Surfaces From Contours: The Correspondence and Branching Problems

David Meyers
Shelley Skinner
Department of Computer Science and Engineering,
University of Washington,
Seattle, Washington 98195

Kenneth Sloan
Department of Computer and Information Sciences,
University of Alabama, Birmingham
Birmingham, Alabama 35294

Abstract

This paper is concerned with the problem of reconstructing the surfaces of three-dimensional objects, given a collection of planar contours representing cross sections through the objects. This problem has important applications in clinical medicine, bio-medical research and instruction, and industrial inspection.

The problem can be broken into four subproblems, the *correspondence problem*, the *tiling problem*, the *branching problem*, and the *surface fitting problem*. We describe our system for surface reconstruction from sets of contours with respect to each of these subproblems. Special attention is given to the correspondence and branching problems.

Résumé

Cet article examine le problème de la reconstruction de la surface d'un objet à trois dimensions à partir d'une série de profiles plans représentants des sections prises à travers l'objet. Ce problème a d'importantes applications en médecine clinique, en recherche biomédicale, dans l'enseignement et en contrôle industriel.

Le problème peut être divisé en quatres parties: le problème de correspondance, le problème de facettage, le problème de branchement et le problème d'interpolation de la surface. Nous décrivons notre solution pour la reconstruction de surfaces à partir de profiles par rapport à chacune de ces tâche. Une attention particulière est portée aux problèmes de correspondance et de branchement.

Keywords: surface reconstruction, tiling, meshes, surface fitting, branching surfaces

[1]This work was supported in part by the National Science Foundation under grants DCR-8505713, CCR-8612543, and IRI-881932.

1 Introduction

The problem of reconstructing a three-dimensional surface from a set of planar contours is an important problem in diverse fields. For example, biologists try to understand the shape of microscopic objects from serial sections through them. Analysis of neural tissue involves attempts to reconstruct the network from serial slices. In clinical medicine, the data generated by various imaging techniques such as CAT, ultrasound and NMR provide a series of slices through the object of study.

The following definitions will be used:

- A *contour* is a simple polygon representing the intersection of the surface of an object and the plane of its *section*.
- A *section* is the set of contours formed by one slice through an area of interest. The contours in a section do not necessarily come from the same object.
- A *canyon* is a region between two contours that merge in an adjacent section. Canyons are formed when two contours are close together along an extended portion of their perimeters.

There are two basic approaches to constructing surfaces from contours: volume based and surface based. Volume based approaches assume data is available as a three-dimensional grid. Surface based approaches assume the data defines the intersection of a surface and a plane of sectioning. Which approach is most applicable depends on the nature of the data. When the available data takes the form of a three-dimensional lattice of density values, as is the case with NMR and other radiological methods, a volume based approach such as the marching cubes algorithm of Lorensen and Cline [10] is applicable. If the available data is in the form of a set of closed contours denoting the surfaces of the objects to be reconstructed, as would be available from hand digitized outlines of the objects in a set of serial sections, then a different approach is preferred. If the spacing between contours is relatively small, the volume based approaches can be used by simply filling the interior of a contour with an appropriate value, distinct from the value exterior to a contour. If spacing between contours is large relative to the resolution within a contour, volume based approaches yield unsatisfactory results. If objects are sliced very obliquely there may be insufficient overlap between contours in adjacent sections to determine connectivity. In such cases, a surface based approach that constructs a triangular mesh from the data points defining the contours is preferred. This paper will concentrate on surface based approaches, and on work aimed at solving some of the problems with such approaches.

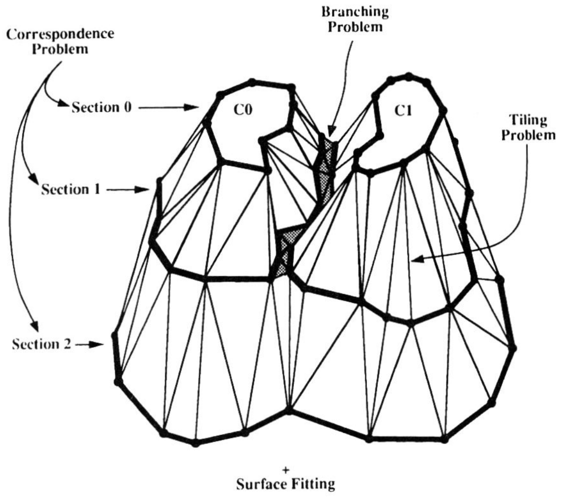

Figure 1: This set of contours illustrates several of the problems that must be solved in order to reconstruct a surface.

The problem of generating a surface from a set of contours can be broken into several subproblems (see Figure 1).

- The *correspondence problem* involves deciding which contours from two different sections should be linked together in the generated surface. A solution to the correspondence problem determines the coarse topology of the final surface.
- The *tiling problem* consists of generating the "best" set of triangular faces using one contour from each of two adjacent sections. A commonly chosen metric for determining what is "best" is minimization of the resulting surface area.
- The *branching problem* involves dealing with pairs of sections in which a contour in one section splits into several contours in an adjacent section. A solution to the tiling and branching problems determines the topology of the surface, and its coarse geometry.
- The *surface fitting problem* involves fitting a "best" surface to the topological mesh determined by the solution to the above problems. A solution to the surface fitting problem produces a detailed description of the geometry of the reconstructed surface.

A brief summary of previous work on each of the above subproblems follows.

The correspondence problem arises whenever there are multiple contours in a section. When this is the case, it becomes necessary to decide how the contours should be organized into groups representing individual objects. In the most general case, these contours may be widely spaced and there may be no information other than the contour boundary available to aid a solution.

Automatic solution of the correspondence problem in its most general form is difficult. Due to the underconstrained nature of the problem, considerable ambiguity can exist with respect to linkage of contours in adjacent sections. For example, consider a set of slices taken through the dendritic tree of a neuron. Tracking the individual strands may well be impossible if the spacing between slices is too large. In order to help constrain the problem, assumptions about the nature of the objects to be reconstructed can be used. See [18] and [2] for descriptions of two approaches.

The tiling problem has been the subject of most of the previous work on reconstructing surfaces from contours.

Keppel [9] first reduced the problem of matching points in successive contours to a search problem on a toroidal graph (see Figure 2). Contours are represented by ordered lists of data points. Edges connecting neighboring points in the same contour are called *contour segments.* Edges connecting a point from one contour to a point from another contour are called *spans.*

The method involves associating spans with the nodes of a graph. The graph is a dense two-dimensional grid, overlaid on a torus. Arcs are allowed only if the two spans connected share an endpoint, and the other two vertices of the spans are connected by a contour segment. With these constraints, an arc defines a triangle consisting of two points from one contour and one point from the other contour. Costs are associated with arcs by using a metric computable from the spans it connects. A surface connecting the two cross-sections is a cycle on the torus (with certain natural restrictions on the type of cycle). See Figure 2 for a depiction of a typical tiling problem and its translation into a search problem.

Virtually every conceivable metric function has been used to distinguish good surfaces from bad. A listing of some of the more important ones includes "Maximize Volume" [9], "Minimize Area" [8], "Minimize Span Length" [3], and "Match Direction" [4]. Sloan and Painter [17] ad-

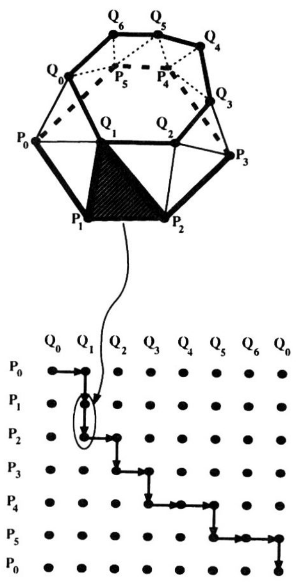

Figure 2: The problem of tiling two contours can be expressed as a graph search. Contours are represented by ordered lists of data points. Edges connecting a point from one contour to a point from the other contour are represented by nodes in the graph. Tiles are represented by arcs between nodes in the graph. Finding a tiling is done by finding a minimal cost cycle in the graph.

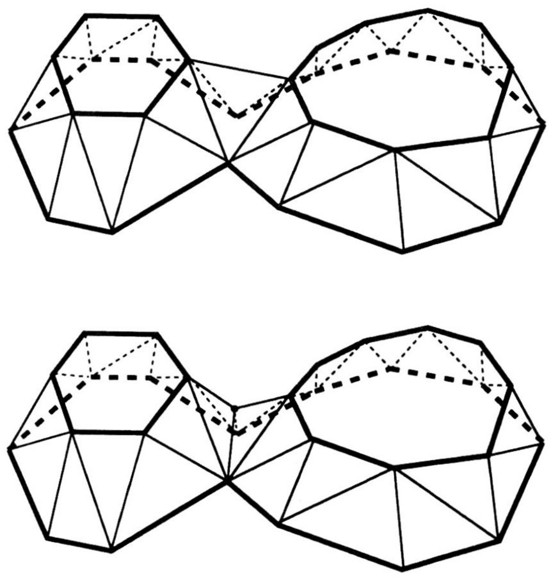
Figure 3: **Upper:** A simple case of the *Branching problem.* Two contours in one slice merge into one contour in an adjacent slice. A possible tiling is shown by the light lines, the contours are represented by the heavy lines. The two contours are merged to allow use of the graph search tiling algorithm. **Lower:** The same example, but the contours are merged as suggested by Christiansen and Sederberg [3].

dressed the choice of metric for the graph cost function, and described a few improvements to the divide-and-conquer algorithm of Fuchs *et. al.* [8].

We consider the tiling problem to be solved; we use the "One Column" method of Sloan and Painter [17], normalize for scale and translation (but not rotation), and optimize for "Minimum Surface Area."

The branching problem arises when several contours from one section merge into one contour in an adjacent section (more generally, when *m* contours become *n* contours). Figure 3 shows a simple instance of the branching problem. Figure 4 shows a more complex example.

Previous solutions to the branching problem have required that complicated instances of the problem be solved interactively. For example, Christiansen and Sederberg [3] describe a method that will handle some branching structures, but that requires user intervention in complex cases.

Boissonnat [1] has proposed an approach to constructing surfaces from contours based on the three-dimensional Delaunay tetrahedralization of the set of vertices of the contours in adjacent sections. The method is capable of handling some branching structures without user intervention.

Solution of the correspondence, branching and tiling problems results in a triangulated mesh in three-space. The *surface fitting problem* involves finding a smooth surface that interpolates (or alternatively, approximates) the vertices of the mesh and maintains the same topology. The choice of interpolation versus approximation depends to a large extent on the intended use of the resulting surface, and on the nature of the input data. If the data are noisy or otherwise imprecise, an approximating method would be preferred. If the data are the precise specifications of some object, then an interpolating method is preferable. Current approximating schemes produce smoother and "prettier" surfaces than the available interpolating schemes [12]. Interpolating surfaces provide for more accurate measurement of properties of the original object.

There are currently a number of methods in use for solving the problem of fitting a smooth surface to a mesh in three-space. The general method uses a series of parametric surface patches in which the vertices of the mesh are the control points of the surface patches, and the topology of the mesh determines which vertices are used in a patch. The surface fitting problem is the subject of much current research, a complete summary of which is beyond the scope of this paper. For a general reference see Farin [6]. Mann *et. al.* [12] compare the properties of various interpolating surface fitting methods.

This paper is a description of our current work on solutions to the correspondence and branching problems.

Additional detail is available in [13].

2 Current Work

What follows is a discussion of our current system, describing the methods we use for solving each of the problems mentioned in the introduction. This discussion will assume the contours given as input are *planar* contours. We will discuss methods for relaxing this restriction in the conclusions.

2.1 Correspondence Problem

We have developed a section description language that allows the description of a solution to the correspondence problem for a set of sections and their associated contours, and is suitable for manual or automated solution of the correspondence problem. The description of a contour set in this language is generated by our correspondence solver and is the input to the part of our system that solves the tiling and branching problems.

As a concrete example of the section description language, an abbreviated description of the set of contours shown in Figure 1 follows:

```
{ SECTION S0 2                       % Section S0 has 2 contours
  { CONTOUR C0 9                     % Contour C0 has 9 points
    { x0 y0 z0 }
    .
    .
    .
    { x8 y8 z8 }}
  { CONTOUR C1 11
    { x0 y0 z0 }
    .
    .
    .
    { x10 y10 z10 }
}}
{ SECTION S1 2
  { CONTOUR CO 17
    { x0 y0 z0 }
    .
    .
    .
    { x16 y16 z16 }
    { ADJNEXT { C1 7 8 } }     % Bridge from C0 - 7 to C1 - 8
  }
  { CONTOUR C1 17
    { x0 y0 z0 }
    .
    .
    .
    { x16 y16 z16 }
    { ADJNEXT { C0 2 14 } }    % Bridge from C1 - 2 to C0 - 14
}}
{ SECTION S2 1
  { CONTOUR C 18
    { x0 y0 z0 }
    .
    .
    .
    { x17 y17 z17 }
    { ALIAS 2 { C0 } { C1 } }  % Tile with C0 or C1
}}
```

This section description language is capable of describing branching structures with any number of branches from one section to the next. There are cases that are not handled; of most importance is the inability to handle "holes" in an object in which one contour is inside another contour from the same section.

We are working on two solutions to the correspondence problem: one is based on Soroka's local processor [18], which assembles elliptical cylinders from contours; the other computes the Euclidean minimum spanning tree of a graph constructed from the contours of the data set.

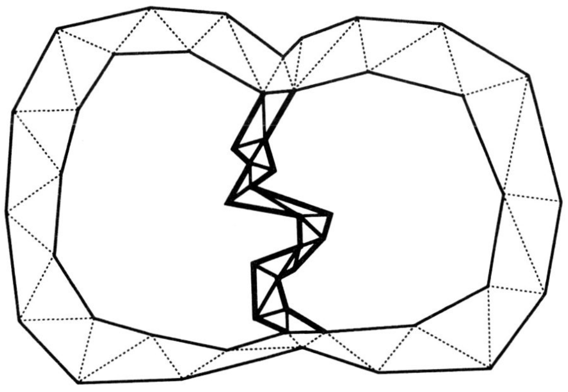

Figure 4: A more complex instance of the *Branching problem*. In this case the canyon (shown in heavier lines) must be handled separately.

We will first describe the local processor solution, and then the minimum spanning tree solution.

An elliptical cylinder (or simply *cylinder*) consists of a set of adjacent, linearly varying elliptical contours. To solve the correspondence problem we use a 3-step process:

1. Assemble elliptical cylinders from the contours.
2. Assemble elliptical cylinders into objects.
3. Process branch points.

In the first step we classify the contours as *elliptical* or *complex* by fitting ellipses to the contours. We compute the ellipse parameters: the center x and y; the major and minor diameters A and B; and the angle θ between the major axis and the x axis using the principle axis method [5] and the dot product space method [16]. Contours that can't be classified *elliptical* are classified *complex*.

We assemble elliptical contours into cylinders in which each of these parameters (except θ, which must be nearly constant to avoid modeling *twisted* cylinders) varies almost linearly with z (defined as having a linear least squares correlation coefficient of no less than 0.8, an empirically determined value that works well in practice). If a contour is elliptical, we use it to attempt to extend an existing cylinder. If an elliptical contour cannot be explained by extending an existing cylinder, a new (singleton) cylinder is created. If a contour is complex, we attempt to create an elliptical *subcontour* that can be used to extend some cylinder. This process continues until all contours are explained.

The second step assembles objects from the cylinders found in the first step, and finds *connections* between contours in different cylinders. If contours are found to be *connected*, they will be tiled together by later steps that solve the tiling and branching problems. If a cylinder cannot be merged with an existing object, a new object is created. This process continues until all cylinders are explained.

In the third step we use the inter-contour connections found in step 2 to find occurrences of branches. A 3-cylinder branch is formed between the end contours of 3 connected cylinders, while a 2-cylinder branch is formed by the connection of an end-contour of one cylinder with an interior contour of another cylinder. A straight cylinder connection occurs when the end-contours of two cylinders are connected only to each other.

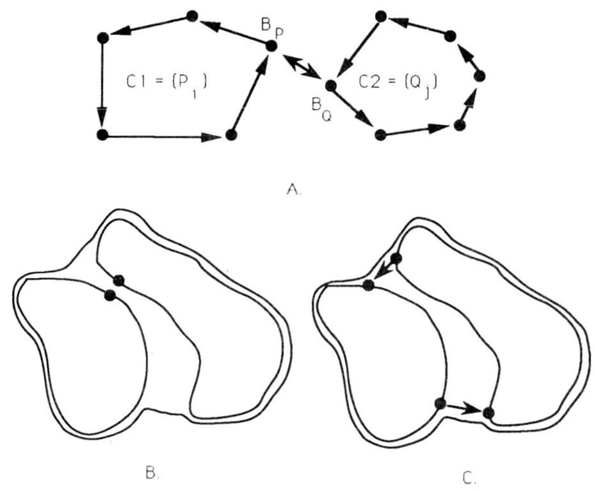

Figure 5: Computing adjacency points. **Part a:** To find adjacency points for composite contours we compute the closest pair of points, one from each contour. **Part b:** This method will not produce good results in the case where two contours approach closely over an extended portion of their perimeters. **Part c:** A good method would select adjacency points more carefully, as shown here, yielding a more accurate tiling of the canyon between the contours.

Adjacency points between the two contours on the same section are computed as shown in Figure 5a. We choose as adjacency points the two most proximate points in those contours.

Results produced by the above method depend on the order in which contours are specified within a section. It may be possible to eliminate this problem by modifying the conditions under which a new cylinder is generated.

We are currently developing a method that uses a minimum spanning tree (MST) to cluster "close" contours. Because only a tree is generated rather than a more general graph, not all edges (representing connections between contours) will be found for an object with one or more cycles (e.g. a torus). However this method seems well suited to data that comprise a single acyclic object, like a branching artery, or a set of such objects.

To build the tree, we associate the input contours with the nodes of a graph. The graph initially contains edges between all pairs of nodes representing contours on adjacent sections. We embed the nodes in the 4-dimensional space (x, y, A, B) defined by the contour's fitted ellipse center and major and minor diameters. We then compute the cost of an edge as the square of the Euclidean 4-space distance between the nodes. Next we compute the MST and break it into segments at its branch points. All contours in a segment of the MST are considered to be a "tube" and are given the same name so that they will be tiled together. Finally, we use the edge information in the MST itself to discover connections to use as input to the same cylinder branch/connection finder used in step 3 by our initial correspondence problem solution. Figure 6 compares results obtained using the two methods.

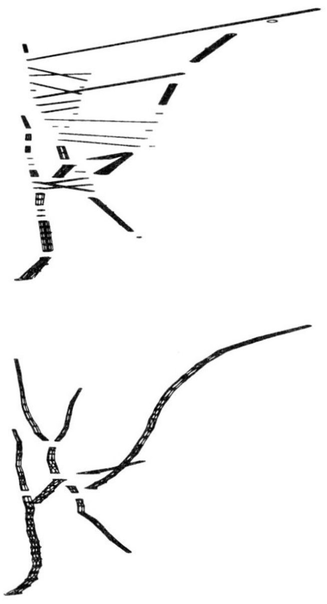

Figure 6: Biological structures obturator artery contours. **Upper:** Elliptical cylinders found using the elliptical cylinder growing algorithm are indicated by lines surrounding groups of adjacent contours. **Lower:** Segments found using the minimum spanning tree correspondence algorithm. The final solution is almost identical with the actual correspondence determined at the time the data were collected.

2.2 Branching Problem

Construction of a tiling at a branch point is accomplished by forming a composite contour from the several branch contours, and using this composite to construct a tiling with the trunk contour. This is the approach used by

Christiansen and Sederberg [3]. The composite contour is only used for tiling the two sections between which the branch occurs.

Figure 3 shows a set of contours in which the construction of a composite is relatively straightforward: the adjacent contours approach closely at a single point. In this case, we construct a composite contour connecting the two adjacent contours at the "close" points. This composite contour is then tiled with the single contour from the adjacent section, (which may itself be a composite contour).

Note that we do not add any points to the data when forming the composite. We take the point of view that the mesh produced by the tiling algorithm is not the final surface. The subsequent surface fitting step will determine where the saddle between the contours should fall. An added point would force the saddle to fall in a particular place and unnecessarily constrain the final surface fitter.

Added data points suffer from an additional problem: the original data points may have associated properties other than position (such as surface normal) that may be difficult to generate for the manufactured data points.

Figure 4 shows an example in which the construction of a composite contour is not so straightforward. If adjacent contours that form a canyon are linked into a composite at a single point as in the simple case, the resultant tiling would not be a good representation of the surface. In cases like Figure 4, we form composite contours by linking the contours across the openings into the canyon between them.

Referring to Figure 1 and the example description of its contours in Section 2.1, a composite contour constructed for tiling the two contours of section S1 to the single contour of section S2 would consist of points 0 - 7 of contour C0 followed by points 8 - 16 and 0 - 2 of C1 followed by points 14 - 16 of contour C0. Points 7 - 14 of C0 and points 2 - 8 of C1 form the walls of the canyon between the two contours. The composite contour is handled by the normal tiling algorithm; the canyon is tiled separately.

Although we have limited our discussion to bifurcating branches, our method is much more general. It is capable of handling cases in which many contours in one section merge into a single contour in an adjacent section as well as cases in which the "single" contour is in reality a composite constructed to handle an incoming branch (for example an X shaped object in which there is no section through the junction of the arms).

2.2.1 Canyon Tiling

We now discuss the problem of tiling *canyons* between contours such as shown in Figure 4.

One way to approach the problem of tiling a canyon is to treat the two walls of the canyon as contours to be tiled together, and use the same graph search method used for tiling contours from separate sections.

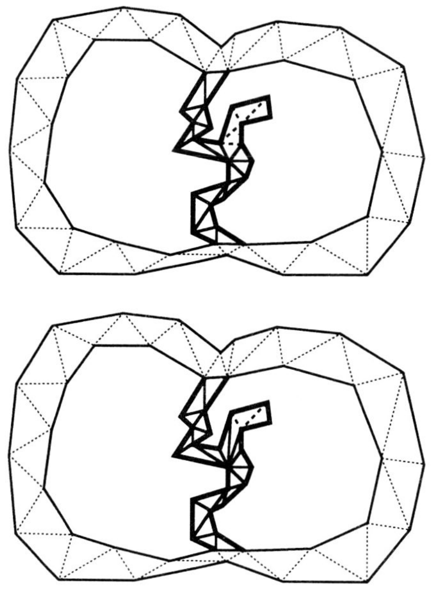

Figure 7: It is not always possible to construct a tiling in the plane using only *Spans* and *Contour Segments*. In such cases a general purpose polygon triangulation algorithm must be used. Two possible tilings are shown. **Upper:** A tiling that seeks to minimize the sum of span lengths. **Lower:** A tiling that seeks to minimize the number of intra-contour spans.

Alas, this open curve graph search method doesn't work for all possible canyons. As shown in Figure 7, it is not always possible to construct a tiling of a canyon that satisfies the constraint required by use of the graph search tiling algorithm.

Triangulation of a simple polygon has been extensively studied, and efficient algorithms exist [7]. They have as advantages their efficiency and that they handle all simple polygons. Their main disadvantage is the difficulty of controlling the nature of the triangulation.

Most known simple polygon triangulation methods produce many "long skinny triangles" [7]. Such triangles are not desirable if the triangulation is to be used as a set of surface patches to be used in construction of a smooth surface, since the influence of a data point on the shape of the surface is not restricted to its local area. Rather, it is desirable to have a set of triangles in which the vertices are "near neighbors" on the surface. The Delaunay triangulation [15] of a set of points in the plane has this property.

We cannot simply use the Delaunay triangulation of the vertices of the polygon, since we must restrict the edges of our triangulation to fall within the original polygon. The Delaunay triangulation would only satisfy this criterion for convex polygons. We start with an arbitrary triangulation, and then modify it in a later optimization step.

Given an initial trianglulation, we improve it by examining cases in which the edge shared by two triangles can be switched so it connects the two vertices opposite the edge. This can be done only when the vertices of two triangles sharing an edge form a convex quadrilateral. Swapping the edge in other cases is not possible, because the result is a pair of non-disjoint triangles. We replace the shared edge with one connecting the vertices opposite the shared edge if the new configuration results in a smaller minimum radius for the circumscribed circles defined by the pair of triangles. This is the criterion used in computing the Delaunay triangulation of a set of points.

The method of optimizing an initial triangulation according to some metric can also be used to control the final triangulation in a fashion similar to the use of a metric to guide the graph search algorithm. By starting from a valid triangulation of the canyon polygon and only allowing modifications that preserve the validity of the triangulation, we can gain some degree of control over the characteristics of the triangulation computed by general polygon triangulation algorithms, at some loss of efficiency. In the preceeding paragraph we described the use of one such optimization criterion, the "Minimize Circumscribed Circle Radius" criterion. More control over the characteristics of the final triangulation could be obtained by designing a more complex evaluation function; for example one could assign added weight to an edge crossing from one canyon wall to the other if one wanted to favor tiles with "cross canyon" edges. By adjusting the weights associated with various parameters of such an evaluation function, one could achieve better control over the nature of the final triangulation.

2.3 Surface Fitting

The final step in the construction of a surface from a set of contours is the surface fitting step. Our approach does not attempt to create a mesh that is itself necessarily a good reconstructed surface. We rely on any of several methods for fitting a surface to data points using each facet of the mesh as the control polygon of a surface patch. Because of this, we are careful not to introduce extra "manufactured" points into the mesh. By avoiding the addition of extra points, we allow our chosen surface fitter to find a "best" surface unaffected by added non-data points. Figure 9 illustrates various choices of surface fitting and rendering techniques applied to the contour sets shown in Figure 8.

2.4 Future Work

We are investigating methods for extending the MST method to handle objects with non-tree topologies, such as a torus. Automatic generation of the adjacency information for complex adjacency regions (the boundaries of the canyons) is a near-term goal.

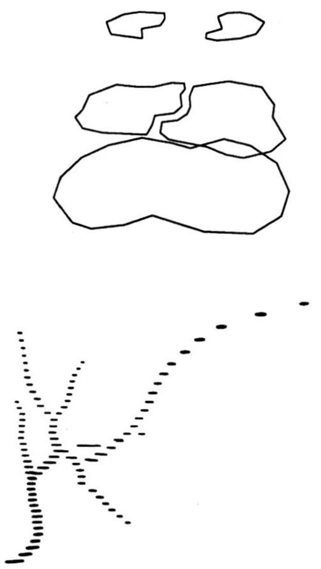
Figure 8: Two sets of contours. **Upper:** A single large object splits into two smaller objects. The intermediate slice is just above the point of separation, and the two parts are still very close together along a moderately complex border. **Lower:** An artery and several branches. Notice especially the lack of data near the branch point closest to the center of the image.

Our current system cannot handle sets of contours in which one contour is enclosed within another contour with the object interior between them. We are currently extending our system to handle this important class of branching problem.

3 Conclusions

We have described a method for reconstructing surfaces from sets of contours that extends previous results by allowing for surfaces in which branching occurs. The key idea is to separate the problem into those parts already handled well by existing methods, and to concentrate on posing and attacking the subproblems that require new approaches.

Previous methods have either not allowed for branching surfaces, or have required substantial user interaction in all but the simplest cases. In most of these methods, points are added to the data during the reconstruction process. We feel addition of non-data vertices to the contours during reconstruction should be avoided. We prefer to allow the surface fitter to determine the precise geometry of the final surface, rather than biasing its result by the addition of non-data points.

We describe two methods for solving the correspondence problem. In one method, we use the elliptical cylinder method proposed by Soroka [18]. The other method has not to our knowledge been previously published. It involves constructing a graph from the contours in the data set, and computing the minimum spanning tree of the graph. The method works well for objects that are "tree like" in their shape. In particular, note the striking improvement in the quality of the solution produced by the MST method as compared to the elliptical cylinders method for our branching artery data set.

We assume a final surface fitting step independent of the steps required to generate a triangulated mesh from the input contours. This step produces the final detailed geometric description of the reconstructed surface. We use a surface fitting testbed that allows easy testing of the suitability of various surface fitting methods for reconstruction of surfaces from contours [11].

Our discussion so far has assumed contours are planar, and that the plane of all contours in any section is the same. These restrictions simplify the presentation, but may be too restrictive in practice. We explicitly use the restriction of planarity only during solution of the correspondence problem and when handling the tiling of canyons between branching surfaces. In both cases, if the data are not planar we can proceed by finding an average plane for the points involved using a least squares method, projecting the points onto this plane, and proceeding as before. We expect this will be sufficient for most practical problems where the concept of a "plane of section" has any meaning. The more general problem of scattered three-dimensional data does not fall into this class and will require entirely different methods.

4 Acknowledgements

We thank Tony DeRose for suggesting the minimum spanning tree as a possible approach to solving the correspondence problem. We would also like to thank our colleagues in the GRAIL lab for their help and support during the course of this work, especially the "gang of seven" who thought it would be an interesting exercise to survey the many existing fine methods of surface fitting and pick one...

References

[1] Jean-Daniel Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics, and Image Processing*, 44(1):1–29, 1988.

[2] Y. Bresler, J.A. Fessler, and A. Macovski. A Bayesian approach to reconstruction from incomplete projections of a multiple object 3d domain. *IEEE Trans. Pat. Anal. Mach. Intell.*, 11(8):840–858, August 1989.

[3] H.N. Christiansen and T.W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics*, 12(2):187–192, August 1978.

[4] Larry T. Cook, P. Nong Cook, Kyo Rak Lee, Solomon Batnitzky, Bert Y. S. Wong, Steven L. Fritz, Jonathan Ophir, Samuel J. Dwyer III, Lawrence R. Bigongiari, and Arch W. Templeton. An algorithm for volume estimation based on polyhedral approximation. *IEEE Trans. Biomed. Engin.*, BME-27(9):493–500, September 1980.

[5] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[6] Gerald Farin. *Curves and Surfaces For Computer Aided Geometric Design: A Practical Guide, Second Edition*. Academic Press, Inc., 1990.

[7] Alain Fournier and Delfin Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3(2):153–174, April 1984.

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

[9] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM J. Res. Develop.*, 19:2–11, January 1975.

[10] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[11] Michael Lounsbery, Charles Loop, Stephen Mann, David Meyers, James Painter, Tony DeRose, and Kenneth Sloan. A testbed for the comparison of parametric surface methods. In *SPIE/SPSE Symposium on Electronic Imaging Science and Technology*, Santa Clara, CA, February 1990.

[12] Stephen Mann, Charles Loop, Michael Lounsberry, David Meyers, James Painter, Tony DeRose, and Kenneth Sloan. A survey of parametric scattered data fitting using triangular interpolants. In Hans Hagen, editor, *Curve and Surface Modeling*, chapter Chapter 1. SIAM Publications, To Appear.

[13] David Meyers, Shelley Skinner, and Kenneth Sloan. Reconstruction of surfaces from contours. Technical report, University of Washington, Dept. of Computer Science and Engineering, In Preparation.

[14] Gregory M. Nielson. A transfinite, visually continuous, triangular interpolant. In Gerald Farin, editor, *Geometric Modelling: Algorithms and New Trends*, pages 235–246. SIAM, 1987.

[15] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*, chapter 5, pages 203–204, 215, 217, 220. Springer-Verlag, 1985.

[16] K.R. Sloan, Jr. Analysis of 'dot product space' shape descriptions. Technical Report 74, Department of Computer Science, University of Rochester, 1980.

[17] K.R. Sloan, Jr. and J. Painter. Pessimal guesses may be optimal: A counterintuitive search result. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):949–955, November 1988.

[18] B.I. Soroka. Generalized cones from serial sections. *Computer Graphics and Image Processing*, 15:154–166, 1981.

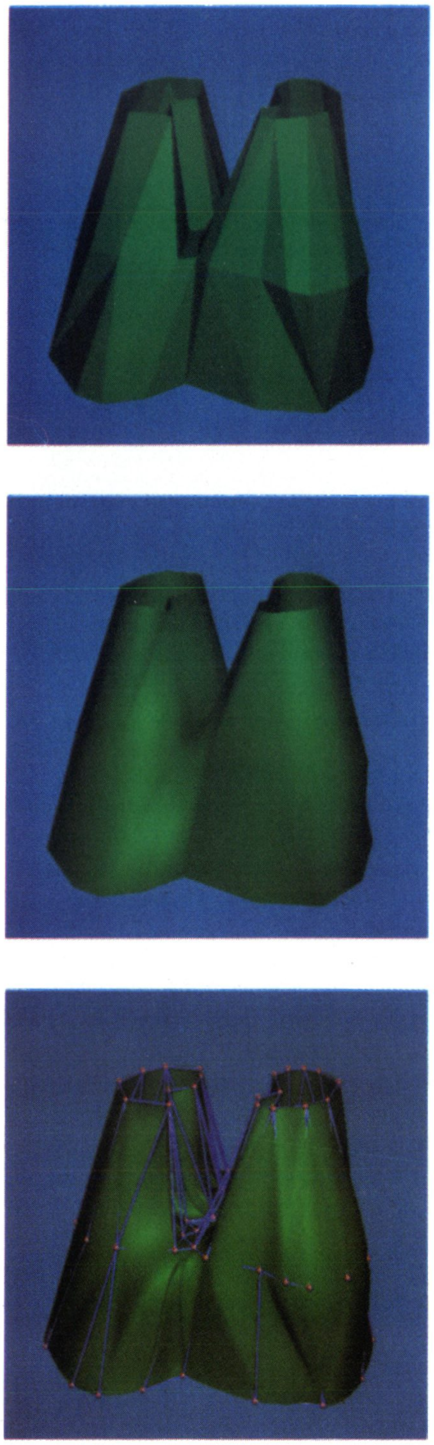

Figure 9: Reconstructed surfaces. **Top:** A flat-shaded rendition of a piecewise planar mesh generated from the data in Figure 8a. **Center:** The same mesh, Gouraud shaded to give the illusion of a smooth surface. **Bottom:** An interpolating surface, generated by the *side-vertex* method of [14]. The mesh is shown as red balls and blue sticks.

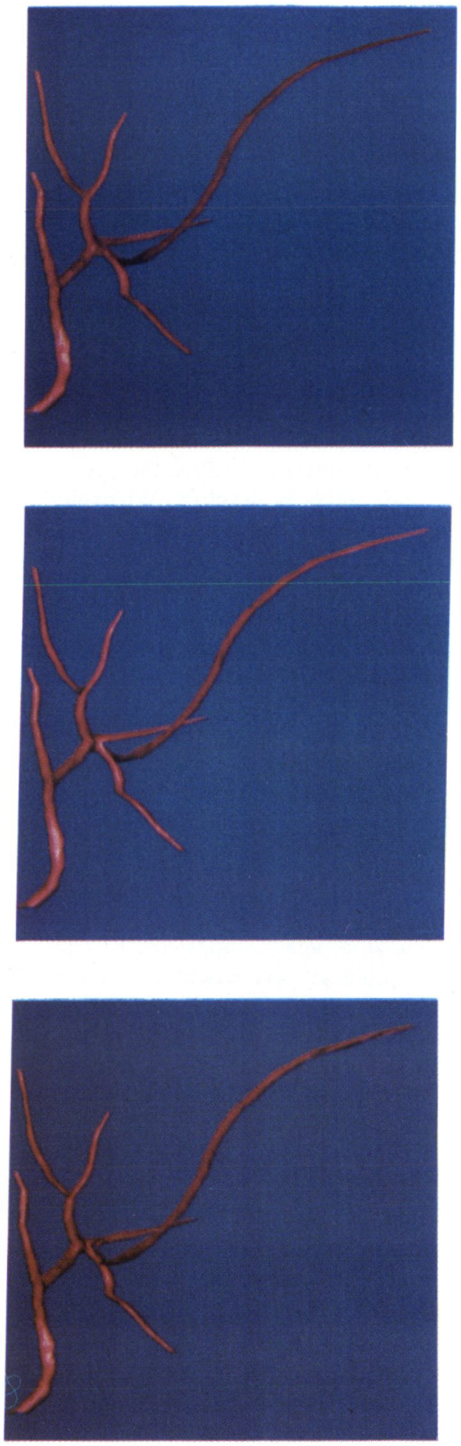

Figure 10: Reconstructed surfaces. **Top:** a flat-shaded piecewise planar mesh generated from the data in Figure 8b. **Center:** The mesh from part d, Gouraud shaded. Except for the branch point closest to the center of the image, this approximation is adequate. **Bottom:** An interpolating surface, generated from the mesh of part d by the *side-vertex* method of [14].