SURFACE RECONSTRUCTION USING VARIATIONAL INTERPOLATION

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Maryruth Pradeepa Joseph Lawrence

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

ABSTRACT

Surface reconstruction of anatomical structures is an integral part of medical modeling. Contour information is extracted from serial cross-sections of tissue data and is stored as "slice" files. Although there are several reasonably efficient triangulation algorithms that reconstruct surfaces from slice data, the models generated from them have a jagged or faceted appearance due to the large inter-slice distance created by the sectioning process. Moreover, inconsistencies in user input aggravate the problem. So, we created a method that reduces inter-slice distance, as well as ignores the inconsistencies in the user input. Our method called the *piecewise weighted implicit functions*, is based on the approach of weighting smaller implicit functions. It takes only a few slices at a time to construct the implicit function. This method is based on a technique called *variational interpolation*.

Other approaches based on variational interpolation have the disadvantage of becoming unstable when the model is quite large with more than a few thousand constraint points. Furthermore, tracing the intermediate contours becomes expensive for large models. Even though some fast fitting methods handle such instability problems, there is no apparent improvement in contour tracing time, because, the value of each data point on the contour boundary is evaluated using a single large implicit function that essentially uses all constraint points. Our method handles both these problems using a sliding window approach. As our method uses only a local domain to construct each implicit function, it achieves a considerable run-time saving over the other methods. The resulting software produces interpolated models from large data sets in a few minutes on an ordinary desktop computer.

ACKNOWLEDGEMENTS

First, I would like to express my sincere and heartfelt gratitude to my supervisor Dr. Eric Neufeld for the great patience he showed while guiding me and for his unwavering support and encouragement in all aspects throughout my graduate studies. I also like to thank him for his wonderful considerate nature and cheerfulness which made my graduate studies an enjoyable experience. I like to express my sincere thanks to my committee members Dr. David Mould and Dr. Michael Horsch for their suggestions and support. I am very thankful to College of Graduate Studies and Research and the Department of Computer Science for providing financial support for my degree. My sincere thanks to the staff of Computer Science department, especially to Jan Thompson, Gina Koehn, and Shane Doucette for their excellent support and for providing a friendly and pleasant environment for my studies. My special thanks to Brennan Rusnell for providing me the additional software needed for my work. I also like to thank Sonje Kristtorn for her suggestions and motivation. I take this opportunity to thank my friends for their encouragement throughout my studies. Finally, I am very grateful to my family for their love, encouragement and support throughout this work.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Use of computer systems in the study and analysis of the anatomical structures has significantly increased over the last few decades. Several clinical applications use advanced computer graphics techniques to model the physiological structures used in medical fields like surgery planning, volumetric analysis, education and research. Image data of these anatomical structures is obtained using techniques such as cryosectioning, computed tomography (CT, CAT scan), magnetic resonance imaging (MRI), and ultrasonic imaging. Cryosectioning is a traditional technique still widely used. The tissues are cross-sectionally dissected to obtain the "slices" and are then digitally photographed and stored as digital data.

A significant community of users builds models from serially sectioned data by tracing contours of objects of interest on images, which are subsequently surfaced. These users are often interested in performing volumetric and surface area calculations of the traced objects, as well as in the model as a teaching or visualization object.

The following illustrates this process. Users at DREM (Digitally Reproduced Embryonic Morphology) are building models of the human embryo at various stages of development. Figure 1.1 shows two consecutive slices from that data set. The user marks boundary vertices with a mouse. Figure 1.2 shows a closeup view of a traced contour. The actual boundaries on slices are not well-defined and tracing must be done by a user with expert knowledge of embryo anatomy.

Once the contours are marked on all the slices, the anatomical surface structure is reconstructed by triangulating parallel tissue slices as shown in Figure 1.3. Given a set of points, triangulation involves connecting them into a mesh of triangles. Several triangulation algorithms that perform surface reconstruction from serial contours have been

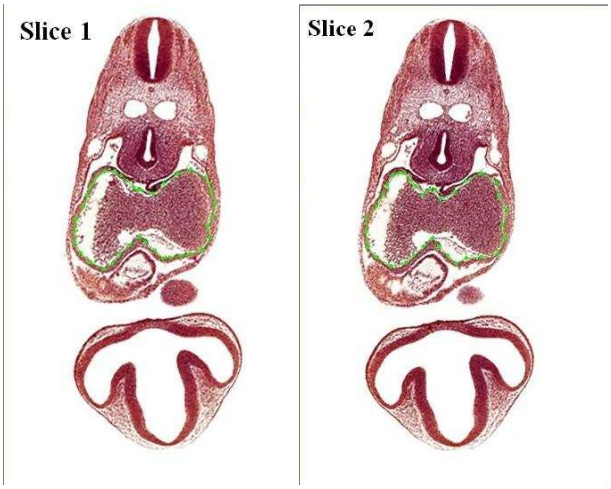


Figure 1.1: Example of slices from DREM data set [1].

explored. Although Figure 1.1 and Figure 1.3 suggest that triangulation of slices is straightforward, many problems arise if the triangulation algorithm does not handle complex data sets. Some of the problems and solutions are discussed in Chapter 2.

These reconstructions were done using Winsurf, a commercial product designed for this purpose. Winsurf surfaces contours by applying a triangulation algorithm to pairwise contours. Figure 1.4 shows a reconstructed embryo heart using WinSurf. Other three-dimensional model building software tools include Maya [7], 3D Doctor [8], trueSpace [9] and 3D Canvas [10]. Winsurf gives the user a choice of triangulation algorithms, which we discuss below.

Our present work uses several data sets. Many of the models shown in this dissertation used parts of objects from a prostate reconstruction by Barry Timms of the University of South Dakota. This model has been under construction for years and shows great detail (Figure 1.5). To obtain their models, Barry Timms's group traces the images onto tissue paper, and then uses Winsurf's magic wand tool to generate contours using a segmentation algorithm. The group uses this technique because they believe that tracing with a pencil gives more accurate contours than tracing with a mouse.

We also used data provided by DREM for the embryonic period of human prenatal development. The embryonic period encompasses the first eight weeks of conception and is divided into 23 stages [1]. The DREM data is obtained by taking cross-sections of the human embryo at $15 - \mu m$ intervals. Hence, each stage consist of hundreds or sometimes

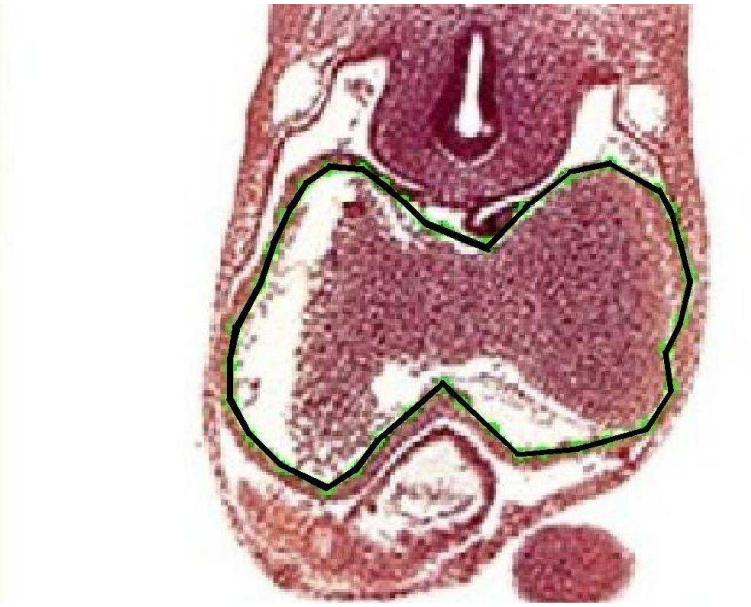


Figure 1.2: Boundary vertices.

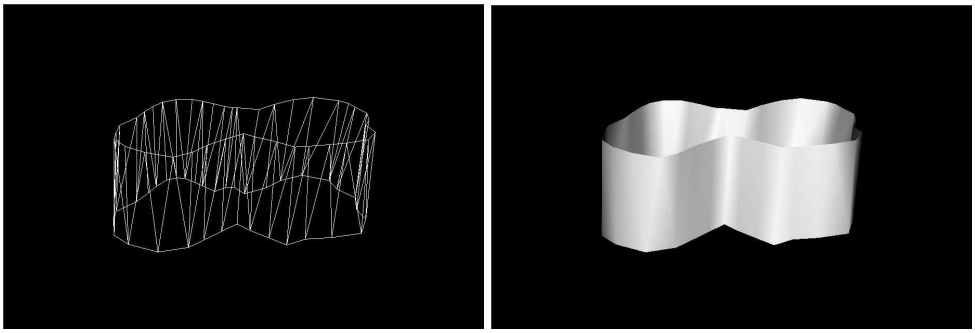


Figure 1.3: Triangulation between the slices.

thousands of cross-sectional slices. The DREM data set is a digitized collection of these sections for all 23 stages. Figure 1.6(a) and Figure 1.6(b) show two stages of embryonic development.

Another important data set used worldwide is the National Library of Medicine's Visible Human dataset [11]. In the mid 1980*s*, the National Library of Medicine (NLM) of the National Institutes of Health [12] developed the *Visible Human Project*. This provided the medical and the graphics communities with a detailed digital data set of the entire human body. The Visible Male data set was sectioned at 1-mm intervals. This thickness was found to be ill-suited for voxel based approaches, so the Visible Female data set was sectioned at 0.33-mm intervals. The Visible Female has slightly over 5000 images compared

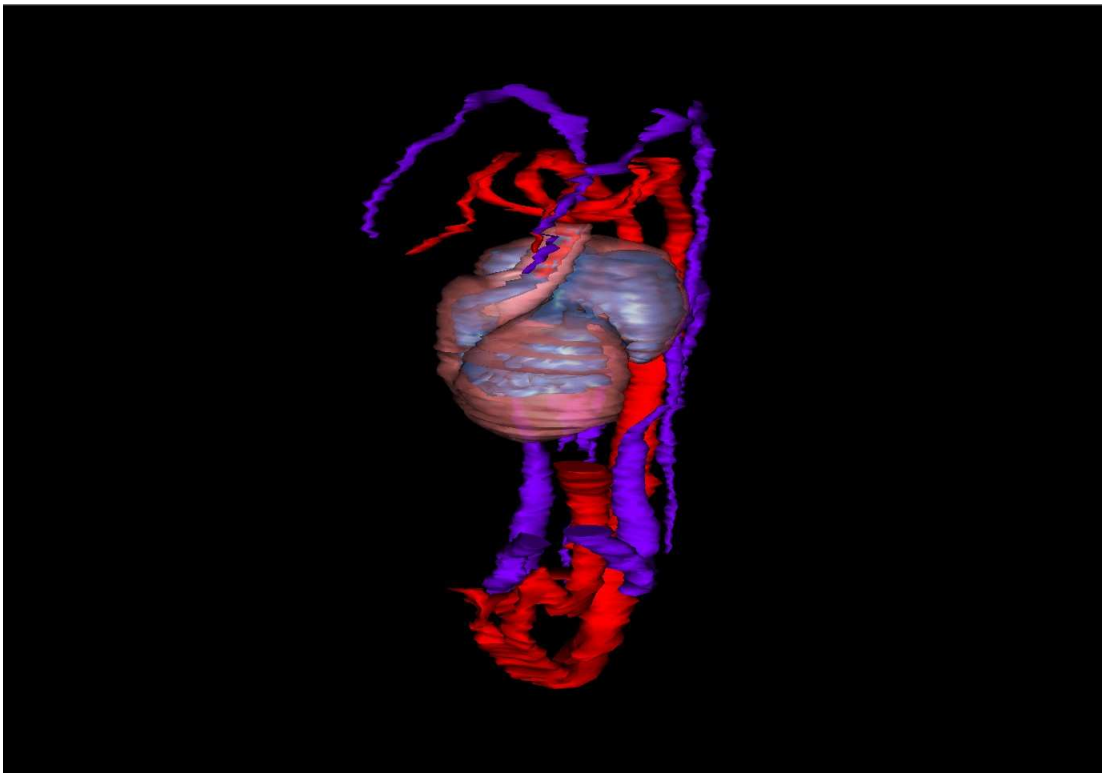


Figure 1.4: Reconstructed embryo heart using WINSURF. Courtesy DREM [1]

to the Visible Male with 1871 images. Despite the availability of this valuable data, the model construction remains a slow process.

Although Winsurf was developed to assist with volumetric and similar calculations, aesthetics has always been important. The kidney model of Doll *et al.* (Figure 1.7) [13] used very high resolution images. As with Timm's prostate model, key areas were drawn by hand with tracing paper to reduce the raggedness of mouse-traced models. This level of detail requires considerable work by the model builders. Thus, one goal of this work is the generation of better quality models without requiring more user input.

1.1 Overview

Models constructed from serially sectioned data tend to exhibit unnatural artifacts. These artifacts arise from several reasons, including misalignment of sections, user errors, and interslice distance. At the outset, the goal of this research in general terms was to produce

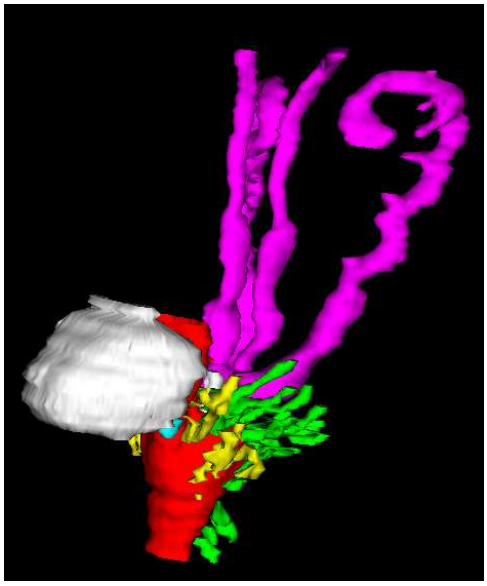


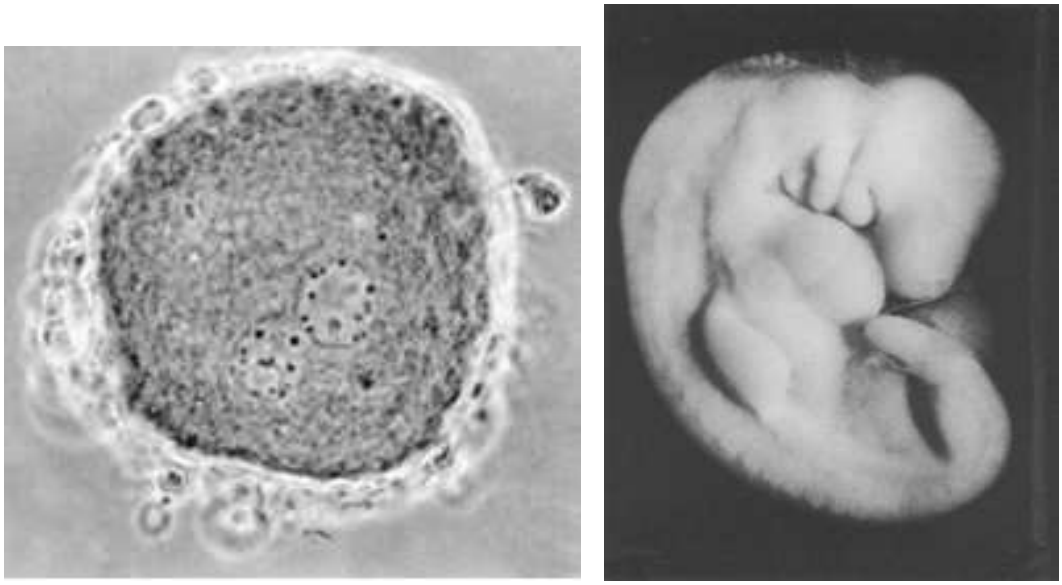
Figure 1.5: Prostate reconstruction using Winsurf.

models from this kind of data, but with significantly higher quality and in reasonable time, so that a model builder working on an ordinary personal computer could quickly get visual feedback while working on a fairly large data set. In specific terms, our goal was to build smooth, organic looking anatomical models from manually generated slice based anatomical data in reasonable time on an ordinary personal computer.

The exploratory phase of the research investigated a variety of possibilities, including more sophisticated triangulation algorithms, and interpolation approaches.

While we found that triangulation algorithms for pairwise slices are virtually instantaneous on a modern desktop computer, even for fairly large data sets, we saw no general way to improve model quality significantly. On the other hand, the interpolation approaches yield high quality (smooth) models, but even modest model sizes can crash a typical desktop machine, or take a very long time to run.

In the end, we came very close to meeting our above stated goal, by using a combination of triangulation and interpolation algorithms. This was accomplished by first using an optimal triangulation algorithm to produce a kind of "first cut" description of the surface, which was then fed into the interpolation algorithm. Then, to avoid the high costs of the interpolation algorithm, we devised a method called the *piecewise weighted implicit functions* in which the surface was computed piecewise by passing a "sliding window" over



(a) Stage 1 of embryo.

(b) Stage 13 of embryo.

Figure 1.6: Stages of embryo. Courtesy The Human Developmental Anatomy Center, Carnegie Collection [1]

the slices and blending the results. The speedup was dramatic and the resulting models do not obviously differ from models not computed piecewise.

The rest of this section gives a high level overview of our work as described in the rest of the thesis.

The first phase of our study investigated the possibility of improving the final models by improving the triangulation algorithms. The early version of Winsurf that we wanted to improve used a greedy algorithm for triangulating the parallel slices. A "greedy" algorithm by definition attempts to satisfy goals by taking the choice with the largest immediate progress. This resulted in the algorithm sometimes making wrong choices when choosing a vertex for subsequent triangulation, and hence, the final model was not visually agreeable. Hence, to improve the early version of Winsurf so as to develop better models, we studied and implemented an optimal triangulation algorithm and a heuristic triangulation algorithm in the course of this work.

A classic and frequently cited example of an optimal algorithm is that of Fuchs *et al.* [3]. An example of optimality criterion for the Fuchs *et al.* algorithm is the minimization of surface area of triangles. Optimal algorithms are typically slower than heuristic algo-

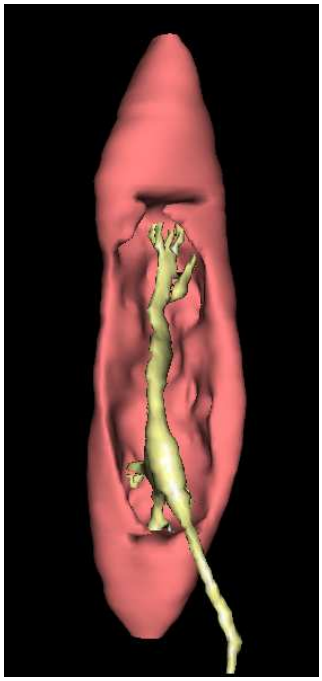


Figure 1.7: Kidney and Ureter.

rithms because they often must explore an entire solution space. The heuristic algorithms on the other hand are not optimal, but are typically faster than the optimal algorithms precisely because they skip over large parts of the solution space. To some extent, the complexity of the data determines which algorithm is suited to a given problem. If the data is simple, that is, if adjacent slices are almost identical, then a simple heuristic can give a quick surface reconstruction. Moreover, the solution found is likely to be close to optimal. On the other hand, a naive choice of optimality criteria can result in models that are expensive to compute and are not pleasing visually: the worst of both worlds. If the data is complex, then optimality criteria or heuristics should be carefully chosen.

Even if the optimality criteria are carefully chosen, in certain situations the surface reconstructed by an optimal algorithm may not be visually pleasing, because users manually trace the contour boundaries which introduce some inconsistencies. Such errors arise because users of software like Winsurf often don't understand the assumptions behind the triangulation algorithms and enter data inconsistently. For example, some slices may have very few points and other slices may have many points. This leads to the reconstructed surface having a faceted appearance where there are relatively few points and a jagged

appearance where there are relatively many points (Figure 1.8). The faceted appearance in case of few points is because the triangles between the slices are bigger and broader, whereas the reason for the jagged look is the uneven distribution of points and elongated triangles.

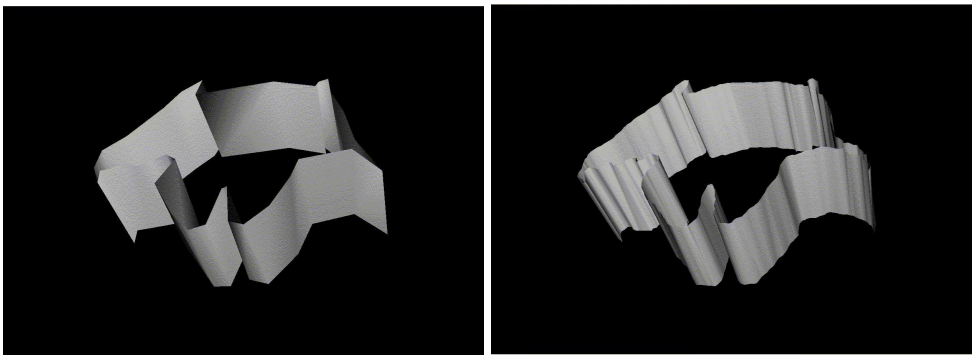


Figure 1.8: Faceted appearance of models with a few distant points and jagged appearance of models with points very close together.

A significant factor that affects the appearance of a reconstructed surface is the thickness of the sectioned slices. Sometimes, if sectioning is done with large intervals, that is, if the tissue slices are thick, the consecutive contours may be quite different, which may result in an uneven triangulation and loss of smoothness.

Two important issues must be addressed to create visually pleasing models. First, the algorithm has to cope with these inconsistencies in the data. Next, consecutive contours should not be very different. These two issues can be accomplished by extending and modifying the *shape transformation* technique of Turk and O'Brien [14], who use a technique called variational interpolation for reconstructing surfaces from point cloud data.

Our work applies the variational interpolation technique to serial contours. This technique takes originally dissimilar input slices and automatically generates numerous slices in between them that are similar to each other, thus achieving smooth transition between slices. Figure 1.10 shows an *X* shape transforming into an *O* shape using variational interpolation (left) and an optimal triangulation algorithm of Fuchs *et al.* (right). The figure on the left generated using variational interpolation is preferred because it looks more natural and organic compared to the one on the right generated using Winsurf. The contour data given as input for this shape transformation is shown in Figure 1.9. It sug-

gests that improvements based strictly on variations in triangulations have limitations. The interpolation algorithm adds considerable smoothness to the sample object. As well, this algorithm may relieve some of the burden on model builders. Of the models shown earlier, the kidney was the smoothest (Figure 1.7). However, achieving this required considerable work on the part of the model builders. Therefore we also looked at ways to incorporate the interpolation methodology into model building.

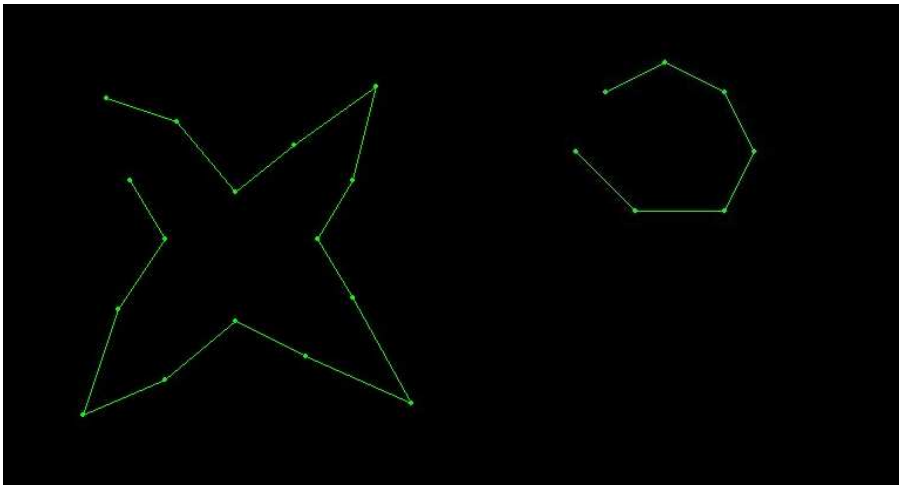


Figure 1.9: Contour data given as input to the system.

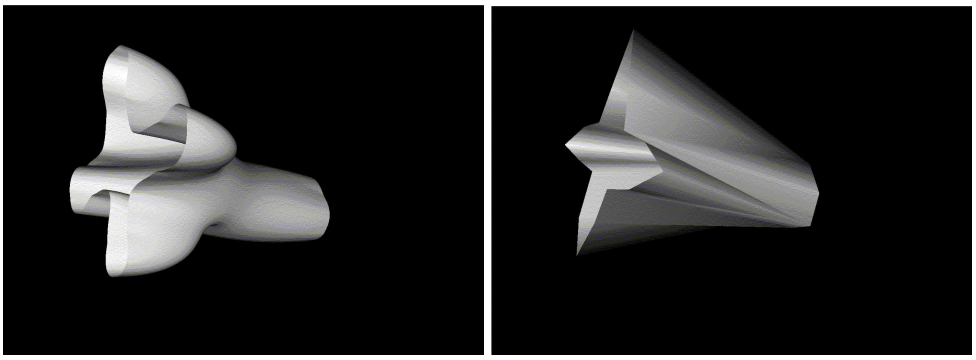


Figure 1.10: Smooth transformation of *X* to *O* using variational interpolation of Turk and O'Brien(left) vs *X* to *O* transformation using WinSurf (right).

The size of some of our models introduced a potential problem of numerical instability when applying the method of Turk and O'Brien. In other cases, a fully automated approach was not ideal because the interpolated object was not consistent with the user's expectations. To deal with both these problems, we investigated a piecewise approach to

variational interpolation, as well as stable methods for dealing with large data sets. In particular, we investigated a fast fitting method of Beatson *et al.* [15], a stable algorithm for solving large systems of linear equations.

Our final solution resulted in an interesting symbiosis between triangulation techniques and variational techniques enhancing the overall usability from the perspective of the naive user, who is usually not knowledgeable about geometric issues. The contribution of this thesis is a combinative method that uses an optimal triangulation and a modified variational interpolation technique called *piecewise weighted implicit functions* that solves the instability problem as well as the slower runtime problem of the previous methods.

The rest of the thesis is organized as follows. Chapter 2 provides the necessary background to the general problem of surfacing models, including triangulation algorithms and different interpolation methods including variational interpolation. Chapter 3 explains two approaches, our weighted implicit function approach and the preconditioned GMRES approach of Beatson *et al.*, to solve the problems of the other surfacing approaches. Chapter 5 discusses our results. Finally, Chapter 6 contains the concluding statements and some probable extensions to our current work.

CHAPTER 2

PREVIOUS WORK

2.1 Introduction to Surfacing

The first problem we studied in depth was triangulation. In simple terms and in our context, triangulation is the problem of how to take contour data from adjacent slices and stitch them together in the most visually pleasing manner and avoid the problems that the greedy algorithm made in the earlier version of Winsurf. Winsurf's predecessor, Surfdriver, triangulated slices piecewise using a Delaunay based approach [16]. In 2D, Delaunay triangulation can be thought of as optimizing a triangulation by maximizing the minimum angle. As we wanted to consider other expressions of optimality, we explored some more optimal algorithms as well as a heuristic algorithm. The optimal algorithm we studied by Fuchs *et al.* [3] performs triangulation by separately determining an optimal surface between each consecutive pair of the slices.

We then turned our attention to interpolation. That is, given pairs of slices, find a sequence of intermediate slices that give a smooth transition between each pair. At the time of writing, the method of variational interpolation was prominent in the literature. Below we review the relevant ideas in both of these sub areas.

2.2 Triangulation algorithms

We studied the triangulation algorithms of Ekoule *et al.* [2] and Fuchs *et al.* [3] with the dual goals of fast real time performance and visually pleasing results. The heuristic algorithm of Ekoule *et al.* claims to do both in addition to handling all shapes. It also claims to deal with branching issues; that is, the problem of a single contour joining multiple

contours on neighboring slices. As WinSurf implements multiple branching by merging contours, we only implemented single branching. We found the algorithm of Ekoule *et al.* was fast, but didn't work for all shapes, including a shape frequently encountered in our data sets. The optimal algorithm proposed by Fuchs *et al.* was also implemented, with better resulting surfaces, and, although slower, in acceptable time.

2.2.1 The heuristic algorithm of Ekoule *et al.*

Ekoule's algorithm is based on the assumption that consecutive contours have similar convex hulls. The *convex hull* of a set of vertices is the smallest convex region that encloses all the vertices [17]. This assumption is often true with the DREM data set [18].

Contours can be *convex* or *non-convex*. A contour is convex if it is identical to its convex hull and is non-convex otherwise. Triangulation between non-convex contours is done by hierarchically decomposing the contours until the given non-convex contour transforms into a convex contour.

Triangulation of convex contours

A contour is represented as an ordered set of points. If C_1 and C_2 are two contours to be triangulated then the edges joining them should have the following properties. Each edge should have one vertex in C_1 and another in C_2, and every two consecutive edges should have only one common vertex and should form a triangular patch.

Let P_i be a point in C_1 and Q_j be a point in C_2 (Figure 2.1). Without loss of generality C_1 has fewer points than C_2. Then the triangulation is done by connecting each point in C_1 to the closest point in C_2. In the explanation that follows, P_i and P_{i+1} represent consecutive points on C_1, $Q_{(j)(i)}$ and $Q_{(j)(i+1)}$ are points on C_2, but are not consecutive. Point $Q_{(j)(i)}$ is closer to P_i and point $Q_{(j)(i+1)}$ is closer to P_{i+1}. The second term of the subscript in $Q_{(j)(i)}$ and $Q_{(j)(i+1)}$ which are i and $i+1$, denote that these points are closer to the points in the i^{th} and $i+1^{th}$ position of C_1.

P_i is connected to $Q_{(j)(i)}$, where $Q_{(j)(i)}$ denotes a point on C_2 closest to P_i, and P_{i+1} is a connected to $Q_{(j)(i+1)}$, where $Q_{(j)(i+1)}$ is the point on C_2 closest to P_{i+1}. Note that $Q_{(j)(i+1)}$ appears later in the sequence of points than $Q_{(j)(i)}$. The point Q_k on C_2 is the

latest point in the sequence such that the distance from P_i to Q_k (d(P_i,Q_k)) is less than the distance from P_{i+1} to Q_k (d(P_{i+1},Q_k)). Also, Q_k lies between $Q_{(j)(i)}$ and $Q_{(j)(i+1)}$. Figure 2.1 depicts the selection of the points. All the points between $Q_{(j)(i)}$ and Q_k are connected to P_i and the points between Q_k and $Q_{(j)(i+1)}$ are connected to P_{i+1}.

If C_1 and C_2 are non-convex, they must be preprocessed before applying the triangulation.

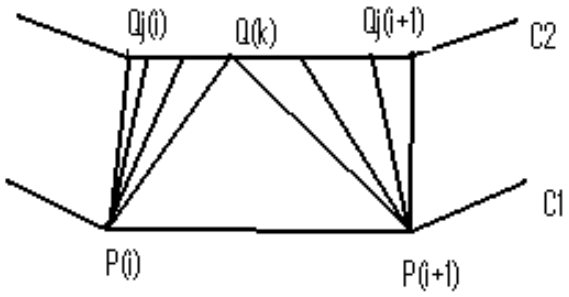


Figure 2.1: Single branching.

Triangulation of non-convex contours

Preprocessing of non-convex contours is done in two separate stages: First, the non-convex contour C is decomposed into its *elementary concavities*, which can be represented in a hierarchical tree structure. Next, the terminal nodes in the tree are projected onto the convex hull of the parent node to get a transformed convex contour C'. The preprocessing is done so that the distribution of the vertices in the transformed contour C' is the same as in the original contour C. This method relies on the assumption that any two contours in the consecutive tissue slices have similar convex hulls.

Decomposition of Contour C Let C^0 be the original non-convex contour C and let P_i, for $1 \leq i \leq M$, be the points in C^0 with anti-clockwise orientation. Let $E^0 = \{P_k \mid k \in K\}$, where K is a subset of $\{1 \ldots M\}$, be the convex hull of C^0. If $C^0 \neq E^0$, then at least one vertex in C^0 is not in E^0. That is, there are at least two points P_{j_1} and P_{i_1} consecutive in

E^0 but are not in C^0. This indicates the presence of a first order concavity, represented as $C^1_{i_1,j_1}$, where (j_1, i_1) are the start and end points of the first order concavity. Let $E^1_{i_1,j_1}$ be the convex hull of $C^1_{i_1,j_1}$. If $E^1_{i_1,j_1} = C^1_{i_1,j_1}$, it is an elementary concavity and denotes the end of decomposition. If $E^1_{i_1,j_1} \neq C^1_{i_1,j_1}$, then there is a second order concavity $C^2_{(i_1,j_1)(i_2,j_2)}$, where (i_2, j_2) represents the start and end points of the second order concavity and is represented in the second level of the tree. In the hierarchical tree structure, C^0 is represented as the root and each n^{th}-order concavity is represented as the n^{th}-level node in the tree.

To summarize the above, the corresponding convex hull $E^n_{(i_1,j_1)\ldots(i_n,j_n)}$ is found for each concavity $C^n_{(i_1,j_1)\ldots(i_n,j_n)}$. When $C^n_{(i_1,j_1)\ldots(i_n,j_n)} = E^n_{(i_1,j_1)\ldots(i_n,j_n)}$, the decomposition cannot go further as the elementary concavity is reached. If $C^n_{(i_1,j_1)\ldots(i_n,j_n)} \neq E^n_{(i_1,j_1)\ldots(i_n,j_n)}$, then the decomposition continues until the elementary concavity is found.

The decomposed original contour now has to be transformed to get the convex contour.

Transformation of Contour C The decomposed contour is transformed by projecting the elementary concavity represented by the terminal node on the convex hull of the contour of its parent node. If $C^n_{(i_1,j_1)\ldots(i_n,j_n)}$ is the elementary concavity, then each point P_i on $C^n_{(i_1,j_1)\ldots(i_n,j_n)}$ is projected onto the line joining P_{i_n} and P_{j_n} as in Figure 2.2. The projection is calculated as follows:

$$x_i^{'} = x_i - R_i(x_{j_n} - x_{i_n}) \tag{2.1}$$

$$y_i^{'} = y_i - R_i(y_{j_n} - y_{i_n}), \tag{2.2}$$

where R_i is the normalized weighting factor and is given as

$$R_i = \frac{\sum_{k=i_n}^{i-1} d(P_k, P_{k+1})}{\sum_{k=i_n}^{j_n-1} d(P_k, P_{k+1})}. \tag{2.3}$$

In Figure 2.2, $A/B = a/b$. These calculations maintain the relative distance between vertices after the projection. The fully processed contour has the same number of vertices as the original contour, and the original contour C has been processed into a convex contour $C^{'}$.

Because $C^{'}$ is convex, the triangulation algorithm for convex contours can be used (first sub-section of Section 2.2.1).

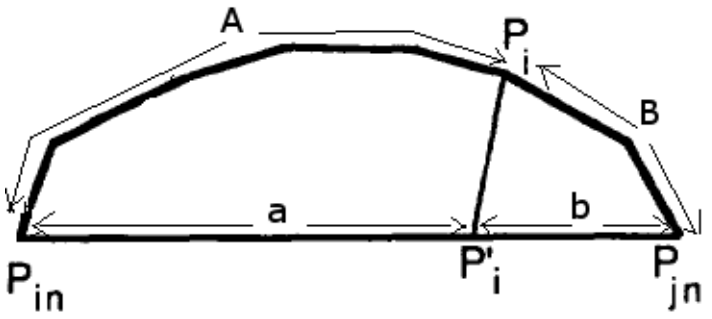


Figure 2.2: Projection of elementary concavities [2].

Implementation

The convex hull of the contour is found using Graham's Scan method [17]. First, the *pivot* of the convex hull is found by choosing the vertex with the smallest *y*-coordinate and the largest *x*-coordinate. The pivot always lies on the convex hull. Let S_T and S_{T-1} be the top two vertices on the stack at any point in the algorithm. We visit each vertex V_i on the contour in anti-clockwise order. If (S_{T-1}, S_T, V_i) represents a "left turn", push V_i on the stack. If (S_{T-1}, S_T, V_i) represents a right turn, pop S_T and push V_i. When the scan completes, this stack contains only vertices of the convex hull.

By recursively calling this method for every concavity, the tree is fully populated with the terminal node representing the n^{th}-order concavity. The transformation function is applied to every vertex and the projections of the points in each level of the tree are calculated. These projections replace the original vertices in the parent node. At the end of this call, the root node contains the transformed or the *projected vertices* of the original input vertices.

When the projection is done, the triangulation of the non-convex contours is done by establishing relations between the vertices of the two transformed contours.

Deployment of Ekoule's algorithm on real models

This heuristic algorithm works on some models and fails on others. Figure 2.3 and Figure 2.4 show models on which Ekoule's algorithm works. The two slices in Figure 2.3 have

similar convex hulls with many non-convex regions. The two slices in Figure 2.4 have identical *y* and *z* coordinates, but one is translated 100 units in the *x*-dimension. This sort of "horseshoe" shape often appears in the embryo models. When WinSurf aligns contours by centering bounding boxes, the corresponding concavities begin and end at the same relative locations. Hence the triangulation for both these cases is correct.

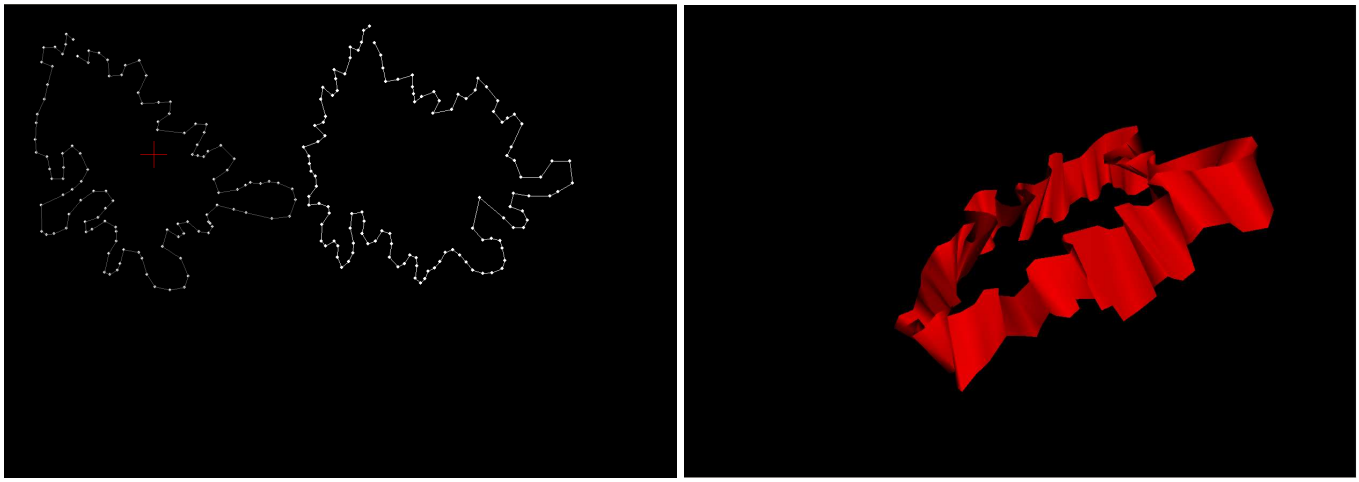


Figure 2.3: An example of a correct triangulation obtained using Ekoule's algorithm.

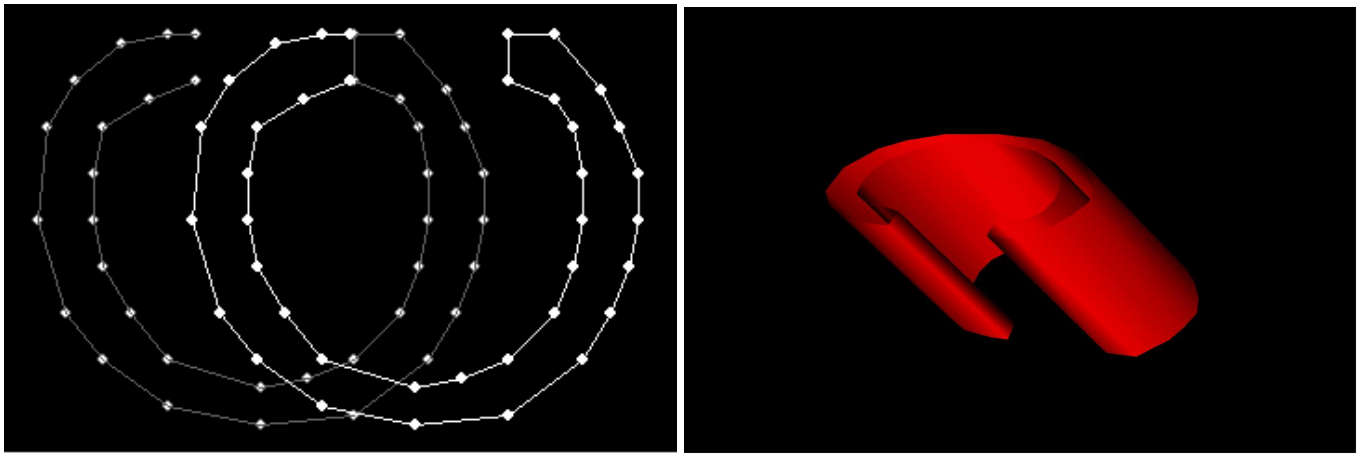


Figure 2.4: An example of a correct triangulation obtained using Ekoule's algorithm.

Example 1 The two slices in Figure 2.5 have almost identical *x*-coordinates. The only difference is that the concavity begins and ends at different relative locations for each slice, and the *z*-coordinates have been translated 100 units for one slice. After projecting

the vertices in the non-convex region on the convex hull, both slices look similar, that is, both will have a circular shape. However, the corresponding local neighborhoods are totally incorrect for the original non-convex regions.

The triangulations of the slices in Figure 2.5 are not pleasing, because the inner vertices of Slice 1 are projected onto the line joining the two outer corner points of the opening. Hence, when the outer vertices of Slice 2 look for nearest neighbors of Slice 1, they choose the projected inner vertices of Slice 1, which is incorrect. Ekoule proposes that, after the transformation of the non-convex contours into convex contours, all the points in the non-convex regions of the contour are projected on the hull and when the nearest points on adjacent contours are chosen, the correct nearest point is determined as the relative distances of the vertices are maintained during the projection. The "horse shoe" example shows that this assumption is not valid even when the convex hulls are similar.

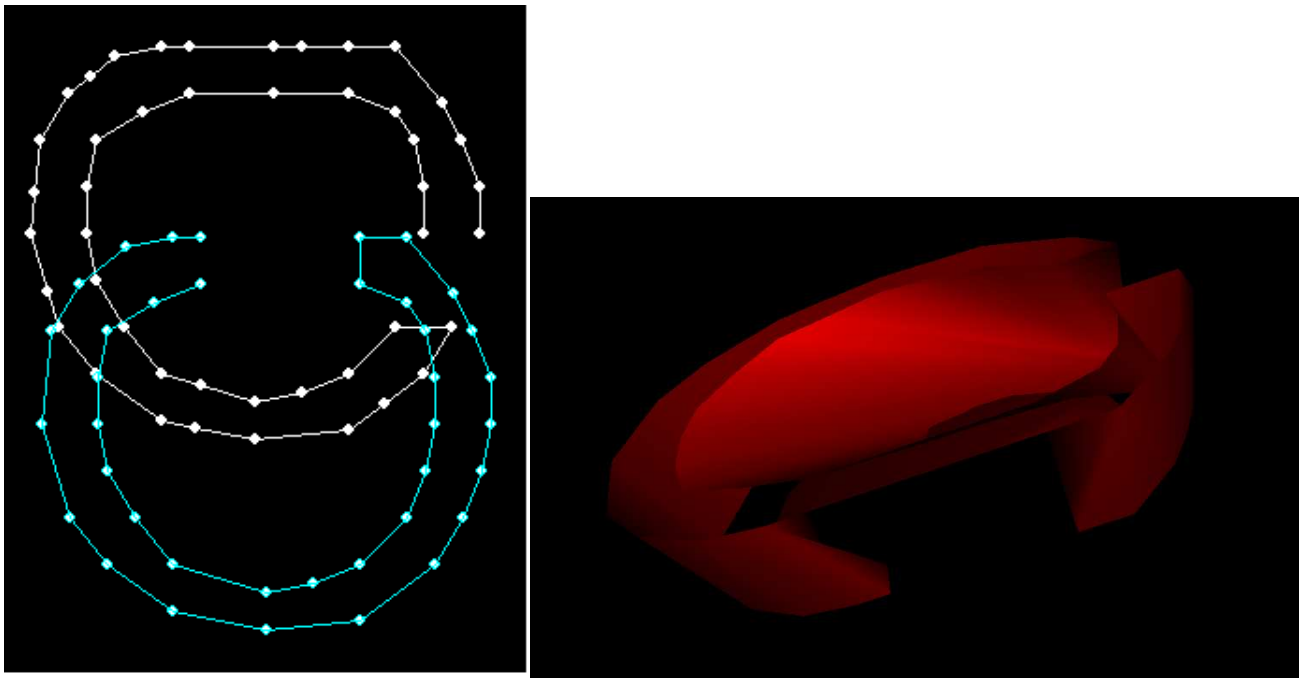


Figure 2.5: An example of an incorrect triangulation obtained using Ekoule's algorithm.

Example 2 Figure 2.6 shows two similarly shaped adjacent contours. One contour has been rotated by a small angle so that the openings do not align. Even though the two contours are almost identical, the local neighborhoods of the projected vertices in

both slices are quite different, which leads to an incorrect mapping between the vertices and hence an incorrect triangulation. When applying Ekoule's algorithm to this type of contour we found that the algorithm did not tolerate rotations well.

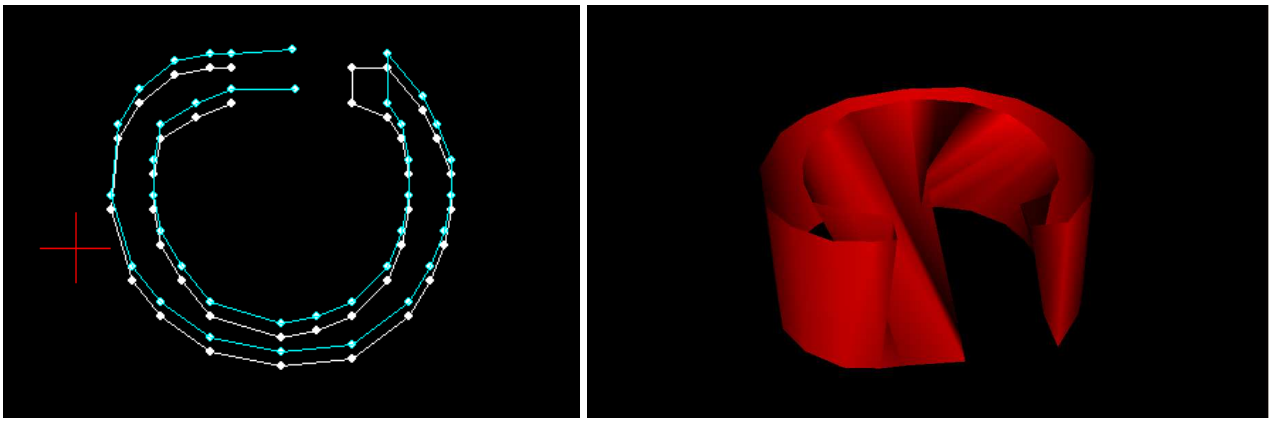


Figure 2.6: An example of an incorrect triangulation obtained using Ekoule's algorithm.

Example 3 In Figure 2.7, the projections of vertices in the inner circle and the projections of vertices in the bulbs (the circular twist on either side) lie on the convex hull. Ekoule also proposes that relative distances are maintained in the transformed contour, but in Figure 2.7 we can see that maintaining the relative distances between corresponding vertices is difficult. Also for these bulb shapes, even though the non-convex regions are projected onto the hull, the intended shapes are not maintained in the final output, because the twists inside the outer curve make the local neighborhood of the transformed contour incorrect. So, when finding the nearest vertex of a point x in Slice 1, the algorithm chooses a vertex that was not in its local neighborhood in the original set, but that is in its local neighborhood after the transformation. The bulb shapes on either side of the slice are rendered incorrectly.

These horseshoe shapes are not obscure. They occur frequently in the DREM data set [18] where they are used to represent wall-like structures. Ekoule *et al.* demonstrated their algorithm on human vertebra, which are not very complex and do not contain horseshoe-like shapes. One possible way of handling such shapes might be to partition the non-convex regions separately and apply the algorithm to the subregions, but this cre-

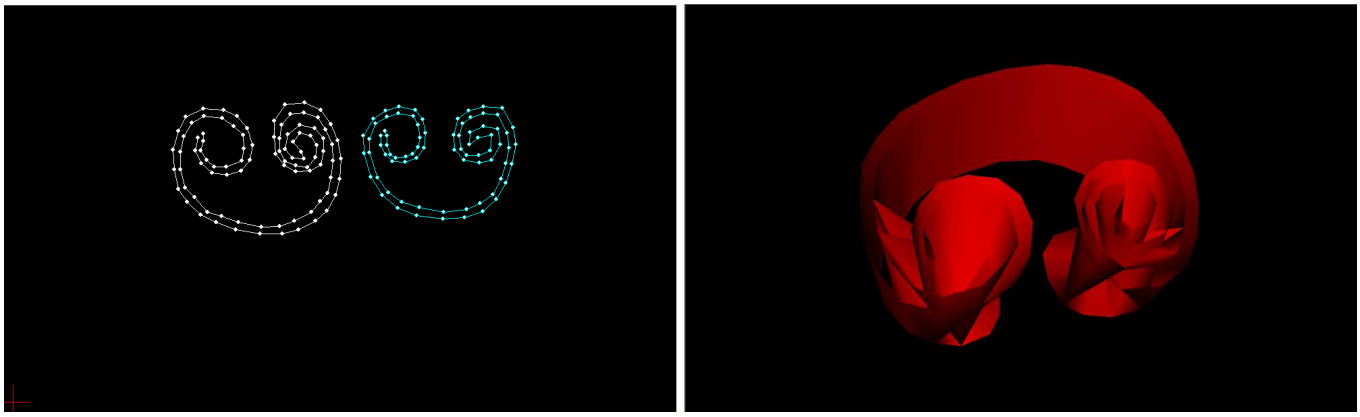


Figure 2.7: An incorrect triangulation generated by Ekoule's algorithm.

ates the new problem of subdividing the non-convex regions. However, the application of such patches to the basic algorithm could soon overwhelm the algorithm's most attractive features - its speed and simplicity.

2.2.2 The algorithm of Fuchs *et al.*

The algorithm by Fuchs *et al.* [3] triangulates adjacent contours using an optimizing criterion without applying any heuristics. Any monotonically non-decreasing optimizing criterion can be chosen by the user. The problem of finding an optimal triangulation can be represented as a problem in graph theory.

Let P and Q be two adjacent contours. P_0,...,P_{m-1} are points of the closed contour P, where m is the number of points in P and P_0 follows P_{m-1}. Q_0,...Q_{n-1} are points of the closed contour Q, where n is the number of points in Q and Q_0 follows Q_{n-1}. A *contour segment* is defined as the section between any two consecutive points in a contour, that is the linear segment between P_i and P_{i+1} or Q_i and Q_{i+1}. A *span* is the edge connecting a vertex from one contour to a vertex in the adjacent contour and is written as $\overline{P_iQ_j}$, where P_i is a vertex in contour P and Q_j is a vertex in contour Q. An *elementary tile* consists of two spans and one contour segment. The two spans connect each end of the contour segment to a common vertex in the adjacent contour. P_i, Q_j, P_{i+1} form an elementary tile, where P_i and P_{i+1} form the contour segment and Q_j is the common vertex in an adjacent contour.

A huge number of sets of elementary tiles can be built between the contours, but the right set is chosen with the following conditions. For the elementary tiles to form an

acceptable surface, each contour segment should be present in not more than one tile of the set, and if a span is a left (right) span for any elementary tile, it has to be a right (left) span for at least one other tile in the set. Even so, there are still many acceptable surfaces. The problem of choosing the best set of tiles among the acceptable sets can be simplified by representing spans and elementary tiles as *vertices V* and *arcs A* of a toroidal graph, and by defining an acceptable subgraph *S* as follows.

In a directed toroidal graph $G = \langle V,A \rangle$, V is set of all possible spans between points in P and Q and A is set of all possible elementary tiles. Figure 2.8 shows a directed toroidal graph. The vertex V_{ij} in G at row i and column j represents the span between P_i and Q_j ($\overline{P_iQ_j}$). An arc in G, written $\langle V_{kl}, V_{st} \rangle$, represents an elementary tile with left span $\overline{P_kQ_l}$ and right span $\overline{P_sQ_t}$, where either $s = k$ and $t=(l+1)\ modn$ or $s = (k+1)\ modm$ and $t = l$. $\langle V_{ij}, V_{(i+1)modm,j} \rangle$ represents a vertical arc from row i to row $(i+1)modm$, and A$\langle V_{ij}, V_{i,(j+1)modn} \rangle$ represents a horizontal arc from between columns j and $(j+1)modn$. The following conditions define an *acceptable subgraph S*. There should be exactly one vertical arc between any two rows, and exactly one horizontal arc between any two columns. For a vertex v, either indegree(v)= outdegree (v) = 0 or indegree(v) $>$ 0 and outdegree(v) $>$ 0, where indegree is the number of arcs incident on a vertex and outdegree is the number of arcs incident from a vertex. Figure 2.9 shows how a set of tiles are mapped into a subgraph.

An acceptable subgraph S corresponds to an *acceptable surface* if and only if (1) S contains exactly one horizontal arc between any two adjacent columns and exactly one vertical arc between any two adjacent rows, and (2) S is weakly connected and for every vertex v, indegree(v) = outdegree(v), that is S is Eulerian. An Eulerian trail in a directed graph is a closed path in which all the arcs occur exactly once.

S can take one of two forms. For every vertex v_{ij} of S, if indegree (v_{ij}) = outdegree (v_{ij})=1, the surface is homeomorphic to a cylinder. If indegree(v_{st})=outdegree (v_{st})=2 for a vertex v_{st}, then for every other vertex v_{ij} in S, indegree (v_{ij}) = outdegree (v_{ij})=1. Such a surface is homeomorphic to two cones glued together at the span $\overline{P_sQ_t}$ where indegree(v_{st}) = outdegree(v_{st}) = 2.

An acceptable surface can also be called an *acceptable trail*. An acceptable trail has

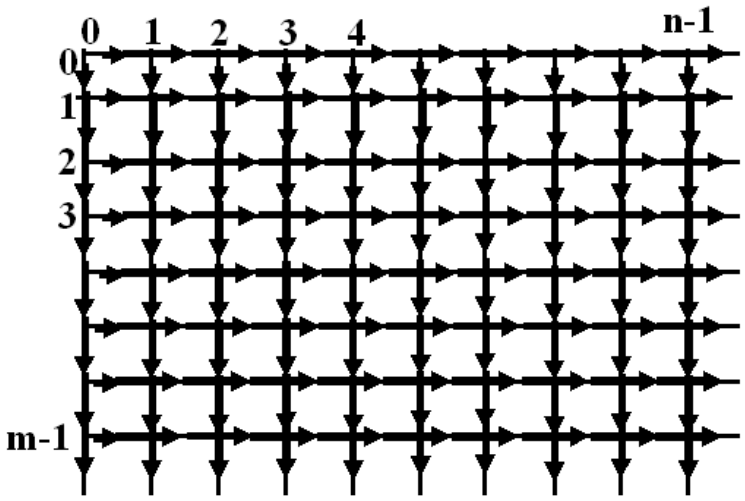


Figure 2.8: Directed toroidal graph representation [3].

$m+n$ arcs, where m and n are the number of points in the contours P and Q. A trail starting at v_{i0} and ending at $v_{m+i,n}$, has m vertical arcs and n horizontal arcs. Fuchs *et al.* proved that there can be $(m+n)!/m!n!$ number of possible paths for the above trail [3] considering all possible permutations. Hence, the number of acceptable surfaces for a graph G is exponential. Therefore, to obtain an optimal surface from all acceptable surfaces, an additional criterion has to be satisfied. To do this, each arc $\mathrm{A}\langle V_{kl}, V_{st}\rangle$ in S is assigned a cost $C(\langle V_{kl}, V_{st}\rangle)$. For example, for any arc, the cost can be the surface area or perimeter of the associated triangle. The cost of a trail is the sum of the cost of the arcs contained in it. The surface of best "quality" is the trail with minimum cost. If the cost chosen is surface area, then the optimal surface among the acceptable surfaces is the one with minimum total surface area.

The problem of finding an optimal *trail* in a toroidal graph G corresponds to finding an optimal *path* in the corresponding planar graph $G^{'}$. A path is a trail in which no vertices are repeated. Hence unlike G, $G^{'}$ has no cycles. It can be obtained by gluing together two copies of G. A path from v_{i0} ends at $v_{m+i,n}$ in $G^{'}$.

The optimal path is found by finding $\pi[i]$ for all $i \in (0, m-1)$, where $\pi[i]$ represents an optimal path starting from $v_{i,0}$ to $v_{m+i,n}$, and then choosing the one with minimum cost from among these m paths. That is, $\pi[0]$, $\pi[1]$,...,$\pi[m-1]$ is found, and the one with the

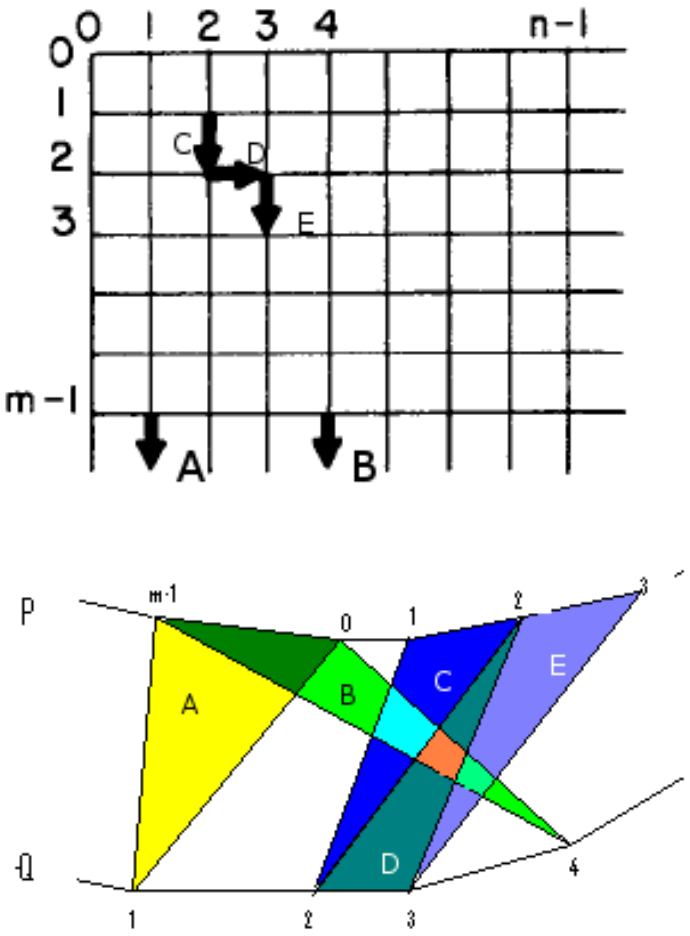


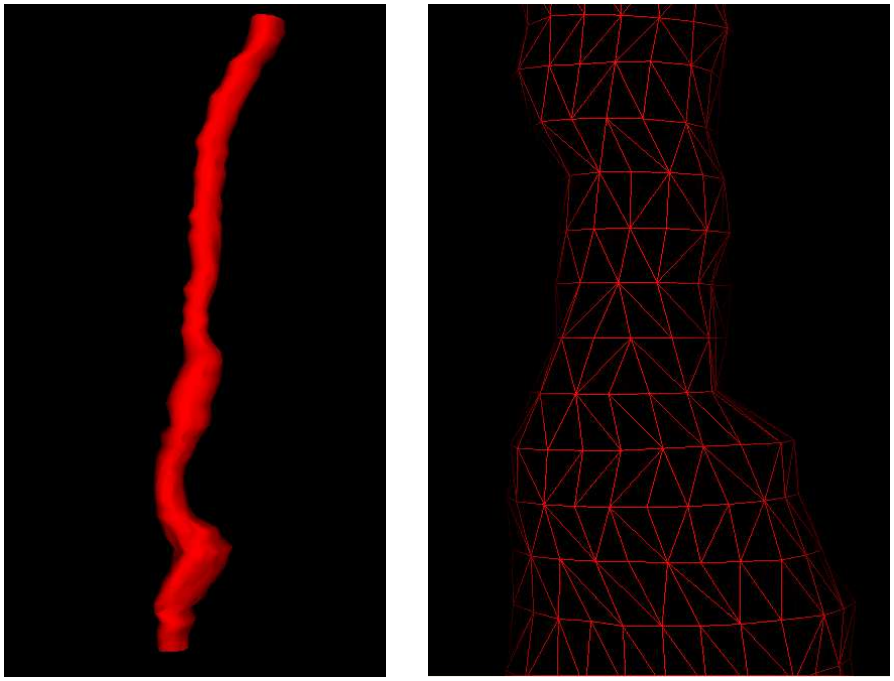
Figure 2.9: Correspondence between a subgraph and a set of tiles [3].

minimum cost is the optimal path π.

The optimal paths $\pi[0]$, $\pi[1]$,...,$\pi[m-1]$, $\pi[m]$ are found using the following theorem: If $\pi[i]$ is the minimum cost path from $v_{i,0}$ to $v_{m+i,n}$, then there exists a minimum cost path $\pi[j]$ from $v_{j,0}$ to $v_{m+j,n}$ which does not cross $\pi[i]$, but can share vertices or arcs with $\pi[i]$ from Theorem 2 of [3]. By this theorem, $\pi[k]$, which is the minimum cost path from $v_{k,0}$ to $v_{m+k,n}$, where $0 < i < k < j < m$, can be found by searching the graph $G'(i,j)$, where $G'(i,j)$ is a subgraph of G' and is spanned only by the vertices between $\pi[i]$ and $\pi[j]$ ($V'(i,j)$). This means that any single minimum cost path $\pi[k]$ between $\pi[i]$ and $\pi[j]$, does not cross either the path $\pi[i]$ or $\pi[j]$.

$\pi[m]$ is found by finding $\pi[0]$ and shifting the path down to m in G'. Thus, $\pi[0]$ to $\pi[m-1]$ can be found by first searching $G'(0,m-1)$ to get $\pi[(m-1)/2]$, and thereby subdividing $G'(0,m-1)$ into smaller subgraphs till all the paths are found. Figure 2.10

shows a vas segment from prostate data generated using this algorithm.



(a) Vas from prostate data. (b) Close up wireframe view.

Figure 2.10: Model of vas segment from prostate data obtained using Fuchs optimal algorithm.

2.2.3 Assessment of the triangulation algorithms

Improvements based only on the variations in triangulations have limitations. That is, for our data-set, the final models obtained using just a triangulation algorithm exhibit a segmented appearance, despite having detailed contours, because of the large inter-slice distance. Even though optimal triangulation algorithms can produce an optimal surface with respect to some metric, a large inter-slice distance gives the models a segmented surface rather than a smooth and organic-looking surface. This is the main reason for exploring various interpolation methods, which produces a sequence of intermediate slices between any pair of given slices so that they are very close to each other as well as geometric similar to each other. The next section explores various interpolation methods.

2.3 Interpolation methods

Data sets like DREM are obtained by sectioning an object at different intervals. Straightforward triangulations of such data using an optimal or a heuristic algorithm do not look smooth for a variety of reasons, one reason being the inter-slice distance. Reducing the distance between individual slices reduces the faceted appearance of models. Shrinking the inter-slice distance causes triangles to be "more equilateral". That is, it eliminates the elongated triangles that can appear in some models due to the large inter-slice distance. Hence, one way of generating smoother models is to automatically generate multiple slices in-between two original slices, and triangulating them using an optimal algorithm. This would replace one row of triangles between any two original slices with $n+1$ rows of triangles, where n is the number of automatically generated intermediate slices between those original slices.

To achieve this automatic generation of multiple slices, the given contour data set has to be interpolated. *Interpolation* is the process of computing new intermediate data values between existing data values [19]. We wish to find interpolated slices between every pair of given slices.

Scattered data interpolation methods are reviewed in [20]. As mentioned in [20], the type of interpolation method to be applied on a particular data set depends on various factors such as density of the data, level of smoothness required, computational costs involved, and the application for which it will be used. Interpolation methods include triangulation based methods, inverse distance methods, and radial basis function methods. The next few sections summarize these methods and explain why variational interpolation was chosen for our work.

Interpolation methods can also be classified as global and local. Global methods use all control points in the original data set to find an interpolation function $f(x)$, whereas local methods use only a neighborhood of points for generating an interpolation function. Hence, global methods are sensitive to changes in data. Insertion or deletion of just a single point will change the interpolation function, thereby changing the values of every surface point. Additionally, using global methods on very large data sets can be computationally

expensive.

Interpolation using local methods can be computationally cheaper than global methods, as they use only a small subset of points in their local neighborhood to find the function. As well, any changes to data outside a local domain will not affect interpolation inside the domain. Choosing the right method should depend on the nature of given problem, for example, whether the problem is interpolation of temperatures in a given space, or if it involves interpolating points for modeling medical data. It also depends on the level of smoothness required (C^0, C^1 or C^2 continuity). The continuity between any two curve segments is determined by the tangent vectors at the point where the segments join [21]. C^0 continuity ensures that the two curve segments join, that is, it ensures that there are no breaks in the defined curve. If tangent vectors (first-order derivatives) of two curve segments are equal at the join point, the curve has C^1 continuity. For a C^n continuous curve, the n^{th} order derivatives of any two curve segments should be equal at the join point. C^1 continuity is a minimum requirement for any two curve segments to join smoothly, whereas C^2 continuity ensures a higher level of smoothness. A polynomial with a degree of at least two (quadratic polynomial) is required to represent the piecewise segments, to achieve C^1 continuity, whereas a polynomial with degree of at least three (cubic polynomial) is required to represent the piecewise segments, for achieving C^2 continuity. Figure 2.11 shows how two curve segments join based on the level of continuity.

2.3.1 Triangulation based methods

Triangulation based interpolation methods are subdivided into linear triangular interpolation, barycentric interpolation, and cubic triangular interpolation.

This method essentially requires a pre-processing step, in which the scattered data is triangulated. Any optimal triangulation that avoids thin and elongated triangles can be used. The piecewise triangular surface generated by a scattered data set on the $\{x,y\}$ plane is called a *triangulated irregular network* or *TIN*.

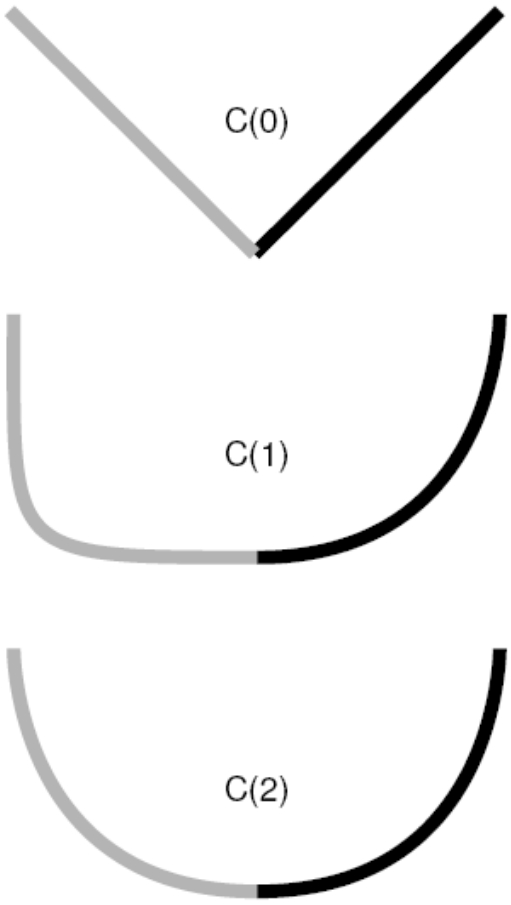


Figure 2.11: C^0, C^1 and C^2 continuity [4].

Barycentric interpolation

Let $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$. Then, an interior point P of triangle $P_1P_2P_3$ can be expressed as a weighted average of P_1, P_2, and P_3 as:

$$P = a_1P_1 + a_2P_2 + a_3P_3. \tag{2.4}$$

The coefficients a_1, a_2 and a_3 are called the *barycentric coordinates*. The interpolated value z at P is the weighted average of values z_1, z_2, and z_3 of points P_1, P_2 and P_3 respectively and is given by:

$$z = a_1z_1 + a_2z_2 + a_3z_3. \tag{2.5}$$

The barycentric coordinates a_1, a_2, and a_3 are obtained by solving the following sys-

tems of equations:

$$x = a_1x_1 + a_1x_2 + a_1x_3$$
$$y = a_2y_1 + a_2y_2 + a_2y_3$$
$$1 = a_1 + a_2 + a_3.$$

The interpolated values obtained from linear triangular and barycentric interpolation methods are identical as the equation of plane passing through three distinct points in space is the same.

Limitations of linear interpolation Even though barycentric interpolation is easy to implement, when this method is used for surfacing models, the surfaces formed using this method have a faceted appearance because of derivative discontinuities at boundaries of adjacent triangles. This method does not round out the surface but only creates more smaller triangles within every triangle of the *TIN*. It gives C^0 continuity, but not C^1 continuity. As these methods find interpolated values within the convex hull of a given scattered data set, it is impossible to extrapolate values.

Cubic triangular interpolation

Surfacing using linear triangular interpolation is not smooth because the planar surfaces of this interpolation are of degree one, which provides only C^0 continuity. To achieve smoother models, the planar surfaces over every triangle are replaced with a curved triangular surface. This is done by applying a triangular Bezier surface [5], [22] on each triangle of the triangular irregular network, using cubic polynomials ([23], [24], [25], [21]) . A recent implementation of this approach is called the *Curved PN triangles* or *Normal patches* [5].

Like linear triangular interpolation, this method requires the scattered data set to be triangulated as a pre-processing step. Once the data set is triangulated, each triangle of the TIN is further subdivided based on a specified *Level of Detail* (LOD). LOD is defined as the number of evaluation points on each edge of a triangle minus two. That is, if LOD is zero, it means there is no further subdivision and the original triangle is returned, if LOD

= 1, then one extra point is added between the two vertices of each edge, subdividing the original triangle into four triangles, and if LOD = 2, there are two points per edge between the corner vertices and the original triangle is subdivided into nine smaller triangles (Figure 2.12).

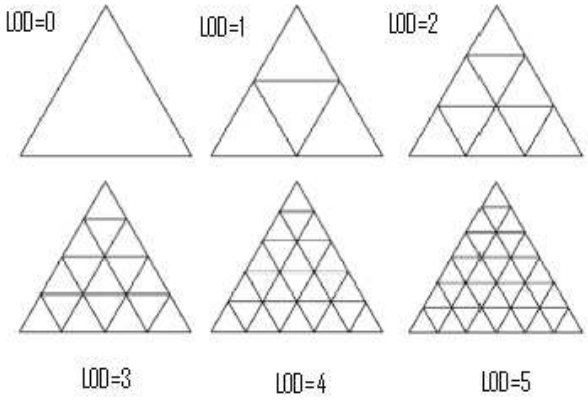


Figure 2.12: Subdivision of original triangle based on LOD [5].

Finding the curved *PN* triangles begins with defining a normal for each corner point of the original triangle as in Figure 2.13. Then a *cubic Bézier patch* is defined over each triangle of the TIN. A cubic Bézier patch is a 3D generalization of a Beźier curve. A cubic Bézier curve is defined using four control points. The curve interpolates the starting and end control points, and the two remaining points influence the shape of the curve but are not interpolated. The common parametric form of a Bézier curve is:

$$p(u) = \sum_{k=0}^{n} P_k B_{k,n}(u), \tag{2.6}$$

where $B_{k,n}(u) = C(n,k)u^k(1-u)^{(n-k)}$ is the basis or the blending function for a Bézier curve and is called a Bernstein polynomial, and where $C(n,k) = n!/k!(n-k)!$ is the binomial coefficient, and the P_k are the control points. Varying u from 0 to 1 generates a smooth curve that blends the P_k.

The curved *PN* triangles are formed using Bézier triangles. A point p in a triangle $<P_1$ P_2 $P_3>$, is expressed in barycentric coordinates as

$$\begin{aligned} p(u,v) &= P_1 + u(P_2 - P_1) + v(P_3 - P_1) \\ &= wP_1 + uP_2 + vP_3, \end{aligned}$$

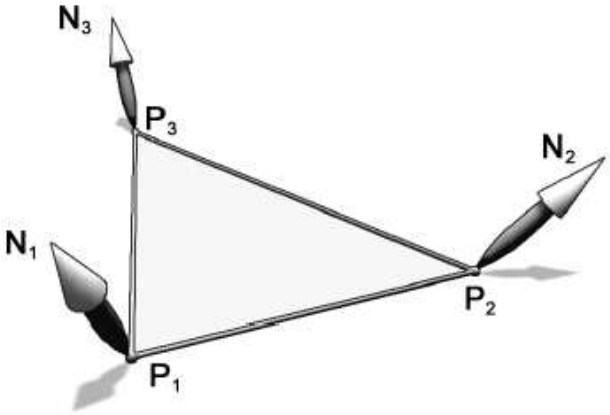


Figure 2.13: Vertices and normals of original triangle [5].

where $w = 1 - u - v$.

Bézier triangles have the form

$$p(u,v) = \sum_{i+j+k=m} B^m_{ijk}(u,v) P_{ijk}, \tag{2.7}$$

where

$$B^m_{ijk}(u,v) = \frac{m!}{i!\,j!k!} u^i v^j w^k, \tag{2.8}$$

$i + j + k = m$.

A Bézier patch is defined using 10 control points and has three important properties. A Bézier patch interpolates the three corner points of the triangle, each edge of the triangle is a Bézier curve defined using four control points in the given edge, and the surface always lies in the convex hull of the control points.

The formula for a cubic Beźier patch is [5]

$$\begin{aligned} p(u,v) = b_{300}u^3 + 3b_{210}u^2v + 3b_{120}uv^2 + b_{030}v^3 + \\ +3b_{021}v^2w + 3b_{012}vw^2 + b_{003}w^3 + \\ +3b_{102}uw^2 + 3b_{201}u^2w + 6b_{111}uvw. \end{aligned}$$

In the above equation, b_{300}, b_{030}, b_{003} are the corner vertices and are also called the *vertex coefficients*. b_{210}, b_{120}, b_{021}, b_{012}, b_{102}, b_{201} are the *tangent coefficients* and b_{111} is the *center coefficient*. They are also called the control points or the control net (Figure 2.14).

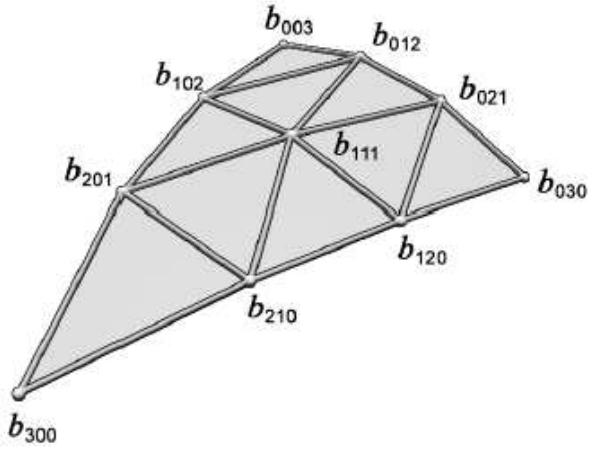


Figure 2.14: Control net [5].

As the curved *PN* triangle interpolates the corner points of the original triangle, the vertex coefficients are left in place to match the corner points. The rest of the geometry coefficients, namely the tangent coefficients and the center coefficient can be determined from the vertex coefficients and the normals at the respective vertex coefficients. The two tangent coefficients near each corner, are projected into the tangent plane defined by the normal at the corner point. The projection of a point X onto a plane with normal N at point P is given as

$$X^{'} = X - wN, \tag{2.9}$$

where $W = (X - P) \cdot N$. Hence, with respect to the above equation, the tangent coefficients are computed using the vertex coefficients and the normals (Figure 2.15) and are given as

$$\begin{aligned}
b_{210} &= \frac{(2P_1 + P_2 - w_{12}N_1)}{3}, \\
b_{120} &= \frac{(2P_2 + P_1 - w_{21}N_2)}{3}, \\
b_{021} &= \frac{(2P_2 + P_3 - w_{23}N_2)}{3}, \\
b_{012} &= \frac{(2P_3 + P_2 - w_{32}N_3)}{3}, \\
b_{102} &= \frac{(2P_3 + P_1 - w_{31}N_3)}{3}, \\
b_{201} &= \frac{(2P_1 + P_3 - w_{13}N_1)}{3},
\end{aligned}$$

where $P_1 = b_{300}$, $P_2 = b_{030}$, $P_3 = b_{003}$, and $w_{ij} = (P_j - P_i).N_i$. The center coefficient, b_{111} is given as:

$$b_{111} = E + \frac{E - V}{2}, \tag{2.10}$$

where $E = \frac{b_{210}+b_{120}+b_{021}+b_{012}+b_{102}+b_{201}}{6}$, and $V = \frac{b_{300}+b_{030}+b_{003}}{3}$. Figure 2.16 shows a vas segment generated using this method.

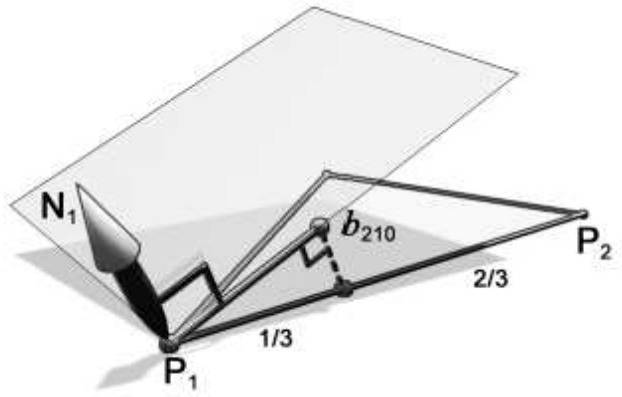
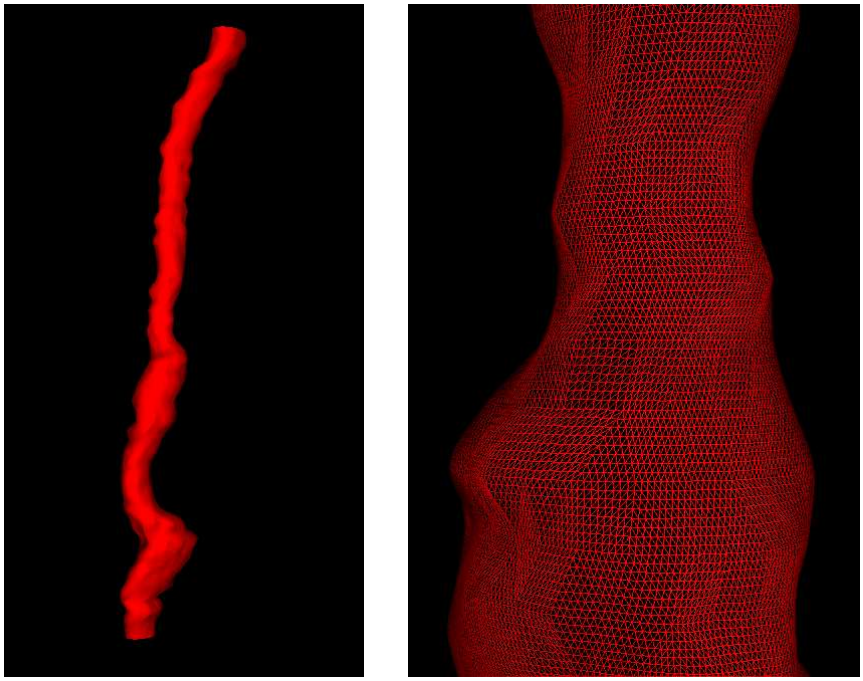


Figure 2.15: Calculating a tangent coefficient[5].

Assessment As this method is local, it can easily accommodate large data sets. As this method is C^1 continuous the surface looks smoother. But it requires pre-processing (triangulation) of the original data set. Like the linear method, this method cannot find extrapolated values. Using it on large models with triangles of greatly variable area requires some care. When used on a whole model, this method also subdivides triangles that are quite small and results in *Z-fighting* problems on some models. Z-fighting is a phenomenon in 3D rendering which occurs when two or more coplanar primitives have similar values in Z buffer, causing random parts of the primitives to be rendered [26]. Figure 2.16 shows a model generated using this method.

2.3.2 Inverse distance weighted interpolation

Inverse distance weighted interpolation, also known as Shepard's method [27] is used because of its simplicity. It is a global method and hence all data points are considered for evaluating the interpolated value of a point P. The interpolated value of P at (x,y) is



(a) LOD model of vas structure from prostate data.

(b) Close up wireframe view.

Figure 2.16: Model of vas segment from prostate data obtained using cubic triangular interpolation.

the weighted average of the values of all scattered data points. The weights depend on the distance between the scattered data points S_i and the point P to be interpolated, where $i \in [1,\ldots,n]$ and n is the number of points in scattered data set. The weight increases if the distance between S_i and P decreases, and decreases if distance between S_i and P increases. Hence, in the inverse distance weighted interpolation method, the value of P at (x,y) is influenced more by nearby points and less by points farther away from P and is given by

$$F(x,y) = \sum_{i=1}^{n} w_i z_i, = \sum_{i=1}^{n} \left\{ \frac{h_i}{\sum_{j=1}^{n} h_j} \right\} z_i, \tag{2.11}$$

where z_i is the value of P_i and w_i is the weight at P_i and is given by

$$w_i = \frac{h_i}{\sum_{j=1}^{n} h_j}, \sum_{i=1}^{n} w_i = 1, where\ h_i = \frac{1}{d_i^{\,k}}, and\ d_i = \sqrt{(x-x_i)^2 + (y-y_i)^2}, \tag{2.12}$$

where (x_i, y_i) is a scattered data point and (x,y) is an interpolated point. Different values of k result in different interpolated values. An extension to the original Shepard's method is given in [20].

A way of localizing this method is given in [28]. The h_i at (x,y) in equation 2.12 is replaced with

$$h_i = \left\{ \frac{[R-d_i]_+}{Rd_i}^2 \right\}, where\ [R-d_i]_+ = \begin{cases} R-d_i & \text{if} \quad d_i < R \\ 0 & \text{if} \quad d_i \geq R, \end{cases} \tag{2.13}$$

where d_i is the Euclidean distance between (x,y) and (x_i,y_i), and R is the radius of influence about (x_i,y_i). This makes the value of the interpolated point P influenced only by the scatter points within this radius. The surface obtained from this interpolation is C^1 continuous [20].

Assessment As this method is based on distance, it tends to give too much weight to data clusters. Unlike triangulation based methods, it is possible to extrapolate outside the convex hull of the given data points.

2.3.3 Radial basis functions (RBF)

The method of *radial basis function* typically uses all control points and is an important method used to perform scattered data interpolation in various medical and graphics applications ([29], [30]). This method is global and can be C^2 continuous and hence is preferred for generating smoother interpolants. This method was first suggested by Hardy [31].

It starts by defining a RBF for every data point, such that

$$f(x,y) = \sum_{i=1}^{n} d_i \phi_i(x,y). \tag{2.14}$$

The response of a *radial function* decreases or increases monotonically with distance from a central point. A radial function is of the form $\phi_i(x,y) = \phi(d_i)$, where d_i is the distance between the input point (x,y) and a data point(x_i,y_i). Choices for radial basis functions include thin-plate splines ($r^2 logr$), Gaussian ($\phi(r) = \exp(-cr^2)$), and multiquadric ($\phi(r) = \sqrt{(r^2+c^2)}$), where r is the radius or distance from the origin.

Given n data points, each point (x_i,y_i) in the scattered data set has an associated value

z_i that the radial basis functions must interpolate as follows

$$\begin{aligned} z_1 &= \sum_{i=1}^{n} d_i \phi_i(x_1, y_1) \\ &\vdots \\ z_n &= \sum_{i=1}^{n} d_i \phi_i(x_n, y_n) \end{aligned}$$

In matrix form, the preceding can be represented as: $z = Md$, $d = M^{-1}z$, where

$$\begin{aligned} z &= (z_1 \ldots z_n); \\ M &= (\phi_i(x_1, y_1) \ldots \phi_i(x_n, y_n)); \\ d &= (d_i \ldots d_n). \end{aligned}$$

Hence, n control points give n equations that can then be solved for the weights d_i. Then the value of any point (x,y) can be found by inserting the weights d_i into Equation 2.14. This interpolation method gives a smooth surface. The pre-processing step involves finding the coefficients of the interpolation function.

Assessment This interpolation method gives much smoother interpolants than all the other methods discussed here. As this method is global, processing a large data set can be expensive. This computational difficulty can be resolved by splitting the data sets into separate domains, each including only a small subset of the data points. The final interpolation function will be the weighted sum of the interpolation functions of all domains. The variational interpolation technique of Turk and O'Brien discussed in the next chapter uses thin-plate splines as the radial basis function.

There are three ways of exploiting RBF's using scattered data interpolation methods. They are the naive methods, such as those of Turk and O'Brien, the fast-fitting methods such as that of Beatson *et al.*, and the compactly supported RBF's [32], [33], [34], [35]. In compactly supported RBF's, the basis functions have a piecewise polynomial profile function and different radius of supports, which depends on the desired continuity of the polynomials and the dimension of the space from where the data is drawn [36].

2.4 Variational interpolation

This section explores the approach of variational interpolation. The basic approach, an application of scattered data interpolation, creates an implicit function that uses a weighted sum of radial basis functions, one per input point. This approach has some problems related to the input data size. One is that solving the coefficients requires solving a system of linear equations. As models become large, simple solvers become unstable. Moreover, the resulting implicit function is expensive to evaluate.

2.4.1 Implicit functions

Consider the equation:

$$f(x,y,z) = 0, \tag{2.15}$$

The function f implicitly represents all (x,y,z) points that satisfy the above equation. Therefore, the surface formed from all the x,y,z points satisfying Equation 2.15 is called an implicit surface [37].

For a given center (c_x, c_y) and radius r, implicit representation of a circle is

$$f(x,y) = (x-c_x)^2 + (y-c_y)^2 - r^2 = 0, \tag{2.16}$$

and parametric representation is

$$f(x,y) = (c_x + r\cos\theta, c_y + r\sin\theta), \theta \in [0, 2\pi], \tag{2.17}$$

where (x,y) are the points on the circumference of a circle. Even though finding a point on the circumference of a circle is relatively easier in a parametric form than in an implicit form, it is much simpler to determine if a point lies on or inside/outside a surface using the implicit form. Because of this advantage over the parametric representation, implicit surface representation is increasingly used in applications such as collision detection [38], shape transformation, and surface reconstruction.

Turk and O'Brien [14] use *variational interpolation* with the implicit function representation to solve the scattered data interpolation problem. Variational interpolation is the generalization of 2*D thin-plate spline* interpolation to higher dimensions.

2.4.2 Variational technique

Figure 2.17 shows a thin plate spline that passes through 15 control points. It derives its name from the behavior of a thin metal plate, because, a metal plate, when forced through a set of control points takes the form in which it is least bent, as in Figure 2.17. The basis function for a thin plate spline is $r^2 \log(r)$, where r is the distance from a center. This basis function increases in value with distance from the center. Using a thin plate spline as a radial basis function is a conventional method for performing scattered data interpolation [39]. It ensures C^2 continuity and hence produces smooth interpolants. Thin plate spline interpolation is used in fields such as medical surface reconstruction where smoothness of a model is a primary concern. It is a global method, since the RBF considers all the control points.

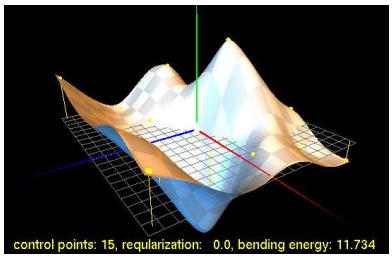


Figure 2.17: Thin plate spline [6].

The problem of scattered data interpolation is stated as follows. Given n data points $\{c_1, c_2, \ldots, c_n\}$ scattered on a xy-plane along with corresponding scalar height values $\{ h_1, h_2, \ldots, h_n \}$, find the smooth surface that interpolates each height at the given locations. That is, find a smooth function $f(x)$ that passes through a given set of data points.

The smoothness of an interpolating function $f(x)$, is determined by the bending energy E, which is a measure of the quality of the interpolating function, and is given as

$$E = \int \int_{\mathbb{R}^2} (\frac{\partial^2 f}{\partial x^2})^2 + 2(\frac{\partial^2 f}{\partial x \partial y})^2 + (\frac{\partial^2 f}{\partial y^2})^2 dxdy, \tag{2.18}$$

where E is a measure of the aggregate squared curvature of $f(x)$ over a region $\mathbb{R}^2$, $\frac{\partial^2 f}{\partial x^2}$ is the second x-partial derivative, $\frac{\partial^2 f}{\partial y^2}$ is the second y-partial derivative, and $\frac{\partial^2 f}{\partial x \partial y}$ is a mixed derivative. Duchon [40] showed that thin plate interpolation minimizes E. Regions with high curvature have higher value of E and regions with less curvature have lesser value of

E. Hence, the problem of scattered data interpolation can be solved using the variational technique, which finds a function $f(x)$ that minimizes E, as well as interpolates each height $\{h_1, h_2, \ldots, h_n\}$ at the given locations $\{c_1, c_2, \ldots, c_n\}$.

Duchon [40] has shown that the function

$$f(x) = \sum_{j=1}^{n} d_j \phi(x - c_j) + P(x), \tag{2.19}$$

minimizes E, in addition to satisfying the constraint

$$f(c_i) = h_i. \tag{2.20}$$

In Equation 2.19, $\phi(r) = |r|^2 log(|r|)$, where r is the distance from a center c_j, d_j are weights, and $P(x)$ is a degree one polynomial. Equation 2.19 is solved for the weights and polynomial coefficients using Equation 2.20 as follows

$$h_i = \sum_{j=1}^{k} d_j \phi(c_i - c_j) + P(c_i). \tag{2.21}$$

The side conditions to this system are

$$\sum_{i=1}^{k} d_j = 0 \qquad \sum_{i=1}^{k} d_j x_i = 0 \qquad \sum_{i=1}^{k} d_j y_i = 0 \qquad \sum_{i=1}^{k} d_j z_i = 0. \tag{2.22}$$

The side condition or the orthogonality condition is interpreted in three ways by Beatson *et al.*: It makes the measure of the smoothness of an RBF finite, controls the rate of growth at infinity of the non-polynomial part, and takes away the degrees of freedom added by the polynomial part [15], [41], [42].

Equation 2.21 can be represented in a matrix form as shown in Equation 2.23 and can be solved using any standard technique.

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \ldots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \ldots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \ldots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \ldots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \ldots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \ldots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \ldots & c_k^z & 0 & 0 & 0 & 0, \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p0 \\ p1 \\ p2 \\ p3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.23}$$

where $c_i = (c_i^x, c_i^y, c_i^z)$ and $\phi_{ij} = \phi(c_i - c_j)$. A standard linear solver LU decomposition method [43], [44] is used to solve the matrix of Equation 2.23. The solution of Equation 2.23 gives an implicit function called the variational implicit function, that minimizes E and satisfies the interpolation constraints of Equation 2.21.

2.4.3 Shape transformation using variational technique

In a shape transformation problem, the initial shape is transformed into the final shape by constructing a sequence of intermediate shapes such that adjacent slices in the shape transformation sequence are geometrically similar. Turk and O'Brien developed a shape transformation technique that uses a single implicit function to describe the transformation of one shape to another, instead of two implicit functions (one for the initial shape and one for the final shape) as required by the older methods [45], [46], [47], [48]. This is done by first defining *boundary constraints* and *normal constraints* (Equation 2.21) on the initial and the final shapes. Boundary constraints, written as $f(b_i) = 0$, consist of surface points b_i and are assigned a height value of zero. Normal constraints, written $f(n_i) = 1$, consist of interior normal points n_i and are assigned a height value of one. The vertices marked on the contour boundaries represent the surface points. The interior normal points are calculated as follows. The parallel slices are triangulated using Fuchs algorithm (section 2.2.2), and the surface normals are found by taking the cross product of the two edges of a triangle. The vertex normal V_N is found by averaging the surface normals surrounding V_N (Figure 2.18), and the interior normal constraint point n_i for a surface constraint point is the point ¡x,y,z¿ in the direction of unit vertex normal V_N from the corresponding surface point.

In Equation 2.23, column $h_1, \ldots, h_k$ is filled with ones and zeros depending on whether the constraint point (c_i^x c_i^y c_i^z) in row i is a boundary or a normal point. The surface obtained using the variational routine exactly passes through all the boundary points.

Figure 2.19 shows the gradual transformation of shape X to shape O. Turk and O'Brien achieve this $2D$ shape transformation by casting a N dimensional problem as a $N+1$ dimensional problem. That is, the $2D$ problem of X to O transformation is cast as a $3D$ shape transformation problem, by embedding the $2D$ boundary and normal constraints of

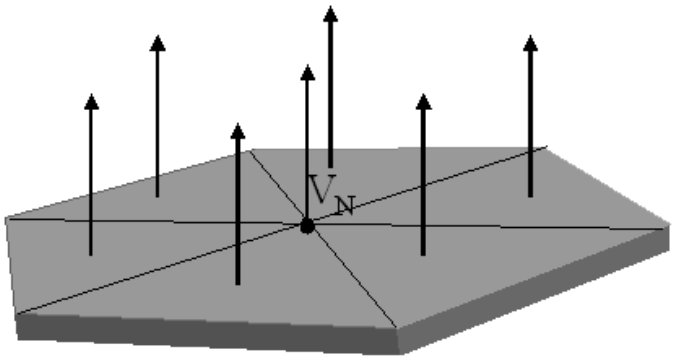
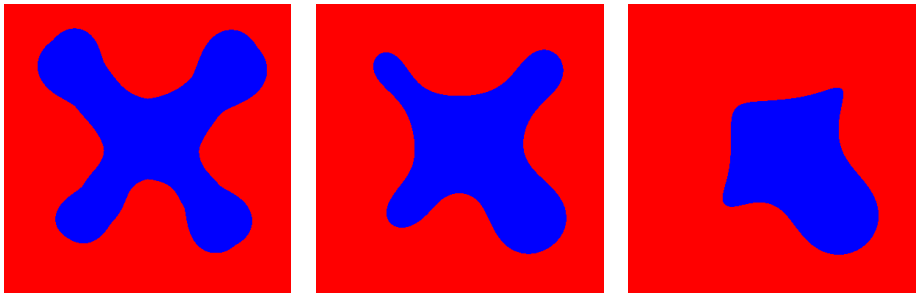


Figure 2.18: Normal constraints

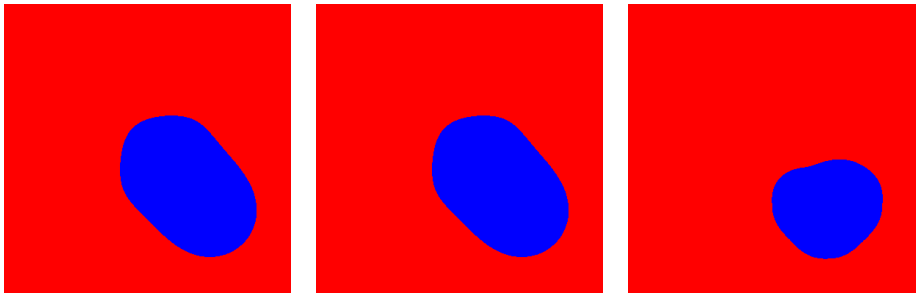
X and O shapes as $3D$ constraints in Equation 2.23 by adding a third coordinate t to each constraint. This coordinate represents the height of each slice. The coordinate t, takes a value of zero for the initial shape, and some non-zero value t_{max} for the final shape. In the X to O shape transformation, the boundary and the normal constraints of X have $t = 0$, and the boundary and normal constraints of O have $t = t_{max}$. Thus, the $2D$ constraint points of both shapes have been changed to $3D$ constraint points. For ease of understanding, this $3D$ representation can be thought of as plotting constraint points of X on the $t = 0$ plane and plotting constraint points of O on the $t = t_{max}$ plane.

If the implicit function of Equation 2.19 returns a value zero for a point (x, y), it means the point lies on the surface. If it returns a value greater than zero, the point lies inside the surface, whereas if the returned value is less than zero, the point lies outside the surface.

A slice of this implicit function taken at $t = 0$ represents the first shape and a slice taken at $t = t_{max}$ represents the final shape. The intermediate shapes in Figure 2.19, are obtained by taking a slice of this function at different values of t. The first shape is obtained by taking a slice of the implicit function at $t = 10$ and the final shape at $t = 200$. The intermediate shapes are obtained by taking a slice of the function at $t = 59$, $t = 79$, $t = 109$, and $t = 159$.



(a) Step 1 (b) Step 2 (c) Step 3



(d) Step 4 (e) Step 5 (f) Step 6

Figure 2.19: Two-dimensional shape transformation sequence

2.4.4 Surface reconstruction using variational technique

Figure 2.20 shows two views of a surface reconstruction done using the variational technique and Figure 2.21, shows similar views of a surface reconstruction done using WinSurf for the same data points.

In medical modeling, a surface has to be reconstructed from several 2*D* slices of contour data. That is, the shape transformation technique has to be applied to all given slices to generate the final surface. An easy way to do it is to perform shape transformation between every pair of slices and stack up the results to get the final surface. Figure 2.22 shows a surface reconstructed using this method. It shows that the surface produced by this method has discontinuities along the plane joining the original slices, hence is not a preferred way to perform surface reconstruction.

Turk and O'Brien's method uses a generalization of the shape transformation technique to perform surface reconstruction from several 2*D* slices . This is done by using the constraint points of all the 2*D* slices to compute the variational implicit function. Figure

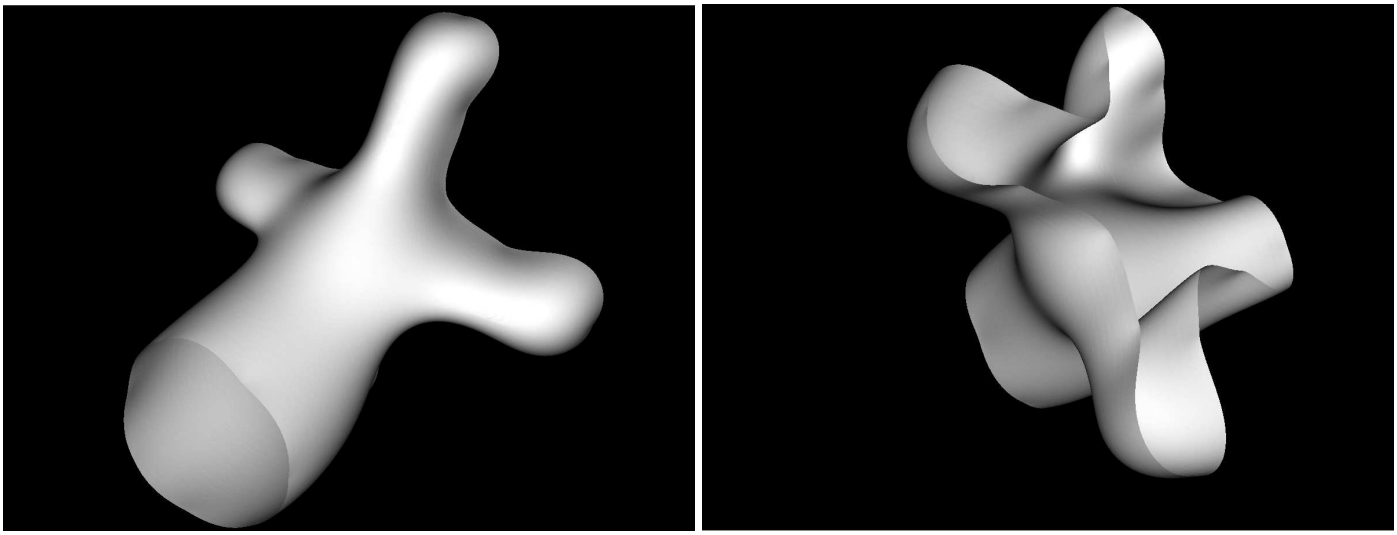


Figure 2.20: 3D view of surface reconstruction produced by Variational Interpolation (showing the *O* and *X* shapes)

2.23 was constructed from the same data as Figure 2.22 using this method.

The implicit function returned by the variational routine describes the entire surface, and it passes through all the boundary constraint points. The value of the implicit function can be found for any positive value of t, even if t extends beyond the planes of the first and the final slices. That is, the reconstructed surface extends beyond the location of the constraint points to give smoother caps at the ends of the surfaces. Figure 2.24 shows a vas segment obtained using this method.

Limitations of this approach

Turk and O'Brien report that their method works well up to a few thousand constraint points, but becomes unstable with large data sets. Because our models often have more than 5000 data points, this method cannot be used directly because the matrix for finding the weights and the polynomial coefficients becomes ill-conditioned.

2.5 Motivation

The triangulation techniques discussed in this chapter created an optimal surface, but as seen in Figure 2.21, an optimal surface does not necessarily mean a visually pleasing

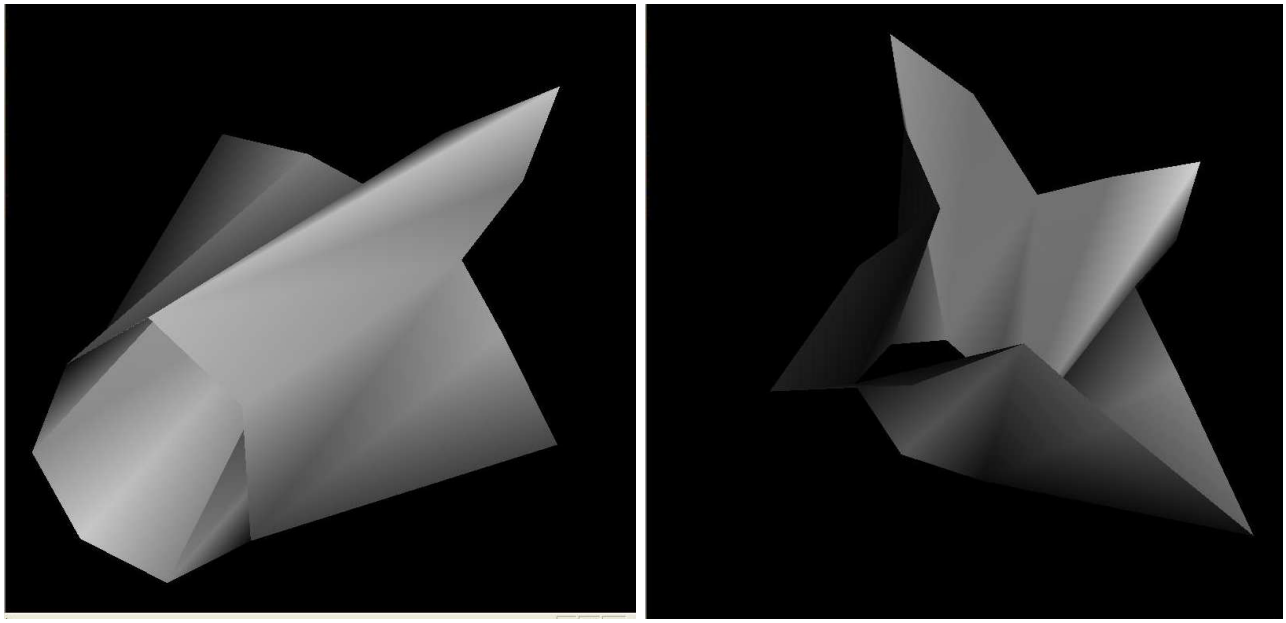


Figure 2.21: 3D view of surface reconstruction produced by WinSurf (showing the *O* and *X* shapes)

smooth surface depending on the optimization criteria. The surface in Figure 2.21 does not look smooth. Hence, triangulating the anatomical slices using just an optimal triangulation algorithm would not accomplish our goal of building smooth anatomical models. Also, even though the interpolation and the RBF techniques discussed in this chapter have disadvantages like slower runtimes, intermediate slices can be generated from the original data using these techniques. This gave us the motivation to pursue a combinative method that makes use of both the triangulation techniques and RBF techniques to solve the problems of instability and the slower runtimes of the previous methods. Our method of *piecewise weighted implicit functions* is explained in the next chapter.

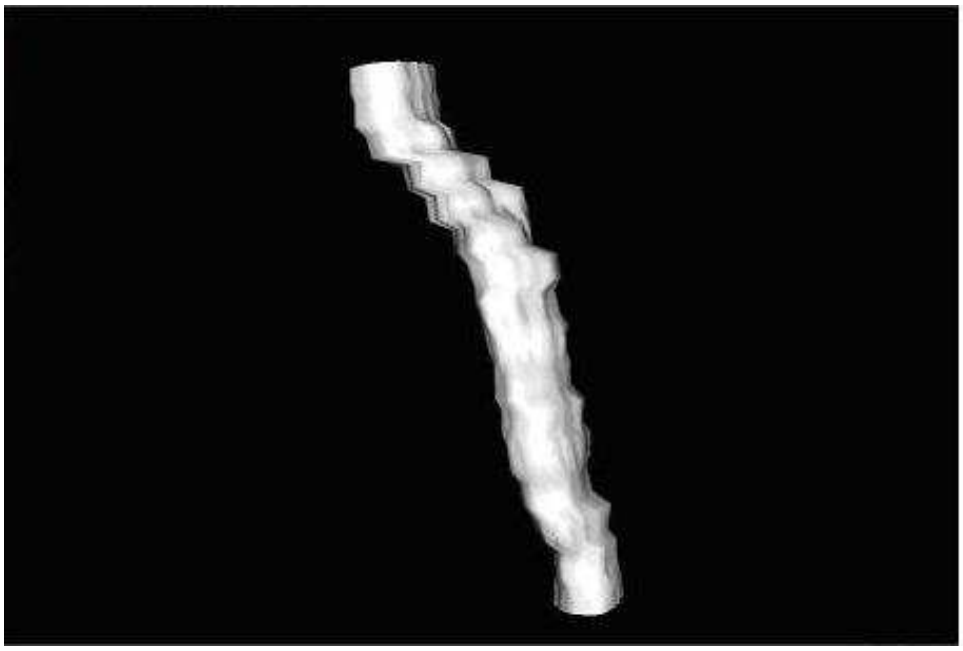


Figure 2.22: Surface reconstruction by performing variational interpolation pairwise on slices

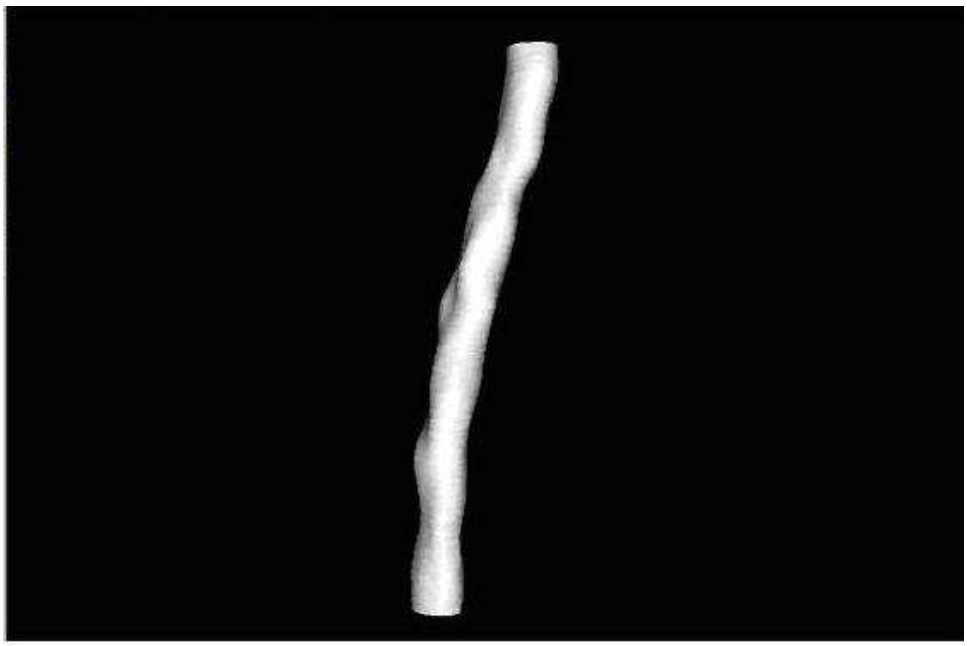
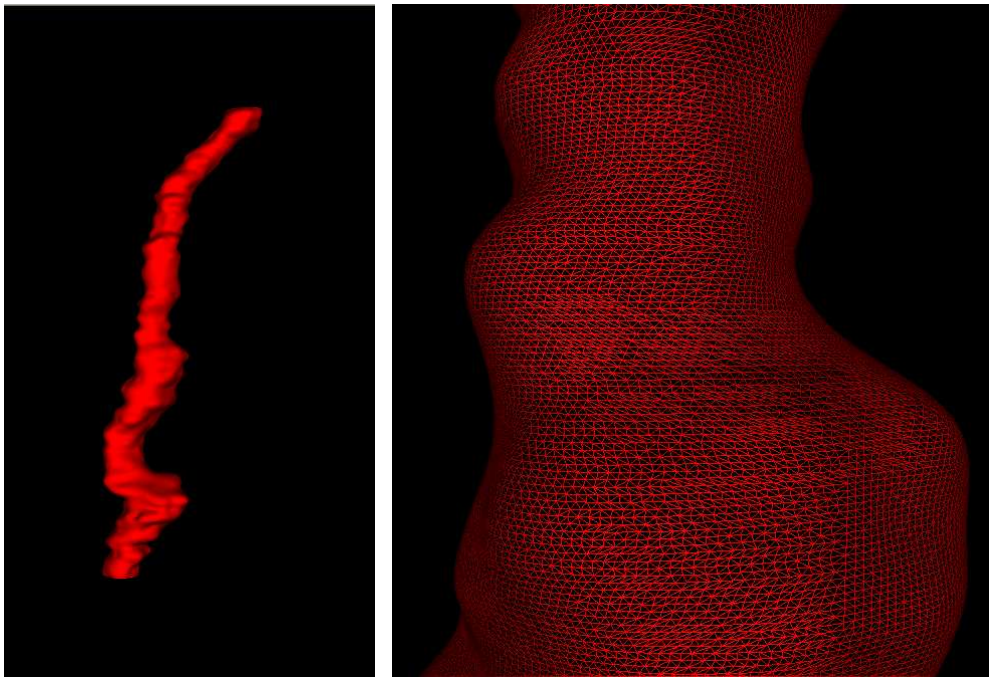


Figure 2.23: Surface reconstruction using variational interpolation on all slices



(a) Vas using original Turk and O'Brien method

(b) Close up wireframe view.

Figure 2.24: Model of vas segment from prostate data obtained using original Turk and O'Brien approach.

CHAPTER 3

STABLE AND EFFICIENT IMPLICIT FUNCTIONS

3.1 Piecewise weighted implicit functions

The variational implicit method of Turk and O'Brien constructs an implicit function consisting of n RBF's, one for each constraint point. The results are very pleasing, but we encounter two problems as models get large, instability and cost of tracing the contours. Turk and O'Brien suggest that for slice-based models all data points be added to the implicit function. To address these problems, we considered two approaches. The first approach uses *preconditioned GMRES iterative* method of Beatson *et al.* [49], which conditions the linear system, but still leaves us with the problem of evaluating the implicit function for large number of constraint points. The second approach is our idea of *weighted implicit functions*. Our solution of weighted implicit functions is both stable to solve and efficient to evaluate. The reason for the instability of the solution matrix (Equation 2.23), is the round-off errors created by the large number of constraint points. Weighted implicit functions avoid this problem by limiting the number of slices given as inputs to the system at a given time. Moreover, using simple triangulations from Chapter 2 generates a fully automatic system that can take contour data, estimate normals and generate interpolated contours.

Our weighted approach finds one implicit function for every window of n slices resulting in k implicit functions. The final surface is represented with k piecewise weighted implicit functions rather than just one as in the Turk and O'Brien approach. The weighting varies with the y-value of the input point, so that the value at each point p is the weighted sum of the implicit functions that influence p along that region. Figure 3.1 shows five implicit functions obtained from eight slices using a sliding window of four slices, and are

computed as follows

$$\begin{aligned} f_1(x,t) &= \textstyle\sum_{i=slice1}^{slice4} d_i\phi(x-c_i)+P_1(x), \\ f_2(x,t) &= \textstyle\sum_{i=slice2}^{slice5} d_i\phi(x-c_i)+P_2(x), \\ &\vdots \\ f_5(x,t) &= \textstyle\sum_{i=slice5}^{slice8} d_i\phi(x-c_i)+P_5(x), \end{aligned}$$

where $f_1(x,t),\ldots,f_5(x,t)$ are the five implicit functions obtained from the eight slices. The weights and polynomial coefficients for each of these implicit functions is obtained by solving them using any standard technique. The size of the sliding window can be set by the user. Such a grouping of the original input slices into smaller subsets avoids the ill-conditioning problem of large matrices. After finding all the implicit functions, the value of a point p is calculated as the weighted sum of the implicit functions influencing p along that region. That is, the final value of a point in a region is obtained as follows:

$$v(x,t) = w_1(t) * f_m(x,t) + w_2(t) * f_{m+1}(x,t) + \ldots + w_l(t) * f_{m+i}(x,t), \qquad (3.1)$$

where $v(x,t)$ is the value of a point at height t, $w_1(t), w_2(t), \ldots, w_l(t)$ are weights of the corresponding implicit functions $f_m(x,t),\ldots,f_{m+i}(x,t)$ at height t, and m and $m+i$ are any values $\in [1,\ldots,5]$.

3.1.1 Determining region of influence

Each implicit function is built using points in a certain region. Likewise, every point contributes to a certain set of functions. Figure 3.2 shows the mapping of all functions $f_1(x,t),\ldots,f_5(x,t)$ to slices $slice_1$ to $slice_8$. In this example, each window consists of four slices. That is, $n = 4$, and $k = 5$. Each function $f_i(x,t)$ is constructed from the points in the slices in the triangle directly under $f_i(x,t)$.

When we render contours, we use a weighted average of all the implicit functions whose triangle contains t. Finally, the weight of the implicit functions at height t is determined by the heights of the associated triangles at t.

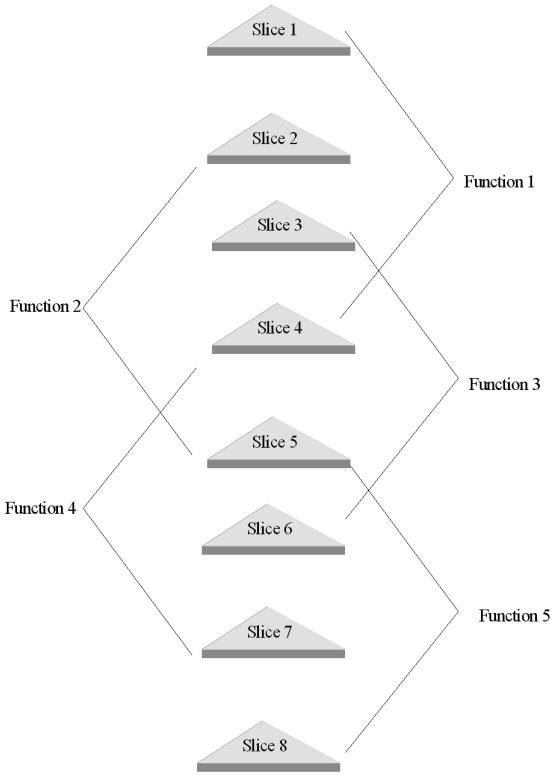


Figure 3.1: Partitioning of slices for n=4

3.1.2 Determination of weights

Figure 3.3, Figure 3.4, and Figure 3.5 shows the influence of $f_2(x,t)$, $f_3(x,t)$, and $f_4(x,t)$ on a point p. As seen in the figures, p is not influenced by $f_1(x,t)$ and $f_5(x,t)$, as the spans of these functions do not include $slice_4$ or $slice_5$. Note that at p, the influence of $f_2(x,t)$ and $f_4(x,t)$ is smaller than the influence of $f_3(x,t)$.

Once the functions influencing p are determined, the influence of each function $f_2(x,t)$, $f_3(x,t)$, and $f_4(x,t)$ at height t, is determined as follows:

$$influence_{f_i(x,t)} = \left\{ \begin{array}{ll} \frac{(t-t_{start_i})}{t_{range_i}} & \text{if} \quad t_{start_i} < t \leq t_{max_i} \\ \frac{(t_{end_i}-t)}{t_{range_i}} & \text{if} \quad t_{max_i} < t < t_{end_i} \end{array} \right\}, \tag{3.2}$$

where $influence_{f_i(x,t)}$ is the influence of $f_i(x,t)$ on a point p at height t, t_{start_i} is the height at which $f_i(x,t)$ begins, t_{end_i} is the height at which $f_i(x,t)$ ends, t_{max_i} is the height at which $f_i(x,t)$ is maximum and is given as

$$t_{max_i} = \left(\frac{t_{start_i} + t_{end_i}}{2} \right), \tag{3.3}$$

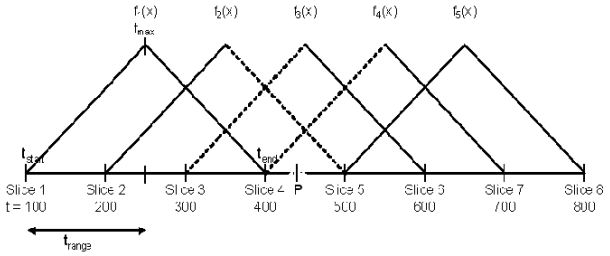


Figure 3.2: Region of influence of all functions based on a point P

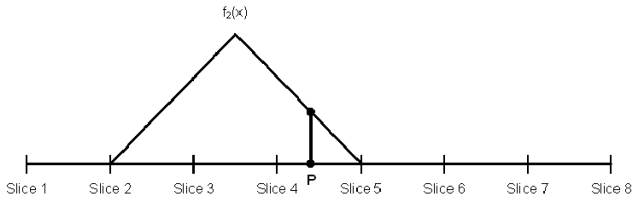


Figure 3.3: Region of influence of function 2

and t_{range_i} is a constant for the size of the region in which the influence grows or declines, that is

$$t_{range_i} = t_{max_i} - t_{start_i}.$$

Usually, we drop the i subscript and write t_{start}, t_{end}, t_{range} and t_{max} when the context is clear.

The total influence of all functions influencing p at t is given as $influence_{total_p}(t)$. The weight of a function is the normalized influence and is given as

$$\begin{aligned} w_2(t) &= \frac{influence_{f_2(x,t)}}{influence_{total_p}(t)}, \\ w_3(t) &= \frac{influence_{f_3(x,t)}}{influence_{total_p}(t)}, \textit{and} \\ w_4(t) &= \frac{influence_{f_4(x,t)}}{influence_{total_p}(t)}, \end{aligned}$$

that is, for a function $f_i(x,t)$ at a point p, $w_i(t)$ is given as

$$w_i(t) = \frac{influence_{f_i(x,t)}}{influence_{total}(t)}. \tag{3.4}$$

The weights add up to one. A graph showing the function influences is given in Figure 3.6. The curved lines show the relative influences of a sequence of implicit functions.

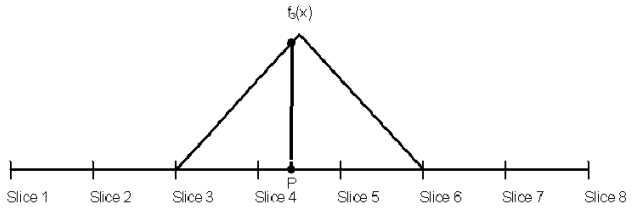


Figure 3.4: Region of influence of function 3

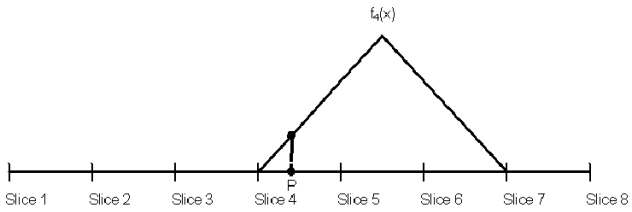


Figure 3.5: Region of influence of function 4

X-axis represent the slices, and Y-axis the relative influences of each implicit function. This graph shows the relative influences of functions for all slices between 2 and 5. The pink plot is the first function, which would be partial, yellow is the second, blue the third implicit function and so on. The region between 2 and 3 is affected by functions 1,2,3. The next region is affected by functions 2,3, and 4, and the next by functions 3,4, and 5. The straight line at the top shows the total weight of the functions, which is 1. We can see that the weights increase and decrease in a linear fashion with some discontinuities at locations of original slices. Despite these discontinuities, the final model appears quite smooth for the size of region of influence we used.

3.1.3 Contour tracing

For a given contour, the intermediate interpolated contours can be found at any desired height level, by tracing the pixels whose value of the implicit function $v(x,t)$ in Equation 3.1 is equal to zero using a contour tracing algorithm called the *Moore-neighborhood* algorithm [50]. For practical purposes, value of an implicit function is considered equal to zero if $v(x,t)$ is very close to zero.

The Moore neighborhood of a pixel *P* consists of the eight pixels that share a vertex

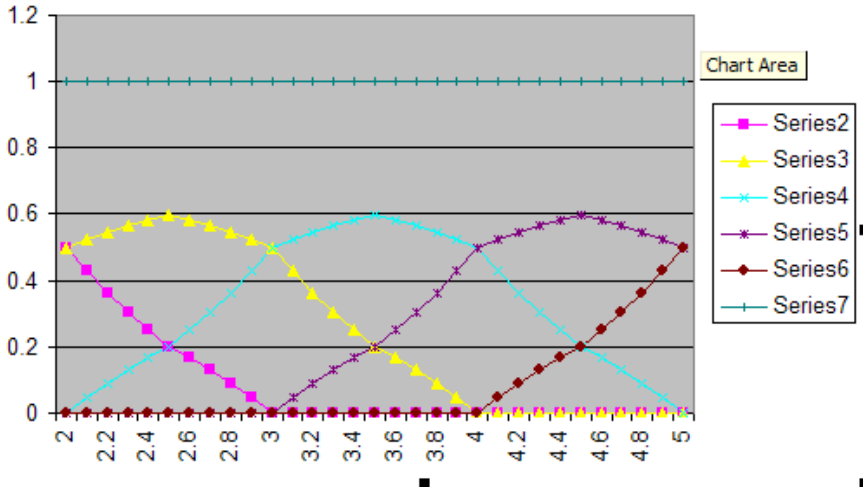


Figure 3.6: Graph showing the influence of functions

or edge with P. Let these pixels be $P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8$. Given a digital pattern on a grid, locate a *start* pixel, one with a value greater than or equal to zero for the implicit function. Locating a start pixel can be done in a number of ways. A brute force method is to start at the bottom left corner of the grid, scan each column of pixels from the bottom going upwards and proceeding to the right until a pixel with a value greater than or equal to zero for the implicit function is obtained. The disadvantage of this search is that, if the grid is very large and the start pixel is located on the extreme right, then the search would take a very long time to locate the start pixel.

This search method slows down the tracing process considerably. Hence, a different method of finding the start pixel is implemented. As the boundary points of the input slices are known, the leftmost bottom point of the first input slice $slice_1$ is marked as the start pixel (sp_1) for $slice_1$. The start pixel of $slice_2$ is found by using sp_1 as follows. Let the coordinates of sp_1 be (x,y). sp_1 can be represented as (x,t,y), where t is the height. The height of slice $slice_2$ will be $t + incr$, where $incr$ is the inter-slice distance set by the user. In $slice_2$, t of sp_1 is replaced by $t + incr$ to get a temporary pixel tp from which the search has to be started. The search then proceeds in circles around tp. If tp is inside the surface then the search continues until a pixel outside the surface is found, i.e., $v(x) < 0$ in Equation 3.1, or if tp is outside the surface the search continues until a pixel inside the surface is found, i.e., $v(x) \geq 0$ in Equation 3.1. This is the start pixel sp_2 of $slice_2$. This

is repeated for all slices.

The next step is to determine the boundary pixels of a given slice starting from the start pixel. Figure 3.7 shows how the Moore tracing algorithm works. Given any pixel P, the Moore neighborhood consists of the pixels labelled P_1 to P_8. These pixels are always visited in clockwise order, although the start position may vary.

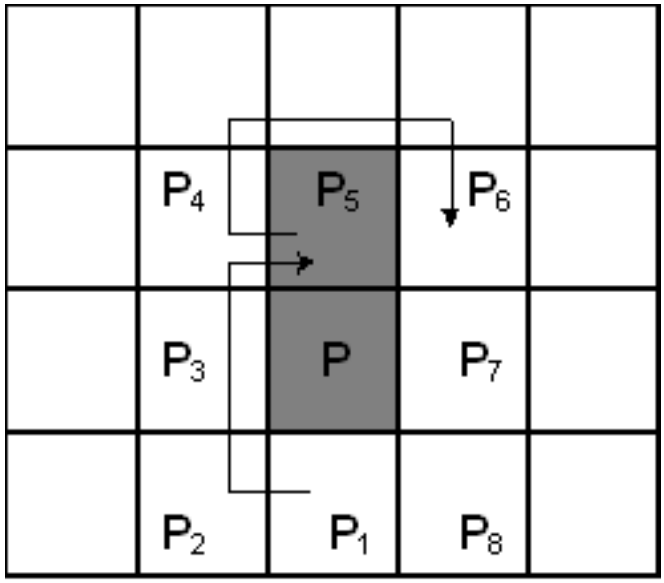


Figure 3.7: Tracing an intermediate contour

The search begins with P set to the start pixel. The Moore neighborhood is visited until a pixel P_i with a value greater than zero is found. Let P_{pre} denote the pixel that preceded P_i in the search of the Moore neighborhood. P_i is added to the contour and P is set to P_i. The algorithm continues except that the search of the Moore neighborhood begins at P_{pre}. The algorithm terminates when the start pixel is visited for a second time. This is repeated for all the required heights. To ensure the smoothness of the models, a pixel with $v(x)$ close to zero is found. This is done by searching between P and P_{pre} for a pixel (P_{final}) with $v(x)$ close to zero. That is, from two points of differing sign we converge to a zero crossing to get P_{final}. The search stops when the start pixel is revisited again.

Concerning the smoothness of models, Winsurf accepts only integer values for pixels. Thus, P_{final} was approximated by Winsurf. This resulted in the models having a stippled appearance. Hence, P_{final} was multiplied by a large number to minimize the round-off problem. Figure 3.8 shows a model generated with an approximation error and a corrected

model.

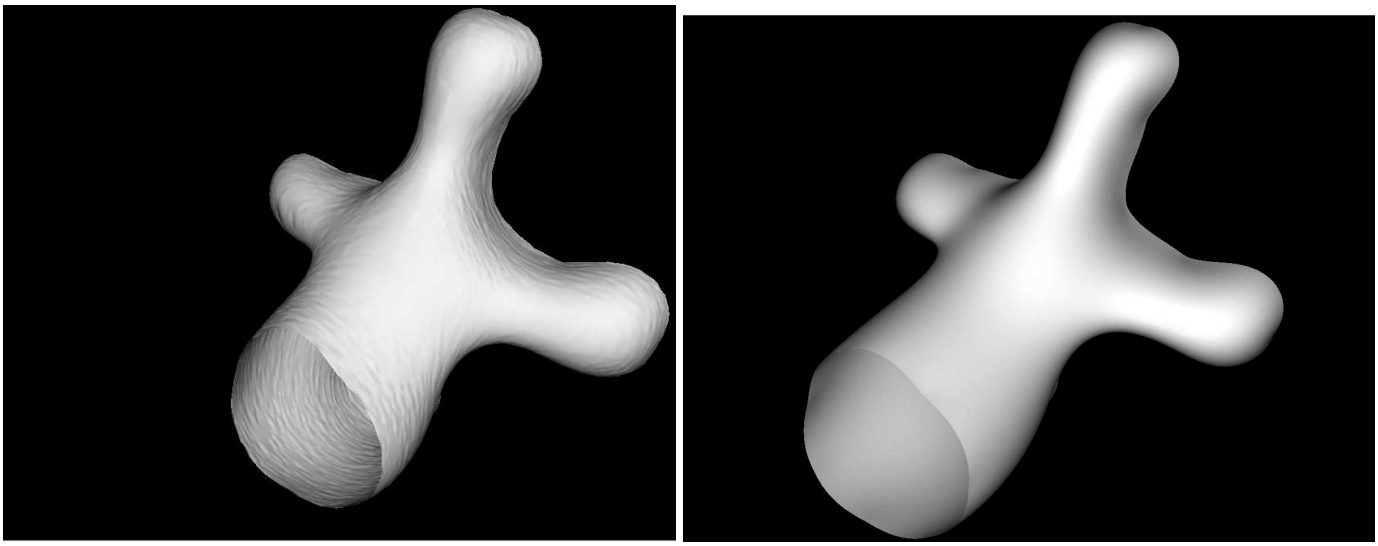
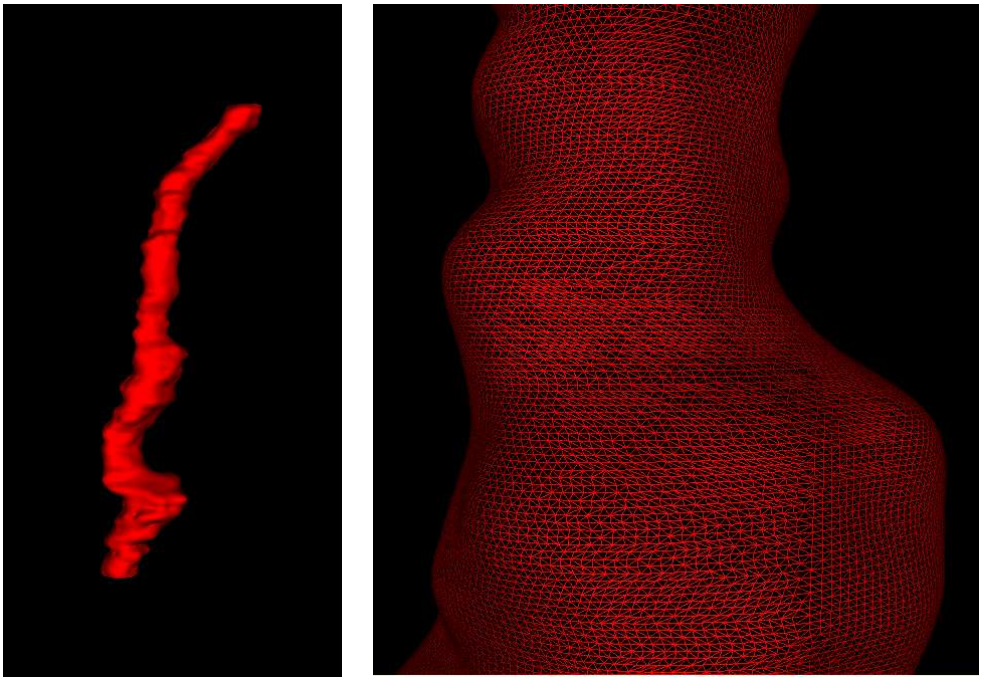


Figure 3.8: Jagged vs smooth appearance of models

Once the intermediate contours are traced, the contours are triangulated to get the final surface. Figure 3.9 shows a vas segment from prostate data generated using this method. Furthermore, as adjacent contours are very similar, it is possible to use simple heuristic algorithms to construct the surface.

3.2 Preconditioned GMRES method of Beatson *et al.*

A fast fitting method implemented by Beatson *et al.* [49] uses an iterative method on a preconditioned interpolation matrix for solving the instability problem of large matrices. A *preconditioned matrix* has clustered eigenvalues that improves the condition numbers of the matrix and also speeds the convergence of the iterative method. Hence, only a few iterations are required to solve a preconditioned interpolation matrix. We investigated the work of Beatson *et al.* [15], which solves such large matrices using the GMRES iterative method (Generalized Minimal Residual Method).



(a) Vas using weighted implicit method

(b) Close up wireframe view.

Figure 3.9: Model of vas segment from prostate data obtained using weighted implicit method.

3.2.1 Preconditioning using approximate cardinal functions

Preconditioning the interpolation matrix is done by changing the unsuitable or "bad" basis (Equation 2.19) of the original approach, to a "good" or suitable basis, that clusters the eigenvalues of the interpolation matrix. For the present discussion, the method of Turk and O'Brien is considered a bad basis, as the basis function of their approach leads to an unstable linear system of equations.

Beatson *et al.* use *approximate cardinal functions* to cluster eigenvalues by defining ψ_j for each constraint point such that

$$\psi_j(x) = p_j(x) + \sum_{i=1}^{\beta} v_{ji}\phi(|x - c_i|). \tag{3.5}$$

That is, for each input constraint point x_j, an approximate cardinal function(ψ_j) is found by choosing β points nearest to x_j. If $\beta \ll N$, Equation 3.5 can be solved using

any direct method like LU decomposition. The constraints for the above equation are

$$\psi_j(x_i) = \delta_{ij}, \tag{3.6}$$

where δ_{ij} is *Kronecker's delta function* ([51]).

$$\delta_{ij} = \begin{cases} 1 & \text{if i = j} \\ 0 & \text{if i} \neq \text{j} \end{cases} \tag{3.7}$$

In a pure cardinal function, each basis element has a value of one at one node and zero at the other $N-1$ nodes. This makes eigenvalues perfectly clustered and the resulting identity matrix is well conditioned and the coefficients can be found in a single iteration. But for large N, this is very difficult to achieve as it would be more expensive than the problem of the original Turk and O'Brien approach because we would have to solve a large system with N nodes, N number of times. Therefore, instead of pure cardinal functions, approximate cardinal functions are used for preconditioning the interpolation matrix which uses a small fixed number of neighbours.

For N constraint points, we obtain the following approximate cardinal functions

$$\begin{aligned}
\psi_1(x) &= p_1(x) + \textstyle\sum_{i=1}^{\beta} \nu_{1i}\phi(x - c_{1i}) \\
\psi_2(x) &= p_2(x) + \textstyle\sum_{i=1}^{\beta} \nu_{2i}\phi(x - c_{2i}) \\
\psi_3(x) &= p_3(x) + \textstyle\sum_{i=1}^{\beta} \nu_{3i}\phi(x - c_{3i}) \\
&\vdots \\
\psi_N(x) &= p_N(x) + \textstyle\sum_{i=1}^{\beta} \nu_{Ni}\phi(x - c_{Ni}),
\end{aligned}$$

where c_{ji} are the constraint points, β is the number of nearest points, and $\nu_{11}, \nu_{12}, \ldots, \nu_{1\beta}$; $\nu_{21}, \nu_{22}, \ldots, \nu_{2\beta}$ up to $\nu_{N1}, \nu_{N2}, \ldots, \nu_{N\beta}$ are the coefficients obtained by solving $\psi_1(x)$, $\psi_2(x)$ and $\psi_N(x)$ respectively.

Then, the values of the $\psi_j(x)$ are used to find μ in the equation:

$$\Psi\mu = f, \tag{3.8}$$

using the matrix

$$\begin{bmatrix} \psi_1(x_1) & \psi_2(x_1) & \dots & \psi_N(x_1) \\ \psi_1(x_2) & \psi_2(x_2) & \dots & \psi_N(x_2) \\ \psi_1(x_3) & \psi_2(x_3) & \dots & \psi_N(x_3) \\ \vdots & & & \\ \psi_1(x_N) & \psi_2(x_N) & \dots & \psi_N(x_N) \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_N \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{3.9}$$

The matrix Ψ has clustered eigenvalues and is solved using the GMRES iterative method in relatively fewer iterations. f contains height values of 0 and 1 analogously to the method of Turk and O'Brien.

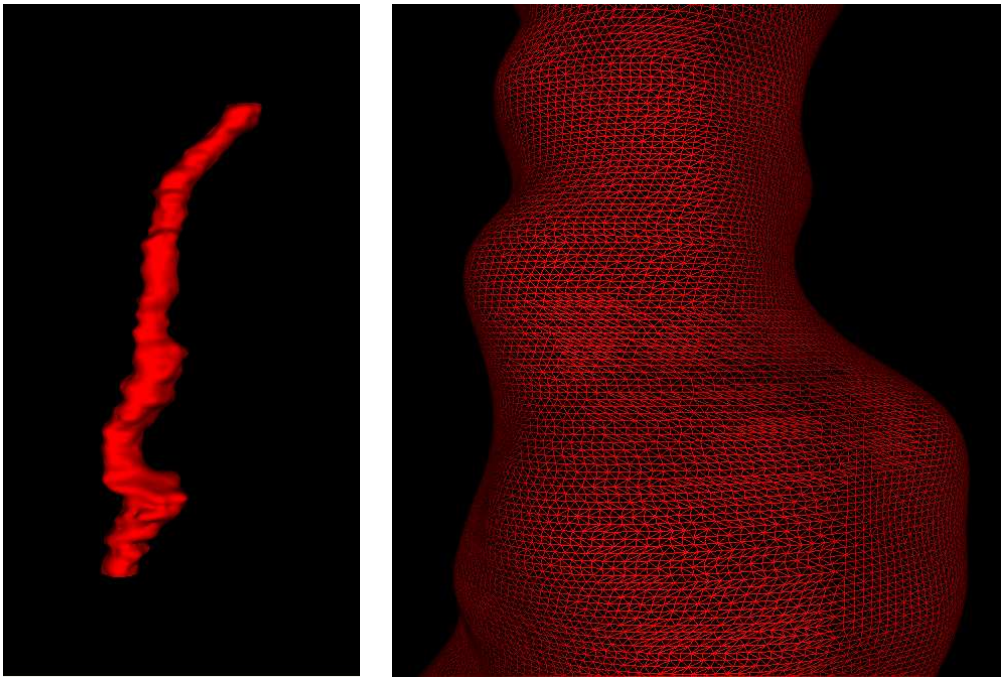
The solution (μ_1, ..., μ_N), is converted back to the original representation of the weights d_i in Equation 2.19 as follows:

$$d_i = \sum_{j=1}^{N} \mu_j v_{ji}, \tag{3.10}$$

and the polynomial coefficients of Equation 2.19 are obtained using the set of equations.

$$\begin{aligned} p_1 &= \textstyle\sum_{j=1}^{N} \mu_j p_{1j}, \\ p_2 &= \textstyle\sum_{j=1}^{N} \mu_j p_{2j}, \\ p_3 &= \textstyle\sum_{j=1}^{N} \mu_j p_{3j}, \\ p_0 &= \textstyle\sum_{j=1}^{N} \mu_j p_{0j}. \end{aligned}$$

Using the method of Beatson *et al*, the weights and the polynomial coefficients of the original implicit function (Equation 2.19) can be found without any conditioning difficulties when N becomes large. Unlike the original approach that required $O(N^2)$ storage and $O(N^3)$ flops, this method requires $O(N)$ storage and $O(NlogN)$ flops. Figure 3.10 shows a vas segment generated using this approach.



(a) Vas using GMRES (b) Close up wireframe view.

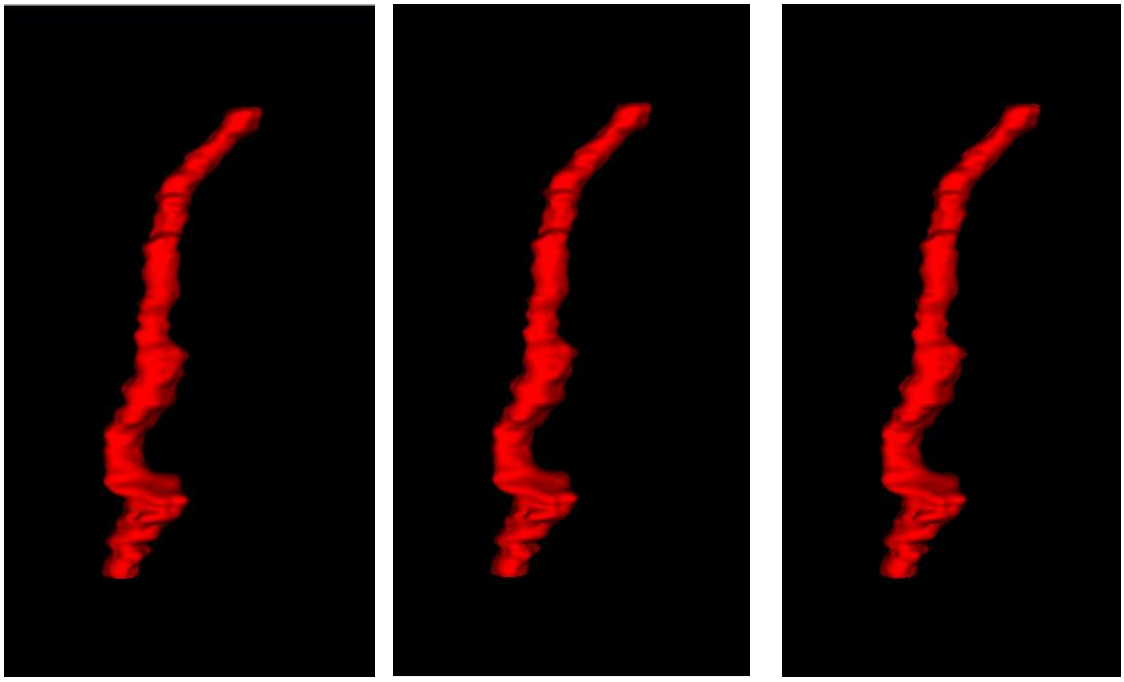
Figure 3.10: Model of vas segment from prostate data obtained using GMRES method.

CHAPTER 4

RESULTS

In the introduction we stated our goal was to produce significantly higher quality models in reasonable time from data that might contain misaligned sections, user errors, or large interslice distances. The two methods studied go a long way towards meeting these goals. This chapter presents our findings.

4.1 Weighted method vs other approaches



(a) Single implicit. (b) GMRES. (c) Weighted Implicit.

Figure 4.1: A comparison of the three approaches.

Figure 4.1 compares the final model obtained from the three approaches (Turk and

O'Brien, Beatson *et al.* and weighted implicit) for the same data set. The original contour has 60 slices, 2320 constraint points, and an inter-slice distance of 14 units, whereas the surface formed using weighted implicit approach (Figure 4.1(c)) has 841 slices, 143609 points, and an inter-slice distance of 1 unit.

Method	Number of slices in final model	Number of vertices in final model	Inter-slice distance
Fuchs	60	2320	14
Single implicit	841	143691	1
Weighted implicit	841	143609	1
GMRES	841	143690	1

Table 4.1: Comparison of the final model generated by four different approaches.

By visual inspection, we can say that there is no appreciable difference in the appearance of the models generated using these various approaches. However, the single implicit approach of Turk and O'Brien can be used to generate surfaces only if the original contour data consists of less than a few thousand data points, whereas the weighted and the GMRES approaches can be used when the original contour data is large. Figure 4.2 shows a structure from prostate data and Figure 4.3 shows a structure from stage 13 of embryo data reconstructed using weighted implicit functions.

Table 4.2 compares the runtimes of the three approaches for a large segment of the vas. The experiments were run on a workstation with 512 MB RAM, and 1.2GHz AMD Athlon processor. The size of the interpolation matrix is 2320. There is a significant difference in the run-times of these three approaches. The table divides the cost into two parts. The first is the creation of the implicit function which requires solving a system of linear equations. The second is the cost of evaluating the function at each point on the resulting model.

For N input constraint points, the complexity of Turk and O'Brien method is O(N^2)to compute the coefficients of the implicit function using LU decomposition and O(MN) to

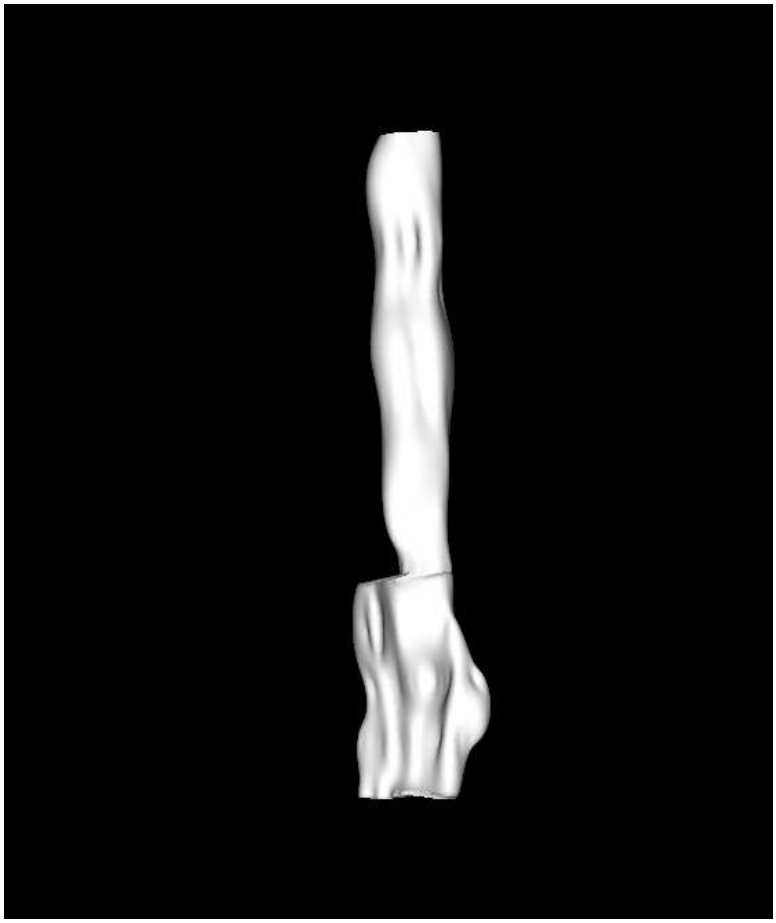


Figure 4.2: Reconstruction of a structure from Prostate data.

evaluate the implicit function at M data points in the final model. For large objects this method is not feasible for real-time model generation.

GMRES can significantly reduce the cost of computing the coefficients, although the potential savings are not reflected here, because the matrix-vector product computation involved in the GMRES implementation was not done using a fast algorithm. Given N constraint points, GMRES first computes N implicit functions of β points using LU decomposition. Relative to N, the cost of this is O(N). The conditional matrix resulting from this is sparse and a clever data structure would result in a very fast iterative solution, though we did not implement this. Note that the cost of tracing the contours to get the final surface is almost equal to the tracing cost of Turk and O'Brien method because it uses exactly the same implicit function. Realtime speedups are possible but these methods involve approximation of the implicit function, and require implementation of complicated

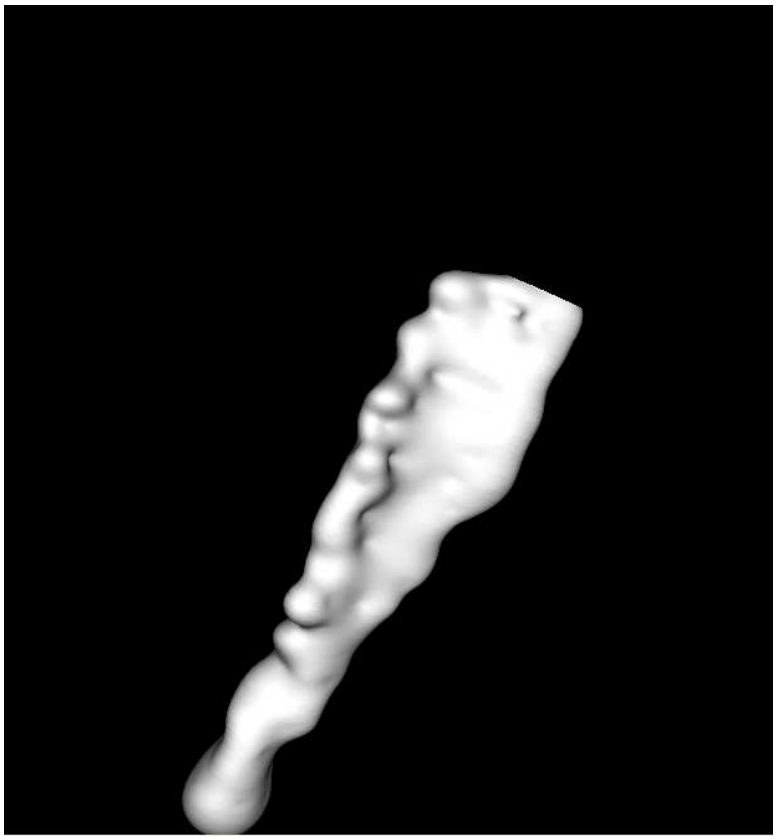


Figure 4.3: Reconstruction of a structure from Stage 13 of Embryo data

data structure. The GMRES method also has other pre-processing requirements such as ensuring that points in a local set do not lie in a single plane. That is, not all β points we choose in Equation 3.5 should lie on the same plane.

The weighted implicit function method achieves a faster runtime than the other two methods avoiding the instability problem encountered by the other two methods, plus is much simpler conceptually and simpler to implement. Dividing the data into small windows makes the cost of finding the overall implicit function linear in the number of constraint points. That is, assuming the number of points per slice is small relative to the size of the whole problem, the cost of solving each individual implicit function is constant. There is one such function per window, and the number of windows grows linearly with the size of the problem. Thus the cost of building the weighted implicit function is about O(N). The cost of evaluating the function at M data points is O(M) since the cost of each evaluation is a large constant under these assumptions.

Table 4.2 shows the savings in time obtained using our approach of piecewise weighted

Method	Time taken to compute coefficients (s)	Time taken to trace contours (s)	Total time (s)
Single implicit	213	1496	1709
Weighted implicit	18.456	311	329.456
GMRES	316	1502	1818

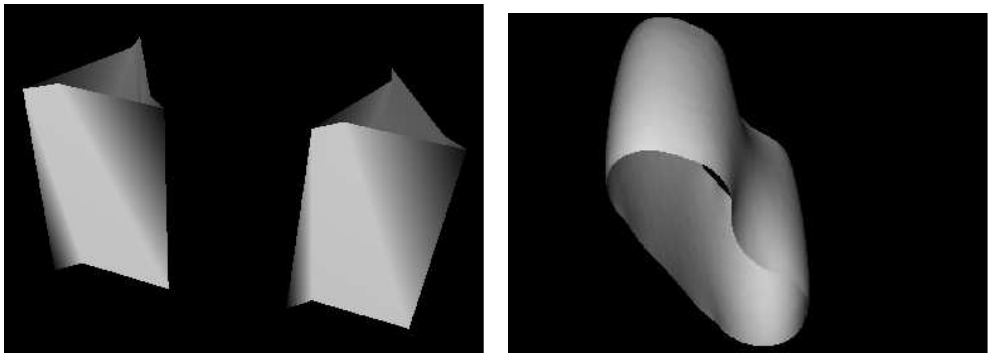
Table 4.2: Comparison of three approaches.

implicit functions as compared to the original Turk and O'Brien approach and the GMRES approach. Thus, the weighted approach produces models very similar to the original Turk and O'Brien approach but has a significant run-time advantage over the original approach, making it more efficient than the original approach when the number of constraint points are quite large.

4.2 Other challenges

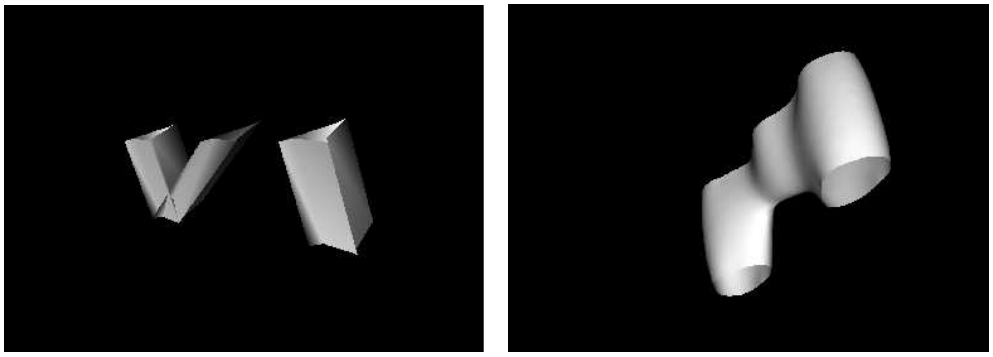
Figure 4.4(b) shows two independent sections (Figure 4.4(a)) blending into a single shape. Even though this kind of blending might be visually correct, it is sometimes inappropriate for our use as blending of different sections is likely to be anatomically incorrect in our setting. This is called the "unwanted blending" problem and is discussed in [52], [53], [54].

Figure 4.5 shows another example of this problem. Figure 4.5(a) shows a Winsurf model generated with two slices, first slice containing three sections and the next slice containing two sections. Figure 4.5(b) shows the automatic branching done using the weighted implicit approach. Such an automatic branching is inappropriate since the user has joined the second and third sections of the first slice to the second section in the next slice. Presumably this has been done for a reason and hence blending of sections is sometimes not appropriate even though it might look correct.



(a) Independent sections. (b) Two sections merged.

Figure 4.4: Merging of independent sections.



(a) Branching in WinSurf. (b) Branching using implicit functions.

Figure 4.5: Branching by implicit functions vs branching in WinSurf.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Summary

This thesis was concerned with the construction of anatomical models from contour data in reasonable time. We studied several triangulation algorithms, but felt that only limited gains could be made by varying the triangulation algorithm as the contour data is relatively sparse. We then turned our attention to interpolation approaches, in particular, the variational implicit function approach of Turk and O'Brien. This approach, based on radial basis functions, seemed promising, but given the size of the models we might deal with, suffered from two problems. The first was that simple LU-decomposition becomes unstable when solving for an implicit function with on the order of 10000 coefficients (i.e., constraints). The second is that the time required to solve such a system, and to evaluate an implicit function consisting of large number of constraint points were high. We first considered GMRES, a numerically stable approach to finding the solution to the implicit function. However, the resulting function is still large for large models, and slow to evaluate when searching for model surfaces. Because the value of the radial basis functions increases with the distance from the center, there was no obvious way of speeding up the evaluation of the final implicit function at a specific point by selectively evaluating RBF's with nearby centers.

5.2 Contributions

Our approach called the *piecewise weighted implicit functions* takes only a few slices at a time to construct the implicit function, hence the interpolation matrix required to compute

the coefficients of the implicit function is kept at a small constant size thereby avoiding the instability problem. Also, as only a few constraint points are involved in the evaluation of an implicit function at any given height, the intermediate slices can be traced more quickly thereby resulting in faster execution time than the other two approaches.

Moreover, the study of triangulation algorithms provided an easy way to find normal constraint points for the implicit function specification. The systems have not yet been completely integrated to fully automate the generation of smooth surfaces from the initial specifications. However, this appears to be a straightforward, if tedious, problem. The following pipeline can be established using this system: the contours specified by the user, that is the input slices, can be triangulated to yield boundary and normal constraint points. They are then fed into the variational interpolation solver, which in turn generates more detailed intermediate contours that are triangulated for the final visualization. Because adjacent contours are so similar, trivial triangulation algorithms can be used at this point.

5.3 Future work

A possible avenue for future work would be to handle the problem of complete extraction of new contours from the implicit function. A variety of simple heuristics can be used to pull out most of the contours in most cases, but in the worst case, a large area must be thoroughly searched for possible new contours. Another useful avenue would be the integration of fast evaluation methods and smoothing methods [30] into this system to achieve more speed ups in runtime as well as to remove noise in the input data. Another constructive area for future work can be the handling of the unwanted blending problem.

REFERENCES

[1] Stages of embryo. url: http://virtualhumanembryo.lsuhsc.edu/HEIRLOOM /Stages/HEP.htm;accessed Jan 2005.

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

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

[4] M. Gleicher. url: http://www.cs.wisc.edu/graphics/Courses/f2003-cs559/splines.pdf; accessed Dec 2004.

[5] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell. Curved PN triangles. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 159–166, New York, NY, USA, 2001. ACM Press.

[6] Jarno Elonen, 2003. url: http://elonen.iki.fi/code/tpsdemo/; accessed Dec 2003.

[7] 3D reconstruction software - Maya. url: http://www.highend3d.com/maya/;accessed Nov,2005.

[8] 3D reconstruction software - 3D doctor. url: http://www.ablesw.com/3d-doctor/;accessed Nov,2005.

[9] 3D reconstruction software - Truespace. url: http://www.caligari.com/;accessed Nov,2005.

[10] url: http://www.amabilis.com/;accessed Nov,2005.

[11] Visible human data set. url: http://www.nlm.nih.gov/research/ visible/visible_human.html;accessed Mar 2005.

[12] Spitzer V.M. and Whitlock D.G. The visible human dataset: the anatomical platfrom for human simulation. *The Anatomical Record*, 253:49–57, 1998.

[13] Doll F, Doll S, Kuroyama M, Sora M.C, Neufeld E, and Lozanoff S. Computerized reconstruction of plastinated human kidney using serial tissue sections. *This article is to appear*.

[14] Greg Turk and James F. O'Brien. Shape Transformation Using Variational Impilict Functions. *Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH 1999*, pages 335–342, 1999.

[15] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics 11*, pages 253–270, 1999.

[16] B. Delaunay. Sur la sphere vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[17] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[18] D. Moody and S. Lozanoff. SURFdriver: A practical computer program for generating three-dimensional models of anatomical structures using a PowerMac. *Clinical Anatomy*, 11(133), 1998.

[19] A. E. Kaufman. *Volume Visualization*. The Computer Science and Engineering Handbook, 1997.

[20] I. Amidror. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of Electronic Imaging*, 11(2):157–176, April 2002.

[21] James D. Foley, Richard L. Phillips, John F. Hughes, Andries van Dam, and Steven K. Feiner. *Introduction to Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[22] G. Farin. Triangular Bernstein-Bezier patches. *Computer Aided Design*, 3:83–127, 1986.

[23] E. W. Weisstein. Cubic spline, from mathworld–a Wolfram web resource. url: http://mathworld.wolfram.com/CubicSpline.html; accessed Feb 2004.

[24] S. McKinley and M. Levine. Cubic spline interpolation. url: http://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall98/SkyMeg/Proj.PDF; accessed Nov 2003.

[25] url: http://www.tinaja.com/glib/hack63.pdf; accessed June 2005.

[26] Z-fighting. url: http://en.wikipedia.org/wiki/Z-fighting;accessed Nov,2005.

[27] Shepard D. A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 23rd ACM National Conference, ACM*, pages 517–524, 1968.

[28] R. Franke. Scattered data interpolation: tests of some methods. *Mathematics of Computation*, 38(157):181–200, January 1982.

[29] W. Richard Fright Jonathan C. Carr and Richard K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions Med. Imag*, 16(1):96–107, February 1997.

[30] J. C. Carr, T. J. Mitchell, R. K. Beatson, J. B. Cherrie, W. R. Fright, B. C. McCallumm, and T. R. Evans. Reconstruction and representation of 3D objects with Radial Basis Functions. *Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH 2001*, pages 67–76, 2001.

[31] R. L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research*, 76(8):1905–1915, 1971.

[32] Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. 3d scattered data approximation with adaptive compactly supported radial basis functions. In *SMI*, pages 31–39, 2004.

[33] Bryan S. Morse, Terry S. Yoo, David T. Chen, Penny Rheingans, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 89, Washington, DC, USA, 2001. IEEE Computer Society.

[34] Nikita Kojekine, Vladimir Savchenko, and Ichiro Hagiwara. Surface reconstruction based on compactly supported radial basis functions. pages 218–231, 2004.

[35] H. Wendland. Piecewise polynomials, positive definite and compactly supported radial basis functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995.

[36] Compactly supported RBF. url: http://www.mathworks.com/access/helpdesk/help/toolbox/mbc/model/rbf3.html;accessed Nov,2005.

[37] Jules Bloomenthal. *Introduction to Implicit surfaces*. Morgan Kaufmann, 1997.

[38] Sunhwa Jung, Min Hong, and Min-Hyung Choi. An adaptive collision detection and resolution for deformable objects using spherical implicit surface. In *International Conference on Computational Science (1)*, pages 735–742, 2005.

[39] T. Gutzmer. Error estimates for reconstruction using thin plate spline interpolants,. (Research report 97-08), May 1997. url: http://e-collection.ethbib.ethz.ch/ecolpool/incoll/incoll_247.pdf; accessed Feb 2005.

[40] J. Duchon. *Splines minimizing rotation-invariant semi-norms in Sobolev spaces. Constructive theory of functions of several variables. Lecture Notes in Mathematics*. Springer-Verlag, 1977.

[41] C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[42] M. Powell. The theory of radial basis function approximations. *Advances in Numerical Analysis. Vol. II: Wavelets, Subdivision Algorithms and Radial Basis Functions*, 2:105–210, 1990.

[43] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C:The art of scientific computing*. Cambridge University Press; 2nd edition (October 30, 1992), 1992.

[44] J. P Moreau. Programs concerning matrices in C/C++. url: http://perso.wanadoo.fr/jean-pierre.moreau/c_matrices.html; accessed Dec 2003.

[45] Taosong He, Sidney Wang, and Arie Kaufman. Wavelet-based volume morphing. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 85–92, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[46] Gabor T. Herman, Jingsheng Zheng, and Carolyn A. Bucholtz. ShapeBased Interpolation. *IEEE Comput. Graph. Appl.*, 12(3):69–79, 1992.

[47] John F. Hughes. Scheduled Fourier volume morphing. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 43–46, New York, NY, USA, 1992. ACM Press.

[48] Rossignac, Jarek, and Anil Kaul. AGRELs and BIPs: Metamorphosis as a Bezier Curve in the Space of Polyhedra. In *Proceedings of Eurographics '94*, pages 179–184, Oslo,Norway, 1994.

[49] R. K. Beatson and W. A. Light. Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372, 1997.

[50] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

[51] Kronecker delta function. url: http://en.wikipedia.org/wiki/Kronecker_delta;accessed June 2005.

[52] Jean-Dominique Gascuel. Implicit patches: An optimized and powerful ray intersection algorithm for implicit surfaces. In *Implicit Surface*, May 1995.

[53] Brian Wyvill, Callum Galbraith, Mark Fox, and John Hart. Towards better blending control of implicit surfaces. *Proceedings of 11th Western Computer Graphics Symposium*, pages 39–45, 2000.

[54] Alexis Angelidis, Pauline Jepp, and Marie-Paule Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. In *Shape Modeling International*. ACM, IEEE Computer Society Press, 2002. Banff, Alberta, Canada.