Surfaces from Contours

DAVID MEYERS and SHELLEY SKINNER
University of Washington
and
KENNETH SLOAN
University of Alabama at Birmingham

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 biomedical research and instruction, solid modeling, and industrial inspection.

The method we describe produces a triangulated mesh from the data points of the contours which is then used in conjunction with a piecewise parametric surface-fitting algorithm to produce a reconstructed surface.

The problem can be broken into four subproblems: the *correspondence problem* (which contours should be connected by the surface?), the *tiling problem* (how should the contours be connected?), the *branching problem* (what do we do when there are branches in the surface?), and the *surface-fitting problem* (what is the precise geometry of the reconstructed surface?). 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. We present a method that can handle sets of contours in which adjacent contours share a very contorted boundary, and we describe a new approach to solving the correspondence problem using a Minimum Spanning Tree generated from the contours.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*boundary representations; curve, surface, solid, and object representations; geometric algorithms, languages and systems;* I.3.8 [**Computer Graphics**]: Applications; J.3 [**Computer Applications**]: Life and Medical Sciences—*biology; health*

General Terms: Algorithms

Additional Key Words and Phrases: Branching problem, branching surfaces, correspondence problem, meshes, minimum spanning tree, surface fitting, surface reconstruction, tiling

1. INTRODUCTION

The problem of reconstructing a three-dimensional (3-D) 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

This work was supported in part by the National Science Foundation under grants DCR-8505713, CCR-8612543, IRI-881932, and IRI-9102860.

Authors' addresses: D. Meyers and S. Skinner, Department of Computer Science, University of Washington, Seattle, WA 98195; K. Sloan, Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294-1170.

sections through the object. In clinical medicine, the data generated by various imaging techniques such as computed axial tomography (CAT), ultrasound, and nuclear magnetic resonance (NMR) provide a series of slices through the object of study. In Computer Aided Design (CAD), lofting techniques specify the geometry of an object by means of a series of contours.

There are two basic approaches to the problem: volume based and surface based. Volume-based approaches assume that data are available as a 3-D grid. Surface-based approaches assume the data define 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 are a dense 3-D lattice of 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 [15] or the geometrically deformed models of Miller et al. [19] may be best. If the available data are a set of closed contours denoting the surfaces of the objects to be reconstructed, as would be available from manual tracing of the objects by a trained observer, then a surface-based approach may be preferred. If the spacing between slices is comparable to the resolution within the plane of the contours, a volume-based approach can be used by superimposing a sampling grid on the contour data and by assigning a different value to grid points falling inside a contour than to grid points falling outside a contour. The grid values can then be used with a volume-based method.

A large separation between the planes of adjacent sections causes problems for volume-based approaches, since they rely on overlap of projected contours for their solution to the problem of determining the adjacency topology of contours in the data set (i.e., which contours should be connected by a surface; see the *Correspondence Problem* discussed below). If widely spaced sections slice obliquely through an object, there may be no overlap between contours representing the object. In such cases, a method that uses more global information to determine topological adjacency relationships is preferred. We concentrate on surface-based approaches and work aimed at solving some of the problems with such approaches.

The input data to a surface-based approach are a series of sections each containing one or more contours. For example, Figure 11 shows two data sets, one of which was constructed to illustrate our discussion, while the other was obtained from anatomical slices through an arterial network.

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 a 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, and an object may be represented by more than one contour in a section. We assume that contours are simple polygons and do not intersect other contours of the data set.

—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.

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

—The *correspondence problem* is solved by determining the topological adjacency relationships between the contours of a data set. A solution to the correspondence problem determines the coarse topology of the final surface.

—The *tiling problem* is solved by generating the "best" topological adjacency relationships between the points on pairs of contours from adjacent sections by constructing a triangular mesh from their points. A commonly chosen metric for determining what is "best" is minimization of the resulting surface area.

—The *branching problem* arises when an object is represented by a different number of contours in adjacent sections, in which case the standard method for solving the tiling problem cannot be used directly. A solution to the tiling and branching problems determines the topology of the surface and its coarse geometry.

—The *surface-fitting problem* is solved by fitting a "best" surface to the mesh computed by solving the above problems. A solution to the surface-fitting problem produces a detailed description of the geometry of the reconstructed surface.

In this paper, we describe work that improves upon existing methods for solving the correspondence and branching problems. We describe a new solution to the correspondence problem, which computes the minimum spanning tree of a graph constructed from the contours in a data set. We also describe a solution to the branching problem, which avoids introducing extra points into the data.

2. SUMMARY OF PREVIOUS WORK

2.1 Correspondence Problem

The correspondence problem arises whenever there are multiple contours in a section. When this is the case, the contours must be organized into groups representing individual objects. In general these slices are widely spaced, and there is no information other than the contour boundary.

Automatic solution of the correspondence problem in its general form is difficult. Due to the underconstrained nature of the problem considerable ambiguity can exist. For example, consider a set of slices taken through a tangled cord or the connections between neurons in nerve tissue. Tracking the individual strands may be impossible if the spacing between slices is large. Assumptions about the nature of the objects to be reconstructed are used often by automated solutions to help constrain the problem, allowing a "reasonable" solution where no single "correct" solution is possible.

The difficulty of an instance of the correspondence problem depends on the resolution of the available data. In the simplest case, the spacing between adjacent sections allows unambiguous solution of the correspondence problem

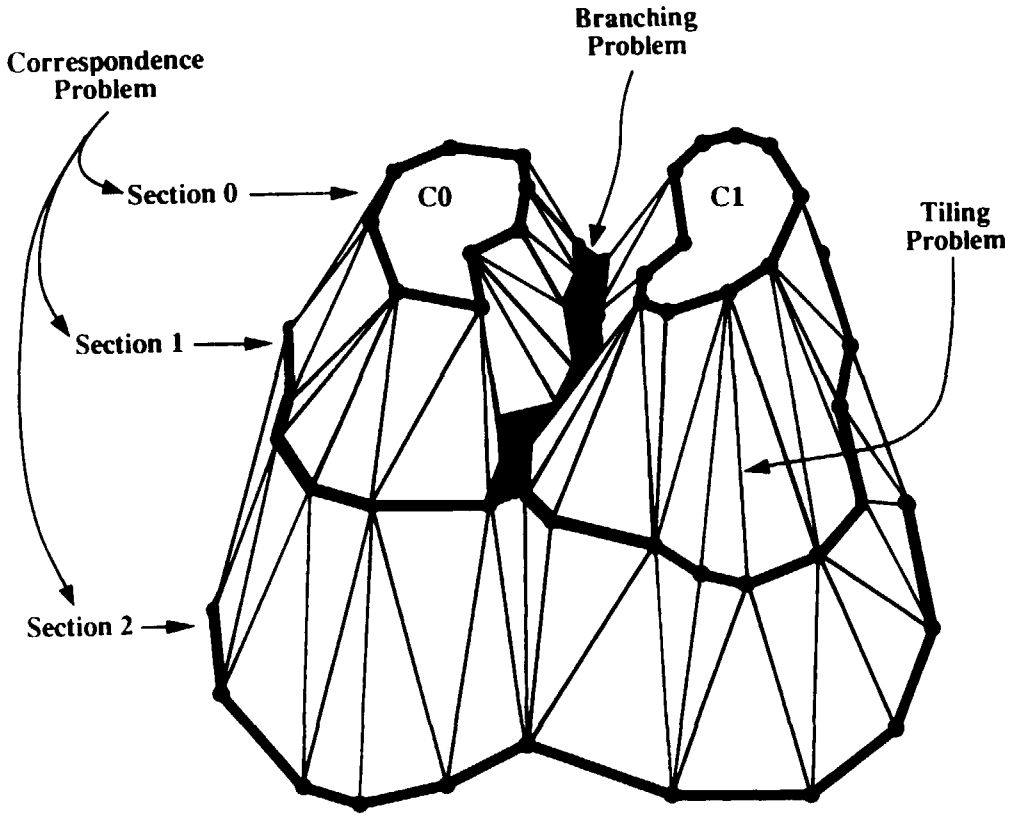

Fig. 1. This set of contours illustrates several of the problems that must be solved to reconstruct a surface. The *correspondence problem* is solved by determining which of the contours from each of the sections should be connected together. The *tiling problem* is solved by choosing a "best" surface connecting contours determined to connect by the solution to the correspondence problem. To solve the *branching problem*, we must determine how to connect the two contours in section 1 to the one contour in section 2, and to each other. After solving these problems, we solve the *surface-fitting problem* by computing a smooth surface to fit the data points of the topological mesh we have constructed by solving the first three problems.

by examining overlap between contours on adjacent sections. Many current solutions to the correspondence problem make the assumption that the available data allow this approach. If the available data are not dense enough to overlap between contours of the same object on adjacent slices, these methods will construct multiple objects.

When available data are insufficient for the overlap method, methods that use a more global view of the data set and incorporate some assumptions about the objects to be reconstructed have been used. Soroka [27] used the concept of a "generalized cylinder" for extracting high-level shape information from a set of contours. He used a special class of generalized cylinder, the "elliptical cylinder," in which the contours are elliptical and the parameters of successive contours vary linearly (within error limits). This approach involves assembling contours into elliptical cylinders, then assembling the elliptical cylinders into objects.

Bresler et al. [2] addressed the correspondence problem for objects that can be described by generalized cylinders. They partitioned a set of contours into "feasible objects" subject to certain constraints that depend on the presumed nature of the objects to be reconstructed. Their use of domain knowledge to group contours into feasible objects helped reduce the exponential complexity of the problem.

2.2 Tiling Problem

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

Keppel [14] first reduced the problem of matching points in successive contours to a search problem on a toroidal graph (see Figure 2). Fuchs et al. [11] provided an extensive analysis of the search problem and developed an efficient search method. They formalized Keppel's approach and applied a "Divide-and-Conquer" technique to speed the search. In this method 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 a graph node with each span. The graph is a dense two-dimensional (2-D) grid, overlaid on a torus. Arcs are allowed only if the spans represented by the connected nodes share an endpoint, and the pair of unshared span vertices are connected by a contour segment. With these constraints, an arc defines a triangle consisting of an edge from one contour connected by two spans to one point from the other contour. Costs are associated with the arcs by using a metric computable from the triangle they define. 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.

The key to the divide-and-conquer algorithm for searching the toroidal graph is that once a minimum-cost cycle passing through a given node is found, that cycle can be used to limit the amount of the graph that must be searched when looking for lower cost cycles not constrained to pass through the node [11].

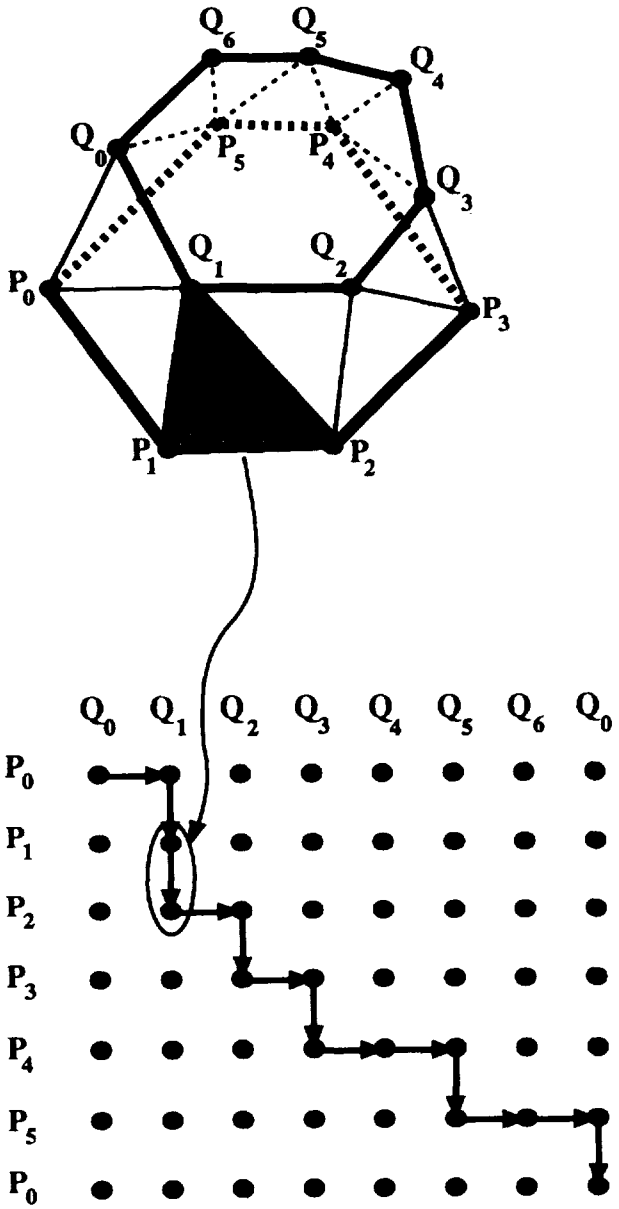

Fig. 2. The problem of tiling two contours can be expressed as a graph search. Contours are represented by ordered lists of data points. Edges connecting neighboring points in the same contour are called *Contour Segments*. Edges connecting a point from one contour to a point from the other contour are called *Spans*. Spans 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.

The tiling problem is severely underconstrained. Many surfaces could give rise to the observed cross-sections. In order to choose one "correct" surface, we optimize with respect to some objective function. The chosen objective function should capture some notion of what a "good" surface is and should be easy to compute.

Virtually every conceivable metric function has been used to distinguish good surfaces from bad. Sloan and Painter [25, 26] addressed the choice of metric for the graph cost function and described a few improvements to the divide-and-conquer algorithm. A brief summary of some of the more important metrics follows:

Volume. Keppel [14] used "Maximize Volume." It is an obvious metric for convex objects, but difficult to use. Indeed, a major contribution of [14] was the demonstration that one could, in fact, calculate the contribution of a single tile to an objective function related to the total volume of an object. Keppel apparently felt that "Maximize Volume" is wrong for the concave regions of an object and designed a multistage algorithm to deal separately with the concave and convex parts of individual contours.

Area. Fuchs et al. [11] used "Minimize Area" for illustration purposes. It is intuitively appealing and probably as good as any metric. However, special care must be taken to handle certain pathological cases (see below). Since "Minimize Area" is easy to compute and produces good results when used in conjunction with normalization of the contours for position and radius, this is our metric of choice.

Match direction. Cook et al. [6] used a metric based on matching the directions of points from their respective contour's center of mass. It is intuitively appealing for concave and museum-viewable objects, but there are problems with severely concave (or convoluted) objects, which Cook et al. went to some lengths to handle.

Span length. Christiansen and Sederberg [5] described a "greedy" method based on "Minimizing Span Length," which does not use the graph search method to construct a tiling. The method incorporated normalization of size and position and was extended to handle some branching structures. The normalization step involved mapping a contour from its bounding rectangle to a unit square, translating the squares for each contour so that they are centered at the origin of the (x, y) plane prior to computing the tiling, and then applying the computed tiling to the original contours. A possible extension to handle complex branching situations, which is similar to the implementation we describe here, was suggested in [5].

Match normalized arc length. Ganapathy and Dennehy [12] described another "greedy" method in which the length of contours being tiled is normalized and in which the positions of the contour vertices along the contour are expressed relative to the normalized length. The spans that constitute the tiling are chosen then so that they are as "nearly vertical" as possible (i.e., the span is chosen which results in the least difference in normalized arc length between the current points on the two contours).

Nonlocal metrics. The metrics discussed above are local metrics, since they assign costs to arcs by considering only the individual tile that corresponds to that arc. As an example of a nonlocal metric, Wang and Aggarwal [28] assigns an initial figure of merit to individual triangles (using "Inverse Span Length") and then uses a relaxation procedure to improve these cost assignments by considering the merit of neighboring triangles. Some triangles have their merit driven to zero and are removed from the graph.

All of the above metrics exhibit poor performance with carefully constructed pathological examples. Some fall prey to rather common cases. For example, consider two identical, circular cross-sections, positioned in parallel planes, with their centers offset by one diameter. Some of the metrics listed above, in particular "Minimize Area" and "Minimize Span Length," will produce something similar to a double cone, joined at a line, rather than the (obvious?) skewed cylinder. Note that this effect is present, but often difficult to see, even when the circular cross-sections are displaced by a small amount. The resulting surfaces are topologically cylinders, but there are creases.

The solution to this problem is to normalize the two cross-sections so that their centers lie on an axis perpendicular to the plane of section, solve the tiling problem to find lines joining points on the two contours, and then use this correspondence to construct a surface from the original contours. See [5], [24], and [25] for details.

Sloan and Hrechanyk [24] considered the problem of "sparse data" and described methods (based on normalization and shape matching) that recognize cases where straightforward tiling is unlikely to be correct. Normalization for position, size, and small rotations were shown to handle many problems correctly. Large rotations were handled by *hallucinating* intermediate cross-sections.

Here, we do not address possible improvements to solution of the tiling problem; we use the "One Column" method of Sloan and Painter [25, 26], normalize for scale and translation (but not rotation), and optimize for "Minimum Surface Area." This method has proven sufficient for all of the naturally occurring tiling problems we have encountered.

2.3 Branching Problem

An instance of the branching problem exists when an object represented by m contours in one section is represented by n contours in an adjacent section where $m \neq n$ and $m, n > 0$. 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, [5] describes a method that will handle some branching structures but which requires user intervention in complex cases. The key idea is to form composite contours, adding fabricated vertices between the adjacent contours to model the saddle surface implied by the contours. In more complex boundary regions, Christiansen and Sederberg [5] resort to user interaction to guide a solution.

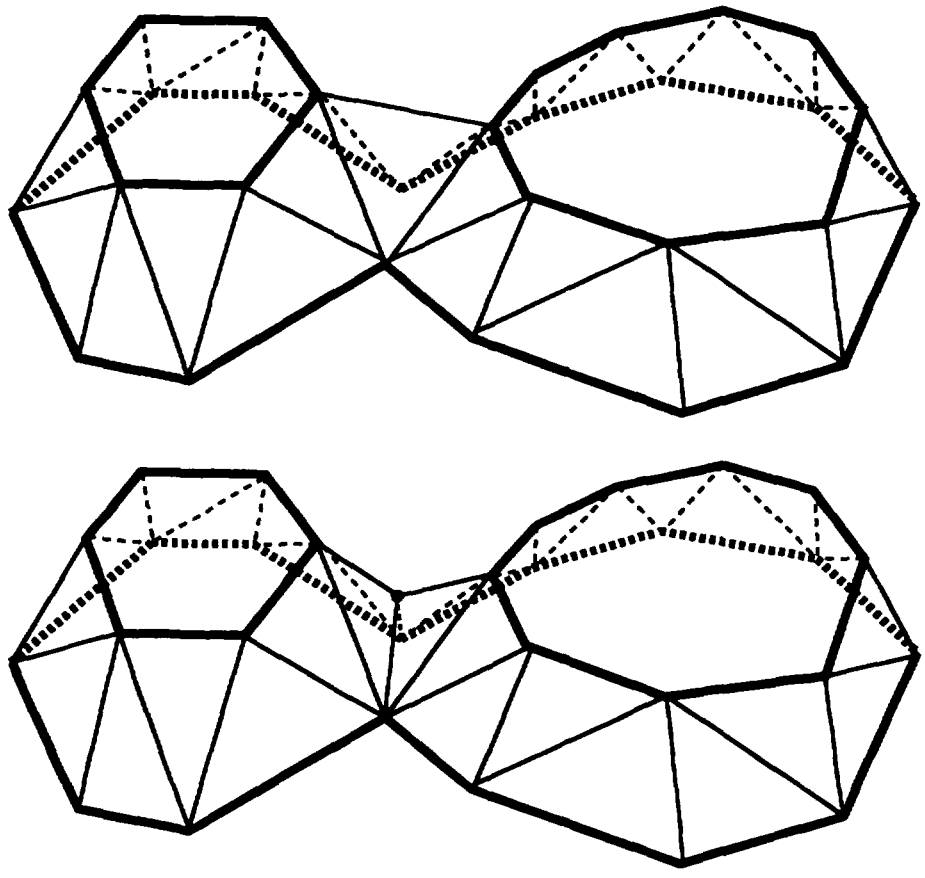

Fig. 3. **Upper:** A simple case of the *branching problem*. Two contours in one slice merge into one contour in an adjacent slice. A possible tiling is shown by the light lines; the contours are represented by the heavy lines. The two contours are merged to allow use of the graph search tiling algorithm. **Lower:** The same example, but the contours are merged as suggested by [5]. Note that a data point has been fabricated between the pair of contours in the upper slice. This new data point improves the piecewise planar surface, but unnecessarily limits a later smooth-surface fitter.

Boissonnat [1] proposed an approach to construct surfaces from contours and that computes the Delaunay triangulation of the set of vertices of the contours in two sections. Extra vertices are added so that the contour edges are contained in the resulting triangulation. This is required because the edges of a polygon are not necessarily present in the Delaunay triangulation of its vertices. Computing the triangulation in three dimensions forms a set of tetrahedra. In most cases, a solid remains after removal of tetrahedra which have a face or edge in the plane of one of the sections that is not contained within the contour. The surface of this solid is used in the reconstruction. The method is capable of handling some branching structures without user intervention.

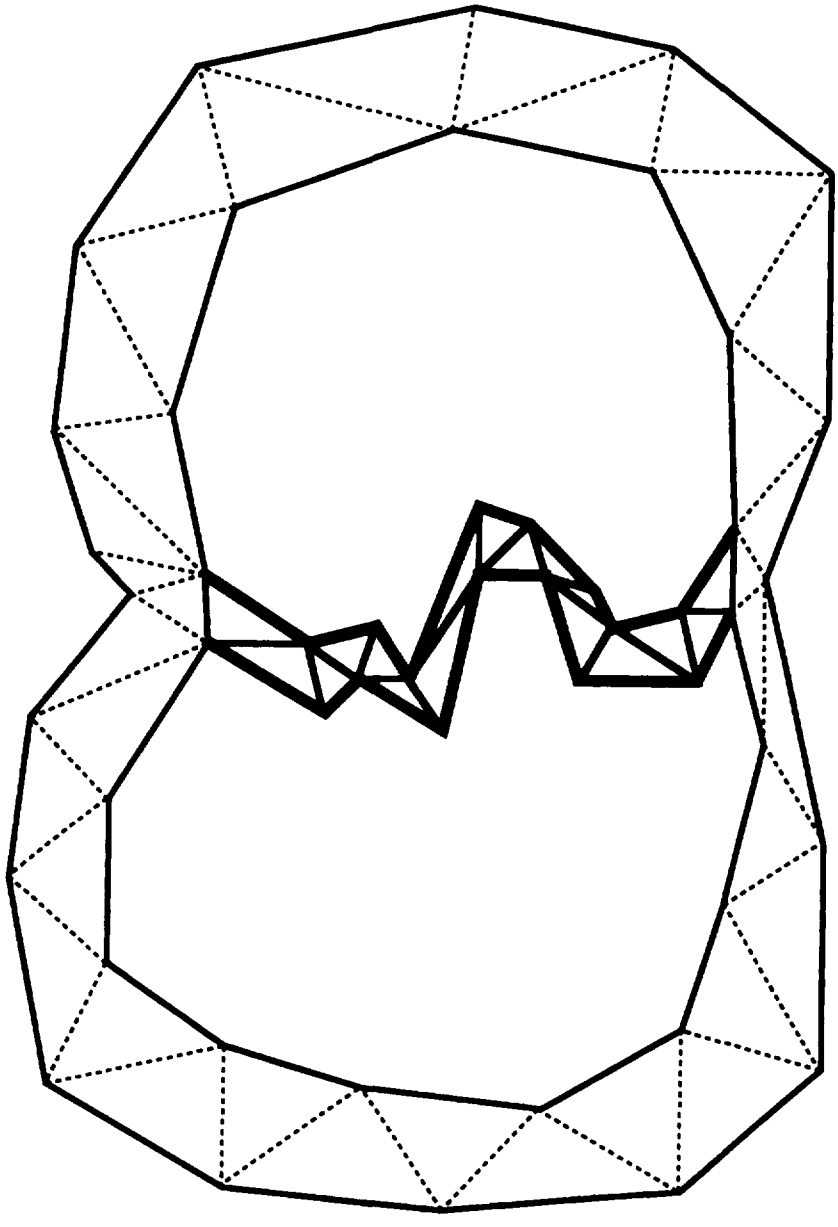

Fig. 4. A more complex instance of the *branching problem*. In this case the narrow corridor between the contours (shown in heavier lines) must be handled separately. The contours are merged at the entrances to the corridor. The merged contour is tiled as in the simpler case (dashed lines), and the corridor is handled separately by tiling in the plane (darker lines).

Related to the branching problem is the problem of tiling a pair of contours with very different shape. O'Rourke and Subramanian [21] has recently shown that, in general, it is not possible to construct a simple polyhedron from two planar simple polygonal faces in parallel planes, using only the vertices of the polygons as the polyhedron vertices. See Ekoule et al. [8] for an approach to this problem.

2.4 Surface Fitting

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 either interpolates or 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 preferable. If the data are the precise specifications of an object, then an interpolating method is preferable. Current approximating schemes produce smoother and prettier surfaces than the available interpolating schemes.

One possible solution to the surface-fitting problem is to use the triangular faces of the mesh as the surface. Unless the contours sample the original surface very finely, this piecewise planar approximation will not be very good. The triangular mesh contains connectivity information that is important in guiding a fitting method to a reasonable surface, but alone it is not usually a satisfactory solution.

Currently, there are a number of methods in use for solving the problem of fitting a smooth surface to a mesh in three-space. The general method is to use 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 each 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 [9]. Mann et al. [17, 18] compare the properties of various interpolating surface-fitting methods and examine the effects of methods for choosing various free parameters on the quality of the resulting surface.

3. 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 that the contours given as input are *planar* contours. We will discuss methods for relaxing this restriction in the conclusions.

3.1 Correspondence Problem

To maximize the flexibility of a system, a method for specifying a solution to the correspondence problem that is suitable for either automated or manual methods should be used. We have developed a section description language, which allows the description of a solution to the correspondence problem for a

set of sections and their associated contours and is suitable for either method of solving the correspondence problem.

As a concrete example of the section description language, Figure 5 shows an abbreviated description of the set of contours shown in Figure 1. The following discussion refers to these figures.

The contour set contains three sections named S0, S1, and S2. The first two sections each contain two contours named C0 and C1. The third section contains one contour named C. Since sections S0 and S1 both contain a contour named C0, those contours will be tiled together, similarly for contours C1. Section S1 contains two contours, while section S2 contains one contour. Three possibilities exist in such a case:

(1) The two contours of S1 and the single contour from S2 represent three separate objects, and the contours should not be connected.
(2) One of the contours of S1 represents the last slice through its object, and the other contour of S1 joins the single contour of S2.
(3) The two contours of S1 merge into the single contour of S2.

In the example, the information within the contours of section S1 indicates that they are adjacent and merge in the next section. Absence of the information indicating merging of contours would mean that case 1 applies. If case 2 applied, then one of the contours in section S1 would have been given the same name as the single contour in section S2. The adjacency information gives the name of the adjacent contour and indices into the list of points of the local and adjacent contours. This information is used in constructing *composite* contours, the details of which will be found in Section 3.2.1. Associated with contour C of section S2 is information indicating that this contour can be tiled to a contour of a different name. Since C0 and C1 of section S1 merge into C of section S2, contour C is given pseudonyms indicating that it can be tiled with contour C0 and C1 of the previous section (the "ALIAS" entry).

This section description language is capable of handling 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, unless the plane of section is oriented so that no contour is contained within another contour.

3.1.1 *Automated Solutions.* We have implemented two solutions of the correspondence problem: One method, based on Soroka's object-understanding system [27], uses a production system to assemble elliptical cylinders from contours one section at a time. An elliptical cylinder (or simply *cylinder*) consists of a set of elliptical cross-sections (contours fitted with ellipses) on adjacent sections whose centers lie along a linear axis. The lengths of the elliptical axes also vary linearly and independently along the cylinder. A second method uses a minimum-cost spanning tree (MST) of a graph constructed from the contours of the data set to compute most of the correspondences simultaneously. The output of both of these implementations is a

```
{ 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
  }
}
```

Fig. 5. An abbreviated section description language description of the example contour set shown in Figure 1.

solution to the correspondence problem described using the section description language discussed above.

3.1.2 *Contour Analysis.* Both methods for solving the correspondence problem involve computing a best-fitting ellipse for each contour. This reduces each contour to a set of five ellipse parameters: center x and y, major and minor half-axes A and B, and orientation θ, the angle that the major axis makes with the x axis. In addition, the Cylinder-Growing method [27] requires that contours be classified as either elliptical or complex.

Ellipse parameters are computed by the following methods: The major-axis inclination, θ, is computed by finding the direction of maximum dispersion of the contour points using the method of principal axes [7]. Using the dot product space method [23], we compute A and B for a contour by finding a rectangle with maximum aspect ratio that only surrounds the contour. The center (x, y) is the centroid of the points defining the contour boundary.

Classification of contours as elliptical or complex is done by computing the standard deviation of the distance from a contour point to the fitted ellipse. If the standard deviation is sufficiently small,[1] the contour is classified as elliptical. Contours that cannot be classified as elliptical are classified as complex and are treated specially during elliptical cylinder assembly.

3.1.3 *Cylinder Growing.* Solving the correspondence problem using the cylinder-growing method is a three-step process:

(1) Assemble *elliptical cylinders* from the contours.

(2) Assemble *objects* from the elliptical cylinders.

(3) Analyze the intercylinder connections within objects to find *branches.*

Major differences between Soroka's system [27] and ours include the following: Soroka assumes the availability of contours obtained by region growing on sections perpendicular to all three axes, and we assume contours perpendicular to only one axis. In Soroka's system, a complex contour may be shared among several cylinders, while, for simplicity, we limit its use to just one cylinder.

We have implemented Soroka's suggestion of using a second production system to discover connectivity between cylinders, and we have subsequently used the connectivity to compute branching information.

Cylinder assembly. After classifying contours, we assemble contours into cylinders by adding contours to the end of existing cylinders or by creating new cylinders. The goal is to produce a small number of cylinders, each one containing as many contours as possible. A contour is said to *support* a cylinder if its ellipse parameters satisfy that cylinder's linear constraints within user-defined error limits.

A cylinder is described by its name, the contours supporting it, their minimum and maximum section numbers, their average orientation, and by

[1] A value of 0.5 yields satisfactory results in practice.

the slopes and intercepts of the linear functions $A(z)$, $B(z)$, $x(z)$, and $y(z)$ that were found using least squares fitting[2] to the ellipse parameters A, B, x, and y. The contours that support a cylinder are stored in a contour list for that cylinder in order of decreasing z.

To model a set of contours as a simple piecewise collection of cylinders, we also require that θ be nearly constant within a cylinder. Otherwise, a grouped set of contours having perceptibly varying orientations may form a *twisted* cylinder. A cylinder with nearly constant θ that also satisfies the above linearity conditions is said to be *valid*.

The production system consists of a database of contours and cylinders, three rules, and a control cycle that selects an unexplained contour and attempts to apply the first rule to the contour in conjunction with each existing cylinder. Each rule is tried in this fashion until the contour is explained or until all rules have been tried.

The first rule succeeds if the selected contour is elliptical, if it lies on a section adjacent to one end of an existing cylinder, and if appending it to this cylinder would yield a valid cylinder. In this case the contour is incorporated into the cylinder. Cylinders are tested in order until one is extended or until they have all been found to be unextendible using the selected contour. The second rule succeeds if the contour is elliptical. Since no cylinder could be extended, a new (singleton) cylinder is created and given a unique name.

If the first two rules both fail, the selected contour is complex, so we attempt to create an elliptical *subcontour* that can be used to extend some existing cylinder. For each nonsingleton cylinder, we extrapolate the cylinder's elliptical parameters onto the section of the complex contour to form a temporary elliptical contour. If the temporary contour intersects the complex contour sufficiently (70% works well in practice), it becomes a *subcontour* that we append onto the appropriate end of the cylinder.

We use sampling to estimate the area of intersection as follows. First, we find a rectangle that surrounds the union of both contours (complex and extrapolated) to use as a base bitmap. We discretize each contour into a separate bitmap and use a polygon-filling algorithm to fill each contour. The AND of the resulting bitmaps gives an estimate of the intersection.

If the current cylinder is a singleton, we simply append to it a copy of the ellipse that best fits the complex contour. When this process terminates, all contours have a unique name within their section. This information will later be used to describe a solution to the correspondence problem using our section description language.

Object assembly. The purpose of the second step of the algorithm is to assemble the cylinders found by the first step into objects by finding connections between cylinders.

Intercontour connections are found using a second production system. This production system consists of only 2 rules. At each cycle it selects an

[2] The use of a minimum correlation coefficient of 0.8 produced reasonable results.

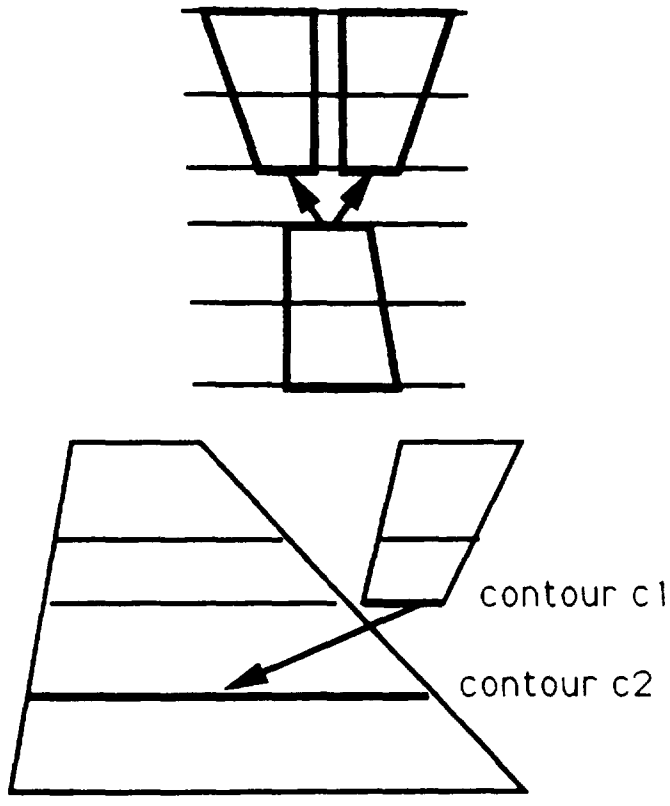

Fig. 6. An illustration of the types of connections between cylinders. **Upper:** An *end-to-end* connection is illustrated. **Lower:** An *interior* connection is illustrated.

unexplained cylinder and attempts to use it to extend an existing object. When all cylinders have been explained, this production system terminates.

The first rule searches exhaustively for all connections between the selected cylinder and each cylinder in every currently existing object and succeeds if any connections are found that explain the selected cylinder.

There are two types of connections: End to end and interior. An end-to-end connection joins end contours of two cylinders. An interior connection joins an end contour of one cylinder to an interior contour of another cylinder.

We use the following criterion to determine if a connection exists: If a contour C1 is an end contour of cylinder CYL1 and if C2, in CYL2, is on a section adjacent to C1, then a link (C1, C2) is said to exist between CYL1 and CYL2 when the orthographic projection (along z) of C1 onto C2 *overlaps* C2 (or if C2 overlaps C1), by a given amount (70% works well in practice). Figure 6 illustrates the connection types.

Whenever a connection (C1, C2) is found to connect an unexplained cylinder to a cylinder from an existing object, we add the unexplained cylinder to the existing object. The connection is stored in a list associated with each

contour and in a global connection list to be used in later analysis. We also maintain a list of the contours involved.

The second rule of the connection-finding production system is applied whenever no connection is found between the selected cylinder and any existing object. In this case the selected cylinder is used to create a new singleton object.

Finding branches. In the third step, we analyze the connections between cylinders found in the second step to locate branch points. We currently only check for bifurcations. The extension to higher-degree branching is straightforward.

Branch locations are found by examining the list of connections created for each contour in the previous step. There are three cases to consider: A three-cylinder branch is formed when the end contours of two cylinders merge into the end contour of a third. A two-cylinder branch is formed by the connection of an end contour of one cylinder with a non-end contour of another cylinder. A pair of simply connected cylinders occurs when the end contours of two cylinders are connected only to each other. Figure 7 illustrates the various types of branches.

A pair of simply connected cylinders is recognized by a connection between the top of one cylinder and the bottom of another. An additional constraint is that the two contours must not be connected to any other cylinders. When this case is recognized the two cylinders are merged. This is accomplished by adding a pseudonym ("ALIAS") to the section description language representation of one of the end contours involved.

A two-cylinder branch is recognized by a connection between an internal contour of one cylinder and an end contour of another, with the added constraint that there are no other connections involving the two contours. When this case is recognized, the required adjacency information is written into the records for the two contours involved.

A three-cylinder branch is recognized by connections between the end contours of two cylinders to the end contour of a third cylinder. This case requires that adjacency information and pseudonym information be added to the contour records involved.

The quality of the results produced by this production system method depend to a large extent on the order in which contours are specified within a section. This is due in part to the use of singleton cylinders. As long as the contour orientations are sufficiently similar, a valid cylinder will always result from appending any adjacent contour to a singleton cylinder. Because a contour only supports one cylinder, a wrong choice will invalidate further use of that contour by another, potentially better, cylinder. If an error occurs there is no chance of it being corrected, since backtracking is not utilized. Thus, one error may easily be propagated. A single unexplained contour and a singleton cylinder alone simply do not provide sufficient information to determine whether the contour should be used to extend the cylinder. Further, it may be too limiting to require that a contour support only one cylinder.

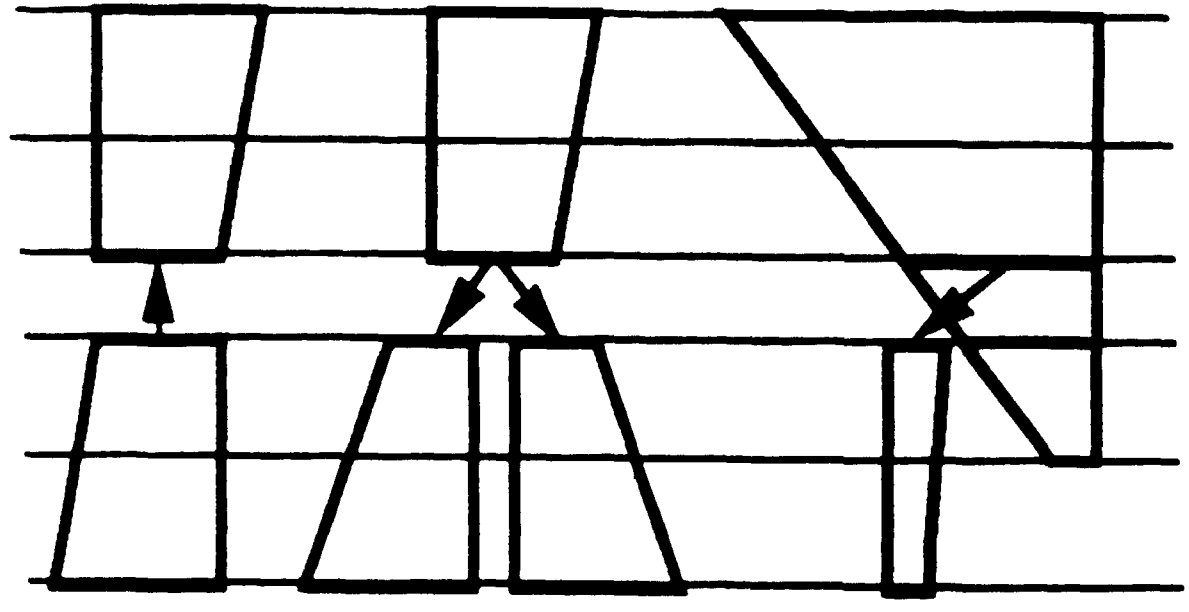

Fig. 7. An illustration of the types of branches. **Left:** A pair of simply connected cylinders. **Center:** A three-cylinder branch. **Right:** A two-cylinder branch. Generalization to multicylinder branches is straightforward.

To deal with these shortcomings, we have considered a modification to the cylinder-growing algorithm: One requiring that cylinders be of length three or greater to be retained in a list of *candidate* cylinders. Rather than being based on a production system, this modification would work sequentially, as follows: We can consider a window of three consecutive sections at a time, hypothesizing all cylinders of length two between the first two sections and keeping only those cylinders that can be extended to length three or more. Cylinders that cannot be extended beyond two contours are deleted. As the window is advanced, cylinders are extended sequentially. Variations on this method might involve changing such things as the number of passes made through the sections, when the cylinders can be hypothesized, which cylinder end(s) can be extended, and whether contours can explain multiple cylinders.

We have not implemented these improvements to the cylinder-growing approach because we feel that the Minimum Spanning Tree approach, which we discuss next, is a more promising method for solving the correspondence problem.

3.1.4 *Minimum Spanning Tree.* In view of the difficulties involved with the cylinder-growing approach to solving the correspondence problem, we investigated a more global technique of finding correspondences. The method we have developed computes the minimum spanning tree (MST) of a graph constructed by linking each contour to all contours in adjacent sections. Costs are associated with the edges by computing distance in the four-space defined by the ellipse center coordinates and major and minor axis lengths.

To solve the correspondence problem using the MST method, we use the following process:

(1) Compute edge costs for a graph G, compute an MST in G, and break the MST into *segments*.
(2) Use G to find intersegment connections.
(3) Analyze the intersegment connections for branches, computing the required adjacency information between branching contours.

As in the first method, we begin by fitting each contour with an ellipse. The MST method considers, however, all contours to be elliptical, so that classification as elliptical or complex is not required.

Having fitted the contours, we then build an undirected graph $G = (V, E)$ with nodes V and edges E. We create initially an edge (i, j) between each pair of nodes i and j for which contours $C(i), C(j)$, represented by nodes i, j, lie on adjacent sections.

By associating each node of G with the ellipse fitted to its corresponding input contour, we embed the nodes in the 4-D space (x, y, A, B) defined by the ellipse center and major and minor axes.

To compute the cost $c(i, j)$ of an edge (i, j), we use the square of the Euclidean distance between the nodes. That is, the cost of an edge (i, j) between nodes i and j having ellipse parameters $(A_k, B_k, x_k, y_k), k = i, j$ is

defined as:

$$c(i,j) = (A_i - A_j)^2 + (B_i - B_j)^2 + (x_i - x_j)^2 + (y_i - y_j)^2 \qquad (1)$$

This metric is desirable because $c(i,j)$ decreases as the centers of the ellipses fit to contours $C(i)$ and $C(j)$ become closer and as their elliptical axes become more similar in size. Once G is initialized, a free minimum-cost spanning tree in G is computed. If edge (i,j) is in the MST, then we say that the contours $C(i)$ and $C(j)$ correspond.

Tree segmentation. To determine correspondences between contours, the MST is broken into segments at the branch points of the tree. To accomplish this, we convert the free MST into a rooted tree by choosing any leaf node as the root. To build a segment, we begin at a leaf node and traverse up the tree toward the root, adding each unvisited node to the segment until we either pass the root, reach a *branch point* (a node with two or more children), or change direction in z.

After the root is passed, we repeat the traversal, starting with another leaf node. We repeat until all the nodes are visited. At a branch point, we close the current segment, initialize a new segment with the current node, and continue to traverse up the tree. Whenever a visited node would cause the segment to be nonmonotonic in z, a new segment is started.

All contours in a segment of the MST are now considered to be a tube and are given the same name so that they will be tiled together. We use the edge information inherent in the MST to locate connections between tubes: a pair of contours (on adjacent sections) from two different tubes that should be tiled together.

In our current implementation, we require that two connected tubes must be joined end to end. With this restriction, we need only examine the MST for edges that connect two tubes to assemble tubes into objects. In general, it will be necessary to remove this restriction, especially when we extend to nontree topologies.

Branch finding. Finally, the connections found in the previous step are analyzed for branching using the algorithm described in Section 3.1.3. Here, branches between the segments (rather than cylinders) are analyzed and used to produce a section description file, as before.

Comparison of MST and cylinder growing. Figure 8 compares results obtained using the cylinder-growing and MST methods.

The MST method is well suited to data from a single naturally branched (acyclic) object, such as an artery. Since the computation of a minimum spanning tree is a global process, the method is not subject to the propagation of local errors as is the cylinder-growing method.

Unfortunately, because it generates a tree rather than a more general graph, not all edges (representing connections between contours) will be found for an object having one or more cycles (e.g., a torus). Likewise, one or more incorrect edges will be found that join a set of disjoint objects.

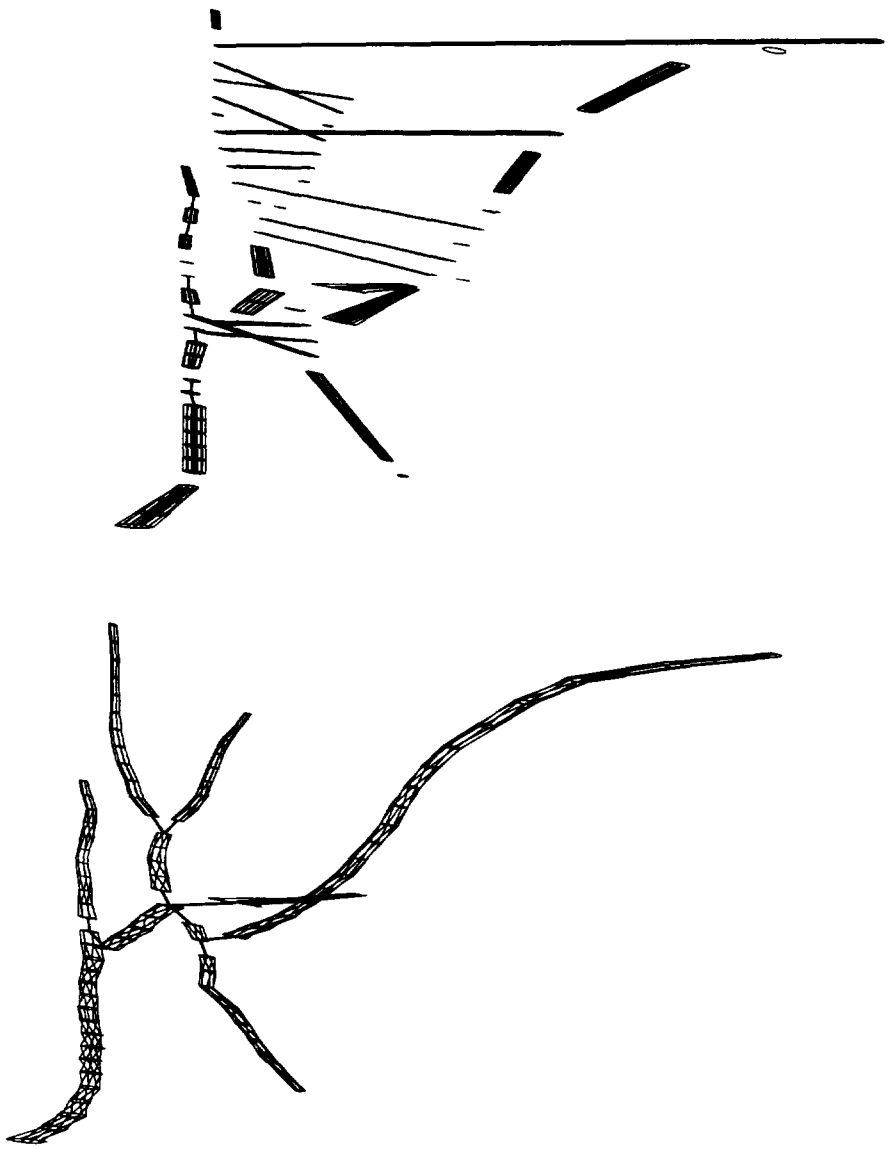

Fig. 8. Biological structures obturator artery contours. **Upper:** Elliptical cylinders found using the elliptical cylinder-growing algorithm are indicated by lines surrounding groups of adjacent contours. **Lower:** Segments found using the minimum spanning tree correspondence algorithm. The final solution is almost identical with the actual correspondence determined at the time the data were collected.

Postprocessing methods can be used to eliminate or add edges as required to alleviate these problems.

3.2 Branching Problem

3.2.1 *Composite Contour Construction.* Our first attack on the branching problem is to construct a composite contour from the set of contours that define the branches, following Christiansen and Sederberg [5]. The composite contour is treated as a single contour and tiled with the corresponding prebranch contour from an adjacent section.

The top portion of Figure 3 shows a set of contours in which the construction of a composite is straightforward; the adjacent contours approach closely at a single point on each. In this case, we construct a composite contour that connects the adjacent contours at the close points. This composite contour is then tiled with the single contour from the adjacent section.

The lower portion of Figure 3 shows the same set of contours merged by the method of [5]. A vertex is added between the contours of the upper section. This has the effect of improving the piecewise planar approximation to the surface formed by the triangular tiling.

The difference between the composite generated by [5] and the one we generate is the added vertex between the pair of contours. Without more global knowledge of the surface from which the contours were obtained, it is difficult to know where this vertex should be inserted. Faced with this difficulty, we choose to avoid insertion of non-data manufactured vertices. We take the point of view that the mesh produced by the tiling algorithm is not the final surface. A subsequent surface-fitting step will determine where the saddle between the contours should fall. An added point forces the saddle to fall in a particular place, and it unnecessarily constrains the final surface fitter. Added vertices suffer from an additional problem: the original data points may have associated properties other than position (such as surface normal) that influence the subsequent surface-fitting step. These properties may be difficult to generate for the manufactured data points.

Figure 4 shows an example in which the construction of a composite contour is more difficult. We define a *canyon* to be an extended region of close adjacency between two contours, such as in Figure 4. If adjacent contours that form a canyon are linked into a composite at a single point, the resultant tiling is not often a good representation of a likely surface. In many cases, some of the tiles constructed will intersect. In cases such as Figure 4, we form composite contours by linking the contours across the openings into the canyon between them. This approach was suggested by [5]. One problem that remains is that of determining where the cross-links between contours should be. Christiansen and Sederberg [5] required user intervention to solve this problem. Our approach to this problem will be discussed in Section 3.2.2.

In the preceding discussion, we have focused on cases in which two contours merge to form one. Our method is more general than this, i.e., handling the general case in which m contours in one section merge into n contours in an adjacent section. If branching is sufficiently complex, an object

may yield a set of contours in which the branch involves multiple contours from each of the sections involved. In cases such as this, our method produces composite contours for both sections involved and then uses the standard tiling algorithm to construct the tiling between them.

3.2.2 *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.

As shown in Figure 9, 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. A more general approach is needed. Since canyons are simple polygons, the most straightforward approach is to use an algorithm that triangulates the interior of a simple polygon.

Triangulation of a simple polygon has been extensively studied, and efficient algorithms exist [3, 10, 13]. The advantages of general triangulation algorithms are their efficiency and that they handle all simple polygons. Their disadvantage is the difficulty of controlling the nature of the triangulation, particularly with respect to favoring the use of cross-canyon tiles.

Most simple polygon triangulation methods produce many high-aspect-ratio triangles; for example see [10]. Such triangles are not desirable if the triangulation is used to define 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 in some sense "nearest neighbors" on the surface. The Delaunay triangulation of a set of points in the plane is a triangulation that has this property [22]. 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 use a simple method for creating an initial triangulation of a polygon and then modify it in a later optimization step.

The method we use for creating the initial triangulation, though not optimally efficient, is easily implemented. The initial polygon is triangulated by examining three adjacent points on its perimeter. If the two distal points can be connected to form a triangle interior to the polygon, containing none of the vertices of the polygon, the connection is made, and the original polygon is split into a triangle and another polygon, containing one fewer vertices than the original. This process is repeated until the remaining polygon is a triangle. Although this algorithm is best-case $\mathcal{O}(n^2)$ and worst-case $\mathcal{O}(n^3)$, performance has been acceptable for the size polygons we normally encounter. If performance becomes a problem with this step, a more efficient algorithm could easily replace this one. A candidate algorithm that also addresses the issue of constraints on the final triangulation is given in [4].

Given an initial triangulation, we improve it by examining cases in which the edge shared by two triangles could be switched so that it connects the two

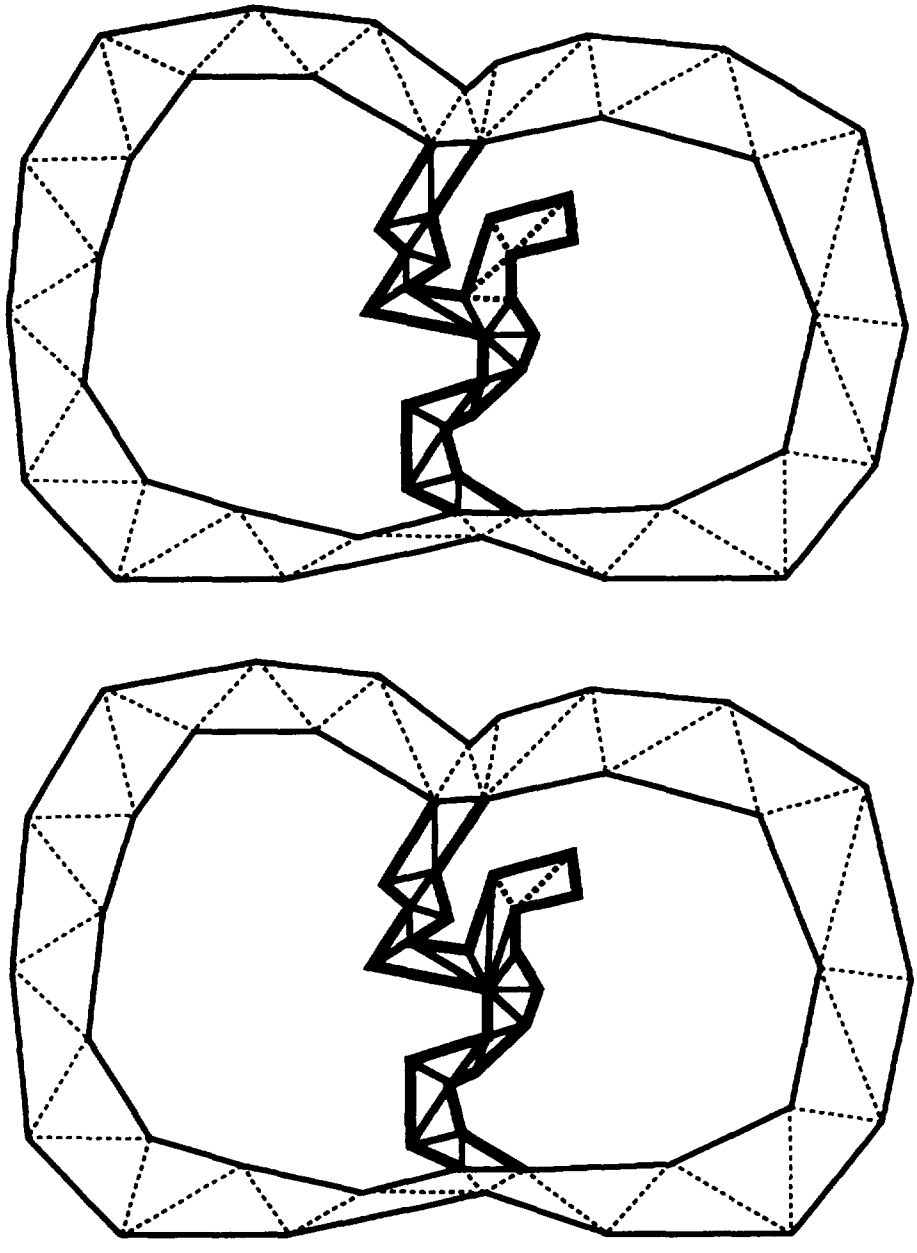

Fig. 9. It is not always possible to construct a tiling in the plane using only *Spans* and *Contour Segments*. In such cases a general-purpose polygon triangulation algorithm must be used. Two possible tilings are shown. **Upper:** A tiling that seeks to minimize the sum of span lengths. **Lower:** A tiling that seeks to minimize the number of intracontour spans.

vertices opposite the edge. This can be done only when the vertices of the triangles that share the edge under consideration form a convex quadrilateral. Swapping the edge in other cases results in a pair of nondisjoint 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 inscribed circles defined by the pair of triangles. This is the criterion used in computing the Delaunay triangulation. Our algorithm proceeds as follows:

(1) Identify all edges shared by two triangles that form a convex quadrilateral. Place these edges on a "suspects list."
(2) While the suspects list is not empty,
    (a) Remove the old edge from the suspect list.
    (b) If swapping the edge reduces the minimum radius of the circumscribed circles defined by the pair of triangles, swap and place each of the four nonshared edges of the two new triangles on the suspect list (if it is shared by a pair of triangles forming a convex quadrilateral).

The method of optimizing an initial triangulation according to a given metric also can 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 control the characteristics of the triangulation computed by general polygon triangulation algorithms, although at some loss of efficiency. We have described the use of one such criterion: minimize circumscribed circle radius. More control over the characteristics of the final triangulation could be obtained by designing a more complex evaluation function. For example, one could favor cross-canyon tiles by modifying the minimization of circumcircle radius test to consider the radius computed for cross-canyon tiles to be smaller than it actually is. Figure 10 shows a canyon with its initial tiling, the tiling resulting from minimization of circumscribed circle radius and that obtained by favoring cross-canyon tiles.

In the preceding discussion of canyon tiling, we have assumed that the boundaries defining the ends of the canyons are known. In general, it is difficult to determine these boundaries automatically. We now discuss a method that works reasonably well in many cases.

The first step is to construct the convex hull of the vertices of the contours defining the canyon (C1 and C2). A key observation is that the convex hull will consist entirely of points from either C1 or C2 but not both, or it will contain two sequences of points, one entirely from C1 and the other entirely from C2. In the second case, there exist two and only two edges in the convex hull that connect a point from C1 to a point from C2. We choose these as candidate canyon boundaries. These boundaries may not be very good. For example, if the contours are two circles, the boundaries chosen will likely be the tangents connecting the two circles, making an undesirably large canyon region. For this reason, we propose improving the boundaries by considering

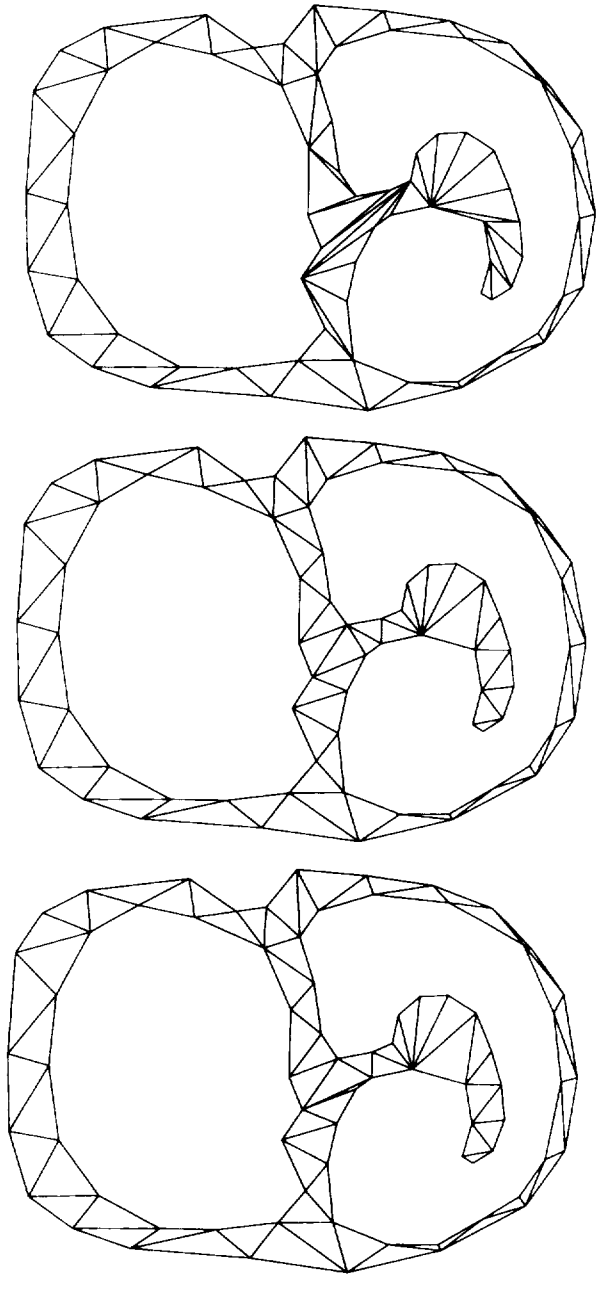

Fig. 10. **Upper:** A canyon polygon shown with an initial triangulation. **Center:** The canyon after optimization using the minimize circumcircle radius metric. **Lower:** The canyon after optimization using a metric which favors cross-canyon tiles by decreasing the circumcircle radius used when comparing with tiles composed of three vertices from the same canyon boundary.

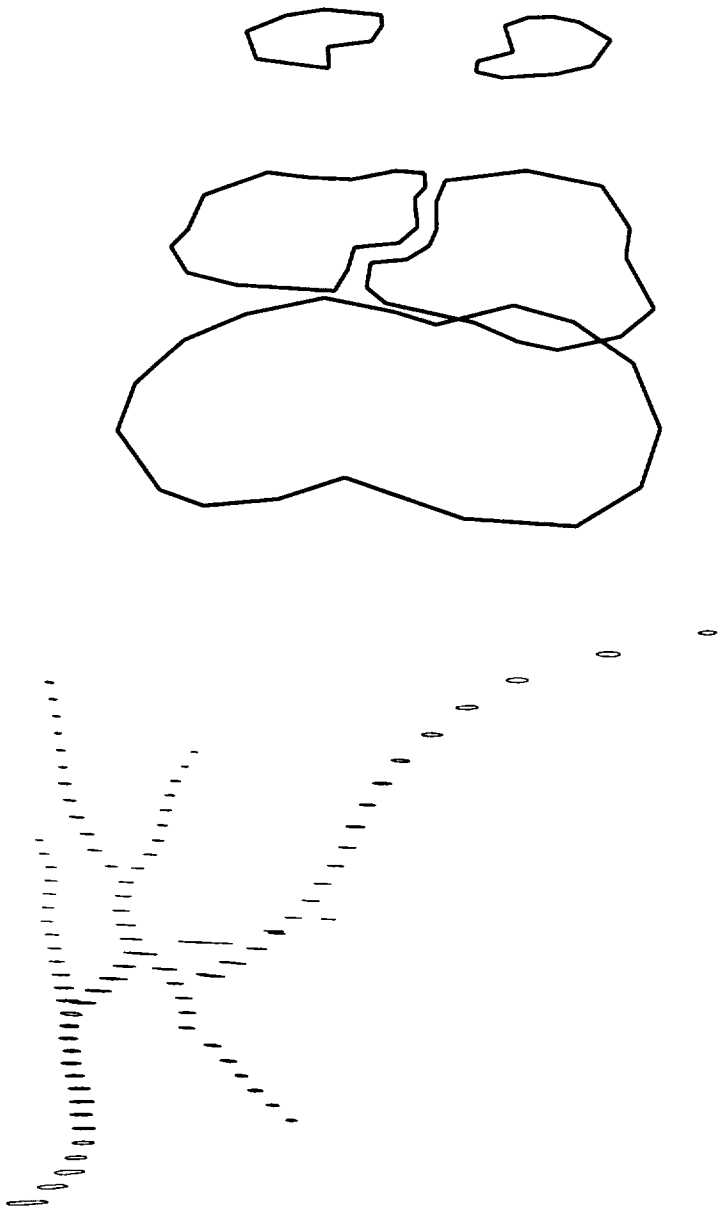

Fig. 11. Two sets of contours. **Upper:** A single large object splits into two smaller objects. The intermediate slice is just above the point of separation, and the two parts are still very close together along a moderately complex border. **Lower:** An artery and several branches. Notice especially the lack of data near the branch point closest to the center of the image.

the corresponding prebranch contour. Consideration of the shape of this contour can be used to improve the candidate canyon boundaries. We are currently investigating possible methods to do this.

3.3 Surface Fitting

As part of a study of the problem of fitting a smooth surface to a triangulated mesh in three-space, we have been involved with the production and use of a flexible software testbed implementing several surface-fitting methods [16, 18]. The contour-fitting system described above produces a mesh suitable for input to this testbed.

The surface-fitting testbed is composed of a number of modules, each of which handles a specific part of the task of producing an image from the input mesh. The modules are:

—A *Fitter*, which produces a stream of Bézier patches from the input mesh.
—A *Tessellator*, which samples Bézier patches, producing a number of planar triangles that approximate each surface patch.
—A *Colorer*, which assigns material properties to the triangles produced by the Tessellator.
—A *Renderer*, which produces a final image from the stream of triangles produced by the Colorer.

The modular structure of this testbed allows easy modification of the properties of the modules involved. Thus, we can use any of a number of implemented surface-fitting methods for the final surface construction step.

Several interpolating as well as approximating methods have been implemented [16, 18]. No one scheme has proven to be perfect; our main point is that the flat, triangular tiling should not necessarily be viewed as the final product. It is useful for previewing, but there are subtle details (such as the precise shape of a saddle surface) that are best left to a global surface fitter. Figure 12 illustrates various choices of surface-fitting and rendering techniques.

4. 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 problems that require new approaches.

Previous methods for solving the correspondence problem have either relied on sufficient data resolution to allow for a solution by overlapping of contours or have used approaches such as elliptical cylinder growing. We have developed a new method for solving the correspondence problem that does not depend on overlapping of contours. This MST method works well for objects with tree topology, but does not recognize connections required for objects with more general topologies, such as a torus. A postprocessing step could be used to remedy this deficiency.

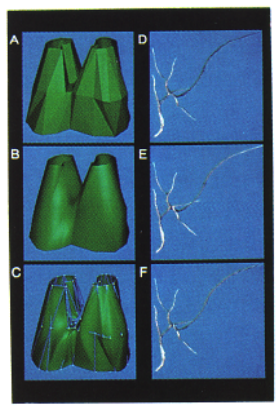

Fig. 12. Reconstructed surface. **Part a:** A flat-shaded rendition of a piecewise planar mesh generated from the data in the upper part of Figure 11. **Part b:** The same mesh, Gouraud shaded to give the illusion of a smooth surface. **Part c:** An interpolating surface, generated by the *side-vertex* method of [15]. The mesh is shown as red balls and blue sticks. **Part d:** A flat-shaded piecewise planar mesh generated from the data in the lower part of Figure 11. **Part e:** The mesh from Part d, Gouraud shaded. Except for the branch point closest to the center of the image, this approximation is adequate. **Part f:** An interpolating surface, generated from the mesh of Part d by the *side-vertex* method of [15].

Previous methods for solving the tiling problem have either not allowed for branching surfaces or have required substantial user interaction in all but the simplest cases. In most of these methods, vertices are added to the data during the reconstruction process. We present a method that can handle complex branching structures without user intervention and that avoids adding vertices to the construction.

Previous work in this area has generally assumed that the triangular mesh computed will be used as the final surface and has often added vertices to the construction for the purpose of improving the quality of this piecewise planar surface. 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. Because we feel that a surface fitter is a better way to improve the final constructed surface, we are careful to avoid introduction of vertices into the mesh we construct, so that the surface computed by the surface-fitting algorithm is not influenced by arbitrary manufactured vertices. We use a surface-fitting testbed that allows easy testing of the suitability of various surface-fitting methods for reconstruction of surfaces from contours.

Our discussion so far has assumed that 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 when handling the tiling of canyons between branching surfaces. If a pair of adjacent contours form a canyon, we triangulate the canyon region assuming that the polygon which defines the canyon is planar. If the data are not planar, we can proceed by finding an average plane for the points involved by a least-squares method, projecting the points onto this plane, and triangulating as before. We expect that 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 3-D data does not fall into this class and will require entirely different methods.

4.1 Future Work

We are interested in extending the spanning tree algorithm to handle objects with general topologies better.

We have developed a method for locating the openings of narrow canyons between branching contours that works reasonably well in many cases. We are extending this method to handle more complicated cases. A related problem arises when two corresponding contours on adjacent sections have very different shape. Recent work on this problem does not handle large differences in shape well. The problem can be seen as a nonbranching canyon-tiling problem. We are extending our canyon-tiling work in this direction.

Our current system cannot handle sets of contours in which one contour is enclosed within another contour with the object interior between them. Such contour sets require a provision for reversing the direction of tiling for one of the contours. We are currently working on extending our system to handle this important class of branching problem.

ACKNOWLEDGMENTS

We thank Tony DeRose for suggesting that a minimum spanning tree might provide a useful construction for solving the correspondence problem. We would also like to thank our colleagues in GRAIL 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 fine existing methods of surface fitting and pick one.

REFERENCES

1. Boissonnat, J.-D. Shape reconstruction from planar cross-sections. *Comput. Vision, Graph. Image Proc. 44*, 1 (1988), 1–29.
2. Bresler, Y., Fessler, J. A., and Macovski, A. A Bayesian approach to reconstruction from incomplete projections of a multiple object 3D domain. *IEEE Trans. Pattern Anal. Mach. Intell. 11*, 8 (Aug. 1989), 840–858.
3. Chazelle, R., and Incerpi, J. Triangulation and shape-complexity. *ACM Trans. Graph. 3*, 2 (Apr. 1984), 135–152.
4. Chew, L. P. Constrained Delaunay triangulations. *Algorithmica 4* (1989), 97–108.
5. Christiansen, H. N., and Sederberg, T. W. Conversion of complex contour line definitions into polygonal element mosaics. *Compt. Graph. 12*, 2 (Aug. 1978), 187–192.
6. Cook, L. T., Cook, P. N., Lee, K. R., Batnitzky, S., Wong, B. Y. S., Fritz, S. L., Ophir, J., Dwyer, S. J., III, Bigongiari, L. R., and Templeton, A. W. An algorithm for volume estimation based on polyhedral approximation. *IEEE Trans. Biomed. Eng. BME-27*, 9 (Sept. 1980), 493–500.
7. Duda, R. O., and Hart, P. E. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
8. Ekoule, A. B., Peyrin, F. C., and Odet, C. L. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Trans. Graph. 10*, 2 (Apr. 1991), 182–199.
9. Farin, G. *Curves and Surfaces For Computer Aided Geometric Design: A Practical Guide*. 2d ed. Academic Press, New York, 1990.
10. Fournier, A., and Montuno, D. Y. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph. 3*, 2 (Apr. 1984), 153–174.
11. Fuchs, H., Kedem, Z. M., and Uselton, S. P. Optimal surface reconstruction from planar contours. *Commun. ACM 20*, 10 (Oct. 1977), 693–702.
12. Ganapathy, S., and Dennehy, T. G. A new general triangulation method for planar contours. *Comput. Graph. 16*, 3 (July 1982), 69–75.
13. Garey, M. R., Johnson, D. S., Preparata, F. P., and Tarjan, R. E. Triangulating a simple polygon. *Inf. Proc. Lett. 7*, 4 (June 1978), 175–179.

14. KEPPEL, E. Approximating complex surfaces by triangulation of contour lines. *IBM J. Res. Dev. 19* (Jan. 1975), 2–11.
15. LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface reconstruction algorithm. *Comput. Graph. 21*, 4 (July 1987), 163–169.
16. LOUNSBERY, M., LOOP, C., MANN, S., MEYERS, D., PAINTER, J., DEROSE, T., AND SLOAN, K. A testbed for the comparison of parametric surface methods. In *SPIE/SPSE Symposium on Electronic Imaging Science and Technology* (Santa Clara, Calif., Feb.). 1990.
17. MANN, S., LOOP, C., LOUNSBERY, M., MEYERS, D., PAINTER, J., DEROSE, T., AND SLOAN, K. A comparison of methods for parametric surface interpolation. In *SIAM Conference on Geometric Design* (Tempe, Ariz., Nov.). 1989.
18. MANN, S., LOOP, C., LOUNSBERY, M., MEYERS, D., PAINTER, J., DEROSE, T., AND SLOAN, K. A survey of parametric scattered data fitting using triangular interpolants. In *Curve and Surface Modeling*, Hans Hagen, Ed. SIAM Philadelphia, P. To be published.
19. MILLER, J. V., BREEN, D. E., LORENSEN, W. E., O'BARA, R. M., AND WOZNY, M. J. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Comput. Graph. 25*, 4 (1991), 217–226.
20. NIELSON, G. M. A transfinite, visually continuous, triangular interpolant. In *Geometric Modelling: Algorithms and New Trends*, G. Farin, Ed., SIAM, Philadelphia, Pa., 1987, 235–246.
21. O'ROURKE, J., AND SUBRAMANIAN, V. On reconstructing polyhedra from parallel slices. Tech. Rep. TR 008, Dept. of Computer Science, Smith College, Northampton, Mass., 1991.
22. PREPARATA, F. P., AND SHAMOS, MI. *Computational Geometry*. Springer-Verlag, New York, 1985, pp. 203–220.
23. SLOAN, K. R., JR. Analysis of "dot product space" shape descriptions. Tech. Rep. 74, Dept. of Computer Science, Univ. of Rochester, 1980.
24. SLOAN, K. R., JR., AND HRECHANYK, L. M. Surface reconstruction from sparse data. In *IEEE Conference on Pattern Recognition and Image Processing* (Aug). 1981, IEEE, New York, pp. 45–48.
25. SLOAN, K. R., JR., AND PAINTER, J. From contours to surfaces: Testbed and initial results. In *Processing of CHI + GI'87* (Toronto, Canada, Apr.). 1987, pp. 115–120.
26. SLOAN, K. R., JR., AND PAINTER, J. Pessimal guesses may be optimal: A counterintuitive search result. *IEEE Trans. Pattern Anal. Mach. Intell. 10*, 6 (Nov. 1988), 949–955.
27. SOROKA, B. I. Generalized cones from serial sections. *Comput. Graph. Image Proc. 15* (1981), 154–166.
28. WANG, Y. F., AND AGGARWAL, J. K. Construction of surface representation from 3-D volumetric scene description. In *Processing of Computer Vision and Pattern Recognition* (San Francisco, Calif., June). 1985.

Received June 1990; revised October 1991; accepted January 1992

Editor: G. Nielson