Topology-Constrained Surface Reconstruction From Cross-sections

Ming Zou* Washington U. in St. Louis
Michelle Holloway † Washington U. in St. Louis
Nathan Carr ‡ Adobe
Tao Ju§ Washington U. in St. Louis

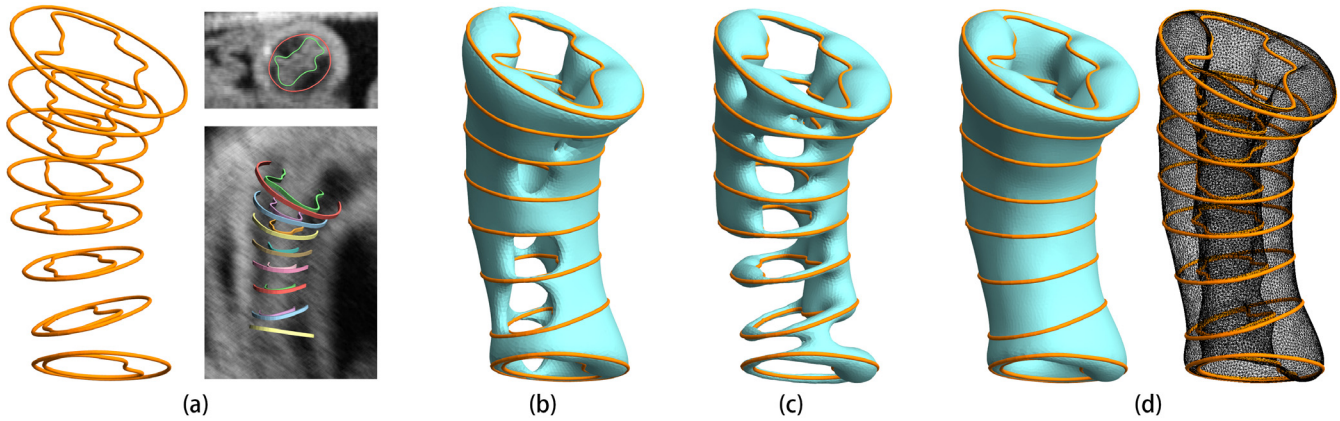

Figure 1: *Non-parallel cross-section curves delineating a developing chicken heart from a CT volume (a) and the reconstructed surfaces using the method of Lu et al. [2008] (b), method of Bermano et al. [2011] (c), and our method with genus-1 constraint without utilizing the CT volume (d). The first two surfaces contain numerous topological tunnels, while ours correctly captures the shell-like shape of the object.*

Abstract

In this work we detail the first algorithm that provides topological control during surface reconstruction from an input set of planar cross-sections. Our work has broad application in a number of fields including surface modeling and biomedical image analysis, where surfaces of known topology must be recovered. Given curves on arbitrarily oriented cross-sections, our method produces a manifold interpolating surface that exactly matches a user-specified genus. The key insight behind our approach is to formulate the topological search as a divide-and-conquer optimization process which scores local sets of topologies and combines them to satisfy the global topology constraint. We further extend our method to allow image data to guide the topological search, achieving even better results than relying on the curves alone. By simultaneously satisfying both geometric and topological constraints, we are able to produce accurate reconstructions with fewer input cross-sections, hence reducing the manual time needed to extract the desired shape.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: Surface reconstruction, cross-section interpolation, contour stitching, topology constraint, dynamic programming

*e-mail:mingzou.cn@gmail.com
†e-mail:mavaugha@wustl.edu
‡e-mail:ncarr@adobe.com
§e-mail:taoju@cse.wustl.edu

1 Introduction

The need to create surfaces depicting a 3D object from its cross-sections arises in many applications. For instance, in order to reconstruct the surface of an organ captured by CT or MRI, radiologists typically start by manually delineating the organ boundary on selected 2D slices of the 3D image volume. The delineated planar curves then need to be connected to form a closed surface. Figure 1 (a) shows an example stack of the delineated curves of a layer of a developing chicken heart from a CT volume (images on the right show two slices through the CT volume).

In many application domains, there exists strong prior information about the object to be depicted. One piece of such information is topology, which describes the object's connected components and their genus. For example, a biological structure usually has a known topology. Most structures have the topology of a sphere, which is a single component with zero genus. Some structures have a more complicated topology. A layer of a developing chicken heart is a cylindrical shell with genus 1. Bones in our body, such as vertebrae and the hip bone, can have a higher genus. Accurately capturing such topology not only helps with recovering the shape of the object but is also important for downstream applications of the reconstructed surface, such as shape matching, fluid dynamics simulation, and mechanical analysis.

Much research has been done for creating surfaces from cross-sections (see a more detailed review in Section 2). While the state-of-the-art algorithms are capable of creating watertight surfaces from even the most complex inputs, no method to date can guarantee that their output has a predefined topology. In fact, it is quite common for these methods to create topological errors, particularly when the cross-sections are not dense (see Figure 1 (b,c)). While it is possible for these methods to achieve the correct topology by providing them with more cross-sections, this will require additional manual labor and the data may not be available at all.

In this paper, we present a novel algorithm for reconstruction from cross-sections that allows precise control over surface topology

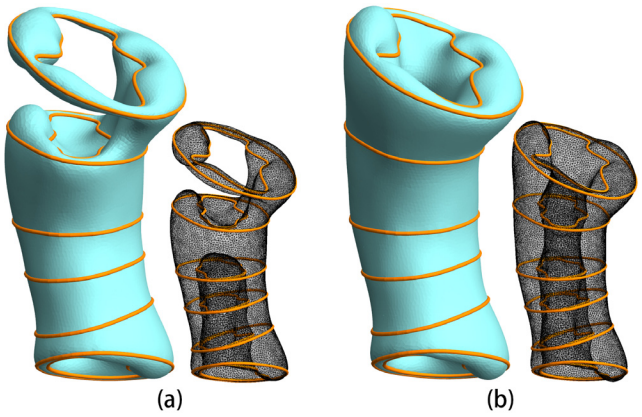

Figure 2: *Reconstructed surfaces of our method with genus-1 constraint, without (a) and with (b) the use of the original CT volume, from the same stack in Figure 1 after removing two cross-sections.*

(genus). We modify the classical divide-and-conquer strategy for solving such problems and combine it with global combinatorial optimization. Like previous methods, we consider the arrangement of the cross-sections, which divides the space into polyhedral volumes (*cells*). Instead of creating a single surface (called a *tile* in this work) within each cell as previous methods do, we explore a family of distinct topologies of tiles and compute a score that assesses the likelihood of each topology. In the optimization step, one topology within each cell is chosen so that the overall surface has the desired genus while the cumulative score from all cells is maximized. Our method can handle both parallel or non-parallel cross-sections, and each cross-section may contain multiple curves with arbitrary nesting relations. The result of our method for the same stack of chicken heart cross-sections is shown in Figure 1 (d).

If 3D image data is available (e.g., original MRI or CT images from which the cross-section curves are delineated), our algorithm can easily utilize it to achieve ever better reconstruction. This is done by biasing the set of tile topologies as well as their scores to respect the strong boundaries in the image. As an example, we tested our algorithm on the same input as in Figure 1 (a) after removing two cross-sections (2nd and 4th from top). Without using the original CT volume, our method creates a surface with the correct genus but fails to capture the shell-like shape of the object (Figure 2 (a)). Utilizing the CT volume allows our algorithm to produce a surface with both the correct topology and shape (Figure 2 (b)). Note that the quality of a real-world biomedical image (as seen in Figure 1 (a)) is often too low for any automatic reconstruction method to produce a reasonable output.

Contributions: We present the first surface reconstruction algorithm from cross-sections that allows global topology (genus) control. Our method enriches the typical divide-and-conquer strategy with an optimization component, and addresses two novel computational problems:

1. How to explore the potentially enormous space of tile topologies within a cell?
2. How to optimally select tile topologies in individual cells under a global topology constraint?

For the first problem, we reduce the exploration space by considering the topology of level sets in a scalar function. We propose a definition of the function based on random walks, which can easily incorporate additional image data, and present algorithms for enumerating and scoring distinct level set topologies. For the second problem, we solve it with a bottom-up dynamic-programming algorithm that is guaranteed to find the optimal solution meeting the topology constraint (if it exists).

2 Related works

Surface from cross-sections: Most methods for this problem take a divide-and-conquer approach based on the arrangement of the cross-sectional planes, and they differ by how the tiles within each cell is computed. If a cell is bounded by two parallel cross-sections and each has one curve, the problem can be formulated as searching for the best correspondence between the vertices on the two curves that optimizes some quality measure [Keppel 1975; Fuchs et al. 1977; Meyers et al. 1992]. However, this formulation no longer applies to cells with multiple, possibly nested curves. There are three common strategies proposed for arbitrary curves. One can consider the Delaunay triangulation of the cell constrained by the cross-section curves [Boissonnat 1988; Cheng and Dey 1999]. This strategy has been successfully extended to cells bounded by non-parallel cross-sections [Boissonnat and Memari 2007]. The second strategy partitions the cell interior by projecting the curves from the cell boundary onto some set of middle sheets of the cell (e.g., medial axis or straight skeleton) [Bajaj et al. 1996; Barequet and Sharir 1996; Oliva et al. 1996; Barequet et al. 2004; Barequet and Vaxman 2007]. Besides non-parallel cross-sections, this strategy can reconstruct non-manifold surface networks that partition the space into regions with multiple labels [Ju et al. 2005; Liu et al. 2008; Barequet and Vaxman 2009]. The last strategy, to which our method belongs, extracts the tile as the level set surface of a scalar function inside the cell that interpolates the labelling on the cell boundary. The function can be defined using linear interpolation [Herman et al. 1992] (limited to cells bounded by two cross-sections), radial basis functions [Turk and O'Brien 1999], or mean-value interpolation [Bermano et al. 2011]. Note that some methods using the projection strategy (e.g., [Liu et al. 2008]) may also be considered as a function-based method where the function is defined by nearest-neighbor interpolation.

All of the above methods offer only *one* choice of tile per cell. The topological correctness of this tile is highly dependent on the density of the cross-sections; a denser sampling usually leads to more likely topology. An excessive number of cross-sections may be required, particularly for thin objects and cross-sections at a non-optimal angle (see a more formal study of sampling conditions by Amini et al. [2013]). Figure 3 demonstrates two typical failure cases for the projection strategy: a thin curve that migrates slightly from one cross-section to the next disconnects the surfaces (top), whereas a change in the shape of the curve introduces topological tunnels (bottom).

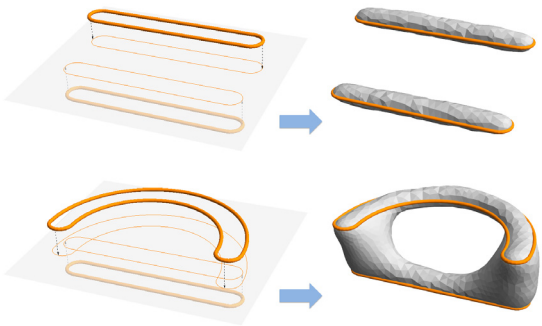

Figure 3: *Two examples of topological errors (right) made by a project-based method [Liu et al. 2008] between two parallel cross-sections each containing a thin curve (left, showing orthogonal projections of the curves onto the middle plane).*

We know of only two pieces of work that consider *multiple* choices of reconstructions per cell, albeit in a lower-dimensional setting. To reconstruct a 2-dimensional polygon from points on 1-dimensional cross-section lines, Barequet et al. [2006] enumerates all possible topologies within each cell by connecting the input points using straight line segments. These topologies are then pruned using a local sampling condition. In the context of vector graphics, Zhou et al. [2014] builds a large polygon by concatenating a sequence of small polygons chosen from a pre-defined set. Dynamic programming was used to select one piece for each location in the sequence so that the resulting pattern has one connected component. In contrast, the 3-dimensional reconstruction problem that we consider is more challenging in several ways. First, we do not have any pre-defined pieces to choose from. Second, connecting free-form curves in 3D by surfaces is much more challenging than connecting points in 2D, not to mention enumerating distinct surface topologies. More importantly, we consider a global optimization problem constrained not only by the connected components of the surface, but also its genus, which is unique in 3D.

Surface reconstruction and topology control: Most surface reconstruction methods (whether from cross-sections or other data) do not have topological guarantees, and hence the correct topology has to be rectified *after* the surface is built [Attene et al. 2013]. The downside to this approach is that topology rectification methods such as [Wood et al. 2004; Zhou et al. 2007] are oblivious of the original data from which the input surface was obtained. Hence the repaired output, although having the correct topology, may lose important properties of the input surfaces (e.g., interpolating the cross-section curves).

Very few existing works provide topology control *during* reconstruction, however, it has been increasingly recognized that topology plays a vital role in extracting the correct shapes. Sharf et al. [2007] developed an interactive reconstruction tool from point clouds in which the user corrects ambiguous topology by scribbles. The recent work of Yin et al. [2014] allows the user to edit skeletal curves to prescribe the surface topology. In medical imaging, methods like active contours, topo-preserving fast marching [Bazin and Pham 2007], and topo-preserving max flow [Zeng et al. 2008] segment an object of interest by morphing an initial template while retaining its topology. Interactions and templates are both utilized in the recent system of Ijiri et al. [2014] for flower modeling. Unlike these works, our method requires no template or additional user interaction besides the input data.

3 Overview

The input to our algorithm is a set of planar *cross-sections*, each containing one or multiple *section curves*[1] along the intersection between the plane with the object. These section curves partition each cross-section plane into *inside* and *outside* regions. The cross-sections can be arbitrarily oriented, and the section curves can have arbitrary shape and nesting topology. An optional input is a 3-dimensional intensity volume where the object's surface is roughly aligned with intensity boundaries.

The output of the algorithm is a closed surface with a user-specified genus that interpolates the section curves. More precisely, the surface should impose an inside/outside labelling of the space that is consistent with the labelling on each cross-section.

Our method combines divide-and-conquer with global optimization. We consider the arrangement of the cross-sections, which partitions the space into convex *cells*. Just like each cross-section, the cell boundary is partitioned by section curves into inside/outside regions. We compute the part of the final surface within each cell, which we call a *tile*. Note that a tile may consist of one or more connected components, each bounded by one or more section curves on the cell boundary. A tile should partition the cell into inside/outside volumes that agree with the labelling on the cell boundary.

The algorithm proceeds in three steps:

Step 1 (Topology exploration): Compute a family of distinct topologies of tiles within each cell and score the likelihood of each topology.

Step 2 (Topology selection): Select one tile topology per cell so that the topology of the surface combined from all cells has the user-specified genus while the cumulative score is maximized.

Step 3 (Surfacing): Output a smooth surface that realizes the chosen tile topology within each cell.

We shall detail each step in the following three sections. As our primary goal is to obtain a correct topology, the focus is placed on the first two steps (which are our main contributions).

4 Topology exploration

Given a cell and section curves on the cell boundary, we would like to explore distinct topologies of tiles that connect the curves. Without any restrictions, the space of such topologies is infinite, since a tile can have an arbitrarily number of handles. Even if we only consider genus-zero tiles, the number of possible tile topologies can potentially be as big as the number of possible partitions of the section curves. This latter number is known as the Bell number, which has double-exponential growth. Enumerating all such topologies, and scoring each of them, can be computationally prohibitive. Moreover, creating the geometry of a surface that realizes an arbitrary topology can be challenging.

To address these challenges, we limit our exploration to a smaller space of topologies that come from the level sets of an *indicator function* defined within the cell. Although these level sets only represent a subset of possible tile topologies, they span a spectrum of connectivity of the object within the cell, and their geometry can be easily reconstructed. As a bonus, our definition of the indicator function offers a natural way to score each tile topology.

While previous methods have also considered level sets of scalar functions for creating tiles [Herman et al. 1992; Turk and O'Brien 1999; Bermano et al. 2011], we make two notable differences. First, we consider not one, but a family of level sets of the function. Second, our choice of the indicator function is different from previous ones and allows easy incorporation of available image data.

4.1 Indicator function

We consider an indicator function f with the following properties. In the interior of the cell, f is continuous and takes on values in the open range of $(0, 1)$. The extension of f to the cell boundary is a discontinuous, piece-wise constant function that is 1 (resp. 0) in the inside (resp. outside) region. Note that a connected component of any level set of f with these properties is either closed or bounded by the section curves.

An ideal indicator function f should capture the likelihood of a point being in the inside of the object. In addition, the level sets of f should form valid tiles (e.g., avoids closed surfaces not bounded by any curves). Last but not least, the definition of f should also

[1] While some previous works call such curves *contours*, we reserve this notion for components of iso-surfaces (as in *contour trees*); see Section 4.

make it easy to incorporate information from an intensity volume if it is available. To meet these criteria, we adopt the *random-walk probability*. Consider a weighted graph where each edge weight acts as the transition probability and a subset of the nodes have been labelled as sources or sinks. We seek the probability of a random walker starting from each node that ends at a source (as opposed to sink) node. Such probabilities have a number of physical interpretations such as steady-state temperature in heat diffusion and voltage in electric circuits. The probability distribution lacks local extrema, which ensures that the level-set of the probability is always connected to the cell boundary. Moreover, by biasing the walker to avoid crossing sharp gradients in an image, the probability would respect object boundaries in the image [Grady 2006]. As a result, the random-walk probabilities fit nicely as our indicator function.

For completeness, we first briefly review computing the random walk probability on a graph (see a more thorough account in [Grady 2006]). We then discuss how to define and compute the indicator function by discretizing the cell.

Random walk on a graph Consider a connected undirected graph where a subset of the vertices, called *seeds*, have been labeled with either 1 or 0. In addition, each edge between vertices i, j is associated with a positive weight w_{ij} that acts as a bias on the random walker. Specifically, for the walker standing at vertex i, its probability to pick vertex j among all incident vertices as its next stop is $w_{ij}/\sum_{j\in N(i)} w_{ij}$ where $N(i)$ denotes the set of vertices incident to i.

The probability, x_i, of a (biased) random walker starting from a non-seed vertex i that ends in some seed vertex with label 1 is the solution to a combinatoric Dirichlet problem. The problem seeks values that coincide with the given labels (1 or 0) at the seeds and has zero graph Laplacian at each non-seed vertex, or

$$x_i - \frac{\sum_{j\in N(i)} w_{ij}x_j}{\sum_{j\in N(i)} w_{ij}} = 0. \quad (1)$$

The above constraints form a linear system of equations from which x_i at each non-seed vertex can be found. The solution is in fact a discrete harmonic function when w_{ij} is a constant value on all edges. To bias the walker to avoid large gradients in an intensity field, we adopt the weight definition from [Grady 2006],

$$w_{ij} = \exp(-\beta(I_i - I_j)^2) \quad (2)$$

where I_i is the image intensity at the location of vertex i, and β controls the fall-off speed of the weight and is left as a user-chosen parameter.

Random walk indicator function We define our indicator function f as a piece-wise linear interpolation of the random walk probabilities computed at the vertices of a tetrahedral mesh. The mesh is created using Tetgen [Si 2007] as a Delaunay Triangulation conformed to the cell boundary and the section curves. We set all vertices on the cell boundary as seeds, and we label those seeds that are either inside or on the section curves as 1 while labelling the rest as 0. The probabilities at the vertices interior to the cell are then computed on the edge graph of the mesh.

To ensure that linear interpolation yields an indicator function, we need to further make sure that an edge connecting two seed vertices with the same label (e.g., 1) lies in the corresponding region on the cell boundary. Otherwise there will be points inside the cell where f attains extreme values (0 or 1) or points on the cell boundary where f is inconsistent with the inside/outside labels. We apply a post process to remove all edges that violate this criteria (called an *inconsistent edge*) by splitting each edge at their midpoints and subdividing their incident faces and tetrahedra.

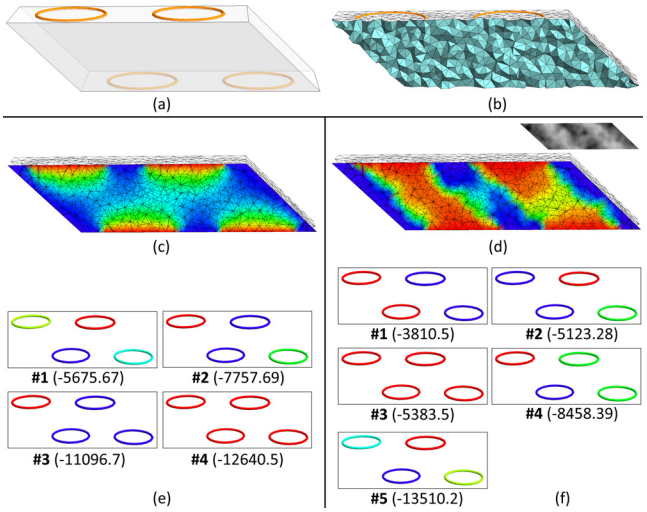

Figure 4: *Topology exploration: (a) a cell with 4 section curves, (b) the boundary-conforming Delaunay tetrahedral mesh constrained by the section curves, (c) the indicator function on a slice of the cell (1: red; 0: blue), (d) the function biased by an intensity volume (a slice is shown in the insert), (e,f) the tiling set topologies ordered with descending scores for each indicator function (a topology is visualized by the coloring the subsets of section curves that are connected by a contour).*

An example of the tetrahedral mesh and the computed indicator function f is shown for a 3D cell in Figure 4 (b,c,d). Observe that f computed in the absence of any additional intensity volume (shown in (c)) approximates the harmonic function in the cell, whereas the use of the intensity volume strongly biased f so that its level sets are aligned with the intensity boundaries (shown in (d)).

An inherent limitation of linear interpolation is that it cannot capture the jump between 0 and 1 across the section curves on the cell boundary. The consequence is that the 2-dimensional level sets of f inside the cell do not exactly interpolate the section curves. However, the level sets are *almost* interpolating: any 1-dimensional level set of f on the cell boundary lies in the one-ring neighborhood of the section curve vertices. Furthermore, in the absence of inconsistent edges, any 1-dimensional level set of f on the cell boundary is homeomorphic to the section curves. In the rest of this section, we shall assume that the level sets of f are interpolating, and interpolation will be enforced at the final surfacing stage.

4.2 Exploring tile topologies

The level sets of our indicator function f are natural candidates of tiles. Each level set $f^{-1}(u)$ with level $u \in (0, 1)$ is a water-tight, intersection-free surface. Each connected component of the level set, called a *contour*, is bounded by some section curves on the cell boundary. The level set induces a natural partitioning of the cell interior into inside/outside volumes that matches the labeling on the cell boundary.

The sequence of level sets parameterized by the level offers a variety of tile topologies. At a level close to 1, each inside region on the cell boundary is enclosed by one contour, forming a maximally connected outside volume. At a level close to 0, each outside region is enclosed by one contour, forming a maximally connected inside volume. A family of tile topologies lies in between these two extreme cases.

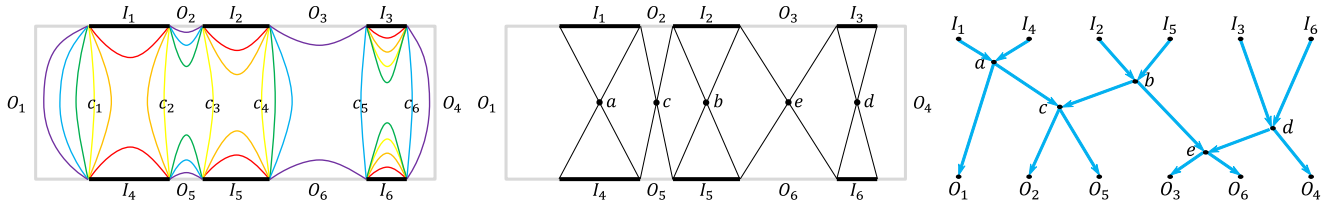

Figure 5: *Level sets topology in 2D: (a) A rectangular cell with 6 inside regions (I_i) and 6 outside regions (O_i) on the cell boundary, the contours of the indicator function (colored by their levels), and a tiling set (c_i). (b) The critical points (a, b, c, d, e) and the contours that contain them (black lines). These contours divide the cell into regions of contours with a common topology. (c) The contour tree that captures the critical points as nodes and regions as edges.*

To further enrich this family, we also consider contours at *different* levels. Note that any set of contours, regardless of their levels, are closed and intersection-free. To form a valid tile, we only need to require that each section curve is used by exactly one contour. We call a set of contours that meet this requirement a *tiling set*. Any level set is a tiling set. Just like a level set, a tiling set partitions the cell interior into inside/outside volumes.

Tiling sets offer a wider range of topologies than level sets. This is particularly important in the scenario where the object to be reconstructed contains multiple components in the cell and the topology of each component is only realized by level sets at different levels. We illustrate this case in the 2D example in Figure 5. Here we consider a rectangular cell bounded by four cross-section lines. The cell boundary is partitioned into several inside regions (I_i) and outside regions (O_i). An intuitive reconstruction within the cell would be three "tubes" that connect the three pairs of inside regions $\{I_1, I_4\}$, $\{I_2, I_5\}$ and $\{I_3, I_6\}$. The level sets of the indicator function with distinct topology are illustrated by colored curves, where contours at the same level are colored the same. Note that no single level set has the desired 3-tube connectivity. This is because the pair $\{I_3, I_6\}$ is smaller than and further away from the other two pairs. Any level set that connects this pair into a tube (e.g., blue) would cause the other two pairs to be merged into a single branching tube. On the other hand, the 3-tube connectivity can be achieved by the tiling set consisting of contours $c_1, \ldots, c_6$ at two different levels.

Our exploration proceeds in two steps. First we will enumerate distinct contour topologies, and they will be combined in a second step to form topologies of tiling sets.

Topology of simple contours We start by enumerating all possible contour topologies in our indicator function. Such topologies are captured by the *contour tree* [van Kreveld et al. 1997], which is a directed tree-like graph that captures the evolution of the contour topology as the level changes. Each node in the tree corresponds to a critical point in the function and each edge corresponds to a set of contours with a common topology (an example is shown in Figure 5 (c)). We use the algorithm in [van Kreveld et al. 1997] to compute the contour tree, which also gives the topology information for each edge of the tree. We encode each distinct contour topology by its genus and bounding section curves.

In our experiment, we observed that a large number of contour topologies have genus greater than zero. These topologies significantly increase the number of tiling set topologies that we will explore next, and they are rarely selected to be used on the final reconstruction. To improve efficiency, we make the assumption that any topological feature of the object (e.g., handles) is sampled by some cross-section, and hence we only consider contours within a cell that are *simple*, or has zero genus. Note that relaxing this restriction does not pose any difficulty to the rest of the algorithm.

Topology of tiling sets Given the contour topologies, we next enumerate ways in which they can be combined to form tiling sets. Recall that a set of contours is a tiling set if each section curve on the cell boundary bounds exactly one contour in the set. The task of finding tiling sets can therefore be formulated as an *exact cover* problem as follows. Suppose we have a set of section curves C and a collection of contour topologies $T = \{t_1, \ldots, t_m\}$, where each t_i is a subset of C that bound the contour. Our goal is find a sub-collection $T' \subseteq T$ such that each curve in C is appears exactly once in some subset in T'.

The exact cover problem is one of Karp's 21 NP-complete problems. Kunth's Dancing Link algorithm [Knuth 2000] can be used to find all possible solutions, but its running time may scale exponentially with the input size. In our implementation, however, we found that Dancing Link runs sufficiently fast (e.g., within a second) for finding tiling set topologies in all our test examples.

4.3 Scoring

To evaluate the "naturalness" of a tile topology, we again make use of our indicator function which approximates the random-walk probability. Conceptually, we consider the inside/outside labelling of the cell interior imposed by a tile, and score the tile by the joint likelihood of the label at each point. We first present our definition of the score function for any given tile in the cell (not limited to tiling sets of the indicator function). Then we discuss how the score is computed for a topology of a tiling set.

Scoring an arbitrary tile To measure the likelihood of a tile T as a labelling of the cell space, we first calculate the likelihood of the labeling at a point x inside the cell. Let f be the random-walk probability, and l_T be the label at x as determined by the tile T. The likelihood of the label at x given tile T can be formulated as

$$h_T(x) = \begin{cases} \log(f(x)) & \text{if } l_T(x) \text{ is inside} \\ \log(1 - f(x)) & \text{if } l_T(x) \text{ is outside} \end{cases} \quad (3)$$

We use the logarithm so that the joint-likelihood of multiple points can be calculated by summation. Integrating the per-point likelihood over the entire cell space Ω gives the continuous formulation of our score for the tile T,

$$h(T) = \int_\Omega h_T(x) dx. \quad (4)$$

In practice, we compute the score discretely over the vertices V of the tetrahedral mesh. To compensate for the variation in tetrahedra sizes in different cells, we compute the following weighted sum

$$h(T) = \sum_{v \in V} h_T(v) w(v). \quad (5)$$

where $w(v)$ is the sum of volumes of tetrahedra incident to vertex v. Since $w(v)$ can be pre-computed from the mesh structure, the only computation involved in scoring a given tile is computing the inside/outside label $L_T(v)$ for each vertex v from the tile.

Scoring a tiling set topology Given a tiling set topology, we need to pick one tiling set that realizes the topology in order to compute the score. This amounts to finding a *representative* contour for each contour topology. Ideally, the representative contour should maximize the score. Note that our score function increases when an inside point x where $f(x) < 0.5$ changes its label to outside or when an outside point y where $f(y) > 0.5$ changes its label to inside. Hence replacing a contour by another (with the same topology) whose level is closer to 0.5 always increases the score.

With this observation, we define the representative contour for each contour topology as one whose level is closest to 0.5. Specifically, suppose the range of levels of the contour topology is $[u_l, u_h]$ (this information is readily available on the contour tree). If the range contains the value 0.5, we shall use the contour at level 0.5. Otherwise, we use the contour at level $u_l + \epsilon$ if $u_l > 0.5$ and at level $u_h - \epsilon$ otherwise, where ϵ is a small value so that the contour avoids the critical points.

Figure 4 (e) shows all tiling set topologies found in the indicator function in (c), ranked by their scores. Note that the family of topologies captures a range of connections between the section curves, from one that disconnects all section curves (#1) to one that connects all of them (#4). On the other hand, the use of the intensity volume encourages topologies (f) that are better aligned with the strong intensity boundaries, many of which are not present in the original family (e).

5 Topology selection

Given a set of tile topologies with scores within each cell, we need to choose one topology per cell so that their union is a surface with a single connected component and the user-specified genus g_u. In addition, we seek the choice that has the maximal cumulative scores among all possible choices. We call such choice the *optimal tiling*. We solve this combinatorial optimization using a bottom-up dynamic programming algorithm, which is guaranteed to find the global optimum. Note the algorithm is independent from how the tile topologies are generated or scored within the cells.

5.1 A dynamic programming problem

Before presenting the algorithm, we first need to show that the problem of finding optimal tiling has the required properties to be solved by dynamic programming, namely *optimal substructure* and *overlapping subproblems*.

We consider a more general problem as follows. Given a volume V formed by the union of a collection of cells, we seek the optimal tiling for cells in V constrained by a given topology T. Here, T is represented by its connected components, and each component is encoded by its genus and the (possibly empty) subset of the section curves on the boundary of V that bound the component. We denote the score of the optimal tiling as $h(V, T)$. Our topology selection problem is simply finding $h(V, T)$ (and the per-cell tile topologies that achieve this score) when V is the union of all cells and T contains a single component with genus g_u.

To see that the problem has an optimal substructure, consider any decomposition of V into sub-volumes V_1, V_2 that are unions of subsets of the cells in V. There could be multiple ways to choose a surface topology in each sub-volume, say T_1, T_2, so that the topology of the union of the two surfaces, denoted as $T_1 \vee T_2$, has the desired topology T. This is illustrated in the 2D example in Figure 6. Since the score of a tile topology in one cell is independent from the other cells, we can write $h(V, T)$ as the maximum sum of the scores over all possible combinations of T_1, T_2:

$$h(V, T) = \max_{T_1 \vee T_2 = T} (h(V_1, T_1) + h(V_2, T_2)) \tag{6}$$

Moreover, note that each subproblem $h(V_i, T_i)$ may be used in defining $h(V, T)$ for different choices of topology constraints T. Hence a dynamic programming algorithm is suited for finding h.

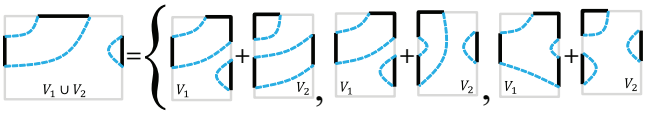

Figure 6: *A 2D illustration that the topology in a volume $V_1 \cup V_2$ (left) can be achieved by multiple combinations of topologies in each sub-volume V_1 and V_2 (right). The thick black segments denote the inside region on the boundary of the volume.*

5.2 A bottom-up algorithm

We perform dynamic programming in a bottom-up fashion, incrementally computing the solution within an increasingly larger union of cells. Given some ordering of the cells, we start from the first cell as the initial *known volume* and grow this volume by taking the union with subsequent cells. As we expand V, we incrementally update and store $h(V, T)$ (and the optimal tiling) for all possible surface topology T within V. Suppose V_1 is the previous known volume and V_2 is the newly added cell, such that $V = V_1 \cup V_2$. We enumerate all combinations of any topology T_1 stored at V_1 and any tile topology T_2 given in the cell V_2. We then use the recurrence in Equation 6 to compute $h(V, T)$ for all distinct union topologies $T = T_1 \vee T_2$. After all cells are included in the known volume V, we output the value $h(V, T)$ where T has a single connected component with the user-specified genus g_u.

A key element of the algorithm is the topological union operation $\vee$, which we shall elaborate first. We then discuss ways to improve the efficiency of the basic algorithm.

Topological union Given two surface topologies T_1, T_2, each consisting of one or more components, we first need to know which of these components are merged to a single component in the union topology $T = T_1 \vee T_2$. Note that a component in T_1 is merged with a component in T_2 if the two components share some common portion of their boundaries. We call these two components *boundary-connected*. Hence a component of T is a maximally boundary-connected set of components in T_1 and T_2.

For each component t of T, our next task is to compute its genus (g_t) and boundary curves (B_t) from those of the un-merged components in T_1 and T_2. Suppose t is made up of m components t_i^1 in T_1 for $i = 1, \ldots, m$ each with genus g_i^1 and boundary curves B_i^1, and n components t_j^2 in T_2 for $j = 1, \ldots, n$ each with genus g_j^2 and boundary curves B_j^2.

Since merging happens along boundaries, the portion of any B_i^1 (resp. B_j^2) that remains on the boundary of the merged surface B_t is precisely the portion that is not shared by any B_j^2 (resp. B_i^1). Hence we can obtain B_t using the *symmetric difference* operator $\oplus$,

$$B_t = (\cup_{i=1}^m B_i^1) \oplus (\cup_{j=1}^n B_j^2). \tag{7}$$

To compute the genus g_t, we resort to the Euler formula for a connected surface with boundary,

$$g_t = (2 - \|B_t\| - \chi_t)/2, \tag{8}$$

where $\|B_t\|$ denotes the number of loops in B_t, and χ_t is the Euler characteristics of t, which is computed as the sum of the number of vertices and number of faces minus the number of edges. Note that χ_t is the sum of Euler characteristics of the component surfaces adjusted by discounting those vertices and edges that are shared by different boundary curves. Let us denote the Euler characteristics for each component t_i^1 or t_j^2 as χ_i^1 or χ_j^2 (which can be computed from $\{g_i^1, B_i^1\}$ or $\{g_j^2, B_j^2\}$ using the Euler formula above). We have

$$\chi_t = \sum_{i=1}^{m} \chi_i^1 + \sum_{j=1}^{m} \chi_j^2 + e(L) - v(L). \tag{9}$$

Here v and e respectively count the number of vertices and edges of a graph, and L is the shared portions of the boundary curves,

$$L = (\cup_{i=1}^{m} B_i^1) \cap (\cup_{j=1}^{n} B_j^2). \tag{10}$$

The calculation of g_t and B_t is illustrated in Figure 7. In this example, two components from T_1 (t_1^1, t_2^1) and one component from T_2 (t_1^2) are merged. One of the boundary curves of t_1^2 completely overlaps a boundary curve of t_2^1, and another overlaps partly with two boundary curves of t_1^1 and t_2^1 (marked by red arrows). The symmetric difference in Equation 7 gives three curves on the boundary of t. To calculate the genus, note that the overlapping portion of the boundary curves is made up of a closed loop and two open curves, hence $e(L) - v(L) = -2$. Also, using Euler formula we have $\chi_1^1 = \chi_2^1 = 0$ and $\chi_1^2 = -1$. Applying Equation 9 gives Euler characteristic $\chi_t = -3$, and applying Equation 8 yields $g_t = 1$, which correctly captures the presence of a topological handle on t.

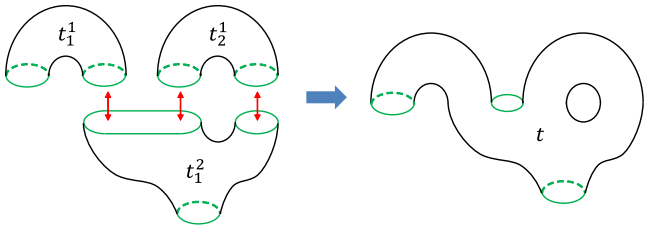

Figure 7: *Merging surfaces along their boundaries (green).*

Complexity and acceleration The time complexity of dynamic programming is upper bounded by the product of three quantities, the total number of cells (n_{cell}), the maximum number of tile topologies explored within any cell (n_{tile}), and the maximum number of topologies of tile unions that the algorithm creates in a known volume (n_{union}). The algorithm stores, for each union topology within a known volume, the optimal tiling of that topology which consists of one tile topology per cell. Hence the space usage is upper bounded by $n_{cell} \times n_{union}$.

The quantity n_{union} is the dominating factor of both time and space complexity. This number can be easily exponential on the number of section curves on the known volume's boundary. We propose two ways to tame the complexity. First, we use a greedy strategy to find the expansion order of cells that would reduce the number of section curves on the known volume. Specifically, the cell to be expanded next is selected as one among all remaining cells adjacent to the known volume that would yield the minimum number of section curves on the boundary of the expanded known volume. Second, we can prune those topologies that are inconsistent with the goal. Note that any topological handle or closed surface created during the algorithm will remain so throughout the expansion of the known volume. Since our final output is a single component with genus g_u, we prune all topologies that either contain closed surfaces or have genus greater than g_u at each step of dynamic programming.

We observed that, in a real-world reconstruction scenario (such as segmenting a biological shape), the object to be recovered usually has a low genus and the number of curves on each cross-section plane tends to be small. In this case, the combination of our cell ordering and pruning strategies can successfully reduce n_{union} to a small amount such that dynamic programming runs efficiently. As shown in Section 7, topology selection takes negligible time for all examples in this paper.

Nonetheless, the computational cost can become prohibitive if the object has a high topological complexity and each plane contains a large number of curves (e.g., a porus material or a neural network). To be able to handle these inputs, one could trade off optimality for efficiency by keeping only the top k topologies (as measured by their scores) in the known volume at each step of the expansion, where k is a user-supplied constant. This simple modification would make the algorithm run in time (and space) linear to the number of cells, regardless of the complexity of the input or the object, but possibly resulting in sub-optimal solutions.

6 Surfacing

Having found the optimal choice of tile topology per cell, we next need to construct the geometry of the tiles that realize those topologies. Ideally, the tile geometry should smoothly interpolate the input contours. A natural candidate is the set of representative contours that we used to score each tiling set topology (see Section 4). The geometry of these contours can be extracted as a piecewise linear surface using an iso-surfacing algorithm (e.g., marching tetrahedra) on the tetrahedral mesh.

While these contours possess the desired topology, they cannot be directly used as the final result for two reasons. First, as mentioned earlier, the contours within a cell don't exactly interpolate the section curves on the cell boundary. To enforce interpolation, we snap the boundary curves of a contour to the corresponding section curves. This can be done during the extraction of the contour. Recall the marching tetrahedra algorithm creates one contour vertex on each mesh edge that crosses the segment. If this edge lies on the cell boundary and only one of the edge's end point lies on a section curve, we simply use that end point as the contour vertex on that edge. Figure 8 (b,c) shows an example of contours extracted without and with snapping.

Second, the contours may exhibit a jagged appearance, in part due to the resolution of the tetrahedral mesh. Also, since the geometry of the contour in one cell is determined independently from contours in adjacent cells, simply gluing contours from different cells may not create a smooth overall shape. Following previous work [Liu et al. 2008], we improve the quality of the merged surface iteratively by alternating between Delaunay-based mesh refinement [Liepa 2003] and fairing based on surface-diffusion flow [Xu et al. 2006]. Both refinement and fairing are constrained by the section curves to maintain the interpolation property. An example result of mesh improvement is shown in Figure 8 (d).

7 Results

We test the ability of our algorithm in controlling surface topology on a variety of synthetic and real-world examples. We compare with the results of two leading methods by Bermano et al. [2011]

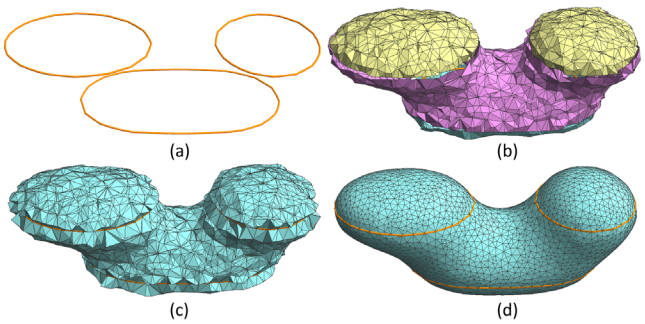

Figure 8: *Surfacing: (a) two cross-sections, (b) surface computed without snapping to section curves, (c) with snapping enabled, (d) after quality improvement.*

and by Lu et al. [2008]. Both methods can be regarded as taking a fixed level set of some indicator function defined within a cell, where the function is obtained by mean-value interpolation (in Bermano's method) or nearest-neighbor interpolation (in Lu's method). Besides the difference in the choice of the indicator function, our method considers a family of contour topologies per cell and strives to achieve a prescribed genus. While we focus on topological correctness in the comparisons, we note that these two methods offer additional features that our method is lacking, such as handling multi-labelled cross-sections and even missing labels.

Figure 9 examines the impact of cross-section sampling density on various methods using a synthetic torus. Without resorting to image data, our algorithm is able to produce surfaces with the desired genus (i.e. 1) in all three sampling densities (shown in column (d)). However, it fails to capture the torus structure in the sparsest input (bottom row). A closer look at a middle cell (see Figure 4) reveals that the correct tile topology in the cell, which should consist of two separate tubes, is missing in the tiling sets of our indicator function. Such tile topology can be recovered by incorporating a synthetic density volume (shown in the top-right), which allows our method to create surfaces with both correct topology and structure (shown in column (e)). Note that the synthetic volume was intentionally made noisy to mimic the quality of typical biomedical data. We also show (in (e) of bottom row) the result of our method with a different target genus (0), in which the torus is broken where the synthetic density is low in the volume.

In comparison, the other two methods produce topological errors for even denser inputs. As discussed earlier and observed in Figure 3, Lu's algorithm easily creates connections (resp. disconnections) between section curves whose projections are barely overlapping (resp. disjoint). On the other hand, Bermano's method appears to create the most disconnections, implying that the method might require a rather dense sampling to maintain the connectivity of the reconstruction.

Similar observations can be made in the chicken heart examples shown earlier. Our algorithm is able to recover the correct topology when the cross-sections are too sparse for the other two methods to succeed (Figure 1), and we can handle even fewer cross-sections when the original CT image is utilized (Figure 2).

Finally, we tested our algorithm (without the use of any additional images) on several more complex inputs depicting both everyday objects and biological structures, shown in Figure 11. These objects all contain thin sheets (e.g., mug and hip bone) or tubes (e.g., hand and vertebrae), which are challenging for existing methods. Our algorithm on the other hand is able to create a single connected surface with the desired (non-zero) genus.

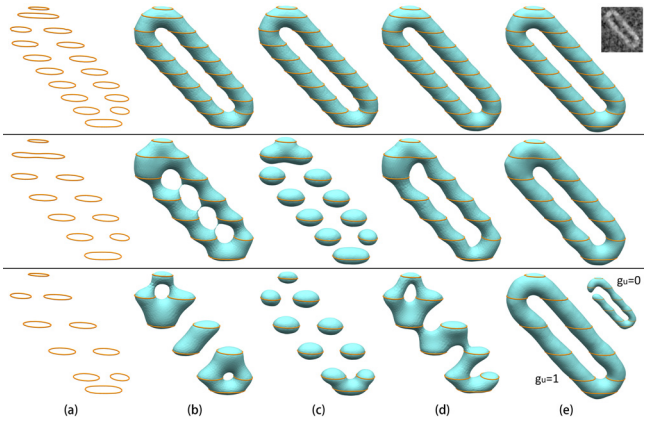

Figure 9: *Cross-sections of a torus at different sampling densities (a) and the reconstructed surfaces using the method of Lu et al. [2008] (b), Bermano et al. [2011] (c), our method with genus-1 constraint without additional image data (d), and our method using a synthetic but noisy image (shown in the top insert) (e).*

Performance Our algorithm was implemented in C++ and tested on a 12-core 2.4Ghz PC with 12G main memory. During topology exploration step, exploration in different cells is executed in parallel. Our implementation runs from a few seconds to less than two minutes for all examples shown in the paper (see last column of Figure 11). This run time is dominated by iterative fairing and refinement during the final surfacing, which in turn depends on the triangle count. Topology exploration finishes within seconds for all examples, while topology selection takes negligible time.

We further examined the performance of the dynamic programming algorithm for topology selection under varying factors such as the number of cross-section planes and target genus. Our input is a set of parallel cross-sections of a Trefoil knot (Figure 10 left). Note that there could be up to 8 section curves on the boundary of a cell and up to 5 section curves the boundary of the known volume during dynamic programming. We varied the number of cross-section planes from 6 to 55 and the target genus from 0 to 3. As seen in Figure 10 right, the running time of the algorithm increases almost linearly both with the number of planes and the target genus. Even with more than 50 planes and a target genus of 3, the algorithm finished in under a second.

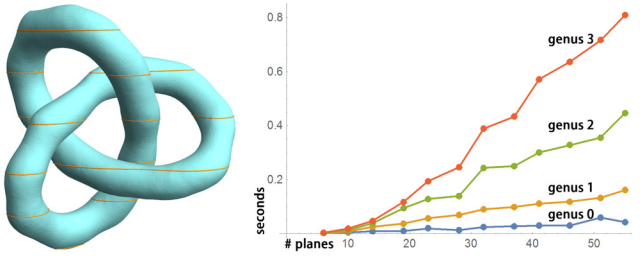

Figure 10: *Left: reconstruction of Trefoil knot from 6 cross-section planes with target genus 1. Right: running time of topology selection with increasing number of planes for different target genuses.*

8 Conclusion and discussion

We present, to our best knowledge, the first algorithm for controlling the topology of the surface when reconstructing from planar cross-sections. We solve the problem in two stages, first generating a family of topologies within each cell of the plane arrangement,

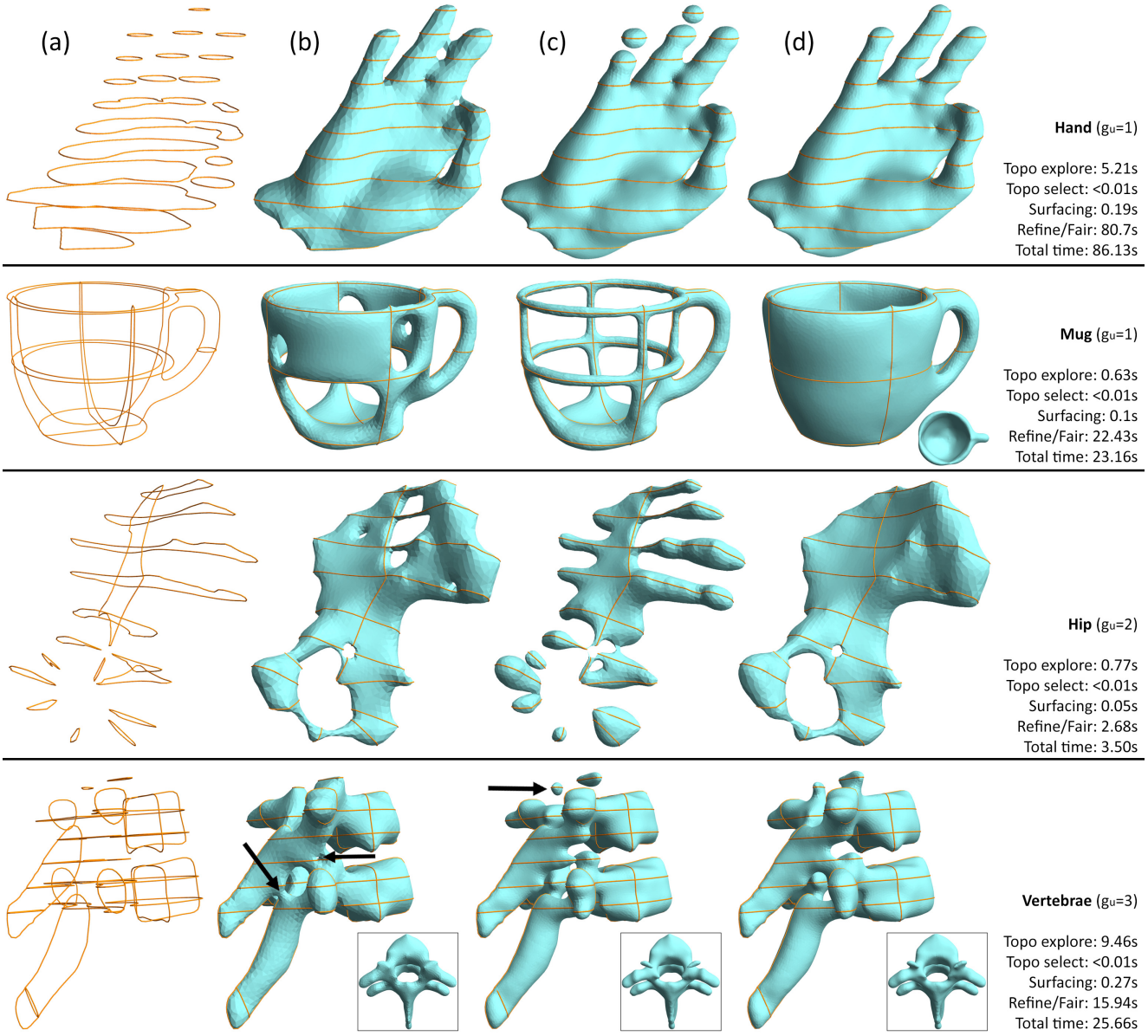

Figure 11: *Reconstructions from input cross-sections (a) using the method of Lu et al. [2008] (b), method of Bermano et al. [2011] (c), and our method with prescribed genuses without additional image data (showing running times) (d).*

then optimally selecting one topology per cell. The algorithm is shown on several examples to be able to capture the correct topology using fewer cross-sections than previous algorithms.

The diversity of per-cell tile topologies and the quality of their scores, as the result of the first stage, directly impact the final result of our algorithm. Our strategy of restricting the tiles to the contours of an indicator function seems to offer a reasonable sampling of the topology space while maintaining computational efficiency. However, our limited sampling may not contain the desired tile topology when the spacing of the cross-sections is too sparse (see Figures 2 and 9). A wider range of tile topologies and better scoring functions may lead to more accurate outputs, but at the possible cost of substantially increased computation time. Alternatively, it would also be interesting to explore a way that allows the user to specify desired tile topology in ambiguous cells.

Finally, we envision that our two-stage paradigm can be extended to offer topology control in other applications involving surface reconstruction, such as from point clouds or scalar functions. The "cells" would be isolated volumes where there is ambiguity in the topology of the reconstructed surface (e.g., near the critical points of the scalar function [Sharf et al. 2007]). By exploring possible topologies within each cell, and scoring them appropriately, our topology selection algorithm can be used to pick the optimal topology per cell that achieves a global genus constraint.

Acknowledgement This work is supported in part by NSF grants (IIS-1302200 and IIS-0846072) and a gift from Adobe.

References

AMINI, O., BOISSONNAT, J., AND MEMARI, P. 2013. Geometric tomography with topological guarantees. *Discrete & Computa-*

tional Geometry 50, 4, 821–856.

ATTENE, M., CAMPEN, M., AND KOBBELT, L. 2013. Polygon mesh repairing: An application perspective. *ACM Comput. Surv. 45*, 2 (Mar.), 15:1–15:33.

BAJAJ, C. L., COYLE, E. J., AND LIN, K.-N. 1996. Arbitrary topology shape reconstruction from planar cross sections. *Graph. Models Image Process. 58*, 6, 524–543.

BAREQUET, G., AND SHARIR, M. 1996. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding 63*, 251–272.

BAREQUET, G., AND VAXMAN, A. 2007. Nonlinear interpolation between slices. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, 97–107.

BAREQUET, G., AND VAXMAN, A. 2009. Reconstruction of multi-label domains from partial planar cross-sections. *Comput. Graph. Forum 28*, 5, 1327–1337.

BAREQUET, G., GOODRICH, M. T., LEVI-STEINER, A., AND STEINER, D. 2004. Straight-skeleton based contour interpolation. *Graph. Models 65*, 323–350.

BAREQUET, G., GOTSMAN, C., AND SIDLESKY, A. 2006. Polygon reconstruction from line cross-sections. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry, CCCG 2006, August 14-16, 2006, Queen's University, Ontario, Canada*.

BAZIN, P., AND PHAM, D. L. 2007. Topology-preserving tissue classification of magnetic resonance brain images. *IEEE Trans. Med. Imaging 26*, 4, 487–496.

BERMANO, A., VAXMAN, A., AND GOTSMAN, C. 2011. Online reconstruction of 3d objects from arbitrary cross-sections. *ACM Trans. Graph. 30*, 5 (Oct.), 113:1–113:11.

BOISSONNAT, J.-D., AND MEMARI, P. 2007. Shape reconstruction from unorganized cross-sections. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, 89–98.

BOISSONNAT, J.-D. 1988. Shape reconstruction from planar cross sections. *Comput. Vision Graph. Image Process. 44*, 1, 1–29.

CHENG, S.-W., AND DEY, T. K. 1999. Improved constructions of delaunay based contour surfaces. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, ACM Press, 322–323.

FUCHS, H., KEDEM, Z. M., AND USELTON, S. P. 1977. Optimal surface reconstruction from planar contours. *Commun. ACM 20*, 10, 693–702.

GRADY, L. 2006. Random walks for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 28*, 11 (Nov.), 1768–1783.

HERMAN, G. T., ZHENG, J., AND BUCHOLTZ, C. A. 1992. Shape-based interpolation. *IEEE Comput. Graph. Appl. 12*, 3, 69–79.

IJIRI, T., YOSHIZAWA, S., YOKOTA, H., AND IGARASHI, T. 2014. Flower Modeling via X-ray Computed Tomography. *ACM Trans. Graph. 33*, 4, to appear. Proc. of SIGGRAPH'14.

JU, T., WARREN, J. D., CARSON, J., EICHELE, G., THALLER, C., CHIU, W., BELLO, M., AND KAKADIARIS, I. A. 2005. Building 3d surface networks from 2d curve networks with application to anatomical modeling. *The Visual Computer 21*, 8-10, 764–773.

KEPPEL, E. 1975. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development 19*, 1, 2–11.

KNUTH, D. E. 2000. Dancing links. *Millenial Perspectives in Computer Science* (Nov), 187–214.

LIEPA, P. 2003. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '03, 200–205.

LIU, L., BAJAJ, C., DEASY, J., LOW, D. A., AND JU, T. 2008. Surface reconstruction from non-parallel curve networks. *Comput. Graph. Forum 27*, 2, 155–163.

MEYERS, D., SKINNER, S., AND SLOAN, K. 1992. Surfaces from contours. *ACM Trans. Graph. 11*, 3, 228–258.

OLIVA, J.-M., PERRIN, M., AND COQUILLART, S. 1996. 3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. *Computer Graphics Forum 15*, 3, 397–408.

SHARF, A., LEWINER, T., SHKLARSKI, G., TOLEDO, S., AND COHEN-OR, D. 2007. Interactive topology-aware surface reconstruction. *ACM Trans. Graph. 26*, 3 (July).

SI, H., 2007. TetGen. a quality tetrahedral mesh generator and three-dimensional delaunay triangulator.

TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 335–342.

VAN KREVELD, M., VAN OOSTRUM, R., BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. 1997. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, 212–220.

WOOD, Z. J., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2004. Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2, 190–208.

XU, G., PAN, Q., AND BAJAJ, C. 2006. Discrete surface modelling using partial differential equations. *Computer Aided Geometric Design 23*, 2, 125–145.

YIN, K., HUANG, H., ZHANG, H., GONG, M., COHEN-OR, D., AND CHEN, B. 2014. Morfit: Interactive surface reconstruction from incomplete point clouds with curve-driven topology and geometry control. *ACM Trans. Graph. 33*, 6 (Nov.), 202:1–202:12.

ZENG, Y., SAMARAS, D., CHEN, W., AND PENG, Q. 2008. Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in N-D images. *Computer Vision and Image Understanding 112*, 1, 81–90.

ZHOU, Q.-Y., JU, T., AND HU, S.-M. 2007. Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics*, to appear.

ZHOU, S., JIANG, C., AND LEFEBVRE, S. 2014. Topology-constrained synthesis of vector patterns. *ACM Transactions on Graphics (Proc. SIGGRAPH Aisa) 33*.