; (ii) the last ring constructed and (iii) the current ring, which is being evaluated for possible selection. Although both our algorithms and Marching Cubes use total storage of $O(n^2)$, our algorithm has a more compact runtime footprint than a typical Marching Cubes implementation. In particular, in order to avoid visiting every single voxel in the volume, the Marching Cubes algorithm keeps a stack of all the voxels on the surface. This stack requires storage of three float values associated with each edge intersection (up to 36 floats per voxel) and three integers per face (up to 12 integers per voxel). In contrast, our algorithm does not require such a stack. Furthermore, we have presented an algorithm in which a distance value is permanently stored for each Surfel. However this is only conceptual, as distance values could be stored temporarily, only for voxels on the *frontier* region of the sweep. The frontier region of the sweep is the region of the surface between the last ring selected to be a part of the mesh and the current ring being evaluated. Even without this modification, our algorithm has a significantly smaller runtime footprint.

Chapter 4

Solver

4.1 Multi-Scale Force-based Solver

Once a coarse mesh with the correct topology is found, the next step of our algorithm consists of turning this initial mesh into a hierarchical triangulation fitting the data with suitable sampling densities and well shaped triangles. This refinement process will make use of a simple force-based multi-scale solver.

4.2 Setup

To solve for the iso-surface one may consider the signed distance function of the volume as a potential field and search for the minimum potential solution [27, 26, 25, 46, 42]. Employing the calculus of variations this results in an *energy minimization* problem intended to bring the current mesh representation to the final desired shape by following the gradient of distance. In this setup the problem does not possess a unique solution independent of the starting position due to the non uniqueness of a minimum distance for non convex sets. Consequently, the problem must be regularized to ensure convergence and a unique solution. Following the practice in variational geometric modeling (e.g., [19]) this is typically done by adding potential energy terms which are functions of first and second derivative magnitudes of the surface. Such thin-plate approaches have been used, for example, by Qin [46]. These additional energy terms also serve to control the size distributions and well-shapedness of the triangles in the mesh.

Unfortunately, this approach has a significant drawback: the tradeoff between closeness to the data and the smoothness of the solution is hard to tune. In essence, smoothness of the solution and faithfulness to the desired goal surface compete with each other. Too much regularization will lead to smooth, unfit surfaces, while not enough regularization will lead to convergence difficulties. In both cases, the overall speed and accuracy is very dependent on fine tuning of parameters. This has been partially addressed by scheduling the regularization as decreasing in time (e.g., [25]). Such strategies help, but still require careful tuning of parameters on a case by case basis.

Computing the gradient of distance is notoriously unstable, especially in the presence of noise. For this reason we have chosen to use the distance itself. The current mesh approximation locally inflates or deflates based on the distance to the zero-contour. I.e., the direction of (local) motion of the mesh is given by its local normal, while the magnitude (and sign) of motion are determined by the distance function itself. This approach, inspired by work in image processing [6], has already been used with success in the context of active implicit surfaces [9]. As a novel element we add a reparameterization technique to control triangle shapes and their variation across the surface. In this way, we obtain adaptive sampling and well shaped triangles without introducing forces which compete with the interpolation constraints. Since the meshes are refined through adaptive quadrisection we have a natural multiresolution structure which we exploit directly for an efficient multiscale solver.

Our setup gives rise to a number of different force terms detailed below. External forces minimize the distance between the mesh and the zero-contour of the data. Internal forces arise from the reparameterization terms.

4.3 External Forces

We begin by considering the force acting on a single triangle before giving the actual equations for the net force on a vertex in the mesh.

Following the balloon strategy, we define the force acting on a triangle T of our mesh as being along the normal of the triangle, with a sign and a magnitude depending on the surface integral of the distances d between the triangle and the actual zero-contour C:

$$F_T = \mathbf{n}_T / \mathcal{A}_T \int_{x \in T} d(x, C)\ dx$$

where $\mathbf{n}_T$ is the triangle normal and $\mathcal{A}_T$ is the area of T. The integral of the distance across the face can be computed exactly in the volume setting, since we assume that the distance varies linearly across a given voxel. In practice this is overkill and we use a much cheaper sampling criterion. Each triangle face is randomly sampled with a uniform distribution whose area density depends on the total area of the triangle. This results in quicker force computations, while preserving the quality of the approximation. Because the sampling is not dependent on the face size we avoid excessive computations for large faces that already fit the underlying zero-contour and we avoid inadequate sampling for small faces that may be poorly aligned with a small feature. Note that the minimum bound on the discretization rate is of the order of a voxel size, since everything is assumed to vary linearly within a voxel. Therefore, we use the following simple sampling strategy:

1. Temporarily quadrisect the triangle into four small triangles and find the four distances d_i for the barycenter of the each of the new triangles. Define the number of samples $m_T = 4$;

2. Estimate the variance $V_T[d]$ of these distances;

3. If $V_T[d] \geq \delta$,

    - stochastically sample the triangle with a uniform distribution and density inversely proportional to triangle size and assign the number of samples m_T

accordingly,

- compute the distances d_i from these samples to the zero-contour.

The variance of a discrete set of distances is computed in the standard way $V_T[d] = E[d^2] - E[d]^2$, where E denotes the mean of its argument. A more sophisticated method, using fully adaptive sampling depending on variance, can be derived, but this simple approach has proved sufficient and has the advantage of being very efficient. The final net force on a triangle would be given by the above mean of the distances

$$\mathbf{F}_T = \mathbf{n}_T E[d].$$

The solver requires forces acting on vertices. To arrive at these we use the above sample points to compute integrals for each vertex by integrating over all incident triangles, weighting each sample point with its respective barycentric coordinate. Figure 4.1 illustrates this idea in the case of a single triangle. Every sample point within this triangle contributes to the force integrals associated with its corner points as follows:

$$\begin{array}{c} 1/m_T \; \mathbf{n}_T d(x_i, C) \; \phi_j(x_i) \\ 1/m_T \; \mathbf{n}_T d(x_i, C) \; \phi_k(x_i) \\ 1/m_T \; \mathbf{n}_T d(x_i, C) \; \phi_l(x_i) \end{array}$$

where $x_i \in T$ is the sample location; (j, k, l) are the corners of T; and the ϕ give the barycentric coordinate of x_i with respect to j, k, and l respectively. Effectively we are

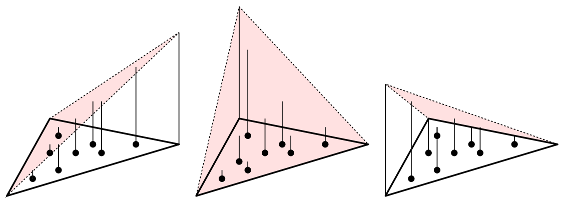

Figure 4.1: *Samples from a given triangle contribute to vertex integrals according to their barycentric coordinates as indicated by the linearly varying weighting ramps.*

using piecewise linear finite elements and stochastic sampling to evaluate the associated integrals. In the implementation one simply iterates over all triangles and accumulates the integrals at each corner.

With this scheme, faces will tend to move towards the zero-contour. If the mesh is coarser than the small details from the zero-contour, it will settle in an optimal position, smoothing the details. The finer the mesh is, the better the fit will be. As mentioned in [26], we also noticed that vertices tend to align with sharp features in the zero-contour.

4.4 Internal Forces

Internal forces are usually added as a regularizing term, to guide the minimization to a desirable local minimum. In our approach internal forces are mainly used to ensure good aspect ratios for the faces and to keep the sampling across the surface smoothly distributed. Usually, springs of zero rest length and identical stiffness are used to keep sample points from clustering locally and ensure uniform sampling [26]. Instead we define *reparameterization* forces which act similarly, but only along the local parameter plane, not in space.

4.4.1 Decoupling Smoothing and Reparameterization

In recent work on mesh smoothing [54, 10], the Laplacian operator has been extensively used to denoise triangulated surfaces, using the approximation:

$$\mathcal{L}(x_i) = \frac{1}{m} \sum_{j \in N_1(i)} x_j - x_i \tag{4.1}$$

where x_j are the neighbors of vertex x_i, and $m = \#N_1(i)$ is the number of these neighbors (valence). Note that this definition is exactly similar to springs with zero restlength whenever the valence is constant throughout the mesh. This Laplacian of the mesh at a vertex can be broken down into two orthogonal components:

- a component normal to the surface, creating *shape smoothing*
- and a component in the tangent plane, fairing the *parameterization* of the mesh.

The normal vector to the surface can be found easily by normalizing the curvature normal vector $\mathbf{K}$ [10, 11]:

$$\mathbf{K}(x_i) = \frac{1}{2\mathcal{A}} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(x_i - x_j). \tag{4.2}$$

For arbitrary connectivity meshes numerical evidence shows that no spurious drifting artifacts appear when the surface is modified only in the direction of $\mathbf{K}$ [10]. This decomposition into normal and tangential components separates motion into one component changing shape and one changing the parameterization. We are only interested in the latter.

4.4.2 Reparameterization as Tangential Laplacian Smoothing

In our context shape smoothing would act *against* the external forces trying to fit the initial data. Thus we are only interested in the tangential motion of Laplacian smoothing in order to improve the quality of the discretization. This reparameterization force is defined as

$$\mathbf{T}(x_i) = \mathcal{L}(x_i) - (\mathcal{L}(x_i) \cdot \mathbf{n})\mathbf{n}, \tag{4.3}$$

where $\mathbf{n}$ is the normalized $\mathbf{K}$ of Equ. 4.2.

Since we use the Laplacian as a reparameterization force, and our refinement scheme is adaptive (the triangles of our surface are not uniformly subdivided) we must account for the irregular parameterization of the conforming edges. Conforming edges are added to avoid t-vertices at the edges between a refined face and an unrefined face. Our scheme is restricted in the sense that neighboring triangles can only differ by one level of refinement. This results in only a small number of configurations that require conforming edges (see Fig. 4.2). Since the Laplacian is designed for the regular case (all vertices with valence 6 and all angles equal to $\pi/3$), we add a term to account for the change in the angle of the triangles generated by the conforming edges:

$$\mathcal{L}(x_i) = \frac{1}{m} \sum_{j \in N_1(i)} (x_j - x_i)(\cot \alpha_{ij}^p + \cot \beta_{ij}^p)$$

where α_{ij}^p and β_{ij}^p are the angles in the parameter plane (see fig. Figure 4.3). These weights are easily precomputed based on the possible conforming edge configurations. Determining the correct weight to use for each edge incident on a vertex is a simple case look-up, depending on how neighboring triangles are subdivided. It is important to note that in the normal uniform subdivision setting when $\alpha_{ij}^p = \beta_{ij}^p$ we find the expected Laplacian, similar to springs with zero restlength, Equ. 4.1.

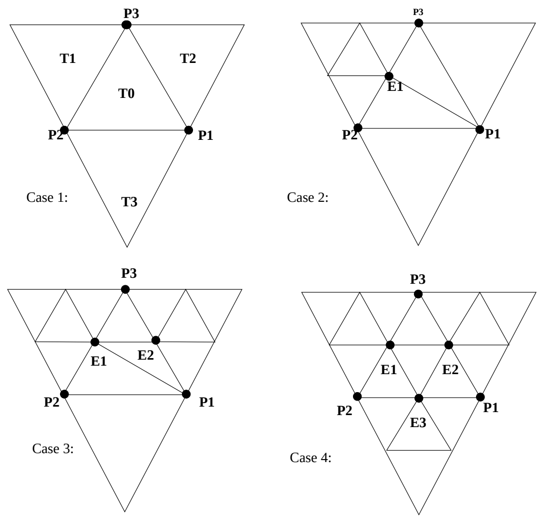

Figure 4.2: *The possible conforming edge configurations with respect to **T0** are invariant under rotation and reflection. In case 1, none of the neighboring triangles are subdivided and no conforming edges are added. In case 2, **T1** is subdivided and we need to add a conforming edge for **P1**. In case 3, **T1** and **T2** are refined and we add two conforming edges, one between the two refined triangles and one for **P1**. Finally, all three neighbors of **T0** can be subdivided and **T0** is not subdivided. In this case we add three conforming edges between the refined triangles.*

We also use the second Laplacian operator $\mathcal{L}^2$ [30, 10] to ensure a smoother variation of sampling rate over the surface. As in the case of the Laplacian, we use the same weights for the conforming edges, and suppress the normal component in the same way. By proceeding as described, we keep internal and external forces distinct, thus simplifying parameter choices.

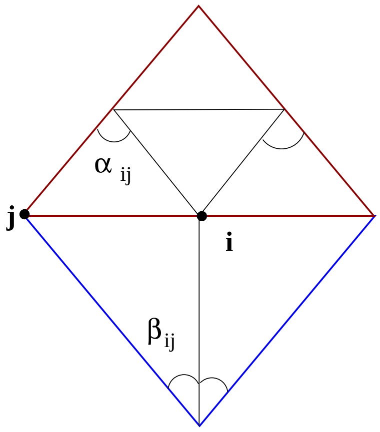

Figure 4.3: *Example of how the weights for the Laplacian are applied depending on the conforming edge configuration. In this situation the top coarse triangle (in red) has been refined and the bottom triangle (in blue) has not. A conforming edge is added, as seen in case 2 of Fig. 4.2. The weights associated with edge* i, j *are based on the angles of the incident triangles measured in the parameter plane. Here,* α_{ij} *is* $\frac{\pi}{3}$ *and* $\beta_i j$ *is* $\frac{\pi}{6}$.

4.5 Refinement Strategy

After an optimal solution has been found for a given mesh, if the mesh does not meet the user supplied accuracy criterion, we evaluate a subdivision criterion over each triangle in the mesh. Any triangles failing the criterion are split 4-to-1. This hierarchy is naturally maintained in a forest of quadtrees, one tree for each original coarsest level triangle. Within the forest we enforce a restriction criterion, i.e., no triangle is allowed to be off by more than one subdivision level from its neighbors. The solver is run anew after any triangles required to subdivide have been refined.

The two criteria used to determine if a triangle should be subdivided are curvature and variance of distance. If the variance of the distance samples for a given triangle is too high, the surface underneath this particular triangle must have high curvature, i.e. is not flat. Subdivision is therefore required since a simple triangular approximation is clearly insufficient. Using a user supplied threshold ϵ_V all triangles T with $V_T[d] \geq \epsilon_V$ are subdivided.

In addition to this criterion, we also test the curvature of the current mesh to ensure good discretization in highly curved areas. If the three vertices of a triangle have too high a curvature compared to the area of the triangle, our solver subdivides the triangle to better adapt to the local geometry. For generality, we add a condition to deal with sharp features in the volume data: we invalidate the test on curvature if the variance of sampled distances is too small. Subdivision will be avoided if we are already describing the surface adequately. Therefore, our second subdivision criterion for a triangle $T = (x_i, x_j, x_k)$ can be written:

$$(|\mathbf{K}(x_i)| + |\mathbf{K}(x_j)| + |\mathbf{K}(x_k)|)\mathcal{A}_T \geq \epsilon_\kappa \text{ and } V_T[d] \geq \frac{\epsilon_V}{10}$$

where ϵ_κ, the maximum discrete curvature, is a user-defined value. The choice of $\epsilon_V/10$

seems reasonable in all our tests, but could be defined by the user if needed, depending on the prevalence of high frequency detail in the iso-surface. It is worth noting that ϵ_V can be viewed as a smoothing factor. For example if the user wants a smoothed version of the surface they can set ϵ_V to a higher number and the system will stop after reaching a solution with fewer triangles that approximate the surface.

4.6 Overall Solver Algorithm

Once forces have been computed for every vertex in the current mesh, vertex positions are updated through an explicit dynamics step:

$$x_i^{(t+\delta t)} = x_i^{(t)} + F_{x_i}\delta t$$

advancing the mesh in time until the approximation error does not decrease further. When advancing the mesh, we also must put a restriction on the time step δt. Specifically, the time step must satisfy the Courant condition (also known as the CFL), that the velocity of change must not travel faster than the minimum detail in the system. This condition is simple to compute in our system and is

$$\delta t = m_e / M_f$$

where m_e is the minimum edge length and M_f is the maximum force. Then the subdivision criteria are evaluated for each triangle and quadrisection is performed as needed. Subsequently we solve again until convergence and continue this process until the user supplied error criteria are satisfied at every point on the surface.

The behavior of the solver is controlled by the relative weightings of distance and reparameterization forces. We have found a factor of 2 in favor of the distance forces to work reliably for a wide variety of data sets. Similarly, error thresholds of $\epsilon_\kappa = 15$ and $\epsilon_V = 10^{-4}$ have proven to work well without the need for tuning (to make the error criteria scale invariant we consider the object to occupy the unit cube).

Chapter 5

Conclusion

5.1 Results

We have applied our algorithm to a variety of datasets and compared the results with Marching Cubes reconstructions as "ground truth." Some of these are shown in Figure 5.1.

The top sequence illustrates the case of a MRI dataset (128^3) which was segmented through a level set method. Construction of the coarsest mesh (186 triangles) took .5 seconds. The intermediate mesh contains 4810 triangles, while the final mesh has 21360 triangles. Comparing our reconstruction against the Marching Cubes mesh (58684 triangles) we find a relative L^2 error of $1.8*10^{-4}$ (Fig. 5.2). The surface is a topological sphere, but requires fairly fine levels of refinement near the ears, attesting to the performance of our solver in the presence of rapidly changing local geometric complexity.

The middle sequence shows an extraction from a 3D scanner generated distance function [8]. The topology of the feline is non-trivial containing numerous handles in the tail region (Fig. 5.3) and demonstrates the performance of our coarsest level mesh extraction and topology discovery algorithm. It also demonstrates the ability of our solver to resolve fairly fine detail such as the mounting posts on the bottom of the paws. Triangle counts are 3412, 13412 and 46996 respectively (Marching Cubes: 72685) for an error of $3.3 * 10^{-4}$. Coarsest mesh extraction time was .34 seconds on a volume of $158 * 74 * 166$ voxels.

Finally the bottom row shows another MRI dataset of a mouse embryo which was segmented with a level set method. The surface has several handles (near both front paws) and numerous concavities. All were resolved successfully. Triangle counts are 1030, 4086, and 26208 respectively (Marching Cubes: 129670) with an error of $6 * 10^{-4}$. Coarsest level extraction took .78 seconds on a volume of $256 * 128^2$. Typical solver times are on the order of a few seconds for the initial meshes increasing to 4 to 5 minutes for the final reconstructions.

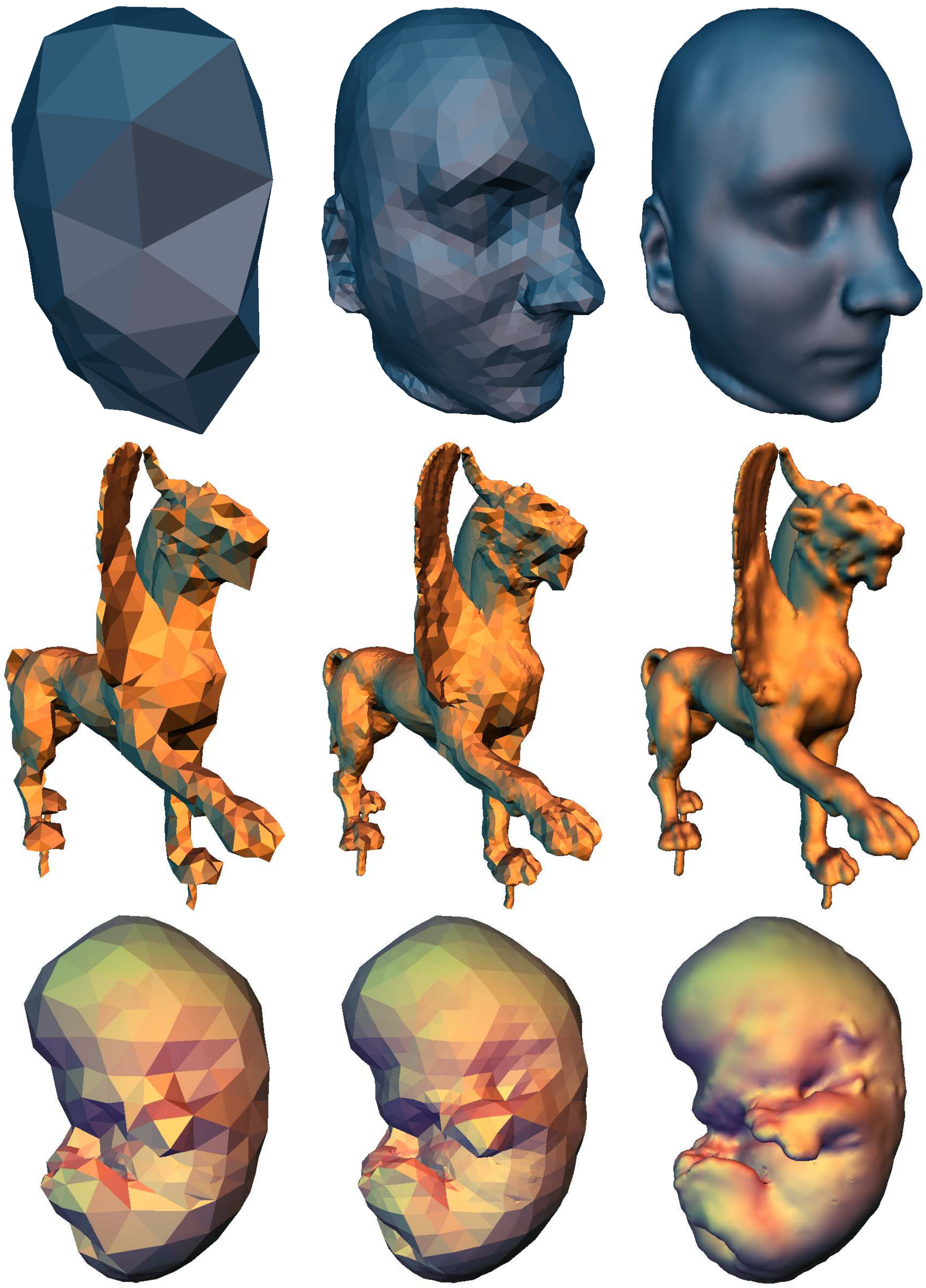

Figure 5.1: *Reconstructions performed with our algorithm on MRI datasets (top and bottom) and a 3D scanner generated distance function (middle). The coarsest mesh is shown on the left followed by an intermediate adaptive mesh and a final result.*

5.2 Summary

We have demonstrated a novel algorithm for the capture of iso-surfaces in the form of hierarchical, adaptive semi-regular meshes. It is based on a new approach to construct a

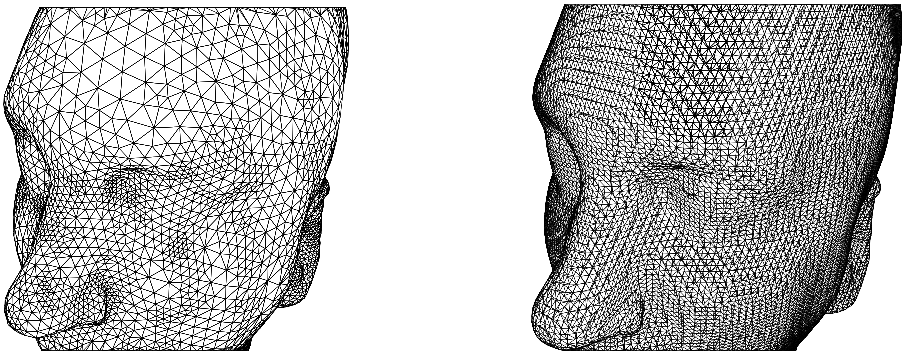

Figure 5.2: *Comparison between our algorithm output and a Marching Cubes mesh. The relative* L^2 *error between these is* $1.8 * 10^{-4}$.

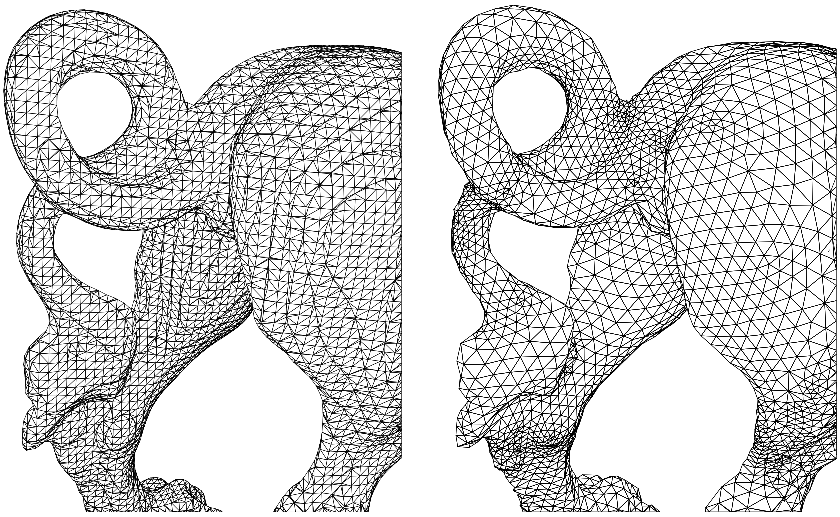

Figure 5.3: *Tail section of feline showing nontrivial topology. Marching Cubes extraction on the left, adaptive semi-regular mesh on the right.*

coarsest mesh with guaranteed topology approximation of the iso-surface using surface wavefront propagation to discover the topology and ensure that it is represented faithfully. Construction of this coarsest mesh is based on a low memory overhead traversal of the volume and does not require or incur the potentially enormous overhead of simplifying a Marching Cubes mesh. In a subsequent solver step we adaptively refine this mesh and optimize its vertex positions based on a balloon inflation/deflation model. In contrast to previous approaches, we use a novel explicit reparameterization force employing tangential components of the first and second Laplacian of the mesh. Thus we do not have to trade off fidelity to the original data and uniqueness of the solution. The resulting meshes have a natural multiresolution structure based on semi-regular meshes. A large number of algorithms are now available which take optimal advantage of such meshes. Examples include editing [61], finite element simulations [5], and progressive coding [28] among many others.

We have demonstrated the algorithm with a number of examples including a distance volume produced by 3D scanning [8] which exhibits non-trivial topology, and two MRI datasets which were segmented with level set methods. A human head with spherical topology (the neck was closed) and a mouse embryo with non-trivial topology.

5.3 Future Work

In order to avoid self-intersection problems during the solution process we have so far relied on coarsest meshes which resolve the geometry reasonably well to begin with. It would be desirable to start with the coarsest possible (in the topological sense) initial mesh and counteract any self-intersection problems in the solver itself. Other interesting areas for future work include

- investigate the use of multiresolution representations of the volume;
- optimization of the solver including adaptive time stepping strategies and automatic selection of parameters such as the relative weighting of reparameterization forces.
- application of the topological graph to irregular meshes to code topology

Appendix A

Proof of Topological Correctness

This appendix demonstrates that our algorithm preserves the topology of the initial discrete surface (consisting of the union of Surfels). In order to do so, we prove that the two surfaces are homeomorphic to one another.

A.1 Set-up, Definitions, and Theorems

All of the following definitions and theorems are from Chapter 21 of **Geometric Topology in Dimensions 2 and 3** by Edwin E. Moise [44].

Let K be a finite complex, of dimension ≤ 2 (refer to [44] for the complete definition of a finite complex, however, recall that K is a collection of simplexes in a space $\mathbb{R}^m$ and specifically, if K is a finite complex, then $|K|$ is a finite polyhedron).

The Euler characteristic χ of K is

$$\chi(K) = V - E + F$$

where V is the number of vertices, E the number of edges, and F is the number of faces. *The Euler characteristic for open cell-complexes is defined in the same way as for complexes.* In our setting the open cell-complexes, C, is comprised of vertices, edges and faces [44].

Next, we recall the following theorems:

Theorem T1: *Let M be a compact 2-manifold with or without boundary. Then all triangulation K of M have the same Euler characteristic.*

Theorem T2: *For open cell complexes, the Euler characteristic is combinatorial invariant*

From these two theorems we can define the Euler characteristic of a compact 2-manifold M as the number $\chi(M)$ which is $= \chi(K)$ for every triangulation K of M. Specifically, we can understand the Euler characteristic of a 2-manifold in terms of the Euler characteristics of its open cell-complexes. This property reflects the main advantage of the Euler characteristic: it is an invariant for any given 2-manifold, or subset of a 2-manifold, *regardless of the discrete representation used.*

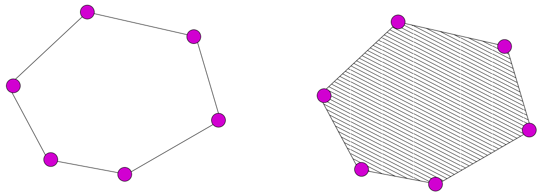

Figure A.1: *On the left is a 1-sphere. On the right is a 2-cell.*

Now, recall that the topology of a compact 2-manifold is completely determined by its genus. The genus of a 2-manifold, M (closed polyhedral surface) is defined with respect the Euler characteristic as follows:

$$\chi(M) = V - E + F = 2(1 - g)$$

where again V is the number of vertices, E the number of edges, F the number of faces and g the genus.

Finally we recall the most important theorem with regards to our construction:

Theorem T3: *Let K_1 and K_2 be finite complexes (in our case, polyhedrons), such that $|K_1| \cap |K_2|$ is a polygonal line J, and so that $K_1 \cup K_2$ is a finite complex. (Recall that $|K_1|$ and $|K_2|$ are finite polyhedrons). In other words if we have two polyhedrons such that they intersect along a polyline, then:*

$$\chi(K_1 \cup K_2) = \chi(K_1) + \chi(K_2).$$

This means that we can compute the Euler characteristic of our final polyhedron $K_1 \cup K_2$ by computing the sum of the Euler characteristics of its parts. Regardless of how we triangulate those regions, we will not affect the Euler characteristic.

Specifically, we use the following:

1-sphere: A 1-sphere is a set homeomorphic to a unit circle. If J is a 1-sphere, $\chi(J) = 0$ since $V = E$ and $F = 0$. In other words the Euler characteristic of a contour, or boundary, is 0 (see Fig. A.1).

Since the 'rings' in our topological graph are actually rings of Surfels (see Fig. A.2), the *boundary* of the rings are 1-spheres.

n-cell: An n-cell is a set homeomorphic to an n-simplex. Therefore, a 1-cell is an arc, while a 2-cell is a disk. $\chi(\textit{any}\ \text{2-cell}) = 1$ since $V = E$ and $F = 1$, (see Fig. A.1).

Any endcap of our topological graph is a 2-cell. (Recall that our initial seed Surfel is also defined as an endcap and thus is also a 2-cell).

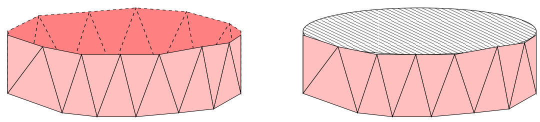

Figure A.2: *A 1-to-1 ribbon (left) and a closed 1-to-1 ribbbon homeomorphic to a sphere (right)*

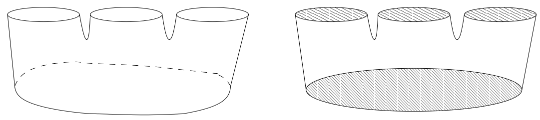

Figure A.3: *On the left is a 1-ring-to-n-rings ribbon. On the right is the closed ribbon, making it homeomorphic to a sphere*

A.2 Main Result

A.2.1 Decomposition of the initial Surfel-tiled surface

We now clarify the relationship between our approach and these definitions and theorems. As metioned earlier the boundary of a ring in our topological graph is a 1-sphere. A ring itself is a ribbon, defined as follows:

A ribbon is comprised of a polygonal surface connecting 1-spheres (for example, see Fig. A.2).

Our algorithm covers the surface by slicing it into a sets of ribbons. We use T2 to decompose the Euler characteristic of the whole surface (union of our Surfels) as a *finite sum of the Euler characteristic of these ribbons and endcaps.* Fortunately, each of these objects are easy to analyse. Let's review the possible cases:

endcaps: These are obviously 2-cells (homeomorphic to a disk), their Euler characteristic is $\chi = 1$.

1-to-1 ribbon : This is the most common case for a ribbon and is comprised of two connected 1-spheres (see Fig. A.2). Its Euler characteristic is trivially equal to 0 by the following argument. Close this tube section by two end caps one at each end of the tube (see Fig. A.2). Clearly we obtain a closed object homeomorphic to a sphere, therefore with a genus 0, therefore with a Euler characteristic of 2. Use now T3 to decompose this 3-part object in order to discover that the tube itself must have $\chi = 0$ since the Euler characteristic of each end cap is 1.

1-to-n ribbon (and vice-versa) : These occur at either a split or merge. For these branchings, the Euler characteristic of the ribbon is slightly more complex. However, it turns out that the exact same derivation used for a 1-to-1 ribbon applies: close the different branches by endcaps, to get an object that is homeomorphic to a sphere. Therefore, for 1-to-n ribbons, where one ring turns into n rings (see Fig. A.3), we have: $\chi = 1 - n$. Notice that we also have to consider the general case: m rings to n rings, which relies on the same arguement as above. However, in our implementation, we separate these cases into a finite set of 1-to-q branchings.

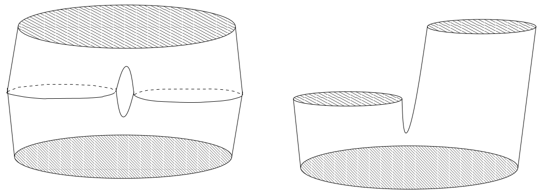

Figure A.4: *Compare the torus on the left with the branching object on the right.*

We now have a very easy way to compute the Euler characteristic (and therefore, the genus): start with 1 (for the initial endcap), add (1-n) for every 1-n or n-1 branchings that happens along the surface, finally add 1 for each end cap. Notice, that since we did not make any assumptions about the placement of the inital cap this proof works for any seed Surfel. No degenerate cases, like those found for the Reeb graph, will take place, guaranteeing a completely automatic process.

This proof can easily be verified on simple geometric objects. A sphere will only have an initial endcap and a final endcap, resulting in an object with genus 0. Since we are specifically interested in objects with genus$\geq$ 1, we make the following observation about a torus. A torus, will also have an initial endcap and final endcap, in addition it will have a split and a merge, (a 1-to-2 ribbon and a 2-to-1 ribbon). Therefore, the Euler characteristic will be: 1+1-1-1 = 0, leading to the correct genus: 1.

A.2.2 Re-tiling of the object with preserved topology

Now, our process of re-tiling the whole object with a sub-sampling of the total rings can be proven to preserve the topology in just a few words. Since a stack of 1-to-1 ribbons have a zero Euler characteristic, we can easily simplfy this down to a single large ribbon between the intial ring and the final ring, without changing the topology. With a split or a merge, we carefully respect the branching by preserving the associated 1-to-n ribbons. Finally, endcaps are retained, guaranteeing an Euler characteric equal to the original surface. Despite severe downsampling, our subsampling of the entire Surfel graph, preserves the correct topology of our initial discrete surface, based on the properties shown in theorems T1 and T3.

A.2.3 A note about tails

We now show that the removal of tails from the rings in the topological graph does not change the Euler characteristic of the surface. A tail can be seen as just a sucession of Surfels, and these Surfels are arbitrarily triangulated (for example, hanging from a complete ring, see Fig. A.5). It is easy to see that each piece of the tail consists of adding 1 vertex, 2 edges and 1 face. Thus the Euler characteristic for a ribbon with a tail is just $\chi(ribbon) + \chi(tail)$ where $\chi(tail) = 1 + 2 + 1 = 0$.

A.2.4 Boundaries

All the previous arguments apply to a closed initial surface. Although our current implementation deals only with this case, the same proofs can be derived for open objects.

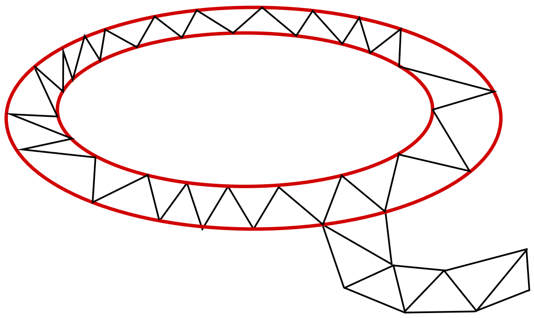

Figure A.5: *A ribbon (or ring) with a tail. The boundaries of this ring, without the tail (shown in red) are 1-spheres.*

Boundaries could exist if the surface represented by the volume data exceeds the limits of the volume. These boundaries could easily be found and identified. Then, similar arguments (adding caps at the boundaries) can be used to derive the general formula: $\chi = V - E - F = 2(1-g) - b$ where now g indicates the genus of the closed object *once the boundaries have been closed*, and b is the number of boundaries. All the rest of the derivation then holds.

Bibliography

[1] 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.

[2] C. Bajaj and V. Pascucci. Progressive isocontouring. Technical Report TR 99-36, University of Texas at Austin, 1999.

[3] Y. Chehadeh, D. Coquin, and P. Bolon. A skeletonization algorithm using chamfer distance transformation adapted to rectangular grids. In *ICPR96*, page B71.3, 1996.

[4] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

[5] Fehmi Cirak, Michael Ortiz, and Peter Schröder. Subdivision surfaces: A new paradigm for thin-shell finite-element analysis. *Int. J. Numer. Meth. Engng.*, 47, 2000.

[6] Laurent D. Cohen and Isaac Cohen. Finite-Element Methods for Active COntour Models and Balloons for 2D and 3D Images. *IEEE Trans. PAMI*, 15(11):1131–1147, 1993.

[7] D. Cohen-Or, D. Levin, and A. Solomivici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

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

[9] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Active implicit surface for computer animation. In *Graphics Interface (GI'98) Proceedings*, pages 143–150, Vancouver, Canada, 1998.

[10] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *SIGGRAPH 99 Conference Proceedings*, pages 317–324, August 1999.

[11] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Anisotropic Feature-Preserving Denoising of Height Fields and Bivariate Data. In *Graphics Interface'2000 Proceedings*, May 2000.

[12] 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.

[13] 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.

[14] H. Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57–97, 1974.

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

[16] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH 96*, pages 209–216, 1996.

[17] S. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 Symposium on Volume Visualization*, pages 23–30. ACM SIGGRAPH, October 1998.

[18] D. Gordon and J.K. Udupa. Fast surface tracking in three-dimensional binary images. *Computer Vision, Graphics, and Image Processing*, 45(2):196–241, February 1989.

[19] G. Greiner. Variational design and fairing of spline surfaces. *Computer Graphics Forum*, 13(3):143—154, 1994.

[20] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99*, pages 325–334, 1999.

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

[22] Paul Heckbert, editor. *Graphics Gems IV*. Academic Press, Boston, 1994.

[23] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University, 1997.

[24] Hugues Hoppe. Progressive meshes. *Proceedings of SIGGRAPH 97*, pages 189–198, 1997.

[25] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. *Proceedings of SIGGRAPH 94*, pages 295–302, 1994.

[26] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH 93*, pages 19–26, 1993.

[27] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. In *1st Conference on Computer Vision*, pages 321,331, London, U.K., June 1988.

[28] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. *Proceedings of SIGGRAPH 00*, 2000.

[29] R. Kimmel and J.A. Sethian. Fast marching method on triangulated domains. In *Proceedings of the National Academy of Science*, volume 95, pages 8341–8435, 1998.

[30] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, July 1998.

[31] Leif P. Kobbelt, Jens Vorsatz, Ulf Labsik, and Hans-Peter Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, 18(3):119–130, 1999.

[32] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96*, pages 313–324, 1996.

[33] J.-O. Lachaud. Topologically Defined Iso-surfaces. Research Report 96-20, Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France, 1996.

[34] J.-O. Lachaud and A. Montanvert. Deformable meshes with automated topology changes for coarse-to-fine 3D surface extraction. *Medical Image Analysis*, 3(2):187–207, 1999.

[35] F. Lazarus and A. Verroust. Level set diagrams of polyhedral objects. In *Proceedings of the Fifth Symposium on Solid Modeling and Applications*, pages 130–140. ACM, June 1999.

[36] 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.

[37] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Computer Graphics (SIGGRAPH '00 Proceedings)*, July 2000.

[38] Peter Lindstrom and Greg Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98–115, 1999.

[39] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[40] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algortithm. *Computer Graphics (Proceedings of Siggraph '87*, 21(4):163–169, 1987.

[41] R. Malladi, J.A. Sethian, and B.C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Trans. PAMI*, 17(2):158–175, 1995.

[42] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[43] James V. Miller, David E. Breen, William E. Lorensen, Robert M. O'Bara, and Michael J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):217–226, 1991.

[44] Edwin E. Moise. *Geometric Topology in Dimensions 2 and 3*. Springer-Verlag, New York, USA, 1977.

[45] B. Payne and A. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.

[46] Hong Qin, Chhandomay Mandal, and Baba C. Vemuri. Dynamic catmull-clark subdivision surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):215–229, 1998.

[47] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, 1995.

[48] J.A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[49] 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.

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

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

[52] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *ACHA*, 3(2):186–200, 1996.

[53] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.

[54] Gabriel Taubin. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358, August 1995.

[55] A. Verroust and F. Lazarus. Extracting skeletal curves from 3d scattered data. *The Visual Computer*, 16(1):15–25, 2000.

[56] Rüdiger Westermann, Lief Kobbelt, and Thomas Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[57] R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *Proceedings of the Third International Workshop on Implicit Surfaces*, pages 19–35. Eurographics Association, June 1998.

[58] Ross T. Whitaker and David T. Chen. Embedded active surfaces for volume visualization. In *SPIE Medical Imaging VIII*, pages 340–352, 1994.

[59] J. Wilhelms and A. Vangelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

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

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