Shape Reconstruction from Contours using Isotopic Deformation

Kikuo Fujimura and Eddy Kuo
Dept. of Computer and Information Science
The Ohio State University
2015 Neil Ave. Columbus Ohio 43210 USA
Phone: 614-292-6730
Fax: 614-292-2911
fujimura@cis.ohio-state.edu ekuo@cis.ohio-state.edu

Abstract

A method for shape reconstruction from contours is presented using isotopic deformation. The reconstructed shape is free of self-intersection and it can incorporate given feature correspondences. The method automatically adds vertices, where necessary, to satisfy these requirements. A new method for handling bifurcation is also proposed, which can handle cases that are problematic for some other algorithms. The method proposed is suitable for terrain modeling, since reconstructed shapes generated by the method do not have overhangs. The running time of the algorithm is dependent on the complexity of the input scene and is shown to be worst-case optimal for the class of task defined. Experimental results are included to illustrate the feasibility of the approach.

Keywords: shape reconstruction, terrain modeling, isosurface, isotopy.

1 Introduction

Shape reconstruction from contours is the problem of constructing a 3D polyhedral shape that has a given pair of contours as its top and bottom faces. Cross sectional data are usually obtained by CT, MRI, or range sensors. Lately, much progress has been made toward rendering such data sets directly using volume visualization techniques. However, polygonal shape reconstruction is also important due to its speed and compatibility with surface-based renderers available today. Shape reconstruction is widely used as a common tool in medical imaging as well as in computerized cartography. In this paper, we present a new method for shape reconstruction which has a number of features that are not fully explored by existing approaches. The method is particularly suitable for terrain modeling applications.

Many methods have been proposed for the shape reconstruction problem in the past 20 years. Most existing approaches use triangular tiling to construct a 3D solid, where each triangular facet is made of two consecutive vertices from one contour and one vertex taken from the other contour. Approaches making use of a triangular tiling may be considered as producing a piecewise linear interpolation between the two contours. Although this model is effective for many reconstruction problems in which consecutive contours are similar in shape, it is subject to some remarks as follows.

- Discontinuity in surface normal. It has been pointed out [32] that sudden changes in surface normal in triangular tiling may at times give rise to a visual artifact in reconstructed shapes. Such a discontinuity is less noticeable for rectangular tiling or free-form surfaces [6]. However, existing algorithms for rectangular tiling usually work only when contours are similar in shape and they are not guaranteed to generate intersection-free solids for an arbitrary pair of contours.

  A related problem is that the surface reconstructed is somewhat sensitive to a specific triangular tiling chosen as illustrated in Fig. 1. Usually, these problems are circumvented by introducing additional vertices to the contours. In this example, adding one vertex in the front edge of the lower contour can solve both problems.

- Self-intersecting shapes. In most applications, it is desirable that the reconstructed shape is simple, i.e., free of self-intersection. However, some existing algorithms do not guarantee this property. As pointed out in [1], it is possible that the output of the toroidal graph method [13] can have a self-intersection for a pair of dissimilar contours, even when it is possible to construct an intersection-free shape by using the same pair of contours by a different approach. For a certain pair of contours, it is not possible to construct a simple polyhedron at all only by using vertices contained in the contours [16]. Usually, this problem is avoided by adding extra vertices in the given contours in a preprocessing stage [1, 3].

- Overhangs. For a certain application such as terrain modeling, the reconstructed shape does not appear natural if it contains overhangs. However, for triangular tilings, such a situation is unavoidable for

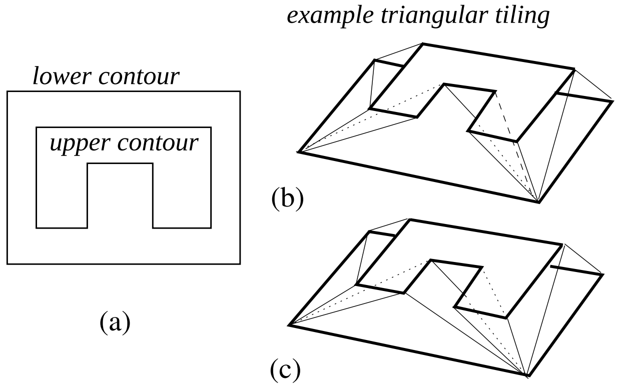


Figure 1: Triangular tiling. (a) Lower and upper contours. (b) A reconstruction example where the concave part is spread out toward the lower contour. (c) A slight shift in the location of the upper contour gives rise to a different triangular tiling. Both shapes have overhangs. (d) Reconstructed shapes such as (b) and (c) are somewhat unnatural. Adding one vertex in the lower contour resolves these problems in this case.

certain inputs as in Fig. 2. This problem cannot be solved by introducing additional vertices in the contours. In such a case, additional steps must be taken such as sampling more between contours or introducing a ridge-like structure between two layers so as to avoid an overhang.

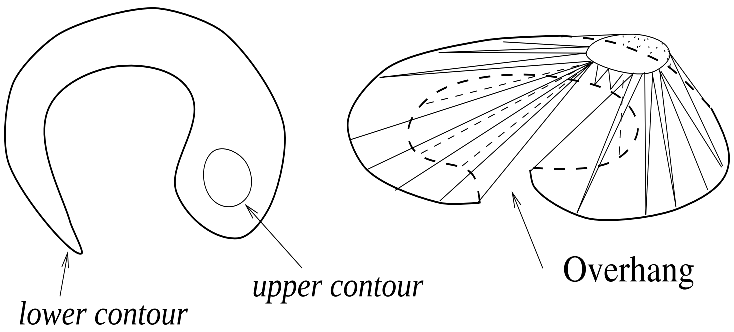


Figure 2: Contours that give rise to an overhang. (a) Lower and upper contours. (b) A reconstruction example. For a triangular tiling, it is not possible to construct a shape without an overhang for the given pair of contours.

- No feature incorporated. For piecewise linear (triangular) tilings, curves or polylines are not to be used to connect vertices of the given pair of contours. This makes it difficult to incorporate feature correspondence. For example, Fig. 3 contains four contours which appear to form a valley. For the valley to be formed, correspondence is to be made between thick portions in contours b and c by connecting A to D, B to E, and C to F. A method based on piecewise linear interpolation cannot achieve this without introducing an overhang in the terrain reconstructed.

In this paper, we propose a method that overcomes these difficulties. Our method is based on rectangular rather than triangular tiling. While triangular tilings are regarded as piecewise linear interpolation, our approach may be phrased as an isotopy-based method. Roughly speaking, isotopic deformation transforms

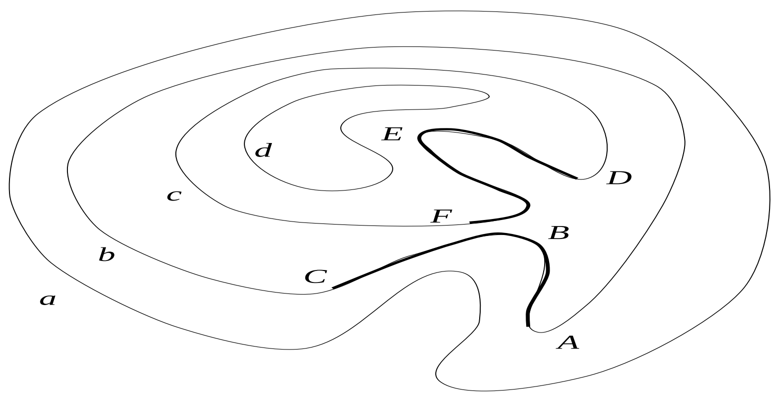


Figure 3: Thick portions in contours b and c can form a valley.

a space (in our case a loop) continuously to another space (another loop), so that no two points will merge into a point at any time during the deformation. The main features of our method are to satisfy the following requirements.

Requirement 1. Reconstructed shapes are guaranteed to be free of self-intersection for an arbitrary pair of closed simple contours.

Requirement 2. The reconstructed shape does not contain overhangs, when contours are properly nested in projection.

Requirement 3. Vertex correspondences can be incorporated in shape reconstruction. If vertex correspondence is not specified, then correspondence between two contours is automatically determined by the algorithm. The method takes any number of vertex correspondences and generates a solid object preserving the correspondences provided, as long as vertex correspondences are consistent. This feature can be used to highlight geometric landmarks such as valleys and ridges.

In Requirement 3, vertex correspondences are said to be consistent if pairs of vertices can be indexed serially (clockwise or counterclockwise) on both contours. These requirements described above are satisfied at the cost of using additional vertices in shape reconstruction. That is, the reconstructed shape has vertices that are not originally included in the given contours. They are used to provide extra degrees of freedom to control shape reconstruction. We show that our algorithm is asymptotically worst-case optimal for the class of algorithms that satisfies all the above requirements. Also, our algorithm runs in an output-sensitive manner, in that for a pair of similar contours, it requires substantially less amount of additional vertices to be used in reconstruction.

Requirement 1, which has been addressed in some past work (e.g., [1]), is critical for most applications, while Requirement 2 is relevant only when contours are nested as in Fig. 3 and is important in particular for terrain modeling. In general, Requirement 3 cannot be fulfilled only by using piecewise linear interpolation.

In this paper, we do not discuss the issue of how to find corresponding features in the given pair of contours. we assume that such information is provided as a part of input. In the literature, the shape reconstruction problem is usually decomposed into three subproblems, namely, the correspondence problem, the branching problem, and the tiling problem. In this article, we concentrate on the tiling problem and the branching problem. We do not discuss the corresponding problem; it is an important consideration for general shape reconstruction problems, while it is of lesser importance for terrain modeling applications.

The rest of the paper is organized as follows. In section 2, we survey related work in shape reconstruction from contours. Section 3 contains our main ideas for the proposed approach and extensions are presented in Section 4. Experimental results and an analysis of the algorithm are shown in Sections 5 and 6, respectively. Section 7 contains some remarks on the method and Section 8 concludes the paper.

2 Background

2.1 Related Work

Reconstructing a 3D solid from a series of contour data was first formulated by using the toroidal graph [13, 21]. In this approach, a triangular tiling is represented as a path in the graph. Then, the shortest path is sought in the graph to generate a tiling such that some global property is minimized such as the total surface area and the total volume enclosed by the solid. Some variations have also been reported based on the toroidal graph formulation. For example, Wang and Aggarwal [35] use A* search to reconstruct surface that minimizes the total edge length. Other heuristics and applications are found in [10, 31, 33].

Christiansen and Sederberg [8] propose a greedy search approach to avoid expensive optimal graph search. After normalizing the two contours, vertices on the two contours are processed sequentially starting from some pair of vertices. The next vertex (either from the top or the bottom contour) with a shorter diagonal is chosen and added to the tiling being constructed. The method runs fast and works well for a pair of similar contours, but may fail to produce a reasonable shape for dissimilar contours as pointed out in [12]. O'Rourke [28] points out that for such normalization, it is best to use a uniform scaling to avoid anomaly.

Some approaches emphasize matching geometrically similar portions of contours more than minimization of a global quantity. Kehtarnavaz, Simar, and de Figueiredo [20] have encoded given contours into the Levenshtein graph, in which higher-level geometric features such as arch-like portions in the contour and protrusions can be captured for shape reconstruction. A graph search is performed to find an optimal solution in the Levenshtein graph and triangular tiling is constructed between matched contour parts. Ekoule, Peyrin and Odet [12] also analyze given contours to find similarity between the contours. They decompose the contour in a hierarchical manner based on convex and concave parts along the contours. Triangulation is then performed by using the information obtained such that similar parts are linked together.

Welzl and Wolfers [36] have posed two criteria for surface reconstruction different from those considered above. The first condition requires that if two contours are convex, then the surface is the convex hull of the contours. The second is that if one contour is obtained by projecting the other from some projection center, then the surface reconstructed is the part of the cone whose apex is the projection center and that lies between the planes containing the contours. They also introduce the angle-consistency condition which generally prevents self-intersection. From a theoretical viewpoint, Gitlin et al [16] have investigated types of input contours for which self-intersection do not occur. They show that for an arbitrary pair of contours it is not always possible to construct a triangular tiling without self-intersection. To avoid such a problem, extra vertices are usually added to the contours.

Some authors address the correspondence problem and branching problem [26, 12] in addition to the tiling problem. Christiansen and Sederberg [8] have proposed to introduce a mid-point between two contours on the same level. The mid-point is used to unite the two contours into one so that existing tiling methods may be applied. This approach faces some difficulty when the projection of the midpoint does not fall inside of the other contour. Meyers et al [26] consider a strategy similar to the approach by Christiansen and Sederberg for the branching problem, while for the correspondence problem they use the minimum spanning tree approach after approximating the contours by elliptical bounding shapes. Zyda et al [37] and Ekoule et al [12] use a bounding-box overlap analysis to determine correspondence.

Some methods handle the three problems at once. Boissonnat [5] has used tetrahedralization of spaces between two contours by using the Delaunay triangulation of pair of successive contours. He then projects each triangulation against the other to determine a tetrahedralization between consecutive slices. After removing internal edges of tetrahedralization, a solid interpolating the two slices is obtained. Lately, Barequet and Sharir [3] have proposed another tiling method that handles the three problems simultaneously. It consists of two major steps. In the first step, geometrically similar parts of the contours are matched by a voting algorithm and then stitched (i.e., triangulated). In the second step, the remaining parts (called clefts) are processed to complete the reconstruction. These methods [5, 3] may generate flat triangles in the same level as the original contours. In such a case, the branch (bifurcation) becomes apparent only after the third level (in addition to the first and the second level contours) is processed.

Bajaj et al [1] also address three problems at once. They introduce three criteria that are desirable for reconstruction from slices. The first one requires that the shape reconstructed contain no self-intersection. Secondly, any line perpendicular to the slice intersects the reconstructed surface at zero point, one point, or along one line-segment. Thirdly, resampling of the reconstructed surface on the slice must reproduce the original contours. They present a multipass method that satisfies all of them. First, they use geometric similarity between successive contours to generate a partial tiling to satisfy the three conditions. Additional passes are taken to handle remaining portions of the contours by introducing medial axis, where applicable. Their approach also handle branching problems and correspondence problems simultaneously. Unlike [3, 5], their method does not use any triangles that are parallel to the contours, which leads to a more natural-looking shape.

Other approaches for shape reconstruction from contours include [15, 23, 19, 9]. Schumaker [30] surveys various methods including nonlinear interpolation methods. See the next section for methods using non triangular tiling.

2.2 Homotopy and isotopy

The concepts of homotopy and isotopy are often useful for describing deformation. Two mappings $f : X \to Y$ and $g : X \to Y$, are called *homotopic* if there is a continuous map

$$H : [0,1] \times X \to Y$$

such that $H(0,x) = f(x)$ and $H(1,x) = g(x)$. The function $H(t,x)$ is called a homotopy. The map $H(t,x)$ may be viewed as a family of functions $h_t(x)$ $(0 \leq t \leq 1)$. Then, $h_t(x)$ can be considered as a continuous deformation from $f(x)$ to $g(x)$ as we vary t from 0 to 1. As a simple example, if Y is a convex subset of a Euclidean space, then we can define $h_t(x) = (1-t)f(x) + tg(x)$ to deform $f(x)$ into $g(x)$. Such a homotopy is called a straight-line homotopy.

An *isotopy* is a homotopy h_t for which every $h_t(x)$ is a homeomorphism. In particular, during an isotopy of a simple curve the image remains simple at every stage [34]. Homotopy is an essential tool for classifying manifolds of low dimensions, while isotopy is instrumental in knot theory [25]. Fig. 4 illustrates the difference between homotopy and isotopy. Here, X is interval [0, 1] on the real line and Y is $\mathbb{R}^2$. The figure shows a curve deforms into another curve (or a point). For homotopy, it is not necessary that the images of $h_t(x)$ remain homeomorphic to each other. The image of $h_t(x)$ (curve in this example) may fail to be one-to-one (Fig. 4(a)) or can cross itself (Fig. 4(b)), while the images under an isotopy are homeomorphic to each other (Fig. 4(c)). For shape reconstruction, triangular tiling is considered as piecewise-linear homotopy, while our method is an isotopic deformation of a loop (Fig. 5).

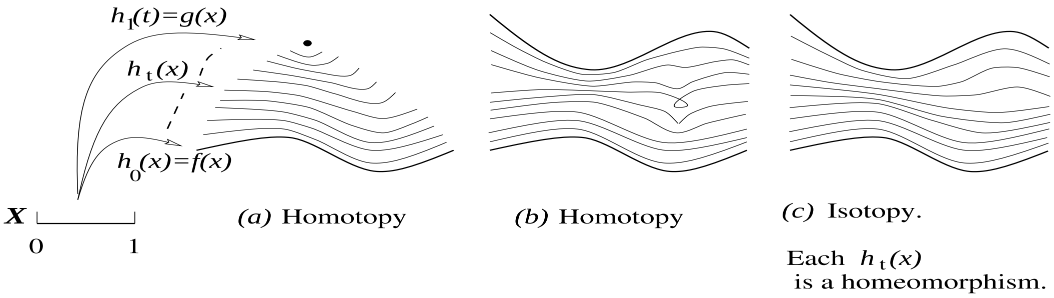
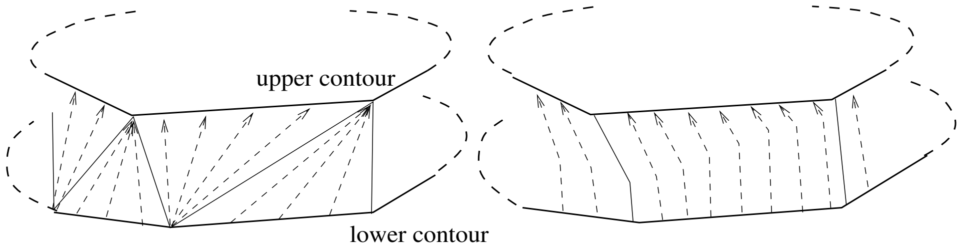


Figure 4: Homotopy and isotopy. A curve may shrink to a point (a) or cross itself (b) under homotopy. Curves remain homeomorphic to each other under isotopy (c).

There are some approaches which consider nontriangular tilings such as ones using free-form surfaces. A bivariate free-form surface $P(u,v)$ may be considered as a homotopy, where a curve parametrized in u deforms along the other dimension v to form a patch. Chang et al [6] consider using Hermite interpolation for surface description and Kehtrarnavaz et al [19] also use parametric surfaces. Shinagawa and Kunii [32]



(a) Piecewise linear homotopy model

(b) Isotopy model

Figure 5: Piecewise linear mapping and isotopy

use a piecewise linear homotopy model based on the toroidal graph for surface reconstruction between two contours by using a continuous version of the toroidal graph. In general, these approaches work well for a pair of contours that are similar in shape. However, their performance for interpolation between dissimilar shapes is not well-understood. Fujimura and Makarov [14] propose a homotopic image morphing method that preserves a topological invariant. Rosenfeld and Nakamura [29] investigate homotopic deformation in digital geometry.

3 Reconstruction Method

To present our basic idea, we assume that we are given a pair of closed simple contours that lie in parallel slices and that one contour is strictly contained in the other when they are projected in a common plane parallel to the slices (as in a topographic map). Later, we discuss heuristic methods for lifting this restriction.

The basic idea is to shrink the outer polygon gradually and seamlessly until it fits the inner polygon. The shrinking process gives a blending between the two polygons. To implement the idea, we use triangulation as our basic structure. We first describe the basic strategy and then a few modifications are introduced to elaborate the approach. Let us use $P = \{P_1, \cdots, P_n\}$ and $Q = \{Q_1, \cdots, Q_m\}$ to denote the two polygons projected in a common plane such that P encloses Q. The following is the outline of our method, which we call as the *snap algorithm*. Let Z be the zone between P and Q.

Reconstruction Algorithm

1. (Triangulation) Triangulate the area Z using vertices of P and Q. It is always possible to triangulate planar polygons by using only vertices of P and Q (i.e., without using any additional vertices) [27].

2. (Snapping) Eliminate triangles of zone Z one by one in an outer-to-inner fashion. Each time a triangle is eliminated, the outer layer shrinks by using one of the snapping operations described below. Repeat this step until the outer loop eventually coincides with the inner loop.

3. (Speed adjustment) The process of eliminating the triangles in zone Z gives us a blending from P to Q.

Adjust shrinking speeds of all points on P so that they start shrinking and terminate at corresponding points on Q simultaneously.

4. (Postprocessing) Process polygon deformation paths to remove sharp turns, where applicable.

We now describe each step in detail.

3.1 Triangulation

There are a number of methods for polygon triangulation. In our case, constrained triangulation is necessary, since inner polygon edges are to be used as triangulation edges. The method by Chew [7] constructs constrained Delaunay triangulation in $O(n \log n)$ time, where n is the total number of vertices. A survey by Bern and Eppstein [4] describes various other methods for triangulation.

3.2 Snapping

The heart of the algorithm is elimination of the triangles. There are two cases depending on whether the triangle to be eliminated has either one or two edges on the outer loop. We describe how to eliminate the triangle for each case.

Snapping Rules

- (Rule 1. One edge, say P_iP_{i+1}, is on the outer loop): Let Q_j be the third vertex in the triangle (Fig. 6(b)). Snap a point on P_iP_{i+1} (say K) to Q_j. Points on P_iK and KP_{i+1} are mapped to P_iQ_j and Q_jP_{i+1}, respectively.

- (Rule 2. Two edges, say, $P_{i+1}P_{i+2}$ and $P_{i+2}P_{i+3}$, are on the outer loop): Snap P_{i+2} to a point (say L) on $P_{i+1}P_{i+3}$ (Fig. 6(b)). Points on $P_{i+1}P_{i+2}$ and $P_{i+2}P_{i+3}$ are mapped to $P_{i+1}L$ and LP_{i+3}, respectively.

- (Rule 3. Forbidden snap): Snap operations are to be done such that the resulting outer loop is never intersecting with itself (e.g., no two vertices coincide, no vertex intersects with an edge other than its endpoint). For example, an edge-snap as illustrated in Fig. 7 is not permitted, since it would generate a triangle ($P_iP_{i+1}P_{i+2}$) on the outer loop. (One could snap P_{i+1} first onto P_iP_{i+2}, then snap P_{i+2}.)

Fig. 6 contains five consecutive legal snaps on a part of a polygon. Whether or not a particular snap is legal can easily be checked by local geometry around the triangle to be eliminated. Let us call a triangle to be *deletable*, if it can be eliminated by using either Rule 1 or Rule 2. For a triangle, if two edges are on the outer loop, then it is always deletable. If only one edge of $\triangle ABC$, say AB, is on the outer loop, then $\triangle ABC$ is deletable only when C is not on the outer loop. Now we have the following claim to ensure that the procedure is sound.

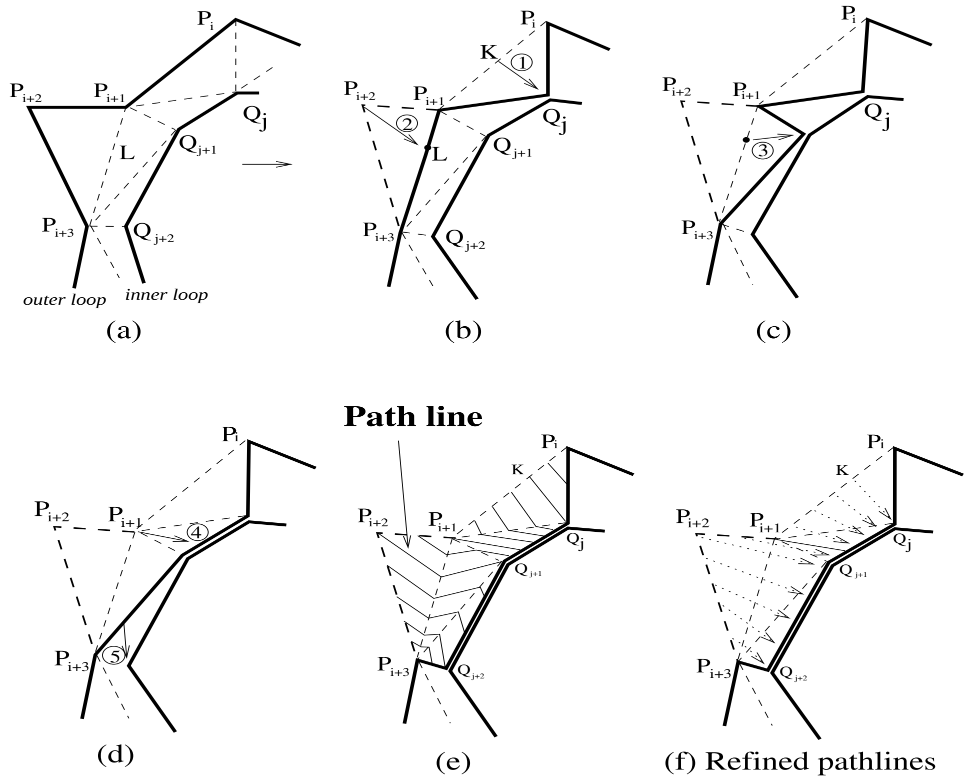


Figure 6: Pathlines. (a) Outer and inner loops with triangulation. (b-e) A sequence of snap operations induces a set of pathlines from the outer polygon boundary to the inner polygon boundary. (f) Stretch pathlines where applicable.

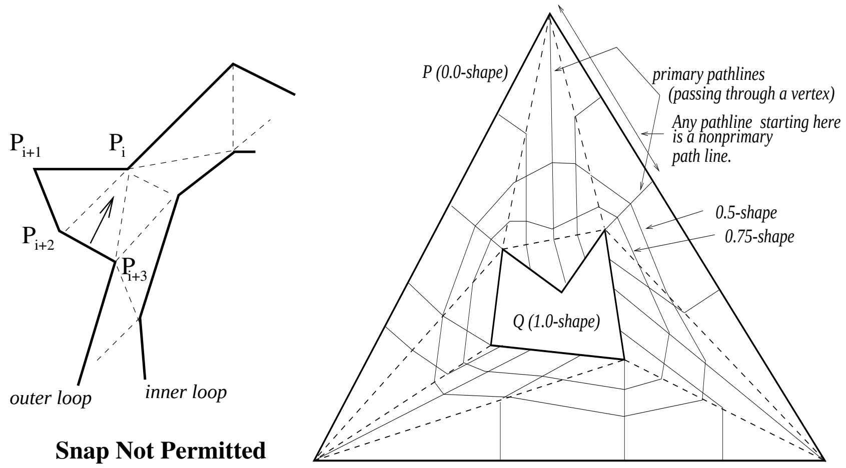


Figure 7: Forbidden snap (left). Examples of t-shapes (right)

Claim 1 *At any stage of the algorithm, there exists at least one deletable triangle.*

Proof: Suppose no triangle is deletable. Then, from the above observation, each triangle has at most one edge on the outer loop as in $\triangle ABC$ in Fig. 8. Such a triangle divides the outer loop into two parts, Part 1 and Part 2 as in Fig. 8. If we repeatedly apply the same argument for Part 1, we will be left with at least one triangle that has two edges on the outerloop, which is a contradiction. Thus, there exists at least one deletable triangle. $\square$

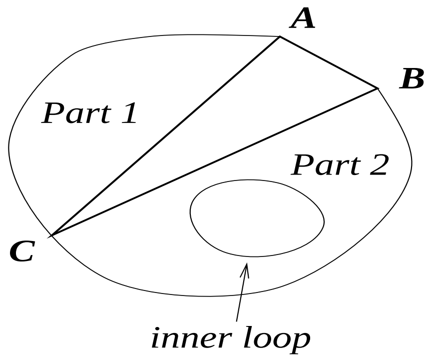


Figure 8: Illustration for Claim 1. $\triangle ABC$ is not deletable.

We now define pathlines. A *pathline* is simply the trajectory of a point on polygon P to its corresponding position on polygon Q. The simplest way to define a pathline within a triangle is to make them all parallel to the trajectory of the point being snapped. For example, in Fig. 6(b), all points on P_iP_{i+1} have pathlines parallel to that of K. A pathline that begins from or terminates at a vertex of either P or Q is called a *primary pathline*. Nonprimary pathlines found between two consecutive primary pathlines may be considered as belonging to an equivalence class in that they all bend on the same triangulation edges. Nonprimary pathlines are derived by linearly interpolating primary pathlines. Pathlines do not intersect with each other as we show below.

Now, let us index a point on polygon P by $\mathcal{P}_s$ ($0 \le s \le 1$). Naturally, $\mathcal{P}_0 = \mathcal{P}_1$. Let us use π_s to refer to the pathline starting from $\mathcal{P}_s$. Also, let $|\pi_s|$ be the Euclidean length of pathline π_s. If we parametrize a pathline by using t ($0 \le t \le 1$), then all points in zone Z may be expressed by (s, t). So, we can write $\pi_s(t)$ to refer to a point in Z. The set $\{\pi_s(t), 0 \le s \le 1\}$ for a fixed t is a polygon and is called the t-shape. Apparently, the 0-shape is P and the 1-shape is Q. Pathlines have a few properties characterized as below.

Claim 2

1. *For s and s' ($s \neq s'$), π_s and $\pi_{s'}$ are disjoint. (That is, two pathlines are disjoint.)*
2. *Function (in s) $|\pi_s|$ is continuous. (Pathline lengths vary continuously along the perimeter.)*
3. *For t and t' ($t \neq t'$), t-shape and t'-shape are disjoint.*

4. *A family of t-shapes ($0 \leq t \leq 1$) defines a seamless deformation from P to Q.*

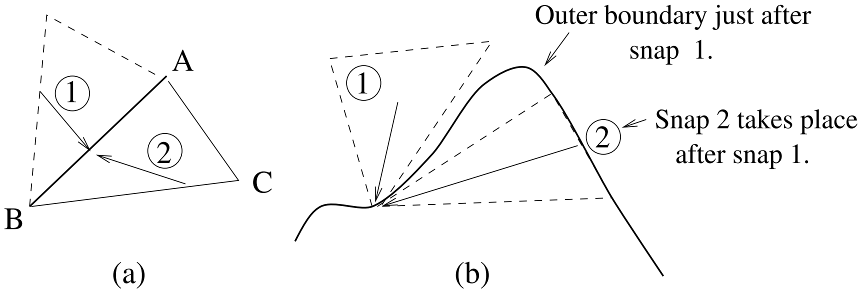


Figure 9: Illustration for Claim 2.

Proof:

1. Clearly, no two pathlines can intersect within a triangle. Thus, if two pathlines intersect at all, then the point of intersection must be either on a triangle edge or at a triangle vertex. Suppose the point of intersection is at a point in edge AB and that snap 1 takes place earlier than snap 2. After snap 1, edge AB must be a part of the (shrinking) outer loop at that stage of the algorithm. Thus, snap 2 as in Fig. 9(a) cannot take place either by using snap rule 1 or 2. Next, suppose that the point of intersection is at a vertex as in Fig. 9(b) and that snap 1 takes place earlier than snap 2. Snap 2 violates snap rule 3. Thus, the point of intersection cannot occur at a vertex either.

2. This claim can be proved by induction on n, the number of triangles in zone Z. When $n = 0$, the outer loop coincides with the inner loop and the claim is true.

   Suppose X is an interior point on a triangle edge e. Either snap rule 1 or 2, when applied to the triangle with e, preserves the continuity of $|\pi(s)|$ at X. Suppose now that X is a vertex of the outer loop. Again, it is easy to see that either snap rule 1 or 2 involving vertex X does not make $|\pi(s)|$ discontinuous at X.

3. Suppose t-shape and t'-shape intersect. At the point of intersection, two pathlines must be intersecting, which is impossible from Claim 2.1.

4. This claim follows immediately from Claim 2.3.

   □

From the fourth property, we have that the family of t-shapes ($0 \leq t \leq 1$) defines an isotopic deformation from polygon P to polygon Q. Finally, note that pathlines are dependent on the order of triangle elimination.

3.3 Speed adjustment

When snap operations are carried out from the outer to inner loops, we need to keep track of the history of primary pathlines so that necessary information can be retrieved for constructing a 3D shape. Each time a vertex V of the inner loop is reached in the snap algorithm, the point V' corresponding to V on the outer loop is to be identified by tracing back the pathline to V. It takes $O(k)$ time to do this trace, where k is the number of triangles on the pathline from V' to V. Usually, k is a small number, although it can be $O(n)$ when the polygons are convoluted. We only need to retain primary pathlines, since other pathlines can be easily obtained by interpolating between primary pathlines. Once the trace is completed for all pathlines, they can be parametrized in $t \in$ [0,1], which makes it possible to generate t-shapes.

3.4 Postprocessing

At times, an intermediate t-shape may have more vertices than it seems necessary. Pathlines can be straightened where possible as a postprocessing operation as in Fig. 6(f), that is, two consecutive segments of a primary pathline are replaced by one segment, if it does not intersect with any other primary pathline nor the contours themselves. This operation removes some unnatural sharp turns in pathlines and serves to lower the complexity of shape representation. Such a refinement is usually possible for shapes resulting from the snap algorithm.

4 Extensions

We describe two important extensions to the basic framework, namely, vertex correspondence and branching.

4.1 Vertex correspondence

So far we have assumed no vertex correspondence is given. In this mode, the terminal position on the inner contour of each pathline is dependent on the snap policy used. Usually, a pathline terminates at some point on Q near its start point on P. However, at times, a pathline can be unexpectedly long and winding, depending on the snap sequence taken. (See Fig. 10 for such an occurrence.) It is possible to enforce vertex correspondence in our algorithm. Such a consideration helps to preserve some geometric landmarks such as ridges in reconstruction. It can also be used to prevent a pathline to become long and winding. Finding matching features may be done by various methods (e.g., [12, 17, 18]) and it is not discussed in this paper. We take the following steps to incorporate vertex correspondences. The general idea is illustrated in Fig. 11.

1. Generate a polyline (called a *vertex path*) connecting corresponding vertices.

2. Retriangulate with added polylines as constraints.

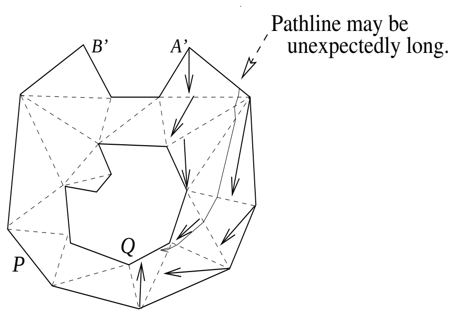
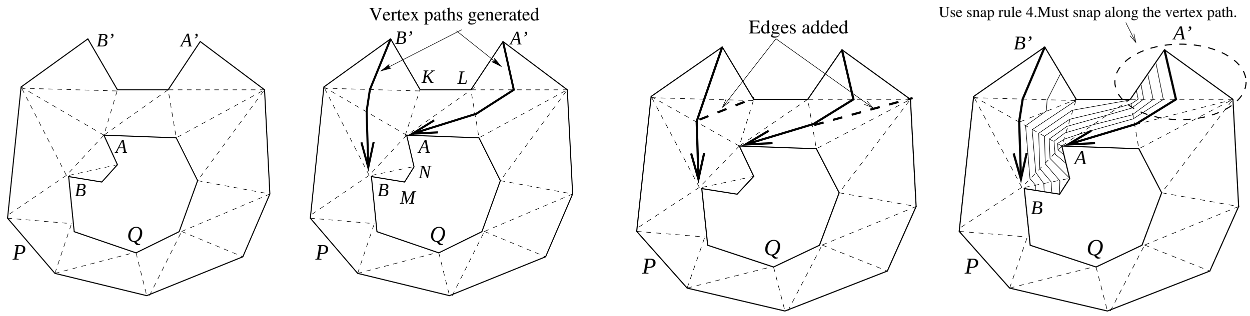


Figure 10: Long pathline. If an arbitrary snap sequence is permitted, then a pathline can travel a distance longer than one would expect.

3. Apply the (modified) snap algorithm.



(a) Contours with triangulation. Vertex correspondence (A-A') and (B-B') are specified.

(b) Vertex paths are generated by using a path planning method.

(c) Retriangulate the area between the two contours.

(d) Generate pathlines by using the (modified) snap rule.

Figure 11: Vertex correspondence. (a) Correspondences are specified for the given contours. (b) Vertex paths are generated. (c) Retriangulate the area along the vertex paths. (d) Use the snap algorithm to generate pathlines.

[**Step 1.** Generating pathlines]: Suppose vertex A on the inner contour and vertex A' on the outer contour are to be connected. If segment AA' is an existing edge of the triangulation, then we can simply use the edge to establish the correspondence. Otherwise, a polyline can be generated by a path planning technique [22] within the zone Z regarding polygons P and Q as the obstacles. Since zone Z has already been triangulated, the induced connectivity in triangulation can be used for path planning. For example, a path can be defined by connecting midpoints of triangular edges except at the start and the end points. (See the polyline from A' to A in Fig. 11(b).) To generate a second pathline, path planning is used again while regarding the first path (in this case the one connecting A to A') plus P and Q as the obstacles, and so on.

[**Step 2.** Retriangulation]: As a result of introducing vertex paths, some triangles are split into two parts. We retriangulate these parts. If a vertex path segment MN splits $\triangle ABC$ into two parts, then we add a new triangulation edge BN as in Fig. 12.

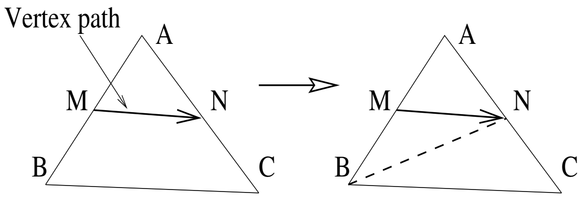


Figure 12: Retriangulation.

[**Step 3.** Apply the Snap Rules]: For cases that vertex paths are introduced, we need to modify the snap rules so as to handle vertex paths.

Snap Rule 4 (edge constraint):

1. Suppose triangulation edge AB and AC are on the outer loop and a vertex path incident on vertex A (as in Fig. 13(a)). Then, A is to be snapped to M.

2. Suppose triangulation edge BC is on the outer loop and a vertex path intersects edge BC transversally (as in Fig. 13(b)) at M. Then, M is to be snapped to A.

3. Suppose vertex M is on a vertex path and $\triangle PQM$ does not intersect the vertex path other than at M (Fig. 13(c)). Then, edge PQ cannot snap to M. This is to maintain the disjoint nature of pathlines.

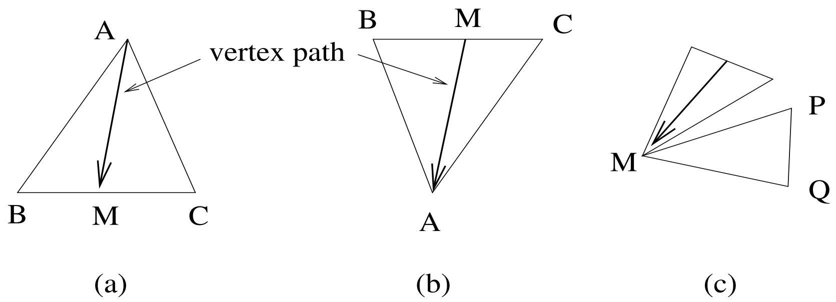


Figure 13: Snap rule 4

As in Fig. 11, when two consecutive vertices A and B correspond to A' and B' on the other contour, the portion between two vertices also corresponds to each other. In this example, polyline $ANMB$ on polygon Q is mapped to segment $A'LKB'$ on polygon P. However, segments LK and MN do not necessarily correspond to each other, unless further correspondences are explicitly specified.

In order for Rule 4.1 to be applied, vertex A cannot have more than three edges incident on it. One consequence of this new restriction is that there are cases that no initial snaps are possible due to the constraints imposed. See Fig. 14 (left). Here, the paths are defined by using existing segments of the triangulation (unlike Fig. 11). Thus, vertex A has four edges coincident to it (unlike three in Fig. 11), which makes it impossible to snap A (same applies to vertices B, C, and D). In such a case, a possible solution is as follows. For a vertex, say B, with more than three edges to it, we add a new vertex P near B and retriangulate the scene such that B has only three edges incident on it as in Fig. 14 (right).

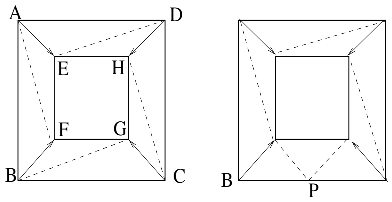


Figure 14: Adding a point to the contour

4.2 Branching

We now discuss how to handle branches. For slices that contain a branching between them, it is not possible to construct an isotopy between them, since two slices contain topologically unequivalent shapes. However, deformation from one slice to the other may be divided into steps of deformation, each of which can be modeled as an isotopy.

In case of branching, we treat each contour separately and then merge the partial results to form the final shape. Let z_1 and z_2 be the heights of bottom and top cross sections, respectively. Let C be the contour on $z = z_1$ plane and C_1 and C_2 be contours on $z = z_2$ plane (Fig. 15(a)).

1. Use the snap algorithm independently for C and C_i to form reconstructed shape S_i ($i = 1, 2$). (Fig. 15(b, c).)

2. Find the heighest point z' at which S_1 and S_2 intersect (Fig. 15(d)).

3. In the $z = z'$ plane, the cross section consists of two contours that are pinched at a single point. We consider the shape as a single simple closed contour and apply the snap algorithm from z_1 to z'.

4. Now we consider the contour on $z = z'$ plane as two separate contours and apply the snap algorithm from z' to z_2 for the two parts separately.

In step 3, two contours may be connected at two points rather than a single point (as in Fig. 16). In this case, we introduce an edge PQ connecting the two contact points. Then, the reconstruction process is divided into two steps. First, the bottom shape is constructed by using contour C and the united contour C_5. Then, the top part is constructed by using two pairs, $\{C_3, C_1\}$ and $\{C_4, C_2\}$.

5 Experimental Results

Based on the idea described above, shape reconstruction results are now presented.

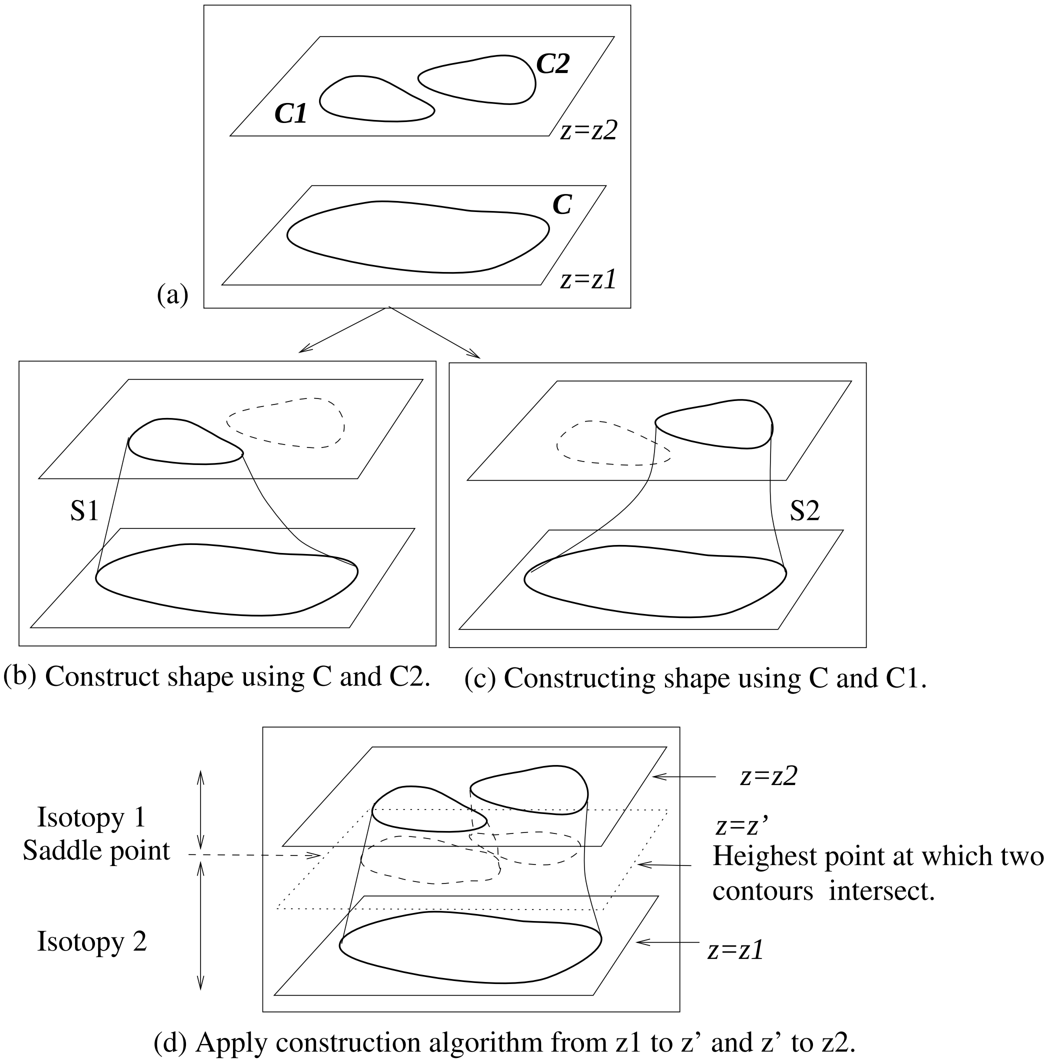


Figure 15: Branching strategy

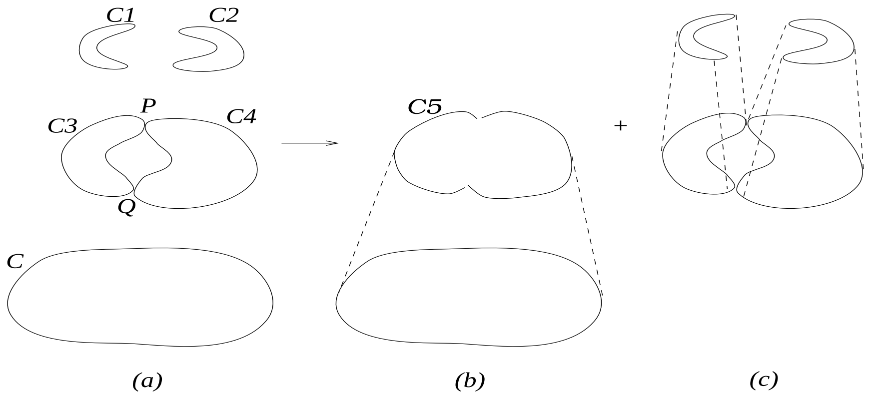


Figure 16: Illustration for branching.

Example 1. Given a pair of contours as in Fig. 17, a method using triangular tiling would connect most of the vertices of the convoluted part of the lower contour to a single vertex of the top contour. As a result, it would generate an 'overhang' in the reconstructed shape. Our algorithm does not generate such an overhang which creates a more natural appearance for terrain modeling.

Figure 17: Example 1. Convoluted shape. Triangulation (left), reconstructed shape by our algorithm (middle), and reconstructed shape by triangular tiling (right)

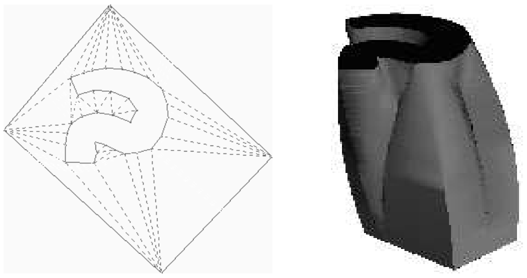


Figure 18: Example 2. Triangulation (left) and reconstructed shape (right)

Example 2. The main algorithm assumes that for a given pair of contours P and Q, contour Q is properly contained inside of polygon P. If this is not the case, a simple idea is to scale Q by a certain factor so that Q can be located inside of P. After the snap algorithm is run, scale all intermediate polygons back by appropriate factors so that the blend sequence terminates with Q at its original scale. Likewise, we can choose to scale P so that Q contains P. When P and Q are dissimilar in shape, we get fairly different results depending on which of P or Q is selected to be scaled. Fig. 18 shows a reconstructed shape using the same pair of contours as in Example 1, but the inner and outer contours here are switched their roles.

Example 3. Fig. 19 contains an example where changes in location of the inner contour give rise to difference triangulation, leading to difference appearances in shape reconstruction.

Example 4 (vertex correspondence). Fig. 20 illustrates the effect of vertex correspondence. The leftmost figure shows the triangulation together with the paths connecting vertices. The middle and the right figures are reconstructed shapes without and with vertex correspondence, respectively. The reconstructed shape with vertex correspondence can represent the valley structure connecting two concave parts of the contours.

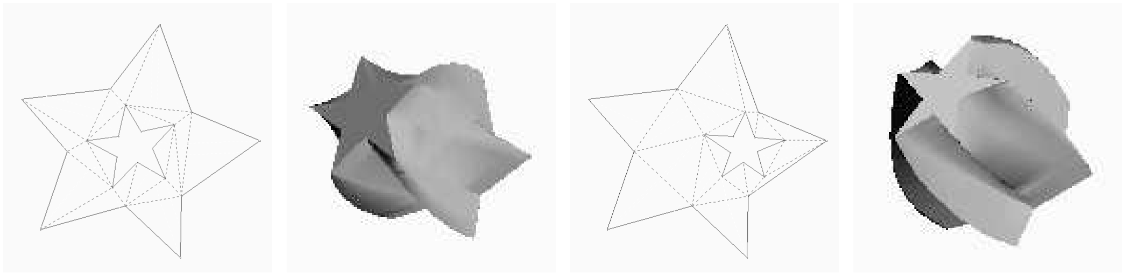


Figure 19: Locations of the inner contour. Triangulation is dependent on the location of inner contour. Thus, the appearance of the reconstructed shape is also affected.

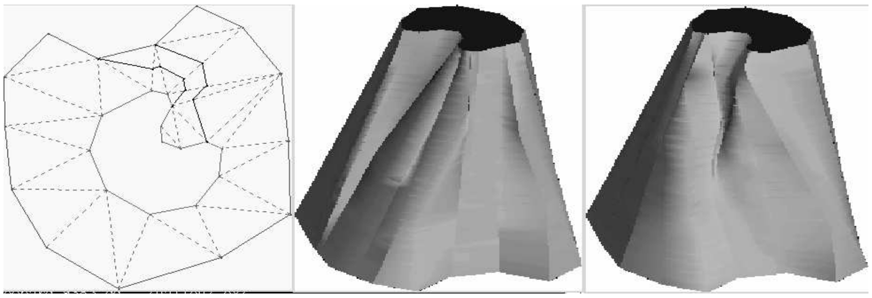


Figure 20: Vertex correspondence. Vertex paths (left), without correspondence (middle), and with correspondence (right).

Example 5. Fig. 21 contains a reconstruction example with more complicated contours.

Figure 21: Reconstruction example from complex contours

Examples 6, 7 (branching).

Fig. 22 illustrates shape reconstruction with branching. The saddle point is automatically determined by the algorithm as described above. Some algorithm would have difficulty dealing with a U-shaped contour as in this figure. Fig. 23 is another example for branching. Here, a united contour (as in Fig. 16) is automatically introduced to generate the branching structure.

Example 8 (Mountain landscape). The attached figure contains a part of an elevation map. There are a

Figure 22: Branching example. Contours for branching (left); top and side views of the reconstructed shape (middle and right).

Figure 23: Contours for branching (left); top and side views of the reconstructed shape (middle and right).

number of locations which can cause an overhang for existing approaches using triangular tiling for shape reconstruction. The attached figure is a reconstructed landscape for the area and contains no overhangs.

6 Analysis of the Algorithm

Let us analyze the running time of our algorithm. We first analyze the case without vertex correspondences. Let n_1 and n_2 be the numbers of vertices in two input polygons, respectively. Triangulation can be done in a number of different manners. For example, generating Delaunay triangulation takes $O(n \log n)$ time, where $n = n_1 + n_2$ and gives rise to $O(n)$ triangles in zone Z. The snapping process takes $O(n)$ steps, since there are $O(n)$ triangles to eliminate. For each triangle snap, we do the following.

1. Determine which triangle to snap. Before the algorithm is run, we enumerate all triangles in zone Z that are deletable and store them in a list. In case of using some heuristic (such as deleting the largest triangle), we sort them and organize them in a priority queue. The initial cost is $O(n \log n)$ time. On each snap, we modify the list by checking neighboring triangles. If one of the neighboring triangles becomes deletable because of the current snap, we add the triangle to the list. Note that there are at most two triangles to check on each snap. The triangle snapped is removed from the list. This local check can be done in constant time and it takes $O(\log n)$ time to enter them in a priority queue.

2. Record which vertex (or edge) is snapped to what location. This step takes constant time, since it can be done within a triangle.

3. Update primary pathlines incident on that triangle. This step can take $O(k)$ time, where k is the number of triangles intersected by the primary pathline. In worst case, $k = O(n)$ when the shapes are fairly dissimilar or convoluted. Otherwise, pathline maintenance takes less than $O(n)$ on reaching a vertex on the inner contour. For example, if both contours are convex, k is usually a small constant. For the shape refinement, we visit each primary pathline at a time. For a primary pathline with d corners ($d < n$), we visit each corner to see if it can be eliminated. Thus, this process takes $O(dn)$ time.

At times, the second refinement can further eliminate corners. As long as we limit the iteration to a small constant, it does not affect the asymptotic running time.

Claim 3 *The snap algorithm runs in $O(n(k + \log n))$ time, where n is the total number of vertices in two contours and k is the maximum number of triangles intersected by a primary pathline and $1 \leq k \leq n$. This algorithm is worst-case optimal for the class of algorithms that satisfy Requirements 1 and 2.*

Proof: It suffices to show that the worst case indeed requires $O(n^2)$ vertices to be added to the polyhedral shape reconstructed. Any algorithm satisfying Requirements 1 and 2 must use $O(n^2)$ vertices to reconstruct a polyhedral shape with top and bottom contours as given in Fig. 24. □

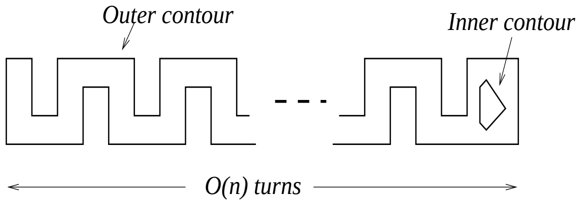


Figure 24: Worst case requires $O(n)$ turns to be made in reconstruction.

We now take a look at the case with vertex correspondence. Suppose there are c pairs of vertex correspondences. It takes $O(n \log n)$ time to generate a path within a triangulated polygon, since the induced connectivity graph has $O(n)$ vertices and edges [11]. Thus, it takes $O(cn \log n)$ time to generate c vertex paths and the resulting triangulation is of $O(cn)$ in size. Using the above argument with n replaced by cn, it takes $O(cn(k' + \log(cn)))$ time to run the snap algorithm, where k' is the number defined as above and $1 \leq k' \leq n^2$. Thus, if c is a small constant, then the running time is not affected much, whereas if $c \approx n$, then the asymptotic running time in the worst case is $O(n^3)$, which slows down the process significantly for a large n.

7 Discussions

This section contains a few ideas and remarks on refining and extending our framework.

7.1 Comparison between triangular and rectangular tilings

Our approach differs from most existing methods in that our algorithm does not use triangular tiling. Some remarks are in order to discuss merits and demerits of triangular and rectangular tiling approaches.

In introduction, we have pointed out that approaches using triangular tiling can suffer from some anomaly coming from a specific triangulation chosen for the tiling. This anomaly is less likely to occur in our approach. Fig. 25 contains two instances of triangulation for the same pair of contours together with some pathlines. (This illustration is without pathline refinement.) In both cases, the concave parts (AB) of the upper contour correspond to similar parts in the lower contour. The situation illustrated as in Fig. 1 is less likely to occur in our algorithm.

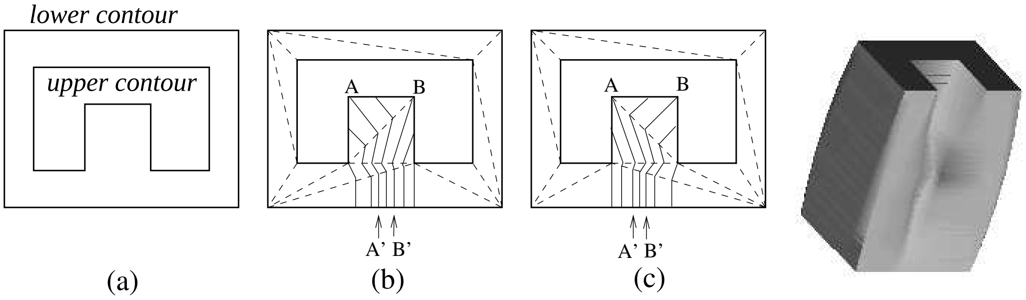


Figure 25: Illustration for the snap algorithm. (a) Dissimilar contours. (b, c) Pathlines for different triangulations (without pathline refinement). (d) Reconstruction example by our method.

Approaches using triangular tiling usually add extra points on the contours to avoid such anomaly as in Fig. 1. Our approach also uses additional vertices that are not provided in the given contours. Methods based on triangular tiling add vertices in the contours, whereas our approach add vertices between contours. This extra degree of freedom also makes it possible to provide flexibility to satisfy Requirement 2, namely, avoidance of overhangs. This flexibility is not afforded unless vertices in some intermediate level are introduced. Whether or not this is a merit for shape reconstruction is dependent on application domains. This feature appears reasonable for terrain modeling, since it makes the reconstructed shape look more natural. According to our analysis, the number of vertices added is not prohibitively high, especially for cases where contours are not severely convolved.

Regarding the requirement that no intersection be produced in the reconstructed shape, the method [1] also satisfies this condition (Their input is more general in that it is not restricted to the case that one contour is enclosed by the other). They satisfy this condition by introducing the Voronoi diagram (or medial axis) for the regions which remain unprocessed after a few passes of tiling. As they point out, since numerically stable implementation of Voronoi diagrams is difficult and involved, their method first decomposes the region in question into a number of convex regions before medial axis is created. As compared with their method, our method does not treat this situation as a special case, which helps to make implementation and correctness proof simpler. In case of a valley-like situation (as in Fig. 26), the valley created by our method is sloped (that is, height of the bottom of the valley decreases monotonically from the upper part to the lower part in the valley), while the method of [1] assigns a constant value for all points on the medial axis and it would require a small additional step to incorporate such a slope.

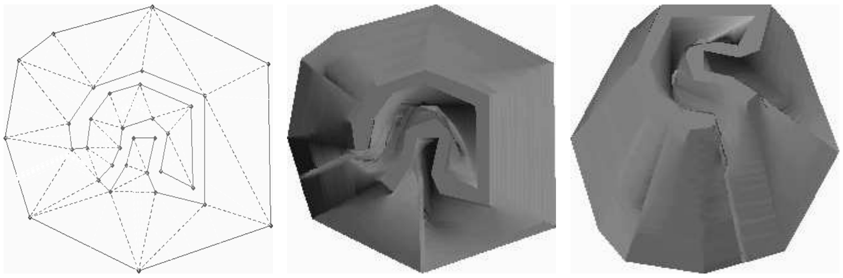


Figure 26: Shape with valley. Contour (left); top and side views of the reconstructed shape (middle and right).

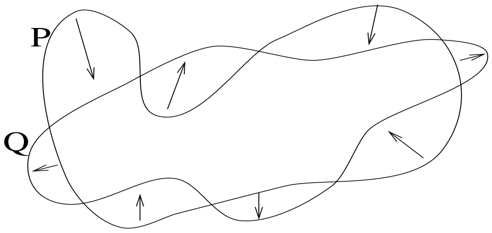
For branching, our method require no special case analysis other than finding branch points. For finding a branch point, the snap algorithm is applied first to each contour independently and the point of highest intersection is considered as a branch. In case there is more than a single highest intersection point, we use a united contour to define a branch curve. Thus, no special treatment is necessary for the U-shaped contour which is difficult to handle in some approach [8].

7.2 Snap strategies

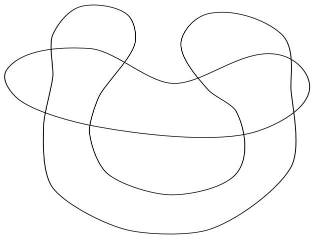
There are a few choices as to how to carry out snap operations outlined in Section 3. For an edge snap as in Fig. 6(b), K can be anywhere between P_i and P_{i+1} to satisfy Requirement 1 (no self-intersecting shape). Thus, a simple choice is just to pick the middle point (midpoint-policy). A more natural choice is to choose K such that $\frac{P_iK}{KP_{i+1}} = \frac{P_iQ_j}{Q_jP_{i+1}}$ (proportional-point-policy). For a vertex snap, L (in Fig. 6(b)) can be chosen in a similar manner. When two contours are similar in shape, these choices produce a reasonable 3D shape. In our experiments using contours with similar shapes, no distinguishable differences are observable between these two policies in the pictures produced.

A related issue is snap ordering (i.e., sequence of triangles to be eliminated). Any legal sequence generates an intersection-free deformation as long as each snap is legal. A few choices will include (i) pick the largest triangle first for elimination and (ii) pick the longest edge to snap (or to be snapped on) first. In our experiment, this does not produce a visible difference in the final appearance and both choices work reasonably well.

In Section 5, we described a scaling heuristic for a pair of contours for which one is not properly enclosed by the other. For some cases, this scaling is unnecessary. When the intersection of the interiors of P and Q is simply connected (as in Fig. 27), we can still use the snap strategy without scaling. Here, we first introduce new vertices for both contours at their intersection points. Then, we inflate or deflate the parts outside of the common region depending on whether the parts are inside of Q or outside of Q. Fig. 27(a) illustrates the situation. This method cannot handle a situation such as in Fig. 27(b). This is a type of input which cannot be processed by the approach of Bajaj et al either, since it violates their Condition 2 [1].



(a)



(b)

Figure 27: Inflation and deflation

8 Concluding Remarks

8.1 Contributions

In this paper, we have introduced an isotopy-based method for shape reconstruction from contours. A provably-correct method has been presented to produce intersection-free shapes from contours. Compared to triangular tiling methods, our method requires additional vertices to be used to construct a solid shape. The additional vertices make it possible to achieve a number of desirable features in shape reconstruction. Important features of our algorithm include the following.

- The reconstructed shape does not contain overhangs which makes this method suitable for terrain modeling.
- Vertex correspondences can be specified, which makes it possible to highlight geometric landmarks such as ridges and valleys in reconstruction.
- The running time of the algorithm is dependent on the shape of contours. The algorithm runs in $O(n^2)$ time in worst case, when contours are highly convoluted. This has been shown to be worst-case optimal for algorithms defined for the task.
- Branching is handled by dividing the reconstruction process into steps each of which can be considered as an isotopy.

Experimental results have been presented to illustrate our approach. Our results show a natural appearance even for non-nested contours by using a scaling heuristic. The method proposed can be either used by itself (as described above) or used together with existing triangular tiling based algorithms. In the latter case, an isotopy is constructed based on the existing triangulation scheme and it makes it possible to add vertex correspondence in the existing framework. Contour-to-contour isotopy can be extended to an isotopy of interiors. Previously, we have studied a method [14] for constructing such an isotopy. Such a consideration is of interest when the internal area is to be interpolated as well as the exterior contour.

8.2 Future work

We now list a number of directions in which our framework may be extended.

Choice of speed: In the previous section, we describe the method in which all points on polygon P arrive at somewhere on polygon Q simultaneously. However, arriving times can vary as long as they are continuous along the perimeter of Q. This means that we can assign selected vertices of Q values between 0 and 1 indicating the arrival times at these vertices. The arrival times at other points on Q can be obtained by interpolating the assigned values. For shape modeling, this means that we no longer have to assume that two slices are parallel to each other. Vertices in the second plane may have different heights.

Once a family of t-shapes is generated, we have a control over speed of deformation. These degrees of control make it possible to handle shape reconstruction from nonparallel slices and incorporating slope information (when handling more than two successive slices at each step) in shape reconstruction. Recently, Barequet et al [2] have considered using more than two consecutive slices in their triangular tiling framework by using different triangulations depending on the branch situation encountered.

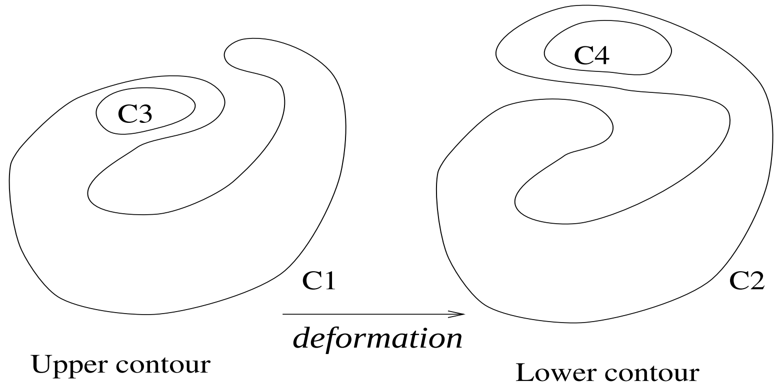


Figure 28: Internal contour

Nested contours: One remaining problem is that of generating deformation paths for internal contours. If a pair of internal contours C_3 and C_4 are positioned in arbitrary locations within boundary contours C_1 and C_2, respectively (refer to Fig. 28), then any piecewise linear interpolation may not be able to interpolate the two slices without self-intersection. In such a case, using a path planner in shape reconstruction may be useful. One such approach, where shape blending is performed amidst environmental obstacles, is described in [24].

Optimization in reconstruction: In some approach, 'goodness' of reconstruction is discussed in terms of minimization of the total surface area or volume of the reconstructed shape [13, 3], while others are more concerned about correctness of reconstruction [1] or visual appearance [32]. In our present work, no attempt is made for optimization. However, such optimization may be incorporated by way of snap order sequencing. Currently, our snap order sequence is determined in a greedy manner (i.e., we pick a triangle with a largest area). We could choose to minimize a certain quantity defined over the reconstruction process. For example, the total length of primary paths is a possible candidate, since a long and winding primary path does not in general lead to a good solution. Clearly, an optimal solution to the shape reconstruction problem contains within it optimal solutions to subproblems (where the outer contour has been shrunken after a number of snap operations). Thus, an optimization approach such as dynamic programming can be applied to determine an optimal snap order sequence (for a fixed snap-point policy).

Many-to-many branching Finally, we are interested in extending our framework to handle the case of many-to-many branching. The process can be divided into subprocesses each of which is considered as an isotopy. We leave these issues as our future research topics.

Acknowledgements. We would like to thank Xiuwen Liu and Paula Stevensen of the Center for Mapping of the Ohio State University for making the contour map data available for our experiments.

References

[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] G. Barequet, D. Shapiro, and A. Tal. History consideration in reconstructing polyhedral surfaces from parallel slices. In *Proceedings of IEEE Visualization*, pages 149–156, 1996.

[3] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding*, 63(2):251–272, 1996.

[4] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D. Du and F. Hwang, editors, *Computing and Euclidian Geometry*, pages 23–90. World Scientific, 1992.

[5] J. Boissonat. Shape reconstruction from planar cross-sections. *Computer Vision, Graphics, and Image Processing*, 44(1):1–29, Oct. 1988.

[6] L. Chang, H. Chen, and J. Ho. Reconstruction of 3d medical images: A nonlinear interpolation technique for reconstruction of 3d medical images. *CVGIP: Graphical Models and Image Processing*, 53(4):382–391, 1991.

[7] L. Chew. Constrained Delaunay triangulation. *Algorithmica*, 4:97–108, 1989.

[8] H. Christiansen and T. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics (Proc. of SIGGRAPH)*, 12:187–192, 1978.

[9] D. Cohen-Or, D. Levin, and A. Solomovici. Contour blending using warp-guided distance field interpolation. In *Proceedings of IEEE Visualization*, pages 165–172, 1996.

[10] L. Cook, P. Cook, K. Lee, S. Batnitzky, B. Wong, S. Fritz, J. Ophir, S. Dwyer, L. Bigongiari, and A. Templeton. An algorithm for volume estimation based on polyhedral appoximation. *IEEE Transactions on Biomedical Engineering*, 27:493–500, 1980.

[11] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw-Hill Book Company, 1990.

[12] A. Ekoule, F. Peyrin, and C. Odet. A triangulation algorithm form arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, 1991.
[13] H. Fuchs, Z. Kedem, and S. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, 1977.
[14] K. Fujimura and M. Makarov. Homotopic shape deformation. In *International Conference on Shape Modeling and Applications*, pages 215–225, 1997.
[15] S. Ganapathy and T. Dennehy. A new general triangulation method for planar contours. *Computer Graphics (Proc. of SIGGRAPH)*, 16:69–75, 1982.
[16] C. Gitlin, J. O'Rourke, and V. Subramanian. On reconstructing polyhedra from parallel slices. *International Journal of Computational Geometry and Applications*, 6(1):103–122, 1996.
[17] A. Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):267–277, March 1996.
[18] A. Kalvin, E. Schonberg, J. Schwartz, and M. Sharir. Two-dimensional model-based boundary matching using footprints. *International Journal of Robotics Research*, 5:38–55, 1986.
[19] L. Kehtarnavaz and R. D. Figueiredo. A framework for surface reconstruction from 3d contours. *Computer Vision, Graphics, and Image Processing*, 42:32–47, 1988.
[20] L. Kehtarnavaz, L. Simar, and R. de Figueiredo. A syntactic semantic technique for surface reconstruction from cross sectional contours. *Computer Vision, Graphics, and Image Processing*, 42:399–409, 1988.
[21] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19:2–11, 1975.
[22] J. Latombe. *Robot Motion Planning*. Kluwer Academic Press, 1990.
[23] W. Lin and S.Y.Chen. A new surface interpolation technique for reconstructing 3d objects from serial cross-section. *Computer Vision, Graphics, and Image Processing*, 48:124–143, 1989.
[24] J. Lu and K. Fujimura. Shape transformation in space-time. *Visual Computer*, 12(9):455–473, 1996.
[25] W. Massey. *A Basic Course in Algebraic Topology*. Springer-Verlag, New York, N.Y., 1991.
[26] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, 1992.
[27] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1993.
[28] J. O'Rourke. On the scaling heuristic for reconstruction from slices. *CVGIP: Graphical Models and Image Processing*, 56(3):420–423, 1994.
[29] A. Rosenfeld and A. Nakamura. Local deformations of digital curves. Technical Report CAR-TA-832, University of Maryland, Center for Automation Research, August 1996.
[30] L. Schumaker. Reconstructing 3d objects from cross-section. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computation of Curves and Surfaces*, pages 275–309. Kluwer Academic, Dordrecht/Norwell, MA, 1989.
[31] M. Shantz. Surface definition for branching contour-defined objects. *Computer Graphics*, 15:242–270, 1981.
[32] Y. Shinagawa and T. L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *Visual Computer*, 7(2-3):72–86, May 1991.
[33] K. Sloan and J. Painter. Pessimal guesses may be optimal: A counterintuitive search result. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:949–955, 1988.
[34] J. Stillwell. *Classical Topology and Combinatorial Group Theory*. Springer-Verlag, New York, N.Y., 1993.
[35] Y. Wang and J. Aggarwal. Surface reconstruction and representation of 3d scenes. *Pattern Recognition*, 19(3):197–207, 1986.
[36] E. Welzl and B. Wolfers. Surface reconstruction between simple polygons via angle criteria. *Journal of Symbolic Computation*, 17(4):351–369, April 1994.
[37] M. Zyda, A. Jones, and P.G.Hogan. Surface construction from planar contours. *Computers and Graphics*, 11:393–408, 1987.