1 Contour Methods

2 Introduction

In a variety of applications the data is available as a series of contours which represent the cross section of a surface when sliced by the planes the contours reside in. The surface construction algorithm determines the surface using the contours. This chapter examines the existing algorithms and presents a new algorithm that has none of the short comings of the existing algorithms. In Section 3 the simple triangulation methods are reviewed. These methods only cope with cases where only one contour exists on each slice. The more general case of many contours on each slice requires more complex algorithms which are the subject of the review of Section 4. Section 5 shows how shapes (cones) can be fitted to the contours to approximate them and aid the reconstruction process. Section 6 introduces the more general problem of reconstruction from unorganised points, and finally remarks on these sections are given in Section 7, along with the identification of major problems. A new algorithm for surface reconstruction is presented in Section 8 that copes with the cases that create difficulties for other algorithms. The context of the problem is given in Section 9 and the approach is described in Sections 10 and 11. The algorithm is tested on classical and real problems in Section 12, and the results are discussed in Section 13. Finally the chapter is concluded in Section 14.

3 Simple Triangulation

The original solution to the construction of surfaces from contour data is due to Keppel [1]. For this method the form the data takes is constrained to be an anti-clockwise sequence of points on the contour. Each contour is a cross section of constant z with the points having positions which are x, y values. Given this definition of the contour, the convexity and concavity of any three successive points can be determined easily by drawing a circle with an arc passing through the points. If the arc has positive curvature it is anti-clockwise and convex, or if it has negative curvature it is clockwise and concave. Using this method the concave and convex regions of the contour can be determined, and treated separately during the tiling process.

Keppel points out that the tiling process must be guided somehow since it is impossible to ennumerate every triangulation of successive contours. For two contours of 12 points each there are around 10^7 triangulation combinations. In order to solve this problem, the contours are represented by a graph (Figure 1), where each vertex in the graph represents a possible span, and each edge is a triangular tile. Any tiling of the two contours is a path through the graph, and the tiling chosen is one such that the path is a minimal cost path measured with respect to some cost metric. A cost is associated with each edge in the graph which when totalled over the path is an indication of how good the surface is with respect to the cost metric.

The cost metric Keppel chooses to produce an optimal surface is known as *maximise volume*. The idea is that each edge in the graph will represent the volume that triangle will add to the surface by treating it as a face of a tetrahedron. The problem of concavities is solved by trying to minimise the volume of the surface in those regions.

This method was later used for a practical application by Fuchs et al. [2], where the graph problem was studied and solved using a divide and conquer algorithm rather than the heuris-

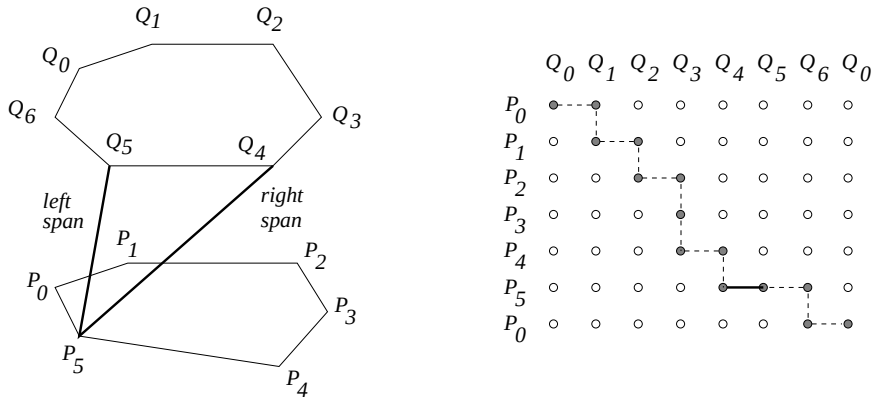

Figure 1: (a) Two adjacent contours. (b) Graph representing contours.

tics of Keppel. They show that the surface can be represented by paths in the graph, and that they can be found by successively subdividing the graph in order to minimise the search space. The measure that they try to minimise is the triangular facet area with additional measures to prevent the twisting of the surface.

Sloan and Painter [3] review the graph solving problem and compare common techniques. They also give a detailed description of a new technique in which they select pessimistic paths in the graph search which they show leads to a quicker solution, as opposed to selecting paths that minimise the search space. They explain that this is because that subdividing unequally results in subgraphs that will take approximately the same time to search, since the larger can take advantage of *bottlenecks* to remove nodes from the search, leaving less nodes for the more complex search for the optimal solution in the smaller subgraph.

The method of Fuchs et al. was also used by Cook et al. [4] with application to measuring volumes of surfaces represented by contour data. Specifically, they were interested in measuring the volume of human organs such as the liver.

Ganapathy and Dennehy [5] use heuristics[1] to triangulate between contours. They use a method of normalising the contour perimeter to 1. The next tile is chosen so that the difference in value of the distance traversed along the top contour with that of the lower contour is minimised. This measure has the advantages that it can be calculated cumulatively using local information and that previous contour spans contribute to the current calculation. It also has the advantage of requiring only $M + N$ steps to calculate the tiling. This is a result of choosing the tiling according to local information, and not minimising some measure of the complete tiling.

Heuristics are also used by Cook et al. [6]. They map the contours onto star-convex regions by assuming contours change direction slowly and smoothly, and have no sharp projections. The centroid of the upper region is determined, and the star-convex region for the lower contour is defined such that every line connecting each point of the lower contour to the centroid of the upper region passes through the region. A tiling between the two mapped contours is determined using heuristics which are based on the dot product of the vectors joining the centroids and the points to be tiled. Once the tiling process is complete, the mapped contours are returned to their original configuration retaining the tiling connections.

[1] Optimal methods solve the problem using graph methods to find a near optimal solution with respect to some measure.

Heuristic methods solve the problem using local information only as a basis for choosing the tiling.

The mesh produced is then used for the display and measurement of the area and volume of the reconstructed object.

4 Correspondence and Branching Problems

The main problem associated with this basic tiling method is the fact that the algorithm is restricted to the generation of surfaces from contour data which have just one contour for each value of z. Any object which is branching or coalescing in nature cannot be tiled. More specifically the problems encountered are referred to as the correspondence and branching problems, and are stated as:

- *Correspondence problem* – given two slices with arbitrary numbers of contours within them, which contours from one slice correspond to which contours on the other slice? Without establishing the correspondence between the contours, the tiling algorithm cannot be used.
- *Branching problem* – given two slices where M contours correspond to N contours $(M, N > 0)$, how should the tiling process be carried out?

Shantz [7] dealt with the many to many branching problem by connecting contours within the slice plane using as few as possible minimum distance connections. This has the effect of creating a single contour on each slice, which represents all the contours on that slice. The tiling then takes place using the method of Fuchs et al. [2] which can handle the case of mapping just one contour to one. The problem with this method is that it can only handle contours that are very similar in shape in adjacent slices. It can only handle multiple contours on a slice if they approach each other at one point rather than having a complex interface (Figure 2).

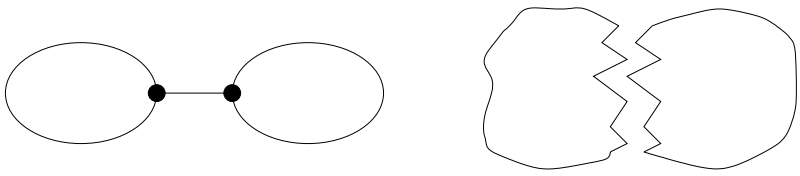

Figure 2: Approach at (a) A single point (b) A complex interface.

Another solution to these problems was proposed by Christiansen and Sederberg [8] in which they determine corresponding contours as those that overlap in any way. Once a correspondence between contours has been determined, contours in successive slices are bounded by boxes and mapped onto a unit square. The reasoning behind this is that successful contour triangulation is more likely when successive contours are mutually centred and similar in size and shape. Triangulation takes place on these contours using the very simple method of selecting the next triangle with the *shortest diagonal*, thus removing the need to resort to costly graph searches for the reasons mentioned above in the method proposed by Ganapathy and Dennehy. Branching is coped with by introducing a new vertex between the closest point on the two branches and combining them along with this point to make one contour. Tiling takes place on these two normalised contours and is then mapped back to the original vertex positions. This method copes well with simple branching but has difficulty when branching contours have a complex interface, which is more often than not the case. In these cases it is suggested the user should manually triangulate the path between the two branches.

Boissonnat [9] creates a surface from contours by projecting the Voronoi diagram of two neighbouring contours onto a plane parallel to the planes of the contours. The resulting 2D graph is used to create the 3D Delaunay triangulation between the two slices. He proves that this is possible for such a restriction on the points (the points occur in two 2D planes and are connected in each plane in the form of contours) and that the method for doing so is optimal. The resulting triangulation is a solid, of which the external faces make up the surface of reconstruction.

This method is very successful in the difficult situations where there are multiple contours with differing numbers upon successive slices. The method fails when adjacent slices differ too much, in addition to which the circumstances for failure cannot be identified. In such cases more slices must be taken through the object. The images presented in the paper show the method working for lungs and a heart, although they are not of a high fidelity.

Ekoule et al. [10] propose an algorithm which handles arbitrarily shaped contours, with multiple contours on each slice plane allowed. For their contour triangulation method they use a simple minimum edge length heuristic which is similar to Christiansen and Sederberg's [8] shortest diagonal heuristic. They divide the contour into convex and concave regions in a similar way to Keppel [1] and create a representative tree structure. Each contour is mapped onto its convex hull with vertices in the concavities mapped using Euclidean distance to retain the relative distances between successive points. The tiling then takes place between the two convex contours trivially, producing a satisfactory triangulation. This triangulation is then retained as the vertices are mapped back to their original positions, thus providing a triangulation of the original complex contours.

The case of multiple branching (one contour to many) is handled by creating a new intermediate contour between the two adjacent slices to be tiled. Each slice is then tiled with this contour to produce the overall tiling. The intermediate contour is produced by creating one closed polygon which is composed of points of the many contours clipped against a polygon joining the centroids of the contours. This polygon is tiled with the single contour and vertices are moved halfway along the spans to create the intermediate contour. The single contour is trivially tiled to this, while each individual contour on the slice with many contours is tiled with its corresponding contour part in the intermediate contour. The centre part of the intermediate contour is then tiled horizontally using existing vertices and the process is complete.

In order to cope with many to many branching they determine which contours should link with which using a *superposition degree* estimate. Those that should link are then tiled together.

This method, whilst dealing with moderately complex cases, will have difficulty when adjacent contours do not have similar convex hulls. In addition to this the tiling is not successful in some cases of many to many branching. The problem occurs when the links do not form disjoint sets. For example (Figure 3(a)) some of the links are $a \rightarrow c + d, b \rightarrow d + e$ and $d \rightarrow a + b$. In this case the available contours are insufficient for their tiling method, they require the contours of Figure 3(b), which have links $c \rightarrow a + b$ and $c \rightarrow d + e + f$.

Giersten et al. [11] use user interaction to determine the connectivity between contours of adjacent slices. During the contouring process the previous slice is overlaid on the current slice in semi-transparent form and the user specifies successive contours. The contour slices are automatically aligned by using the centroid information provided by specifying corresponding contours. The slices are rotated and translated to achieve the correct alignment and overlap, and scaled to compensate for objects which may have been compressed as the result of the slicing process. They then proceed to reconstruct the data on an object by object basis. For

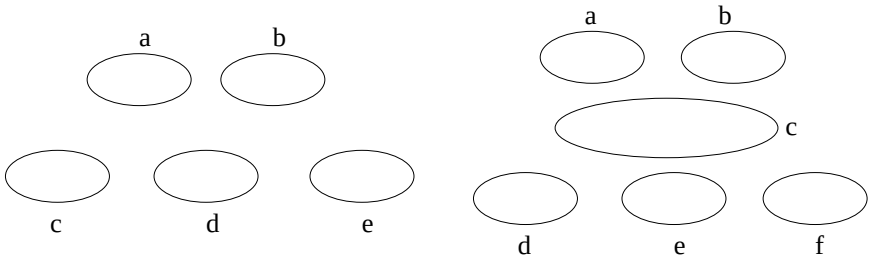

Figure 3: (a) Insufficient data (b) Sufficient data.

each object they have a subgraph which indicates the corresponding contours on each slice, and their connectivity. Where a contour branches, or contours coalesce, they use a simple contour branching scheme based on that of Christiansen and Sederberg [8], and as such can only handle simple one to many cases. In addition to reconstruction they also apply 3D filtering techniques in order to produce smooth object surfaces.

5 Cones

Soroka [12] investigates the use of Generalised Cones (GCs) when applied to the reconstruction of objects from contour data. The three components of the GC are the *spine*, *cross section* and a *sweeping rule*. Soroka uses elliptical cones (ECs) which are described using elliptical cross sections, and by allowing the minor and major axis of the ellipse to vary linearly and independently over time. Each contour is determined to be either elliptical or complex. Complex contours are discarded and play no other part in surface reconstruction. On finding an elliptical contour the algorithm tries three rules in order, firing the first rule that applies. First it tries to extend an existing EC by adding the elliptical region to the end. Secondly it tries to create a new cone from the newly found single elliptical region. Thirdly it splits off a subregion of the cone in order to grow the cone into a complex region. Essentially this process attempts to model the general topological and geometrical information provided in the data, but suffers from a few problems.

The method can only model simple convex objects. Any contours or parts of the object regarded as complex are omitted from the final reconstruction. The practical application that Soroka presents is that of the reconstruction of a canine heart. The resulting model has the right ventricle missing because the contours comprising that object are classed as complex in that region. Another problem is that fact that the ECs cover the object, and not partition it. This is due to the fact that regions may be shared by several cones, and the result is that it would be impossible to use this geometric description to perform any quantitative analysis about the volume of the reconstructed object.

Myers et al. [13] present progress on the correspondence and branching problems. They use a descriptive language to indicate the contours present on each slice and the connection to contours on adjacent slices. This connectivity information is the result of an analysis of contour correspondence which is performed either manually or automatically. Two methods are suggested for automatically determining correspondence, namely the minimum cost spanning tree (MST) method and an algorithm using elliptical cylinders (ECs).

The method they use for ECs differs from that of Soroka [12] in that complex contours are not ignored. Rather they are approximated by a subcontour that is elliptical, and which is then treated in the same way as the other ellipses, either to extend a cylinder, or to create

a new singleton cylinder. The goal of this process is to produce a small number of cylinders containing as many ECs as possible.

The next stage is to link cylinders together to create objects. They establish that cylinders can be linked in three ways. The cylinders A and B could link if one end of cylinder A joins to one end of cylinder B. The cylinder A could link to B and C if one end of cylinder A joins to the ends of cylinders B and C. Lastly the cylinder A could link to B if the end of cylinder A joins to an interior contour of cylinder B. In all cases the cylinder can be added to an existing object. If a cylinder does not connect with any existing object it is used to create a new object.

Their method also differs from Soroka's in that they use each contour only once when creating a cylinder. This results in the cylinders being a disjoint representation of the surface rather than a cover of the surface as was the case with Soroka's method. This restriction, while avoiding the problems of Soroka's method, creates problems of its own, in that the result is order dependent. The order in which the ellipses are considered has a great bearing on the cylinders produced since a valid cylinder can be created by an ellipse being added to any singleton cylinder, when perhaps it would be better used elsewhere. They suggest examining all cylinders that could be created from three sections and keeping only those that extend to a length of three.

They also suggest a method using an MST. Firstly an ellipse is fitted to every contour. A graph is created using the contours as nodes, and a four dimensional value (x, y, A, B) is associated with each node, where (x, y) is the position of the contour, and A and B are the major and minor axes. All edges (i, j) are added where contours i and j lie on adjacent sections, and as such could possibly correspond. The cost for each edge (i, j) is calculated as the Euclidean distance between the nodes. The result is that the more similar contours are in position and orientation, the closer to zero the cost is. A MST is then computed, and if edge (i, j) exists in the spanning tree, contours c(i) and c(j) are expected to correspond. This tree contains interobject connections which are removed by traversing paths through the tree from unvisited nodes. After this stage the contour connectivity and object information is available in the form of that produced by the EC growing method. The problem with the MST method is that it does not find all the connectivity information because a tree rather than a graph is used. Also because connections are rejected on some criterion, rather than created, the tree may be left with edges that result in incorrect joins of disjoint objects.

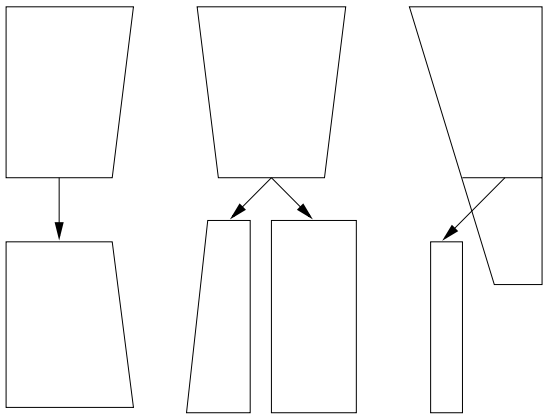

Figure 4: Branching – (a) One to one (b) Two to one (c) Internal.

From this model the branching structure of the object can be classified. In their method they

only allow bifurcations. The three types of branching are – two cylinders are connected (Figure 4(a)), one cylinder branches (Figure 4(b)) and one cylinder interconnects (Figure 4(c)).

Where branching is involved they divide the possible cases into two – those that involve contours which have a simple interface (Figure 2(a)), and those that have a complex interface (canyon) (Figure 2(b)). The simple case can be tiled using the method of Christiansen and Sederberg [8]. The more complex case is solved by triangulating the polygon representing the canyon. The two contours are then connected and tiled using the one to one method. The canyon is found by finding the convex hull of the combined contours. This results in a list of points where the two edges from a point on one contour to a point on the second contour are the edges that close the canyon polygon.

The techniques they describe are quite elegant (MST) and seem to work very well for many cases that other methods fail on. The main problem with this method is that although they indicate it can handle cases where many contours correspond to and branch into many, they give no examples. The canyon tiling for such arbitrary cases would become complex, assuming the correct connectivity could be found. Their method also suffers from the problem that it cannot handle major shape changes between slices, which is usual for these methods. Finally it cannot handle cases where one contour is entirely contained in another contour defining some hole within the object.

6 Reconstruction from Unorganised Points

Surface reconstruction from contour slices can be regarded as a particular case of the more general problem of reconstruction from disparate points. The general problem can be considered to be the reconstruction of the surface from a finite set of points in space. The restricted problem of surface reconstruction from contour slices is the reconstruction of a surface from a finite set of points where the points occur in a finite number of planes and are connected in some way.

The restricted problem arises when the data has been collected in a slice-by-slice manner, either automatically or manually. The general problem occurs when the data is collected by some object scanner or sensors (eg. cyberware scanners). This problem occurs during applications of reverse engineering – the automatic generation of CAD models from laser range data.

The problem has been approached by Boissonnat [14], using Delaunay triangulation. His first method works well for smooth objects with only minor variations in curvature. A point and its k-closest neighbours are projected onto a plane and the 2D Delaunay triangulation of the points is constructed. The triangulation is retained when the points are mapped back to their original configuration in order to give the surface triangulation. The second method, which works for a wider class of objects, creates the 3D Delaunay triangulation for the points. This triangulation consists of a tetrahedral description of the convex hull of the object. If the object is not equivalent to the convex hull, tetrahedra must be eliminated from the polyhedron until all the original sampled points are on the boundary. At this point the external faces of the tetrahedra is the triangular mesh description of the object.

Edelsbrunner and Mücke [15] also use 3D Delaunay triangulation to create the mesh, but have an additional control, α, which controls the level of detail. With $\alpha = \infty$ the mesh produced is identical to the convex hull of the original set of points. As α is decreased, the mesh decreases, until $\alpha = 0$ at which stage only the original points are present. A piece of the

mesh disappears when α becomes small enough so that a sphere with radius α can occupy the space without enclosing any of the original points.

Turk and Levoy [16] produce meshes from $m \times n$ range images by considering each sample in the image to be a candidate vertex in the mesh. Four points in neighbouring rows and columns are selected and checked to see if they create triangles. If the distance between three points is greater than some threshold no triangle is created since the surface may involve an unseen crevice in that area. The meshes from several scans from different angles are joined together in a process they call *zippering*. The final stage to the process is to optimise the mesh (see Section **??** of Chapter **??**).

Most of these methods are computationally expensive – Turk and Levoy [16] state that the process of digitising an object and producing a mesh requires "*five minutes of user interaction and a few hours of compute time*". Boissonnat [9] applied the Delaunay triangulation method to the restricted problem, and took advantage of the restriction to reduce the computational expense. A future project could be to try to apply these methods to the restricted problem in an attempt to both accelerate these methods, and solve the restricted problem.

7 Remarks

From the above review of the various approaches it should be apparent that most strive to solve specific problems. The problem of tiling between two well behaved contours seems to be successfully managed, but by simply increasing the shape complexity of the contours, erroneous tilings start to occur. This problem can be overcome by mapping the contours to shapes that can be well tiled, for example Ganapathy and Dennehy's normalised length, Christiansen and Sederberg's mapping to unit square, and Ekoule et al.'s mapping to convex hull. The mapping of contours to some other shape is carried out so as to preserve the order and relative distances of vertices. The tiling process is carried out on the mapped contours, and the tiling is retained once the contours are returned to their original configurations. This allows a new class of contours to be handled, but the tiling process still requires contours to be similar in shape, and so if their mapped shapes are not similar, the method still does not produce a satisfactory tiling. A good test of any tiling technique is that of a spiral contour in one slice, and a convex contour in the next slice. This is regarded as being such a difficult case that J. Rossignac[2] of IBM research set a competition to produce a surface from such contours. None of the methods mentioned thus far can create a satisfactory surface from such contours, solely because they differ too greatly. Attempts have been made but as Boissonnat reports [9] the usual surfaces produced, intersect themselves.

In addition to the fundamental problem of tiling between two contours most methods have trouble tiling branching contours. The approaches can be divided into those that do not cope with branching, Sloan and Painter's, Fuch et al.'s, Ganapathy and Dennehy's, and Soroka's, those that cope with simple branching (one to many, only when contours approach at a point), Shantz's, Christiansen and Sedeberg's, and Geirsten's, and those that cope with more complex branching (some many to many cases and contours that approach at complex interfaces) Boissonnat's, Ekoule's, and Myers et al.'s. Even the advanced methods do not cope well with moderately complex branching cases, and often require many slices in areas of mild complexity. The problem is compounded if contours have complex shape or branch arbitrarily many to many, and it seems that the more complex the case to be handled is, the more complex the solution is. Boissonnat [9] points out that mostly the inadequate solutions

[2]Personal communication

are a result of the methods not being able to tile horizontally under the contours in order to form bridges.

All the methods reduce the problem to that of determining the surface between two adjacent contours and carry out the tiling using heuristic or optimal methods and in the case of Boissonnat [9] Delaunay triangulation.

In addition to all of these problems, contour methods also usually lack ability to create surfaces with a definite thickness, for example a hollow cylinder as would be produced from the contours of Figure 5.

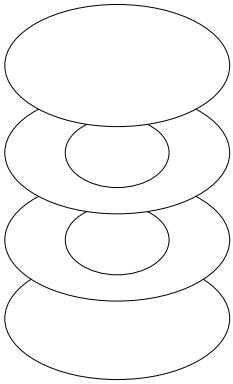

Figure 5: Contours with internal structure.

MacLeod et al. [17] were faced with the task of reconstructing the human thorax from over a hundred MRI slices. They describe the process as time consuming, since after digitisation of the contours, and alignment of the points, the resulting triangulated mesh must be edited by hand. User intervention is required in order to correct any inconsistencies of the mesh, involving a slice by slice check by hand of the whole surface that has been generated by the tiling process. To produce a surface from such data is tedious and can take of the order of one man day to do so.

The fact that so many problems exist for such a tiling method would suggest that an altogether new approach to the problem is required. The new approach should be able to cope with arbitrarily shaped contours with ease and produce *good* surfaces from the contour data quickly and with a minimum of user interaction. It should also be able to derive surfaces from many to many branching contours that is a smooth representation, and gives a good idea of what the original object was. Operation with a minimum of preprocessing to orientate the contours would also be a good requirement.

8 A New Approach to the Construction of Surfaces from Contour Data

In this section a new approach to the construction of surfaces from contour data is introduced. The process neither establishes a correspondence between vertices on one slice with those of the next as all other methods do, nor does it require preprocessing of the contours, or the contours to be similar in shape. The method is shown to be accurate, in that the surface produced, when intersected by planes corresponding to the original contour slices, produces the original contours. It is consistent in that the surface is not dependent upon the order in which the slices are processed, or the ordering of the segments of any of the contours.

It is efficient with regard to computation and ease of implementation. It is also flexible in that it handles arbitrary shaped contours that do not need to be convex, does not require the segments to be organised in such a way that they are placed end to end, joining the last vertex of the last segment to the first vertex of the first segment, and it does not require the segments to be organised such that a point is inside the contour if it is to the right, or outside otherwise. In fact it only requires that the contours be non-self-intersecting and closed, which is far less restrictive than the other methods.

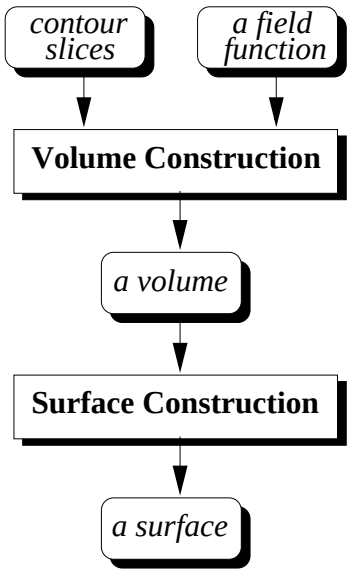

Figure 6: Two phases of the algorithm.

This new algorithm is organised as in Figure 6, which shows the input to the volume construction phase as a stack of contour slices and a field function. The result of this stage is a volume which is then used to create the surface in the surface construction phase. The background which lead to the development of this method is given in Section 9. Details about the input requirements are given in Section 10. Efficient calculation of the field function is the topic of Section 11 and results for some classical and practical problems are given in Section 12. Finally the approach is compared and summarised in the conclusions, Section 13.

9 Background

The impetus behind this research is the requirement for a good, robust method for producing surfaces from contour data. The data involved is that of a set of just 18 MRI slices through a human torso, from which models of the thorax, liver and lungs were required. It was also a requirement that the surface be as accurate and smooth a representation as possible. The MRI data did not contain any sharp delineation of objects, and as such was not suited to other methods of visualisation (Chapters 2 and 3). This *fuzziness* was due to the thickness of the slices. In addition the contours varied greatly between slices due to the fact that there was such a large interval between scans (3cm). As a result of the fuzziness problem, it was best to identify the organs by eye, and outline them using digitised contours. Each slice was displayed in turn, and the contours of interest were outlined using mouse presses to indicate the positions of vertices on the boundary. The fact that the contours varied greatly between slices suggested that standard methods would not work well since this is the case they have

most difficulty with. The need to visualise and perform mensuration upon surfaces from such contours led to the development of a new approach to construct surfaces from contour data.

10 Input Requirements

The input to the *volume construction* phase is a stack of contour slices S_1, S_2, ..., S_{N_z} and a field function $f(x, y)$. The function $f(x, y)$ is defined over all points of a plane in which contours of a slice reside such that

$$f(x,y) \begin{cases} < 0 & \text{if } (x,y) \text{ is outside all contours} \\ = 0 & \text{if } (x,y) \text{ is on a contour} \\ > 0 & \text{if } (x,y) \text{ is inside a contour} \end{cases} \tag{1}$$

In this phase, a two dimensional grid of size $N_X \times N_Y$ is uniformly mapped onto every contour slice. By defining a z-axis in the direction of the contour stack, each grid point can be considered as a voxel in an $N_X \times N_Y \times N_Z$ volume. The value associated with each voxel at location (x, y, z) is defined as $f(x, y)$ computed against all contours residing in plane z. The grid size determines the resolution of the surface to be constructed. The optimal accuracy can be obtained by choosing a fine grid or a grid with unequal intervals such that every contour vertex is located at a grid point. The original contours in each slice can easily be extracted from such a grid by taking all grid points where $f(x, y) = 0$, and linearly interpolating points that are not located at any grid point. This is valid since a contour is composed of a set of line segments, each of which is a linear interpolation between successive vertices of the contour. However, in practice, efficiency in computation and implementation can be achieved without sacrificing much accuracy by using a relatively coarse grid with equal intervals and an appropriate field function. The consideration of choosing a field function is to be investigated in the next section.

In the surface construction phase, the volume representation constructed from the contour slices is regarded as a continuous volume space, where the value of a non-voxel point is a trilinear interpolation of values of its eight neighbouring voxels that make up a cube containing the point. In comparison with the contour-based representation, this allows more general and consistent operations to be carried out. The process of extracting a surface from the volume is simply to identify all the points at which the trivariate function has a value of zero. The surface tiling algorithms of Chapter 2 may be used to construct such a surface in the form of a mesh of triangular facets which can then be rendered. The inbetweening of two successive contour slices becomes a trivial task of interpolating between two successive images in the volume.

11 Efficient Calculation of the Field Function

In the volume construction phase, a field function is computed with each voxel (i.e. grid point) in an $N_X \times N_Y \times N_Z$ volume. One simple field function is a decision function that determines if a point is interior, exterior to or on the contour boundary:

$$f(x,y) = \begin{cases} -1 & \text{if } (x,y,z) \text{ is outside all contours} \\ 0 & \text{if } (x,y,z) \text{ is on a contour} \\ 1 & \text{if } (x,y,z) \text{ is inside a contour} \end{cases} \tag{2}$$

The advantage of this function is that a polygon fill algorithm can be employed to scan grid points in each of the contour slices and determine their topological states in relation to contours. However, the surface produced is highly dependent upon the density of the chosen grid, and moreover, the finer the grid, the more discontinuous the surface in the z direction. This problem is illustrated in Figure 7, which shows two adjacent contour slices (Figure 7(a)) and the cross section of a surface constructed with each of three different grids (Figure 7(b)). Figure 8(a) shows a blocky surface resulting from the use of the function. The *blockiness* exhibited by the surface in Figure 8(a) is a result of the fact that essentially a binary decision has been made at each voxel – either it is inside the surface or not. This leads to a restricted number of possible surface configurations and as such surface normals, leaving the surface with large areas of flatness and sharp edges instead of a smoother continuous surface.

A far better field function is a distance function defined as

$$f(x,y) = \begin{cases} -dist(x,y) & \text{if } (x,y) \text{ is outside all contours} \\ 0 & \text{if } (x,y) \text{ is on a contour} \\ dist(x,y) & \text{if } (x,y) \text{ is inside a contour} \end{cases} \quad (3)$$

where $dist(x,y)$ is the distance from (x,y) to the closest point on the contour, or contours if there are more than one on the slice. This results in a volume that represents a continuous function sampled at discrete grid points and allows a surface to be constructed based on linear interpolation. As shown in Figure 7(c), the shape of the surface is not affected by the fineness of the grid in the same way as the simple field function. In contrast to Figure 8(a), Figure 8(b) shows a much more satisfactory surface produced using the distance field function. It is a much more natural surface than that of the one in Figure 8(a), and more like the surface one would have expected to have produced the contours.

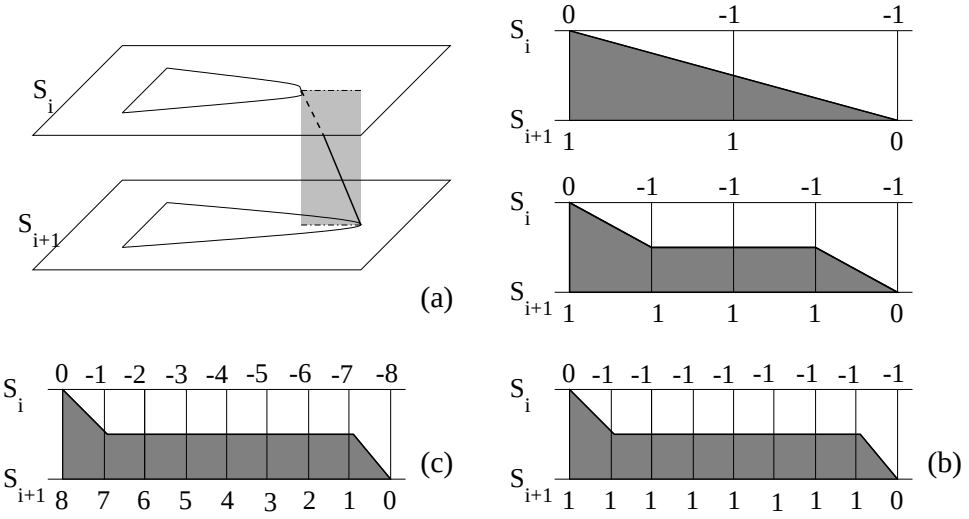

Figure 7: (a) Two adjacent contour slices. (b) Results of the simple field function with three different grids. (c) Result of the distance field function.

The method described above is simple to implement, but implemented naively would result in an algorithm of $O(\sum_{i=1}^{N_Z} N_X \cdot N_Y \cdot N_{C_i})$, where N_Z is the number of slices, N_X and N_Y are the dimensions of the grid and N_{C_i} is the number contour segments in each slice S_i.

The calculation of the distance field would be computationally expensive, as for every grid-point, the distances to every contour segment in the same slice have to be calculated in order

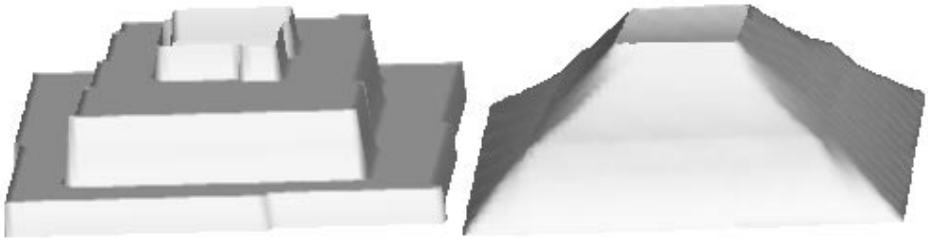

Figure 8: (a) Result of using simple field function. (b) Result of using the distance field function.

to find the minimum. Typically, for a volume data set with $200 \times 200 \times 25$ grid points and an average of 100 contour segments in each slice, 100 million distance calculations need to be performed. Since each distance calculation involves a computationally expensive square root this is obviously prohibitive and the prerequisite that the method should be quick to compute is not fulfilled.

The complexity of the algorithm can be reduced by partitioning each slice S_i into N_{C_i} sectors, each of which contains a contour segment C_i and those grid points closer to C_i than any other segments. The distance field value of each grid point can then be calculated using the field function against only one contour segment. Figure 9 shows an example of partitioning a plane consisting of two simple contours. The partitioning process is almost identical to that for constructing a Voronoi diagram except that bisectors between pairs of contour segments as well as vertices are calculated. The optimal solution to the Voronoi diagram is known as $O(N \cdot logN)$ [18]. Therefore the complexity of the partitioning stage is $O(\sum_{i=1}^{N_Z} N_{C_i} \cdot logN_{C_i})$ and the volume construction phase has a complexity of $O(N_X \times N_Y \times N_Z)$.

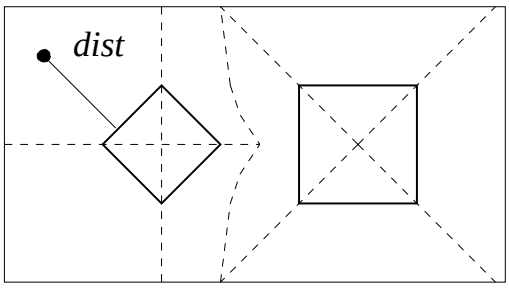

Figure 9: Partitioning a contour slice.

Although the number of segments to be considered for each grid point is reduced to one, the distance function still has to be computed for every grid point in each slice. As the implementation of a partitioning algorithm is far from being trivial, in most applications the actual saving on the cost of implementation and computation is usually very little if there is any.

By simple observation it can be seen that it is not necessary for the value of every grid point to be accurately computed. In the surface construction phase, the algorithm examines each cube, bounded by eight grid points. When it encounters a cube which contains the object interface it performs surface construction by linearly interpolating these grid points to find the polygon vertices of the surface, and also the surface normal at these vertices is calculated.

Voxel values at the eight vertices of the cube need to be known in order to determine the

surface. The 6-neighbours[3] (two in each of the x, y and z directions) of each voxel also need to be known in order to calculate the surface normal. In other words, only the values of voxels near the actual surface are required in order to display the surface. At all other voxels an indication of whether the point is inside or outside the surface is sufficient. This leads to a practical method for reducing the computational cost by restricting the distance computation to those voxels near the surface interface.

Each voxel is associated with two fields, namely state and distance. The state field indicates whether a voxel is interior (1), exterior (-1) or on the surface mesh (0), and is first computed by scan-converting the object. The state function (Equation 4) can be calculated for each voxel (x, y, z).

$$f(x,y,z) = \sum_{k=z-1}^{z+1} \sum_{j=y-1}^{y+1} \sum_{i=x-1}^{x+1} \text{state field of voxel } (i,j,k) \tag{4}$$

For a voxel (x, y, z) there is no need to apply the distance function (Equation 3), if:

- its state field is zero, or
- $f(x, y, z) = 27$ or -27 and $f(x', y', z') = 27$ or -27 for each 6-neighbour (x', y', z').

It is obvious that there is no need to apply the distance function when the state field is zero which indicates the voxel is on the surface. The second condition shows that if a voxel and all its 26-neighbours[4] are all interior (or all exterior) to the surface and all the 26-neighbours of its 6-neighbours have the same state, its distance value will not influence the surface display in any way. The first part states that a voxel need not be known if it is not used during linear interpolation of position, and the second part states that a voxel need not be known if it is not used during determination of the surface normal. This process eliminates many voxels from the expensive distance computation and identifies those voxels, and only those voxels, for which the distance function must be calculated.

Although the pre-processing stage (i.e. scan conversion) still has a worst case complexity of $O(\sum_{i=1}^{N_Z} N_X \cdot N_Y \cdot N_{C_i})$, if the number of voxels on each slice S_i for which the distance must be calculated is R_i, the number of distance calculations is now $\sum_{i=1}^{N_Z} R_i \cdot N_{C_i}$. Usually R_i is less than both X and Y, and is much less if a fine grid is used, therefore this method results in a reduction of an order of magnitude of the number of distance calculations. Furthermore, the complexity of the distance calculation can be reduced to $O(\sum_{i=1}^{N_Z} R_i)$, if combined with the partitioning method. In practise this partitioning stage is difficult to implement and compute. A far easier method is to divide the grid up into sectors, and distribute each contour segment into its appropriate sector. Measuring the distance to a contour then becomes the simple task of finding the distance to the closest contour segment within the sector. Any sectors outside of this distance can be trivially removed from a list of candidate sectors, thus greatly reducing the number of contour segments to compute the distance against, and hence speeding up the process.

[3]The 6 neighbours of a voxel comprise just its face neighbours, when considered as a cube.

[4]The 26-neighbours of a voxel comprise its 6 face neighbours, 12 edge neighbours and 8 vertex neighbours.

12 Results for Classical and Practical Problems

In this section the surfaces constructed for various contour data sets are examined and discussed. The contours have been deliberately chosen such that they show how this approach can cope with contours that other methods cannot. They have also been chosen to show how *good* the surfaces are, that are produced by this approach.

The branching problem has already been shown to be a complex problem, particularly when many contours branch into many. To demonstrate how well this method copes with branching, five cases are presented. Firstly the simple case of one contour branching into two (Figure 10(a)). Secondly, one contour branching into two, and approaching at a complex interface (Figure 10(b)). Thirdly, one contour branching into four (Figure 11(a)). Fourthly, a particularly difficult case of three contours branching into five (Figure 11(b)), and finally one contour branching into two contours and then into three (Figure 12(a)).

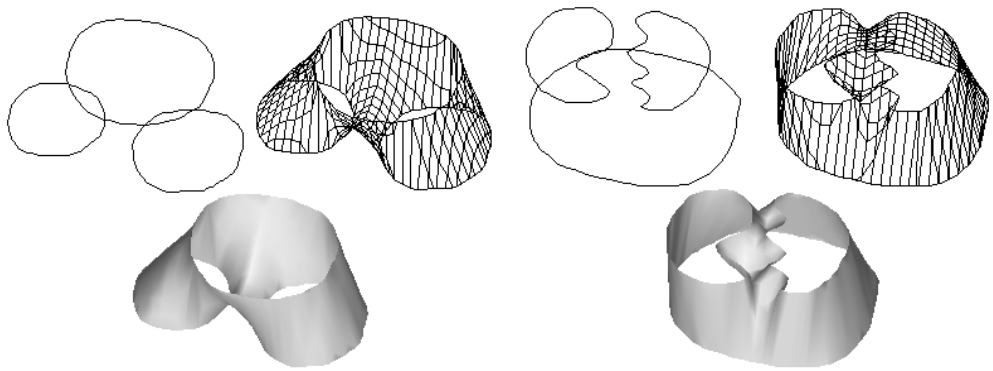

Figure 10: One contour branches into two at a (a) Simple interface. (b) Complex interface.

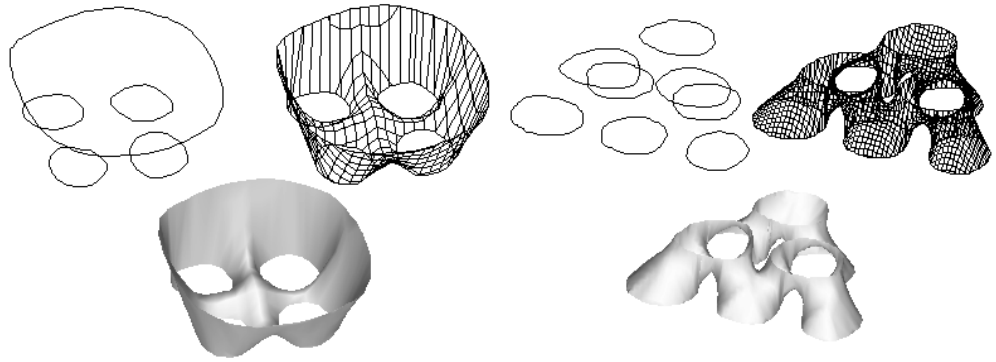

Figure 11: (a) One contour branches into four. (b) Three contours branch into five.

One contour branching into two can be effectively coped with by most tiling algorithms and the example is given to show that the surface is reconstructed just as effectively by this method. A far more difficult problem is that of branching into two contours where the contours approach at a complex interface. Previously user intervention or a complex tiling algorithm [19] would be required, but this method creates a surface without resorting to such techniques. To demonstrate one to many branching, one contour branching into four has been chosen. The resulting surface is a smooth representation of what one would expect the surface to look like. The difficult case of three contours branching into five also results in

a natural looking surface and suggests that this method truly handles arbitrarily many to many branching without the complexity or failings of other methods. The final branching of one contour to two and then three contours has been highlighted to show that a case Ekoule et al.'s method cannot cope with (Section 4), can be solved using this approach. He stated there was insufficient information available to tile the case of two contour branching into three, and that more slices would be needed in the vicinity in order to determine the surface representation. As can be seen, this method requires no such information and produces a very natural looking surface. In fact, by adding the additional contour on top of the other two slices, it is shown that the method correctly tiles around the *hole* that is present in the surface. The tiling is also very smooth and natural looking, which is a result of the distance function. By observation, this method robustly handles difficult many to many contour tiling cases with ease. It requires no extra information to be present with the contours, no user interaction and no tricky special procedures to create the tiling. The data is treated uniformly and consistently in every case in order to produce the surfaces.

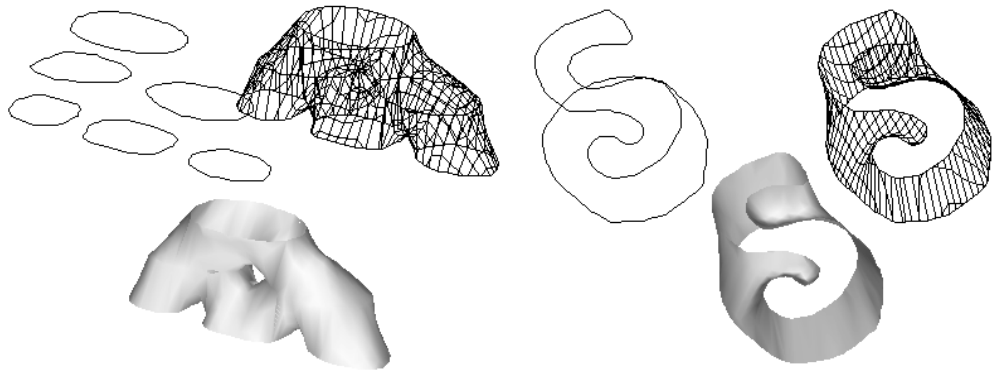

Figure 12: (a) One contour branches into two and then into three. (b) S to a circle.

The problem of greatly differing contours is also a major stumbling block for the tiling methods. To show that this method creates satisfactory surfaces for such situations, a number of cases have been chosen that involve quite complex contours in successive slices. The first is an S-shape corresponding to a circle (Figure 12(b)). Secondly, a thin slice successively becoming a square, a triangle and a circle (Figure 13(a)). Thirdly, a spiral becomes a circle (Figure 13(b)).

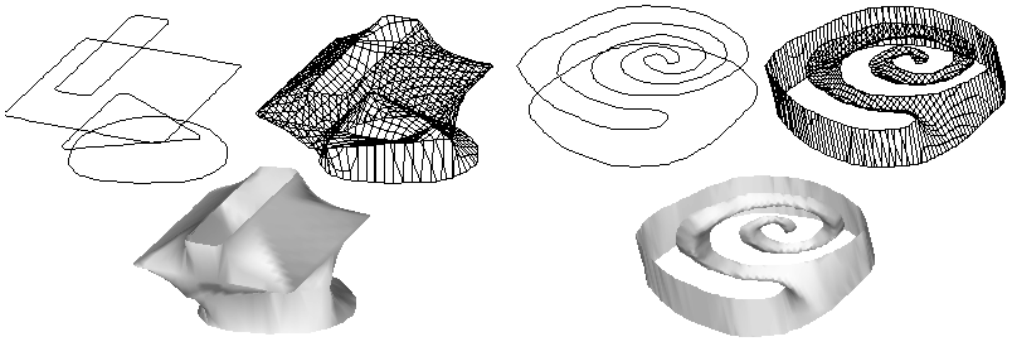

Figure 13: (a) Tube to square to triangle to circle. (b) Spiral becomes a circle.

The first example of an S-shape corresponding to a circle demonstrates that a mildly complex case produces a very reasonable, smooth surface. Such a contour pair would result in

undesirable tilings using most methods. The method of Ekoule et al. [10] would produce the correct surface since both contours would be mapped to their convex hulls, and the tiling would take place between these contours. Problems would arise if the convex hulls of the contours differed greatly, and such a situation is given in the second case. Again the surface produced is convincing as a candidate surface. Finally as stated in Section 4 the mapping of a spiral to a convex contour is an extremely difficult situation, but as can be seen by this particular surface, one that can be handled very well by this method.

The next example seeks to demonstrate how contour correspondence is determined. In the first example two similar contours do not overlap when viewed perpendicular to the plane, and result in two separate objects (Figure 14(a)). In the second case they do overlap very slightly and are treated as one object, namely as a tube (Figure 14(b)). The results may or may not match with a desired shape, but since both surfaces could be valid, it is difficult to say which is correct or not. However, it is commonplace to treat contours that do not overlap as separate objects and those that do as joined objects. This method certainly handles such cases consistently.

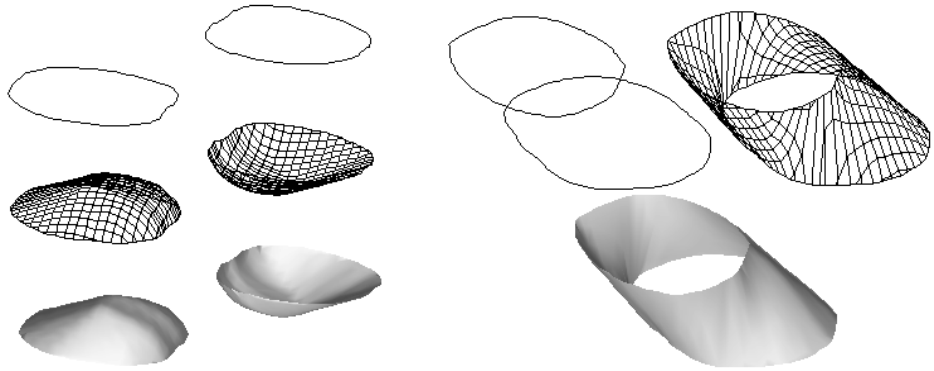

Figure 14: (a) Contours do not overlap. (b) Contours do overlap.

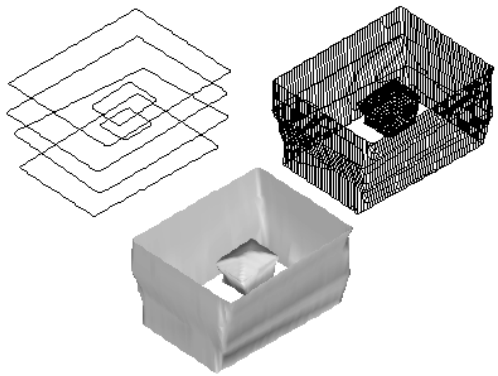

Figure 15: Internal object structure.

The final artificial problem shows a contour that has been constructed interior to another (Figure 15). This contour is present on the two middle slices, but not on the two outer ones. The reconstructed object has an exterior surface made from the joining of the four external contours, and an interior closed object made from the two internal contours. This method differs from the majority of other methods in that it handles such contour situations as contours that are describing the internal construction of an object. Effectively, this method can produce objects that have a determinable *thickness*. The two surfaces represent the

boundary of the object, and are produced from the volume data, having their surface normals calculated from the data. The surface normals do indeed point outwards from the surface, and so for the internal surface correctly point *inward* towards the empty centre of the object.

The practical application chosen is that of reconstructing the torso (Figure 16) and lungs (Figure 17) of a human male from just 17 and 12 contour slices respectively. The slices were obtained using a MRI scanner, and were first described in Section 9. Each contour was obtained by outlining the object within the slice using a mouse. The stack of slices were converted into a volume, and then a surface was reconstructed from them. Considering so few slices were used for the reconstruction the resulting surfaces are fairly detailed and smooth, with notable pectoral muscles, shoulder blades and abdomen on the torso.

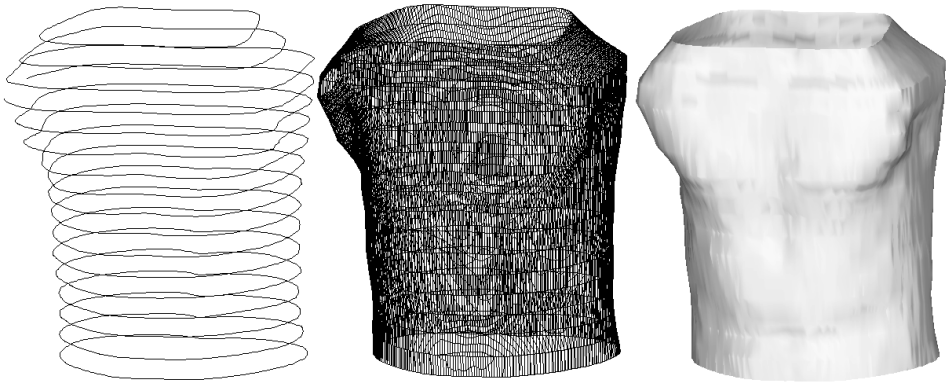

Figure 16: Torso reconstruction from 17 MRI slices.

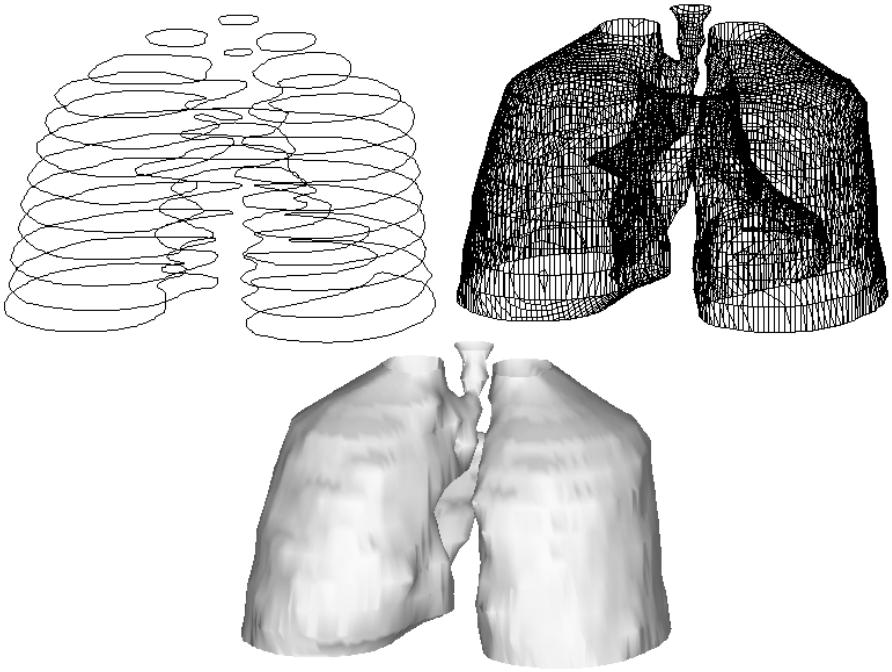

Figure 17: Lung reconstruction from 12 MRI slices.

The results of all this testing on a Dec Alpha model 3000/400 using C are recorded in Table 1, with timings given in seconds.

Fig.	Case	Contour Resolution	No. of Slices N_Z	Grid Size $N_X \times N_Y$	No. of Triangles	Volume Construction	Surface Construction
10(a)	1 to 2	200×200	2	50×50	531	0.133	0.02
10(b)	1 to 2	200×200	2	50×50	694	0.183	0.03
11(a)	1 to 4	200×200	2	50×50	630	0.133	0.02
11(b)	3 to 5	240×240	2	60×60	2892	1.000	0.07
12(a)	1-2-3	200×200	3	50×50	889	0.150	0.02
12(b)	S to O	200×200	2	50×50	770	0.200	0.02
13(a)	C-hulls	200×200	4	50×50	2332	0.300	0.05
13(b)	Spiral	240×240	2	60×60	2188	1.050	0.05
14(a)	Offset	200×200	2	50×50	1312	0.217	0.03
14(b)	Overlap	200×200	2	50×50	774	0.133	0.02
15	Interior	260×260	4	65×65	3768	1.083	0.10
16	Torso	512×512	17	64×64	8682	4.017	0.32
17	Lungs	420×420	12	84×84	10413	5.050	0.38

Table 1: Table showing testing results

13 Discussion

This method treats contour data in an altogether more consistent way than the previous method. The use of the field function converts the data into a volume of numerical data, from which the surface is derived. Since the surface is determined locally on a cube by cube basis, decisions about how *good* the tiling is, are redundant. Since the tiling can follow the surface arbitrarily through the volume, horizontal tilings are created automatically, without the need for user interaction or special tiling algorithms. The previous methods generated the surface by considering the contours of two slice planes at a time, whereas this method could be regarded as ignoring such structures and producing the surface globally from the volume as a whole by using local surfacing operations on cubes.

The result of using this new approach is that the data is treated consistently with no need for preprocessing. As an example it is often required that the contour segments are listed in an anti-clockwise manner, with the start vertices in close proximity. Such preprocessing is removed, and the contour segments and contour slices can be processed in arbitrary order.

The branching problem is handled quite well with all of the problematical cases being tiled efficiently. Satisfactory surfaces are created for complex branching situations involving complex contour to contour interfaces. The problem of tiling massively varying contours between slices is essentially solved. The method even produces a satisfactory surface for the highly complex spiral problem. The example tilings have also shown that objects with interior structures can be reconstructed from contour slices that have contours interior to other contours, and that the surface created encloses the solid object, and has the correct surface normals.

Finally, from Table 1 it can be seen that the time to create such surfaces is quite acceptable, even for the large problems of the lungs and torso. Taking as example the problem of reconstructing the torso from MRI slices (first mentioned in Section 7). Using this method, one can rely on the surface produced from the contour data, and therefore there is no need to manually adjust badly placed tiles. A smooth surface can be produced from very few contour slices, and therefore the time required to isolate the object in the MRI scans is reduced. In the particular case of the torso reconstructed from 17 MRI slices, it took about 5 minutes

to outline the object in all of the slices, create the volume data, and reconstruct and display the surface. The 5 minutes compares quite favourably with the day taken by MacLeod et al. mentioned in Section 7. The resulting surface is smooth, and compares very well with the surface they produce.

The advantages of this method enable it to be useful for the rapid visualisation of contour data. It is particularly suited to applications where few contour slices are available, and a good, smooth surface is required quickly. The user can select a few slices from the data, create the contour outlines and produce the surface automatically. Using other methods this would not be possible because contours would differ too greatly between slices, but using this method such a factor is not so important.

14 Conclusion

This chapter has reviewed the techniques that exist for the reconstruction of objects from contour data (Sections 3–7). It has shown that the failings of such methods are their inability to handle complex branching and differing contour shapes on successive slices. In addition to these main problems, these methods fail to reconstruct solid objects with holes, and require many restrictions on the contour data to be adhered to. In contrast a new method for the reconstruction of surfaces from contour data has been presented (Sections 8–13). This method correctly tiles complex cases, and is able to reconstruct objects with interior structures. It requires only two restrictions to be present with the data, namely that the contours are closed and non-self-intersecting, which is far less restrictive than other methods. The surfaces produced are smooth, and natural looking, and the method produces them by treating the data in a consistent and appealing manner. The many test cases demonstrate that this method is reliable, automatic, effective and computationally inexpensive when producing surfaces from contour data.

This work [20] was presented at Eurographics 1994 (Oslo, Norway), and appeared in Computer Graphics Forum 13:3, pp C-75–C-84, under the title "A New Approach to the Construction of Surfaces from Contour Data".

References

[1] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, January 1975.

[2] 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.

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

[4] L. T. Cook, P. N. Cook, K. R. Lee, S. Batnitzky, B. Y. S. Wong, S. L. Fritz, J. Ophir, S. J. Dwyer, L. R. Bigongiari, and A. W. Templeton. An algorithm for volume estimation based on polyhedral approximation. *IEEE Transactions on Biomedical Engineering*, 27(9):493–499, September 1980.

[5] S. Ganapathy and T. G. Dennehy. A new general triangulation method for planar contours. In *Proc. SIGGRAPH '82 (Boston, Mass., July 26-30, 1982)*, volume 16(3), pages 69–75. ACM SIGGRAPH, New York, July 1982.

[6] L. T. Cook, S. J. Dwyer, S. Batnitzky, and K. R. Lee. A three-dimensional display system for diagnostic imaging applications. *IEEE Computer Graphics and Applications*, 3(5):13–19, August 1983.

[7] M. Shantz. Surface definition for branching contour-defined objects. *Computer Graphics*, 15(2):242–270, July 1981.

[8] H. N. Chrisiansen and T. W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. In *Proc. SIGGRAPH '78 (Atlanta, Georgia, August 23-25, 1978)*, volume 12(2), pages 187–192. ACM SIGGRAPH, New York, August 1978.

[9] J. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics and Image Processing*, 44:1–29, 1988.

[10] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, April 1991.

[11] C. Giersten, A. Halvorsen, and P. R. Flood. Graph-directed modelling from serial sections. *The Visual Computer*, 6:284–290, 1990.

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

[13] D. Myers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[14] J. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, October 1984.

[15] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994.

[16] G. Turk and M. Levoy. Mesh optimization. In *Proc. SIGGRAPH '94 (Orlando, Florida, July 24-July 29, 1994)*, volume 28(2), pages 311–318. ACM SIGGRAPH, New York, July 1994.

[17] R. S. MacLeod, C. R. Johnson, and M. A. Matheson. Visualization of cardiac bioelectricity - a case study. In *Proc. Visualization 92*, pages 411–418. IEEE CS Press, Los Alamitos, Calif., 1992.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[19] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Transactions on Graphics*, 3(2):135–152, April 1984.

[20] M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 13(3):C–75–C–84, September 1994.