



National Library  
of Canada

Acquisitions and  
Bibliographic Services Branch

395 Wellington Street  
Ottawa, Ontario  
K1A 0N4

Bibliothèque nationale  
du Canada

Direction des acquisitions et  
des services bibliographiques

395, rue Wellington  
Ottawa (Ontario)  
K1A 0N4

Acquisitions and Bibliographic Services

Direction des services bibliographiques

## NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

## AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**UNIVERSITY OF ALBERTA**

**SURFACE MODELLING OF TRUNK DEFORMITY  
ASSOCIATED WITH SCOLIOSIS**

**BY**

**ZIXI ZHANG**



A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**EDMONTON, ALBERTA**

**FALL, 1993**



National Library  
of Canada

Acquisitions and  
Bibliographic Services Branch

395 Wellington Street  
Ottawa, Ontario  
K1A 0N4

Bibliothèque nationale  
du Canada

Direction des acquisitions et  
des services bibliographiques

395, rue Wellington  
Ottawa (Ontario)  
K1A 0N4

Author's Authorisation

Author's Authorisation

**The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.**

**The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.**

**L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.**

**L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

ISBN 0-315-88047-3

**Canada**

**UNIVERSITY OF ALBERTA**

**RELEASE FORM**

**NAME OF AUTHOR:** Zixi Zhang

**TITLE OF THESIS:** Surface Modelling of Trunk Deformity

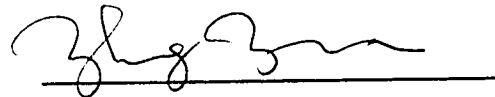
Associated with Scoliosis

**DEGREE:** Master of Science

**YEAR THIS THESIS GRANTED:** 1993

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



(Student's Signature)

301, 1 Nanxinli, XiangZhou

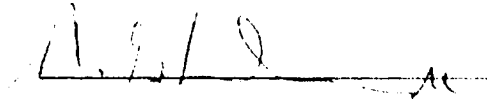
Zhuhai City, China

Date: Sept. 30, 1993

UNIVERSITY OF ALBERTA

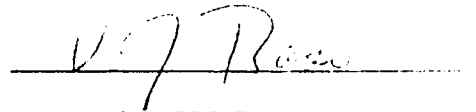
FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Surface Modelling of Trunk Deformity Associated With Scoliosis** submitted by **Zixi Zhang** in partial fulfillment of the requirements for the degree of **Master of Science**.



Dr. N.G. Durdle

Co-Supervisor

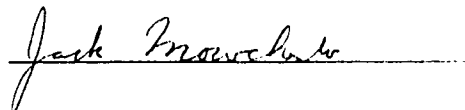


Mr. V.J. Raso

Co-Supervisor



Dr. A. Basu



Dr. J. Mowchenko

Date: *Sept. 30, 1993*

## ABSTRACT

Surface modelling can provide clinicians with a three-dimensional graphical representation of surface features of the trunk for the assessment of spinal deformities. For fast acquisition of 3-D representations of trunk surfaces, a computer aided analytical tool has been developed for a better understanding of trunk deformity due to scoliosis.

A regular grid of dots is projected upon a child's back. This grid is distorted where the surface shape changes. 3-D coordinates are obtained for each dot location. The coordinates of dots are used to model trunk surfaces. Surface modelling from these discrete data points depends on many factors: the distribution of data points, the order of points and the topological appearance of the object. Any of these features can significantly affect the performance of the modelling algorithms. A good surface modelling algorithm must overcome these problems and rapidly produce an accurate model from the data.

Different approaches for surface modelling were reviewed and compared. An efficient method using triangulation has been implemented to construct a 3-D trunk model. The algorithm has been modified to reduce boundary problems and to handle more complex structures. The software is implemented with a user interface for boundary processing and surface model assessment. Combined with internal spinal alignment information obtained from radiographs, the 3-D trunk surfaces can be used to improve the understanding of the effects of treatment, to document the natural history of scoliosis, and to demonstrate the 3-D deformity for educational purposes.

**To my parents, for their patience and support;**

**And to Landdy, for her understanding and encouragement.**

## ACKNOWLEDGEMENTS

The author would like to thank his supervisors, Dr. N. Durdle and Mr. J. Raso, for their guidance and encouragement throughout the graduate program.

Special thanks are due to Mr. D. Hill for valuable technical assistance on RISC 6000 workstations during the software development, and for proof-reading the thesis.

The author is grateful to the committee members, Dr. A. Basu and Dr. J. Mowchenko, for their valuable suggestions and recommendations in producing this finalized thesis.

Last but not the least, the author would express his appreciation to Brian Oostenbrink for the assistance in producing the hard copies of images, Edmond Lou for installing some drawing programs on the PC, and Ron Shute for proof-reading a manuscript.

Financial support was given by the University of Alberta and the Alberta Microelectronics Center.



## TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION .....	1
1.1 Scoliosis--A General Description .....	1
1.2 Surface Modelling of Human Body .....	10
1.3 Objective .....	11
1.4 Structure of the Thesis .....	12
2. SURFACE MODELLING.....	13
2.1 Modelling of General Objects.....	13
2.2 Surface Representations.....	15
2.3 Triangulation for Surface Modelling .....	18
2.3.1 Triangulation and the Property of Optimal Triangulation .....	19
2.3.2 Assessment of Triangulation Techniques .....	22
2.3.3 Criteria for Triangulation .....	23
2.4 A Comparison of Algorithms for Surface Modelling .....	25
2.4.1 Introduction .....	25
2.4.2 Recursive Insertion Approaches .....	26
2.4.3 Hierarchical Triangulation .....	28
2.4.4 Polygon Decomposition Approaches .....	31
2.4.5 McLain's Algorithm .....	35
2.5 Conclusion .....	38
3. 3-D SURFACE CONSTRUCTION .....	39
3.1 Modelling in 2-D Instead of 3-D .....	39

3.2 Possible Problem Areas .....	40
3.2.1 Modification of Line Directions .....	41
3.2.2 Points Sharing the Same Circumcenter .....	42
3.3 Implementation Issues .....	43
3.3.1 Input/Output and Data Structures .....	43
3.3.2 Format of Data Points .....	45
3.3.3 List Operations .....	46
3.3.4 Direction of Triangulation .....	49
3.4 Unconstrained Triangulations .....	51
3.5 Boundary Processing .....	55
3.5.1 Introduction and Review of Literature .....	55
3.5.2 Proposed Solution.....	61
3.5.3 Boundary Generation .....	64
3.5.3.1 Boundary Generation Problem .....	64
3.5.3.2 Preliminary Boundary List .....	65
3.5.3.3 Improvement of Boundary List Using Angle Constraints .....	72
3.5.4 User Interface for Boundary Modification .....	74
3.5.4.1 Why the User Interface? .....	74
3.5.4.2 User Interface Overview .....	77
3.5.4.3 Event Handling .....	79
3.5.5 Results of Triangulation with Boundary Constraint .....	85
4. ASSESSMENT OF THE SURFACE MODELLING METHOD .....	89
4.1 Quality of Modelling.....	89
4.2 Speed of Triangulation .....	92
4.3 Generality of the Modelling Method .....	95

<b>5. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK .....</b>	<b>98</b>
<b>5.1 Conclusion .....</b>	<b>98</b>
<b>5.2 Limitations .....</b>	<b>99</b>
<b>5.3 Recommendataions for Future Work .....</b>	<b>104</b>
 <b>BIBLIOGRAPHY .....</b>	 <b>107</b>
 <b>Appendix A.1 Pseudo Code for Unconstrained Triangulation .....</b>	 <b>118</b>
<b>Appendix A.2 Pseudo Code for Triangulation with the Boundary Constaint.....</b>	<b>126</b>
<b>Appendix A.3 Display and Shell Script Program .....</b>	<b>135</b>

## LIST OF FIGURES

Figure 1.1:	A Scoliotic Trunk.....	3
Figure 1.2:	Block Diagram of a Video-Based Trunk Measurement System .....	7
Figure 1.3:	A Light Pattern with Constant and Known Spacing .....	8
Figure 1.4:	Light Pattern Produces Targets on Body Surface When Projected onto the Trunk.....	9
Figure 2.1:	Examples of Different Polygons.....	21
Figure 2.2:	Circumcircles.....	22
Figure 2.3:	Different Partitioning Methods for a Quadrilateral .....	24
Figure 2.4:	Insertion of a Point Affects Existing Triangles .....	27
Figure 2.5:	Partition of a Convex Quadrilateral by Sibson's Method.....	28
Figure 2.6:	A Ternary Triangulation and its Tree Representation .....	30
Figure 2.7:	Calculation of $\Pi$ .....	33
Figure 2.8:	Rejection of a Split .....	34
Figure 2.9:	McLain's Triangulation Algorithm .....	36
Figure 2.10:	McLain's Algorithm in 3-D Space.....	37
Figure 3.1:	Triangulation of 3-D Points Achieved by Triangulation in 2-D .....	40

Figure 3.2:	Modification of Line Directions .....	41
Figure 3.3:	Points Sharing the Same Circumcircle.....	43
Figure 3.4:	Input and Output Data.....	44
Figure 3.5:	Data Structures for Triangulation .....	46
Figure 3.6:	List Operation When the Point is <i>Unconnected</i> .....	48
Figure 3.7:	List Operations When the Point is <i>Connected</i> .....	50
Figure 3.8:	Change of Triangle Directions .....	51
Figure 3.9:	Triangulation of a Concave Region Becomes a Convex .....	53
Figure 3.10:	Artifacts in a Trunk Model .....	54
Figure 3.11:	Windowing to Reduce Boundary Effect .....	56
Figure 3.12:	Convex Hull Decomposition of a Concave Polygon.....	58
Figure 3.13:	A Pre-Defined Threshold Used to Reject Triangles .....	59
Figure 3.14:	A Problem with Threshold Method.....	60
Figure 3.15:	Detection of Invalid Triangles.....	63
Figure 3.16:	Different Boundaries for the Same Data Set.....	65
Figure 3.17:	Too Small Resolution Produces Intersecting Boundaries.....	67
Figure 3.18:	Boundary Generation by Griding Points.....	69
Figure 3.19:	Modification of Boundary Line Segments .....	70

Figure 3.20:	More Than One Point outside the Boundary Lines .....	72
Figure 3.21:	Angle Constraint for Boundary List .....	73
Figure 3.22:	An Example That Causes Boundary Intersecting .....	75
Figure 3.23:	A Problem When Reducing the Resolution .....	77
Figure 3.24:	A Window Layout .....	79
Figure 3.25:	Event Hierarchy .....	80
Figure 3.26:	Zooming Sub-Menu .....	81
Figure 3.27:	Rotation Sub-Menu .....	81
Figure 3.28:	A Trunk Surface Model .....	88
Figure 4.1:	Average Ratios for Different Data Sets .....	91
Figure 4.2:	Number of Points to be Evaluated is Reduced by a Half .....	93
Figure 4.3:	Computation Time for Different Data Sets .....	94
Figure 4.4:	Lewis' Algorithm Handles More Complex Structures .....	97
Figure 5.1:	Jagged Areas Generated When Modelling Manifolded Data .....	101
Figure 5.2:	Problem with Rejecting Triangles Beyond Boundary .....	103
Figure A.1:	Programs and Their Input/Output .....	139
Figure A.2:	Package Drive Program .....	140

## 1. INTRODUCTION

### 1.1 Scoliosis--A General Description

Scoliosis is an abnormal lateral curvature of the spine coupled with axial rotation of the vertebra. This abnormal curvature and rotation cause deformity of the rib cage, which is visible as asymmetry of the trunk. In spite of numerous investigations into possible causes, eighty percent of children with scoliosis have it for unknown reasons [1]; it is more common in females and has the highest risk of progression during times of rapid growth. Children with mild scoliosis should be monitored to detect if the scoliosis is progressing in order to intervene with treatment at the appropriate time. Researchers have not been able to discover the factors that cause this progression, although these factors are now thought to be different from those which initially cause the deformity [2].

The biomechanics of the human spine are complex due to both its static and kinematic functions [3]. In biomechanical terms, the spine is: a) a structural element connecting the upper limbs, head, and lower limbs, whose function is to transfer all the static and dynamic actions to the pelvis; b) a structure which enables the trunk to kinematically perform 3-D movements, particularly rotation and bending; and c) a structure protecting the delicate spinal cord. In the frontal plane, the normal spine is perfectly aligned and balanced, with symmetry between the right and left sides. The same is true for the coronal plane, with no rotation between segments. The junctional zones between the sagittal segments are smooth. This results in a smooth sagittal surface profile.

With scoliosis, the bony structure of the spine is asymmetrical. This asymmetry is usually accompanied by axial rotation of individual vertebrae, which causes complex deformities of the trunk that present themselves as asymmetrical elevations of the shoulders and hips, prominence of one of the shoulder blades, a skin fold on two concave sides and a subtle twisting of the trunk (*Figure 1.1*). These features of trunk deformity are difficult to quantify in a growing child although they are likely to become more noticeable during the adolescent growth spurt.

Braces, casts and surgery are used in the treatment of scoliosis. Monitoring and measuring this deformity is necessary to determine when to intervene with treatment, to understand the etiology and assess the effect of the treatment. It is now well agreed that the internal spinal deformity is a three-dimensional deformity [4, 5], but the relationship between the internal spinal deformity and the external trunk deformity is poorly understood [6]. Some patients with severe spinal deformity may have very little trunk deformity, while some patients with minor spinal deformity may have a significant trunk deformity. To better understand of the relationship between spinal deformity and trunk deformity, as well as the three-dimensional nature of these deformities, monitoring and measurement of these deformities must be made in three dimensions.



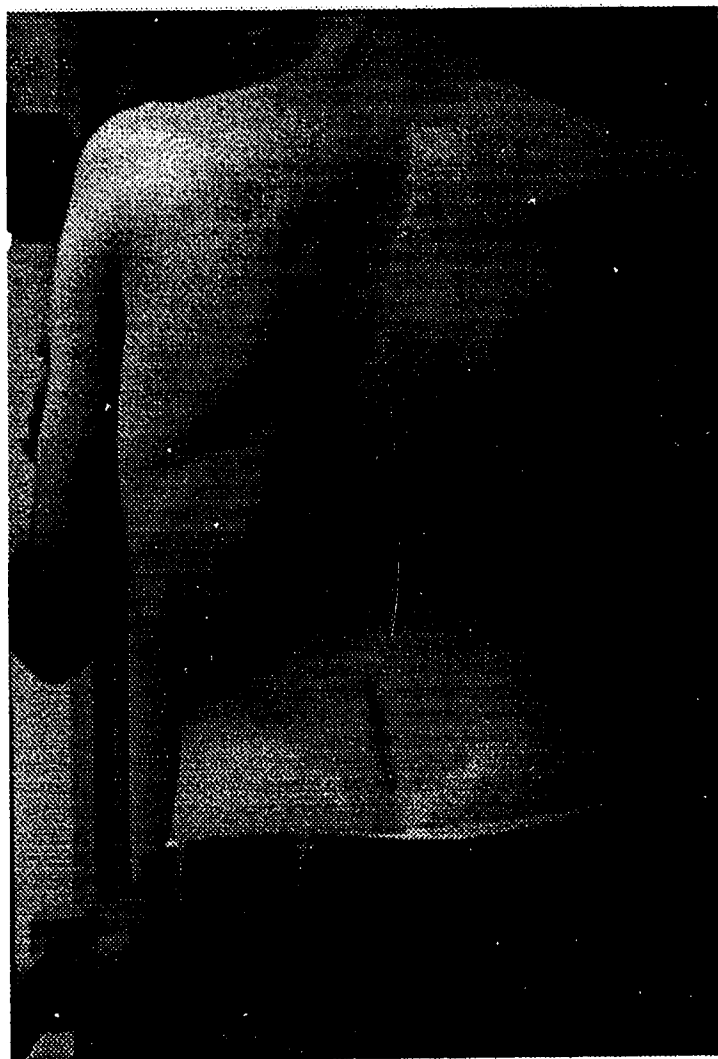


Figure 1.1. A scoliotic trunk.

The study of spinal deformity is image-oriented. The internal spinal configuration can be determined using radiographs. While this traditional measurement produces an image of the spinal curvature, it portrays the three-dimensional anatomy as a two-dimensional film image. The advantage of radiographs is to allow people to see the actual deformed structure of the spine. There are several disadvantages to radiographs. Firstly, it is an invasive technique and risks associated with radiation do not permit frequent monitoring of the patient. Secondly, a two-dimensional representation of a structure which is deformed in three-dimensional space is not an accurate description [3, 5, 7]. Thirdly, radiographs do not produce an image of the trunk surface.

The study of spinal deformity requires not only an analysis of the internal spinal configuration, but also the cosmetic defect which is visible as the trunk deformity. Cosmesis is a critical factor for the patient with adolescent idiopathic scoliosis, because body appearance can be responsible for psychological distress, and the degree of trunk deformity correlates well with the severity of the psychological disturbance [8]. Children with scoliosis and their families are generally most concerned about the cosmetic effect during treatment. In this aspect, control of back shape may be more important in terms of patient satisfaction than correction of spinal deformity [3, 9]. As a result, many scoliosis clinics measure surface deformity using instruments such as the Scoliometer [1], and the Formulator Body Contour Tracer [10] to measure features of the deformed trunk. These methods attempt to condense the features of a complex trunk surface to a single index. Shortcomings of these methods include the intensive labor involved in the measurement, and long operation time to acquire the surface shape data. These shortcomings limit their applications in the clinical setting.

Surface topography is popular in clinical setting because it is non-invasive and produces a representation of the three-dimensional deformity of a trunk surface. The 3-D surface can be used to assess the asymmetry of the trunk. Clinical studies show a correlation of surface asymmetry with the distribution of vertebral rotation changes along the spine [5]. Drerup has classified the measurement and the evaluation of trunk deformity with surface topography into three steps: acquisition, image reconstruction, and shape analysis. Due to the three-dimensional nature of the deformity, the three steps must also be performed in 3-D space [7].

Moiré techniques have been used for surface topographic analysis. They were introduced into clinical assessment of scoliosis around 1970 [11]. Moiré fringe patterns can be generated by projecting structured light onto the surface of the back. This fringe pattern represents three-dimensional topographic information in two dimensions. Using an appropriate imaging system, the information is recorded and digitized to describe the three-dimensional geometry of the surface of the back. The technique is simple and inexpensive. However, the image may be misleading because of its sensitivity to patient positioning; very different topograms could be obtained from the same patient due to motion [12, 13]. This ambiguity and the difficulty in automation (i.e. computerized processing) have limited the clinical application of this method in surface measurement of trunk deformity.

Hierholzer, Drerup and Frobin [12-14] first introduced video-based techniques for the measurement of trunk surface. They used cameras for capturing the image of the trunk surface for the clinical assessment of trunk deformity. Geometric transformations and spatial triangulation are necessary to obtain the coordinates of these points. Two cameras were used for the determination of three-dimensional coordinate of a point. The third camera was used to detect areas not visible on one of the two cameras. Because of its

accuracy and efficiency, this technique is more and more popular and is replacing Moiré techniques. Later improvement to this technique has reduced the number of cameras to only one while maintaining the measuring accuracy to 2 mm. At the Glenrose Rehabilitation Hospital, the images are acquired with a computer (Mac II) using a frame grabber (Scion image capture II) and the software furnished with the frame grabber. Through appropriate digitization, the image is processed into 3-D coordinates of surface points. *Figure 1.2* shows a block diagram of the imaging system for trunk measurement.

In the imaging system shown in *Figure 1.2*, a light pattern is projected onto the body surface. The light pattern is a grid of targets of constant and known spacing (*Figure 1.3*). Depending upon the surface appearance of the trunk, the targets have irregular spacing when they are projected onto the trunk (*Figure 1.4*). These targets on the body surface can be determined using spatial triangulation geometry, which produces a set of data points, each with the three dimensional coordinates  $x$ ,  $y$  and  $z$ .

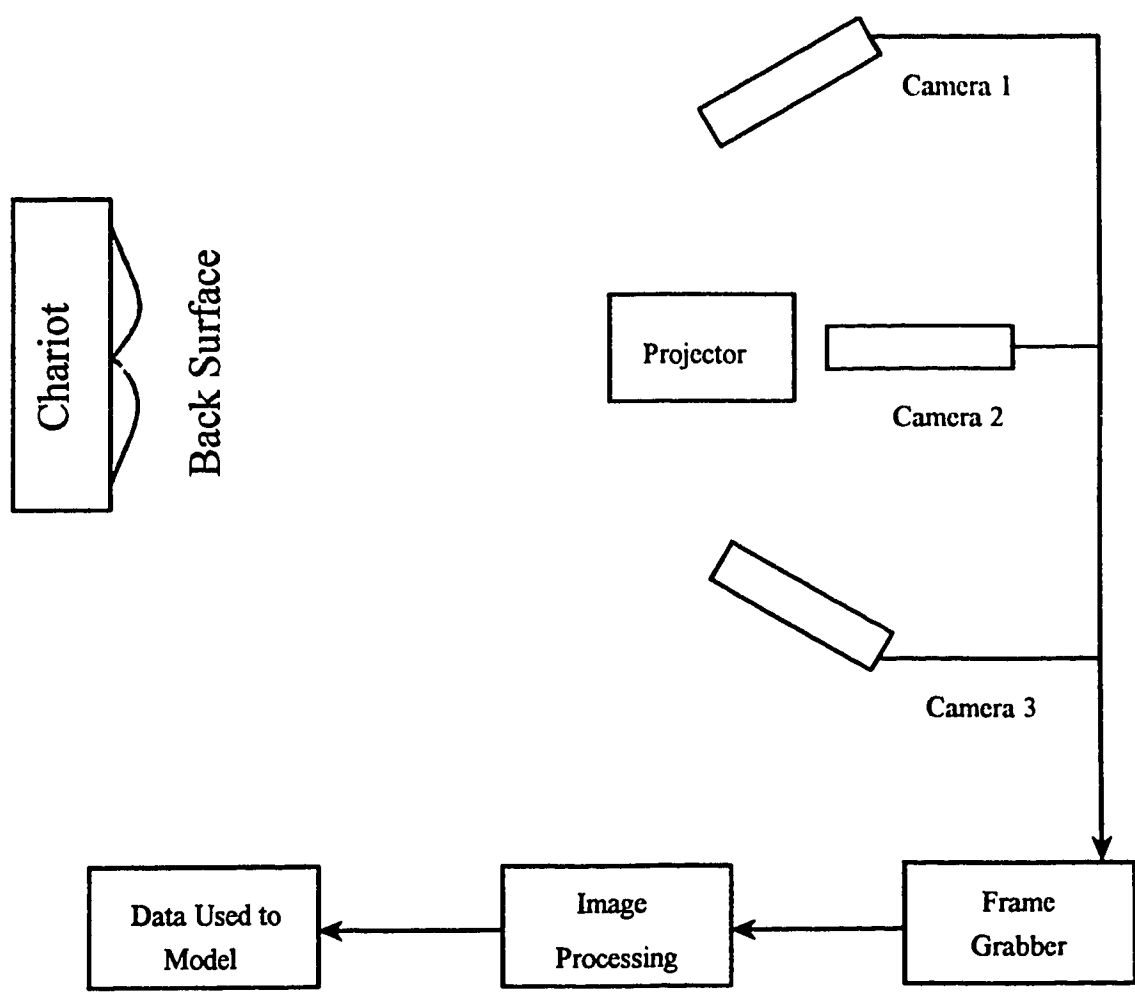


Figure 1.2. Block diagram of a video-based imaging system for trunk measurement

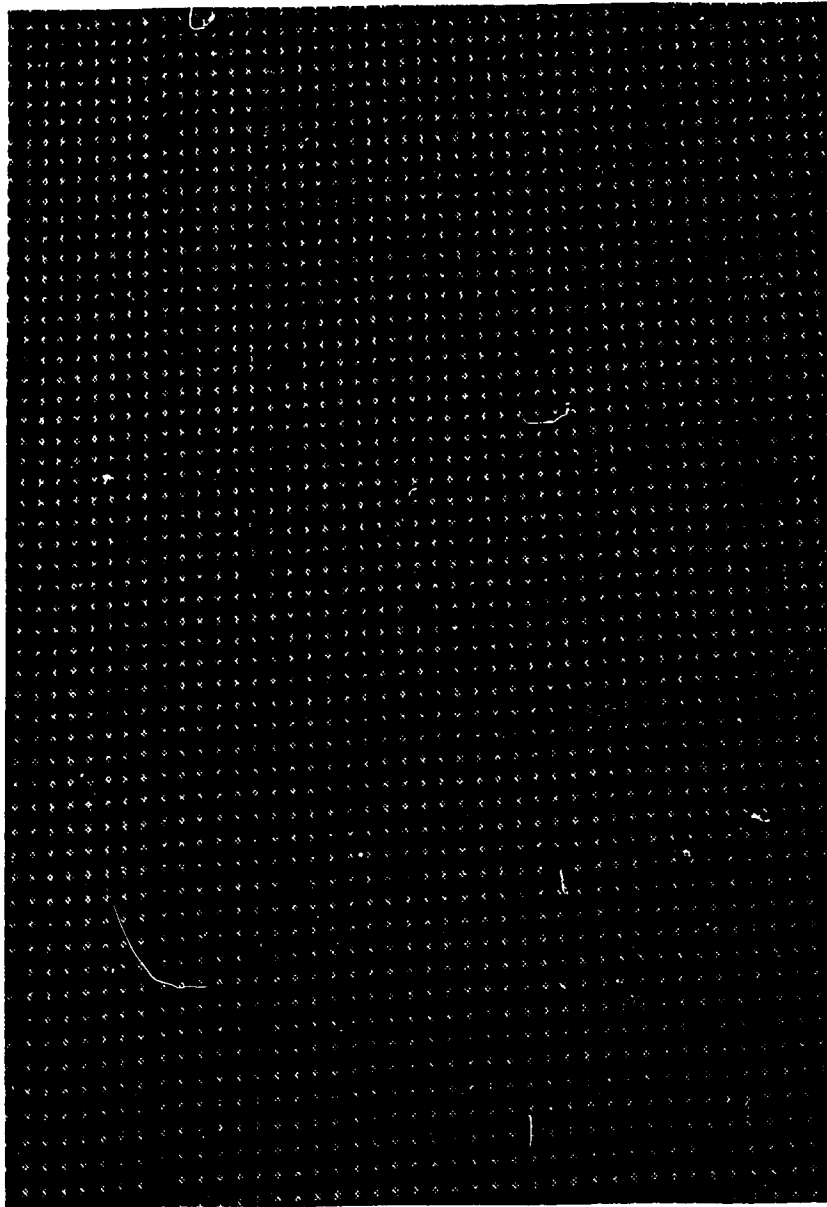


Figure 1.3 A light pattern with constant and known spacing



Figure 1.4 Light pattern produces targets on body surface when projected onto the trunk

## **1.2 Surface Modelling of Human Body**

Regardless of how data are obtained, there is a problem as how to accurately represent the surface. Never the less, surface models of anatomic structures have a great potential for educational and research applications in surgery, because surgery is essentially a three-dimensional, image-oriented specialty [15].

The history of representing 3-D anatomic surfaces on a computer has been dominated by contouring methods. For a decade, computers have been used to process radiographic data to produce cross-sectional displays, first of head, then of the whole body. Computer Tomography (CT) and magnetic resonance imaging (MRI) have grown into important diagnostic modalities. Contours, which are simple polygons representing the intersections of the surface of an object and the plane of a section, are automatically acquired from CT or MRI data by detecting discontinuities at adjacent densities of the object. These contours are processed further for modelling the object. The pioneering effort of modelling the surface of the human body with a computer was done by Keppel [16], who approximated the complex surface of a skull by triangulating the contour lines measured from a patient. Since then, there have been numerous algorithms to construct surfaces from contour lines [17-20]. With improved computational efficiency and the capability of handling complex structures, these algorithms have been used to model skulls [16], legs [21], and other anatomic structures [15, 22, 23]. Limitations of this kind of surface modelling method include the difficulties in differentiating adjacency of cross-sections and in dealing with branching structures where contours end, and the inefficiency of generating the contours, especially when sharp changes over the surface occur, causing very high density of points and contours.

In the research of scoliotic deformities, little has been done concerning surface modelling of deformed trunks. Hierholzer and Drerup [14, 24] first generated a three-



dimensional "surface" of the trunk. The "surface" was represented by a set of curves in the horizontal direction, a set of curves in the longitudinal direction, and the intersections of the two sets of curves. Pearson [9] used image processing algorithms and a spatial triangulation method to produce the three-dimensional data points. Similar systems can be found in literature [23, 25-27]. With these methods, no actual surface is available because the model is represented by either 3-D curves or dot patterns. As a result, there is lack of good visualization of the three-dimensional deformity and the model can be difficult to interpret. However, surface modelling of deformed trunks from discrete data points has not been well studied. With the popular video-based techniques used for 3-D measurement of trunk deformity, surface points with coordinates can easily be generated for clinical purposes. Therefore a method to construct a surface from these points is necessary to complete the techniques in the clinical setting.

### **1.3 Objective**

The data acquisition system for the trunk measurement in the Glenrose Rehabilitation Hospital uses a regular light pattern of dots and produces the 3-D points on the body surface. These 3-D points form the basis of surface modelling of the trunk.

The purpose of this thesis is to develop a method for modelling the trunk surface from discrete data points taken from a patient using the existing data acquisition system and to quickly provide clinicians with an accurate and meaningful representation of the trunk surface for the assessment of trunk deformity.

### **1.4 Structure of the Thesis**

In this work, different surface modelling approaches were compared. McLain's triangulation algorithm was selected to approximate a 3-D surface. This algorithm was modified to reduce the boundary effect allowing it to handle more complex structures. A surface modelling package was developed to generate a trunk model for clinical assessment of trunk deformity. In this package, an interface provides a user-friendly tool to portray and analyze the three-dimensional trunk deformity. The package has been designed to minimize human intervention.

Chapter 1 covers a general description of scoliosis and the objective of the thesis. In Chapter 2, different possible methods for modelling general objects are reviewed. A triangulation algorithm, which produces optimal triangulation, is selected for surface modelling. Chapter 3 describes software design issues in which important implementation considerations are presented. This chapter introduces the boundary effect of surface modelling as results from the original algorithm. The second part of the chapter concentrates on boundary processing, which includes literature review on the problem and the proposed solution to the problem when modelling the trunk. This chapter also deals with software design of the user interface for boundary modification program. An assessment of the proposed modelling method is covered in Chapter 4. Chapter 5 presents a conclusion of the thesis and discusses the limitations of the modelling method. As a result, the recommendations for further work are at the end of the chapter. Bibliography follows Chapter 5. Pseudo codes for the triangulation algorithm, and the modified algorithm with the boundary constraint are listed in Appendices A.1 and A.2. Appendix A.3 presents the flowchart of the whole modelling package, which links four programs used in modelling the trunk into one program. Source code for all the programs is available from Dr. Nelson G. Durdle in the Department of Electrical Engineering at the University of Alberta.

## **2. SURFACE MODELLING**

Object modelling is the technique of constructing the surface of an object based on partial information about the object. Reconstruction problems of this sort occur in diverse scientific and engineering domains, either to aid in the comprehension of the object's structure or to facilitate its automatic manipulation and analysis. These domains include geographical surveying [28] , computer simulation [29], industrial design processing (CAD/CAM) [29-30], and medical and biological research [31].

It is easier and more flexible to derive the geometric properties or attributes that the object might possess by analyzing the model rather than directly analyzing the actual object. Shape information of an object is an important attribute which describes the appearance of the object. In many applications, people are more interested in the shape information--especially shapes that do not have unique features, and whose characteristics are not well defined. It is also true in the understanding of trunk deformity, and in the understanding of the relationship between spinal deformity and trunk deformity.

### **2.1 Modelling of General Objects**

There have been three basic methods to represent a three-dimensional object: wireframe, surface and volumetric representations.

#### **1. Wireframe Models**

A wireframe model represents the boundary edges of an object and consists entirely of points, lines, and curves. There is no concept of surface: each facet is considered as a closed polygon without an interior area. Wireframe models are easy to construct and are computationally efficient. This method can be used when the object is

simple. However, using this method to model a general three-dimensional object has limitations [30, 32]:

-- It produces an ambiguous model because there is no concept of surface. Uniqueness of a model refers to the representation of an object where there is no question as to what is being modelled or represented, and that a given representation should correspond to one and only one object. Uniqueness of models is necessary for testing the equality or inequality of two objects [33]. Different objects can have the same wireframe model. For example, a rectangular solid in 3-D space is represented as a rectangle with wireframe model, while a rectangle consisting of four line segments in the 3-D space is also represented as the rectangle. Without other information, it is impossible to distinguish the solid from the four-line-rectangle with the wireframe models.

-- It lacks graphic or visual coherence. It is impossible to define a surface normal with only lines and curves. So hidden lines and surfaces can not be determined and further illustrated. As a result, depth information is lost.

These disadvantages make the technique inappropriate for representing clinical objects where unique, accurate three-dimensional models of the objects are necessary.

## 2. Surface Models

In this method, solid surface patches are used to represent the surface of an object. These solid surface patches produce a clear, unique model of the object. With hidden surface removal in computer graphics, a good visual image can be obtained.

Surfaces may be approximated by planar patches, parametric surface patches, or quadric surface patches. Because of the solid nature of the surface, this method is less-

efficient than wireframe modelling in computation time and memory requirements. Many investigations have contributed to improving the efficiency of this method. Various algorithms for surface modelling are presented and further discussed in Section 2.3.

### 3. Volumetric Models

Another method for modelling an object is volume-based modelling. With this method the object is considered a three-dimensional region consisting of a closed surface and the interior volume. Only volumetric models contain information about interiors, and thus they are used when the interior is of interest. Because of intensive computation involved for each pixel of the interior volume, the volume-based method is the most expensive in both computation and storage efficiency among the three modelling methods. For example, to manipulate a cube with a computer, the frame buffer required to store the cube has  $n^3$  pixels where  $n$  is the length of one side in pixels. Each pixel of the frame buffer must be calculated during the manipulation. If represented as a surface model, only  $6 \times n^2$  pixels are necessary for the cube.

For clinical assessment of trunk deformity associated with scoliosis, it is more desirable that the modelling system produce the accurate and unique surface model in a relatively short time. So the surface modelling approach is the most appropriate among the three approaches and so is selected to model deformed trunks.

## **2.2 Surface Representations**

Three of the most common surface representations are polygon mesh surfaces, parametric surfaces, and quadric surfaces.

A polygon mesh surface is a set of connected polygonally bounded planar surfaces. Open boxes, cabinets, and building exteriors can be easily and naturally represented by polygon meshes. Polygon meshes can be used, although less easily, to represent objects with curved surfaces; however, the representation is only approximate. Surfaces of objects are defined by arrays or meshes of polygonal facets. The more data points used, the more accurate the model is. Displaying meshes as filled polygons generates the visual effect of the surface being modelled. This effect is desirable in surface modelling. Manipulations of the resultant surface, such as rotation, scaling and translation, are the most efficient among the three representations because each polygonal patch has one and only one normal, which makes it much easier to define the direction of the patch, and to determine a 3-D point inside or outside a patch. Many graphics workstations now possess specialized hardware for the fast rendering of these surfaces [34].

Another class of surface representations is that of parametric surfaces. There are several important ones which have received attention in computer graphics in the last decade. These are B-spline surfaces, Bezier surfaces and Hermite spline surfaces.

The common denominator of the parametric surface representations is that they utilize a grid of control points, typically rectangular. Each set of control points describes a surface patch. Adjacent patches share control points on their edges. The differences between the representations in this class occur in how the surface defined by the control points relates to these control points. The two major classifications are those with surfaces which pass through the control points, and those that do not.

B-spline surfaces are described by a control polyhedron. The surface typically does not pass through its control points. This can make this representation difficult to use when

trying to fit the representation to data. B-spline surface representation was discussed intensively in [35-37].

The Bezier surface is also described by a characteristic polyhedron. The corner points of this polyhedron lie upon the surface. All other points, those along the edges and interior, define the 3-D curvature of the surface. More detail is available in the references [30, 33].

The Hermite spline surface [38] will pass through alternate control points. The surface passes through the first and third control points along each direction of the rectangular grid. The intermediate points serve to define control vectors which are tangent to the surface at the point where the surface passes through the control grid.

Parametric surface representations can define a large range of surface topologies more concisely than polygonal representations. Another important property of this class of surface representations is that transformations of a patch can be easily performed by transformations of the control points. Objects with a very large degree of surface irregularity may, however, require many small patches making this surface representation less efficient than polygonal representations. Though it is possible to determine on which side of a patch a point lies, it is difficult to classify points as inside or outside of a surface defined by multiple parametric patches if the surface is allowed to be concave [33, 34]. This makes calculation of volume, mass and center of gravity or other physical parameters of solid objects modelled via parametric surfaces more difficult. Moreover, if the modelled surface must be solid, not just surface outlines, the computation required to manipulate the resultant surface is much more extensive because the normal of each patch is not a constant to the patch, which makes it particularly difficult to render the surface. The above disadvantages of parametric surfaces make them difficult to use for certain tasks in

biomedical computing, which include surface modelling of trunk deformity discussed in this thesis.

The last type of surface representation is the quadric surface as described by Powell [39-40] and Barr [41]. They are an analytic representation of a surface. The quadric equation that describes the representations is typically utilized for the definition of solid volumes, though it can also be used to define surfaces of revolution as discussed by Uptill [38]. Primitive surface types such as spheres, cones, disks, cylinders and others can be represented with these surfaces. It is necessary to ensure closure of the volumes defined when using combinations of these surfaces to define solids. It is possible to easily classify 3-D points as inside, on or outside the volumes defined by these surfaces. However, the regular nature of the shapes which these representations can describe makes it poorly suited to many modelling and visualization tasks in biomedical applications, in which more irregular topologies need to be simulated.

In this thesis, data points used for modelling trunk surfaces are mostly irregularly (randomly) distributed. This property of data points make it difficult to use parametric or quadric representations because there is no regular control grid available. To analyze trunk deformity may involve many manipulations of the surface model. Therefore, being efficient in manipulation of resultant surface models, polygonal representations of trunk surfaces seem to be the most appropriate among the three possible surface representations.

### **2.3 Triangulation for Surface Modelling**

When using polygonal surface representations, triangulation is the best technique because triangles are simple geometric objects that can be manipulated and rendered easily and efficiently. Use of only three points per patch ensures that the coplanar restriction of a



facet is met. This restriction is necessary to eliminate ambiguities in the location of the surface and to allow for ease of rendering [34, 38].

### 2.3.1 Triangulation and the Property of "Optimal" Triangulation

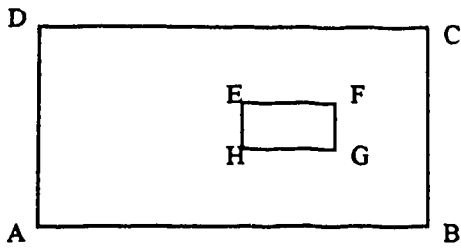
Two types of triangulation have been investigated in the literature: triangulation of data points and triangulation of polygons. In some cases, these two types of triangulation are combined to triangulate more general structures.

Let  $S$  be a finite set of points in the Euclidean plane. A triangulation of  $S$  is a maximal straight line plane graph whose vertices are the points of  $S$  [42, 43]. By maximality, each face is a triangle except for the exterior face which is the complement of the convex hull of  $S$ . Occasionally, a triangulation is called a general one to distinguish it from a constrained one for which a triangulation of a finite point set has some edges prescribed. A special case of a constrained triangulation is polygon triangulation where  $S$  is the set of vertices of a simple closed polygon where the edges of the polygon are prescribed.

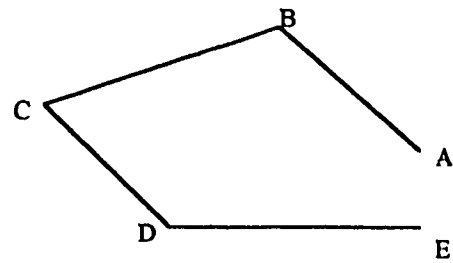
A triangulation of a simple polygon  $P$  with  $n$  edges is a partition of its interior into exactly  $n-2$  triangles. There have been various algorithms to triangulate simple polygons [44-47]. The fastest algorithm found in the literature for triangulating simple polygons is in the order of  $n \log n$ . Given a set of data with a pre-defined boundary represented as a simple closed polygon, one general technique to triangulate the data is to triangulate the polygon first, then insert the rest of the data points into the triangulated polygon [43, 48, 49]. In this case, the process of triangulating polygons plays an important role in triangulation of the given data.

A simple closed polygon is not multiply connected. With a simple closed polygon, there is no hole; when traversing the boundary of this polygon, every vertex is visited exactly once, no vertices are missed and the endpoint is the same as the start point. *Figure 2.1* shows a simple polygon and some examples of polygons that are not simple closed polygons. In geographical survey, a lake in an area to be modelled may be considered as a hole in the area. This area is not a simple polygon and thus a more general technique of surface modelling must be used.

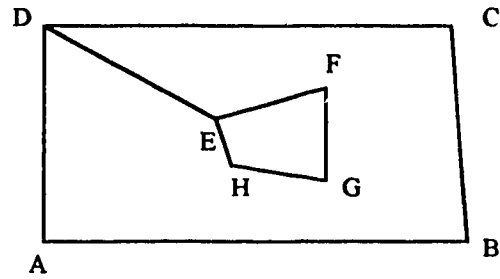
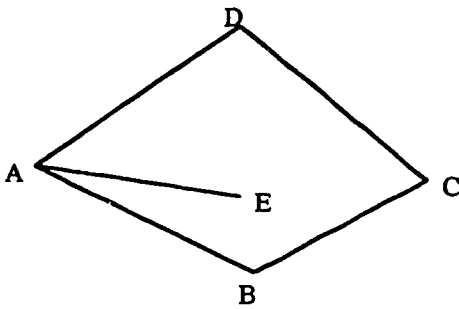
Some multiply connected polygons, which are not simple polygons, can be processed by sub-dividing the polygons into simple polygons. To triangulate a polygon, it is necessary to identify whether the polygon is simple, or multiply connected. In case of a simple polygon, most polygon triangulating algorithms can be directly used.



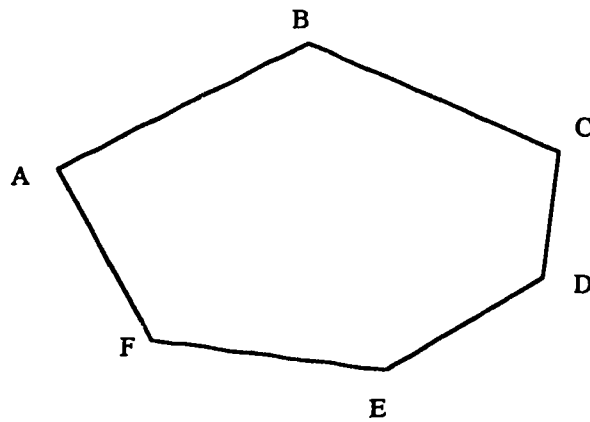
(a) a polygon with a hole.



(b) an un-closed polygon.



(c) multiply connected polygons EABCD AE and EDABCDEF GHE.



(d) simple closed polygon ABCDEFA.

Figure 2.1 Examples of different polygons. (a)-(c) are not simple closed polygons.  
 (d) shows an a simple closed polygon.

There are many different ways to group data points into triangles. It is often considered an "optimal" triangulation when used for surface modelling if a Delaunay circle restriction can be met [50-51]. A Delaunay circle is the circumcircle of a triangle where there are no other data points closer to the three vertices of the triangle. This property is also described as that the circumcircle of a triangle contains no points in its interior [42]. An illustration of this property is shown in *Figure 2.2*

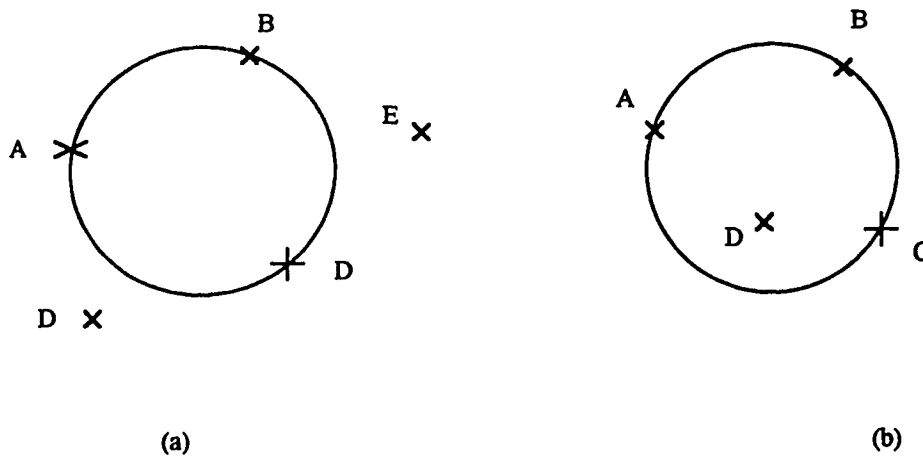


Figure 2.2 Circumcircles. (a) Delaunay circle ABC. (b) ABC is not a Delaunay circle because point D is inside the circle.

### 2.3.2 Assessment of Triangulation Techniques

Many algorithms can be used to approximate three dimensional surfaces. Different algorithms may be chosen based on particular requirements for the surface model. To select an algorithm, or algorithms, the following must be considered:

1. Quality: For the best approximation of a surface, triangulation algorithms must produce 'optimal' triangles for a given data set. Equilateral (or equiangular) triangles are optimal for the best interpolation over and between triangles. Therefore, triangles that are close to equilateral (equiangular) triangles are better than long thin ones [42].

2. Speed: The algorithms should be efficient, but more importantly, the computation time required should increase as slowly as possible as the number of data points ( $n$ ) increases. Many algorithms are at an order of  $n^2$ . There are some algorithms at an order of  $n \log n$ .

3. Generality: The algorithms should be capable of dealing efficiently with complex structures. In medical and biological research, complex surfaces, for which no pre-defined mathematical functions exist, need to be modelled. Surface modelling of the trunk deformity discussed requires a general algorithm which can handle complex structures.

### 2.3.3 Criteria for Triangulation

A set of points may be grouped into triangles in many different ways, producing triangles of various shapes and sizes, which affect the quality of the resultant surface. The simplest case can be shown with a quadrilateral. A convex quadrilateral  $ABCD$  (Figure 2.3) may be divided into triangles in two ways, ( $\triangle ABC$  and  $\triangle ACD$ ), and ( $\triangle ABD$  and  $\triangle BCD$ ). It is the purpose of the criteria to choose one of the two ways so that the resultant triangulation is optimal for the given data set.

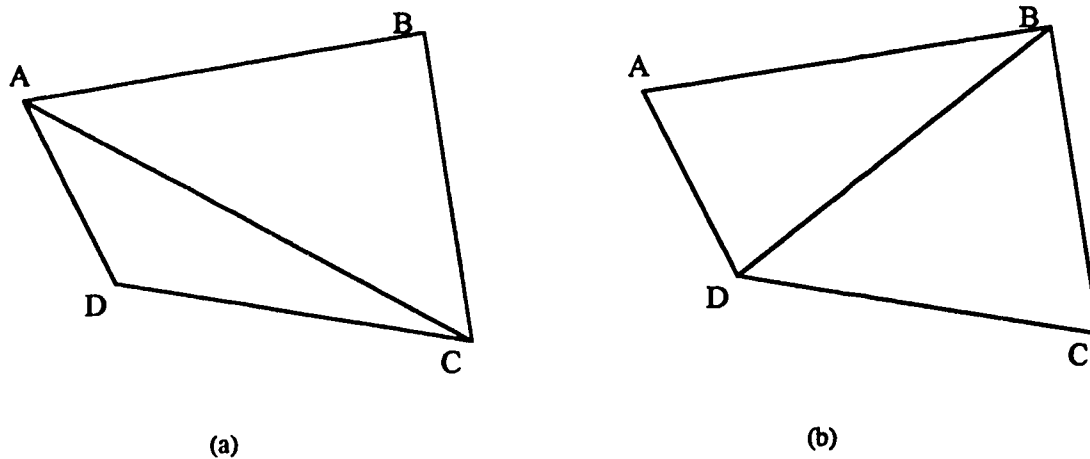


Figure 2.3 Different partitioning methods result in different triangles for the same quadrilateral. (a) produces a long thin triangle ACD which is considered not good for modelling a surface while (b) produces better triangles.

Three of the most common criteria for triangulation are: minimize diagonals, maximize minimum angles and minimize maximum angles.

1. Minimum overall length approximation and shortest diagonal partitioning: Rhind [52] stated an approach to minimize the total length of the triangle sides. Similarly, Gold [53] optimized the approximation by choosing the partition of the quadrilateral with shortest diagonals. These however are unsatisfactory as very thin triangles could be produced [54]. Sibson [55] combined these optimization methods into his "locally equiangular triangulation" approach in which a unique equiangular triangulation of the quadrilateral can be produced. Unfortunately there are few algorithms of this category available.

2. Maximize minimum angle: This is typically used in the Delaunay triangulation schemes. The criterion is applied to all triangulations of the same point set. This result can be extended to a similar statement about the sorted angle vector of the Delaunay triangulation [42] and to the constrained case [51]. With numerous algorithms available, the Delaunay scheme remains the most popular in triangulation of point sets. Preparata et. al. [56] showed that the Delaunay triangulation of  $n$  points in the plane can be constructed in the order of  $(n \log n)$ .

3. Minimize maximum angle: Edelsbrunner [42] studied the problem of constructing a triangulation that minimizes the maximum angle, over all triangulations of a finite point set, with or without prescribed edges. Although avoiding large angles is related to preventing the generation of small angles, this criterion does not maximize the minimum angle as the Delaunay triangulations do. Again, four points are sufficient to give an example to this effect (*Figure 2.3*). Edelsbrunner's algorithm runs in the order of  $(n^2 \log n)$ , which is the fastest one among those using this criterion. Triangulations that minimize the maximum angle have applications in the areas of finite element analysis and numerical methods [42, 57-58].

## **2.4 A Comparison of Algorithms for Surface Modelling**

### **2.4.1 Introduction**

There are many triangulation algorithms for surface modelling. Some have been used to model surfaces from contours, while others are available for approximating surfaces from discrete data points. In this thesis, emphasis is placed on the algorithms that construct surfaces from discrete data points because for surface modelling of trunk

deformity, the original data contains no additional information other than the three-dimensional coordinates of points. These data points are normally irregularly distributed.

#### **2.4.2. Recursive Insertion Approaches**

In this approach, each data point is sequentially inserted into an appropriate existing triangle such that the existing triangle is subdivided into three new triangles. The triangle list must be exhaustively searched for each insertion because more than one triangle may be affected by the insertion of the point. In *Figure 2.4*, when inserting point  $P$ , triangles  $\triangle ABC$ ,  $\triangle ACG$ ,  $\triangle GCF$ ,  $\triangle FCE$  and  $\triangle ECD$  are affected by the insertion. Thus the triangle list for triangulation is must be updated to  $\triangle ABP$ ,  $\triangle APG$ ,  $\triangle GPF$ ,  $\triangle FPE$ ,  $\triangle EPC$  and  $\triangle ECD$ . The last triangle ( $\triangle ECD$ ) is not affected by the insertion.

The general steps for triangulating a point set using a recursive insertion approach are: (1) find the triangle that contains the point to be inserted; (2) search for other triangles affected by the insertion; (3) construct new triangles after the insertion; (4) update the triangle list.



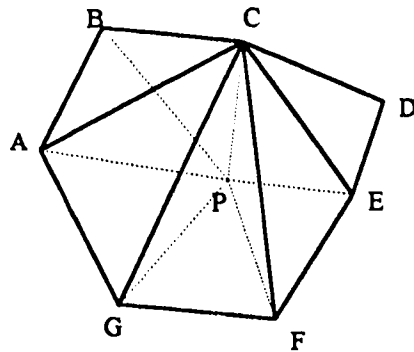


Figure 2.4 The insertion of point P affects existing triangle. The triangle list must be updated after the insertion of point P.

Some criteria are necessary to define triangles affected by the insertion of points. These criteria are derived from the criteria for "optimal" triangulation of a data set, which has been explained in the previous sections. Different algorithms use different ways of searching for the triangle containing the point, and different ways of updating the triangle list.

Sibson's [55] optimization method is based on a circumcircle to determine the partitioning of the quadrilateral when a new point  $X$  is inserted (*Figure 2.5*). The criterion selects  $BC$  as the diagonal of the quadrilateral  $ABXC$  if and only if  $X$  lies strictly outside the circumcircle of  $ABC$ ; it selects  $AX$  as the diagonal if and only if  $X$  lies strictly inside the circumcircle; and it allows either  $AX$  or  $BC$  to be selected if and only if  $X$  lies on the circumcircle.

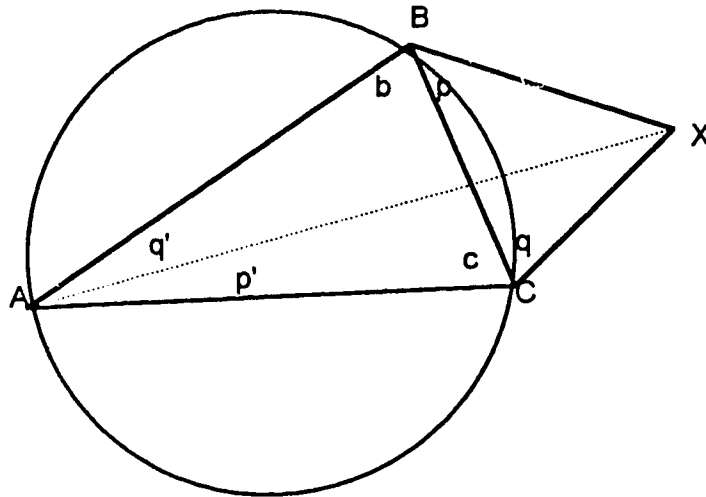


Figure 2.5 Partition of a convex quadrilateral by Sibson's method

Sibson used the Max-min angle criterion to decide if the point was inside, outside, or on the circumcircle of an existing triangle. A proof of uniqueness of the triangulation method was presented. However, Sibson could not present a practical algorithm which applies this method to triangulate a data set, because searching for the triangles affected by the point to be inserted contributes the majority of the computational requirements of the triangulation process.

### 2.4.3 Hierarchical Triangulation

Hierarchical triangulation has been being investigated for surface representation [49, 59-62]. With this approach, the triangulation is described by a segmentation tree. The

root corresponds to the initial enclosing triangle, whereas other nodes represent triangles resulting from subdivision of their parents. The hierarchical models of the surface representation are based on the recursive subdivision of the surface into nested triangles. Hierarchical triangulation provides representations of a surface at different levels of resolution, which allows a reduction in the number of points needed to describe the shape of the surface. Hierarchical triangulation is particularly appropriate for modelling a surface with variable-resolution, or for monitoring the rendering efficiency when displaying the surface.

The most common forms of hierarchical triangulations are the *ternary* and the *quaternary hierarchical triangulations*. In a ternary triangulation, a subdivision of a triangle  $T$  consists of joining an internal point  $P$  to the three vertices of  $T$ , and thus forming three sub-triangles incident at point  $P$  (*Figure 2.6*). While in a quaternary triangulation each triangle is subdivided into four sub-triangles formed by joining three points, each lying on a different triangle side. The major problem with a ternary triangulation is the elongated shape of its triangles, which leads to inaccuracies in numerical interpolation [63]. A triangle in a quaternary triangulation may have more than one neighbor along each edge. The resulting surface is generally not continuous, except when all of the triangles are uniformly split [62]. These shortcomings of hierarchical triangulations prevent them from accurately modelling a smooth surface.

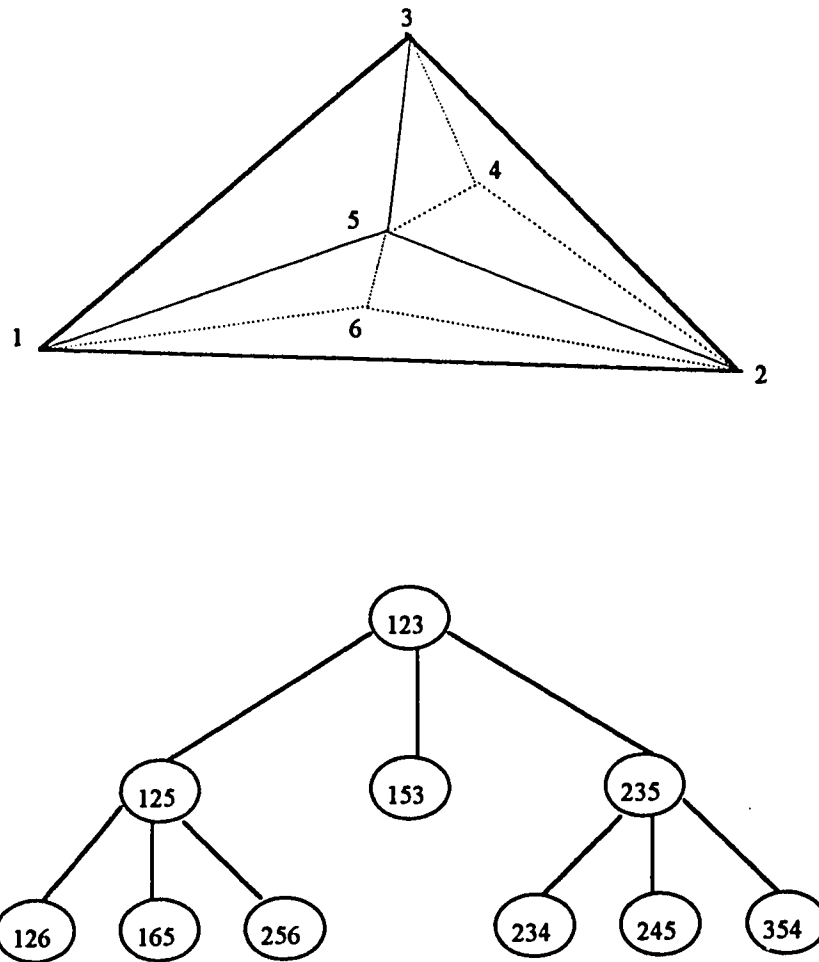


Figure 2.6 A ternary triangulation and its tree description

Palacios-Velez [64] combined recursive insertion and hierarchical subdivision for computing the Delaunay triangulation of data sets. An improvement in searching for the triangle containing the query point is made by applying the Oriented Walk Search. With this search, the sign of the area of a triangle is used to decide which of the neighboring

triangles contains the query point. The triangles affected by the insertion are searched using the criterion that a Delaunay triangle does not contain any points inside its circumcircle. All triangles whose circumcircles contain a new point will be affected by the insertion. However, rounding errors in computing the areas may cause inconsistent sign determination and lead to cycling in the search process [65-66]. Some additional process must be applied to solve this problem.

#### **2.4.4 Polygon Decomposition Approaches**

In these approaches, the region supporting the data points may be considered to be a polygon. Some techniques triangulate the data set by decomposing the polygon and then inserting each point into the triangulated polygon. Another technique, Lewis' algorithm, splits the polygon into sub-polygons until all sub-polygons become triangles.

##### **1. Polygon Triangulation and Point Insertion**

With this approach, the polygon is first triangulated using a polygon triangulation technique. Points are sequentially inserted into appropriate triangles. Then triangles are updated. The fastest algorithm to triangulate a simple polygon is in  $O(n \log n \log n)$  [45]. However, this approach produces long thin triangles which are not suitable for smooth interpolation. To modify all possible triangles in the polygon when insertion occurs, however, is computationally intensive and may be in the order of  $(n^2 \log n)$  [42].

##### **2. Lewis' Algorithm**

Lewis [67] triangulates the data set by: (1) splitting the region into sub-regions by creating a new boundary across the region, and (2) solving the triangulation problems for each sub-region separately.

There are numerous ways to split a region. Lewis' algorithm attempts to make the split which divides the region into two equally sized halves while keeping the halves as circular as possible. This is achieved by applying a product  $\Pi$  whose factors are the signed distances of the boundary points from the proposed split line. The  $\Pi$  is calculated by the two parts with each for one half of the region. For example, in *Figure 2.7*,  $\Pi_1 = L_1L_2$ ,  $\Pi_2 = S_1S_2S_3$  and so  $\Pi = \Pi_1\Pi_2$ . By calculating  $\Pi$ , it is also determined in which half of the original region points belong to. Points with the same sign as  $\Pi_1$  are all in the first half, and points with same sign as  $\Pi_2$  belong to the other. When calculating the partial product, say  $\Pi_1$ , some splits are rejected if one of the elements for  $\Pi_1$  is opposite in sign to that of its predecessor. This rejection avoids a split as shown in *Figure 2.8* where  $S_1$  and  $S_2$  are on different sides of the split line. Therefore the partition of the region is most likely to be equally sized.

All possible splits of the region are considered. The chosen split maximizes the expression

$$\Pi * E^b$$

where  $E$  is the minimum of

- (a) half of the average distance between the boundary points, and
- (b) the distance from the split line to the nearest interior points contained within any triangle having the split line as a side.

Using the above expression tends to force the split into equally sized halves, achieves the greatest problem reduction, and makes further split of regions easier.

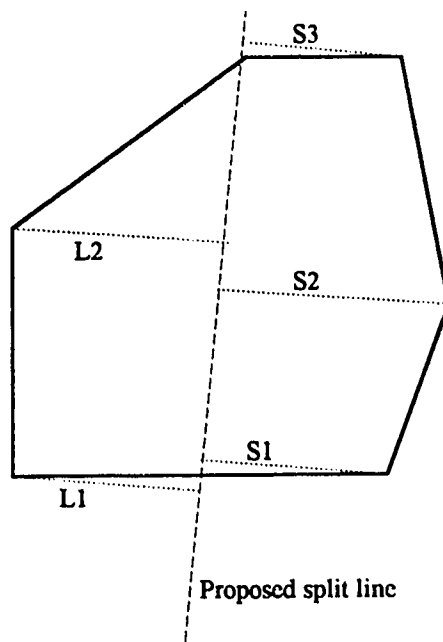


Figure 2.7 Calculation of  $I$  and determining to which half the internal points belong to achieve "equally sized partition" of the polygon.

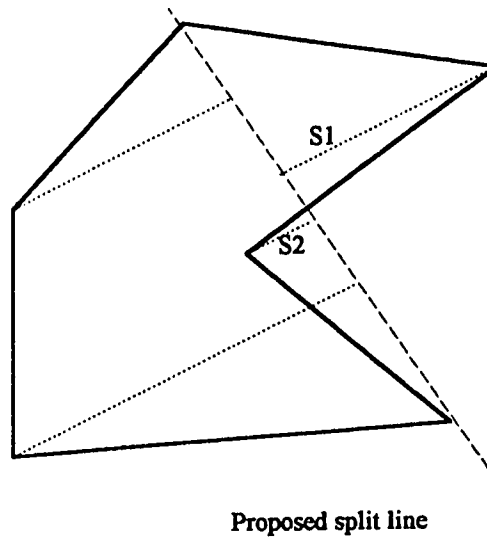


Figure 2.8 Rejection of split:  $S_1$  and  $S_2$  on different sides of the proposed split line, the split is rejected.

Other algorithms for splitting regions are given in references [44, 68-70]. In these algorithms, a polygon is first divided into convex sub-polygons. Then each convex sub-polygon is triangulated separately. Finally all triangulated convex sub-polygons are merged together with the sub-polygons which are concave parts of the original polygon deleted. A problem with these algorithms is that with a complex polygon, it is not possible to detect which part is convex and which is concave.

Lewis' algorithm is faster and more general than other polygon decomposing approaches. Depending on the data points, however, the resultant triangulation may not be "optimal" because no Delaunay circles are used when grouping triangles [71]. Another shortcoming of the algorithm is that data structures used in the implementation are very complex, because the boundaries of regions being split and to be split must be stored and are heavily used, while the directions of the boundaries must be kept consistent. The



initial boundary of the discrete data points must be decided accurately even before the triangulation can start.

#### **2.4.5 McLain's Algorithm**

Unlike the preceding approaches, McLain [72] triangulates the data set from a small area and grows outwards, increasing the triangulated area when new triangles are added. At each step, the algorithm uses a Delaunay circle of an existing triangle to search for the nearest point to be connected into a new triangle with one of the edges of the existing triangle. The three vertices of the new triangle are the two endpoints of one of the edges of the existing triangle being evaluated and the nearest neighboring point of the existing triangle in the data set. *Figure 2.9* shows an example of this process for a two dimensional case. Note that the line  $BC$  is an edge of exactly one existing triangle.

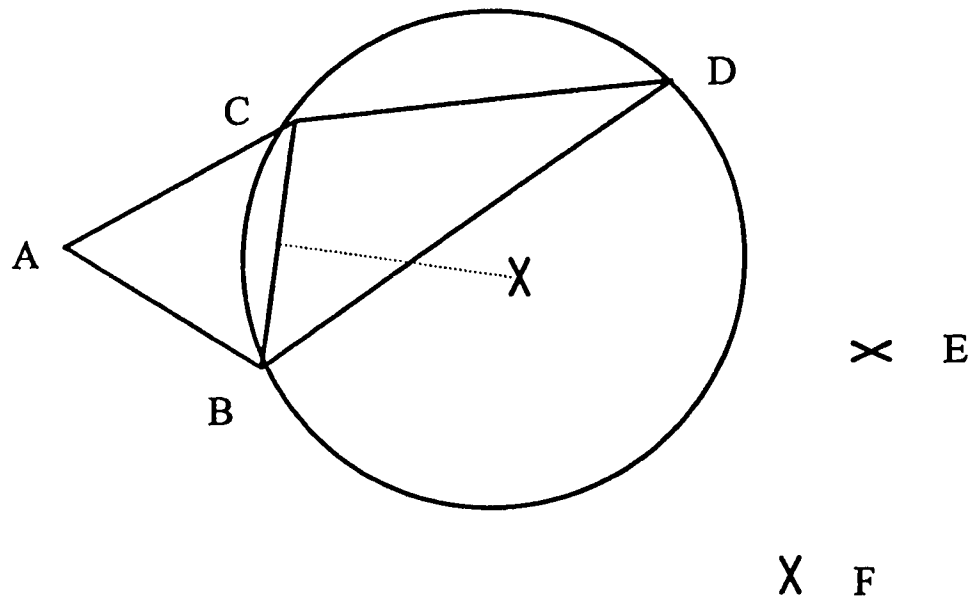


Figure 2.9 McLain's triangulation algorithm. Searching for the next point from D, E and F for edge BC of the existing triangle ABC. D is selected due to the minimal distance from the circumcenter of BCD to the edge BC.

When evaluating a point with a line, the signed distance from the circumcenter to the line is considered. The point which has the minimum distance is selected as the nearest point for the line and connected to the next triangle. The distance is measured in a way that the distance from the facing vertex of the existing triangle to the line under evaluation is negative, e.g., for line  $BC$ , distance from  $A$  to  $BC$  is negative. All points and circumcenters on the other side of line  $BC$  opposite to  $A$  have a positive distance. All points on the same side of line  $BC$  as  $A$ , which have a negative distance, are not considered for line  $BC$ . Each triangle is ensured to be a Delaunay triangle.

The algorithm starts from either a line on the boundary of the region or with a line joining any point to its nearest neighbor. It terminates when no point can be found on the appropriate side of any of available lines.

In three dimensional space, the algorithm triangulates a point set by grouping the points into fours and partitioning the region into tetrahedrons. Instead of a circle, a sphere through four points and its center are used to determine the nearest point for the next tetrahedron (*Figure 2.10*). In this case, the distance from the sphere center to the plane must be calculated. While the number of the tetrahedrons increases, the triangulated 3-D region approximates the region of the object, whereas facets of tetrahedrons approximate the 3-D surface of the object.

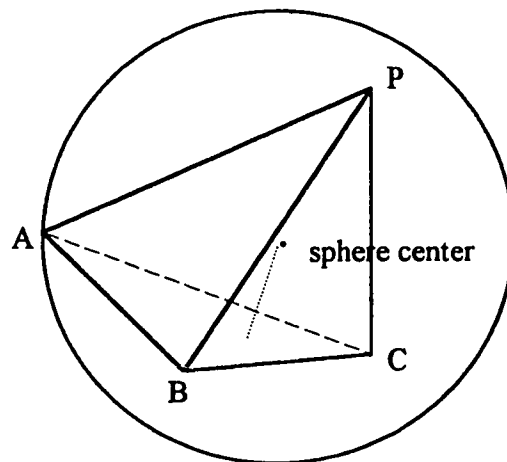


Figure 2.10 McLain's algorithm in 3-D space. A sphere through A, B, C and P is found.

Distance from the sphere center to the plane ABC is used to determine the nearest point for the next tetrahedron.

McLain's algorithm produces smooth surfaces because of Delaunay triangulation. The data structures are relatively simple compared to previously described algorithms. Because signed distances from points to lines are used to evaluate valid points for appropriate triangles, the algorithm can not terminate until the triangular representation of points becomes a convex. Therefore even if the region of points is concave, the algorithm will produce a convex representation of the region. Consequently, the algorithm itself is not capable of handling complex structures in which regions of concavity are often inevitable.

## **2.5 Conclusion**

McLain's algorithm is attractive because it produces smooth surfaces due to its optimal triangulation. It uses a least square fit of the weighted average of the three functions of a triangle to ensure a smooth transition between triangles. The data structures are simple compared to the other algorithms described. With the modifications described in Chapter 3, the algorithm can also handle relatively complex cases.

McLain's algorithm was chosen as the most appropriate method to model the trunk. The following sections describe the algorithm and discuss implementation issues.

### 3. 3-D SURFACE CONSTRUCTION

#### **3.1 Modelling in 2-D Instead of 3-D**

McLain's algorithm triangulates three dimensional data points by tetrahedralizing the points. The basic operation is calculation of the sphere center, which is a third-order polynomial calculation compared to a second order calculation in two dimensions. The number of combination for four points in the data set to decide a sphere in 3-D cases is much larger than the number of combination for three points in forming a circle in 2-D. Thus, the algorithm is more computationally intensive in 3-D cases than in two dimensions.

Children with moderate or severe scoliosis may have a folded waist region. This folding over is referred to as manifold. Since the manifold area can not be detected using the current video-based imaging technique, surface modelling of trunk deformity does not need to handle this complex case. By not handling manifolds, the trunk surface can be modelled more simply with 2-D triangulation by applying Boissonnat's proof of the equivalence [48], which states that the triangulation in 3-D can be achieved by triangulating the points in 2-D if the points are not folded over (*Figure 3.1*).

Surface points can be analyzed in a 2-D projection plane by ignoring one of the three coordinates. By triangulating points in the 2-D plane, the three dimensional trunk surface will be approximated. The computation required for triangulation is reduced considerably by: a) calculating the second order polynomials (circles) instead of the third-

order polynomials (spheres), and b) the decreased number of possible triangles compared to the number of possible tetrahedrons.

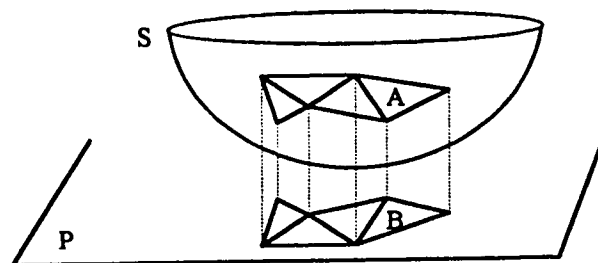


Figure 3.1 Triangulation of 3-D points achieved by triangulation in 2-D.

There is an exact mapping between the 3-D triangles on a sphere and the 2-D triangles on its projection plane (P).

### **3.2 Possible Problem Areas**

Some potential problems may occur depending upon the distribution of data. If these problems are not solved properly, they can either cause major computer system problems such as System Down, or produce incorrect triangular representation of given data.

### 3.2.1 Modification of Line Directions

As described in McLain's algorithm, signed distances are considered when determining the point to be used to form the next triangle. Each time a triangle is connected, the direction of each of the two new lines for this triangle must be updated (*Figure 3.2*).

Suppose triangle  $\triangle BPC$  is the next triangle for line  $BC$ . Lines  $BP$  and  $PC$  must be updated according to their facing vertex of the triangle for further triangulation. The equation of a line (say  $BP$ ) " $ax + by + c = 0$ " is determined by its two endpoints ( $B$  and  $P$  in the example). With this equation, the distance from the facing vertex ( $C$ ) to the line  $BP$  can be positive. If this is the case, line equation parameters ( $a$  and  $b$ ) must be modified so that the distance from the facing vertex ( $C$ ) to the line ( $BP$ ) is negative. A similar process is applied to line  $PC$  with a facing vertex  $B$ .

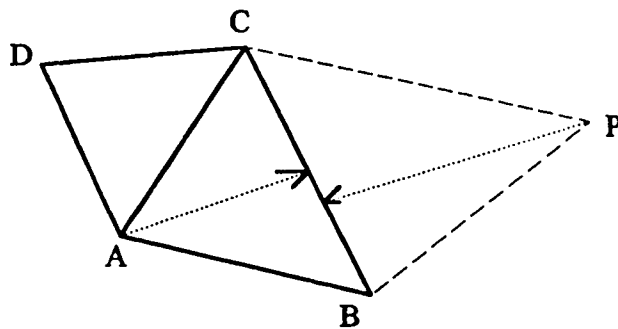


Figure 3.2 Modification of line directions

The distance from a point to the line is used to decide if the point is valid for a line, i.e., if the distance is negative, which means that this point is on the same side as the line's facing vertex, then the point is rejected and no circumcircle is calculated for this point; All points having a positive distance to the line are considered valid and circumcircles are calculated for these points to determine which point is appropriate for the next triangle.

### **3.2.2 Points Sharing the Same Circumcenter**

There is ambiguity if more than one point is equidistant and closest to the line under examination. This ambiguity may cause the triangulation process to cycle by adding more and more new lines, producing intersecting triangles. An example of the cases is shown in *Figure 3.3* where, when evaluating a line  $AB$  for a nearest point, more than one point ( $D$ ,  $E$  and  $F$ ) are on the same circumcircle. In this case, the distance from each of the circumcenters to the line  $AB$  is the same. A decision must be made as which point will be the new vertex.



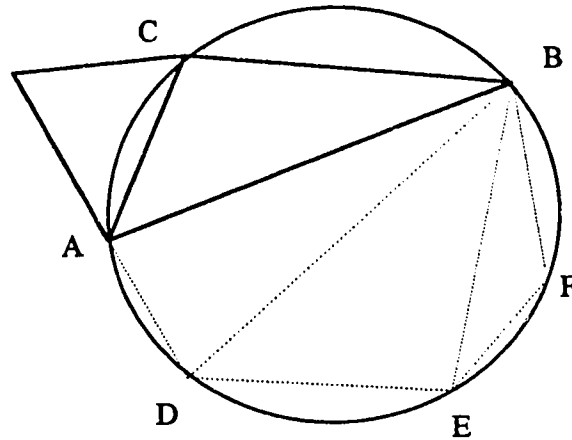


Figure 3.3 Points sharing the same circumcircle

A maximum angle criterion is used in this case. The procedures are : a) calculate the angles  $\angle DAB$ ,  $\angle EAB$ , and  $\angle FAB$ ; b) sort the angles into decreasing order; c) connect triangles according to the sorted angles, i.e., triangles  $\triangle ADB$ ,  $\triangle DEB$  and  $\triangle EFB$ . This criterion ensures the triangulation to be performed properly. An angle between two lines can be calculated using the dot product of their position vectors.

### **3.3 Implementation Issues**

#### **3.3.1 Input/Output and Data Structures**

##### **1. Input/Output**

The input data are a set of data points each with three coordinates  $x$ ,  $y$ , and  $z$ . The output of the modelling program is a set of triangle connections. For better efficiency and

simpler processing, instead of writing all three coordinates of each vertex of a triangle, only the index of each vertex of the triangle is written into the output file (*Figure 3.4*).

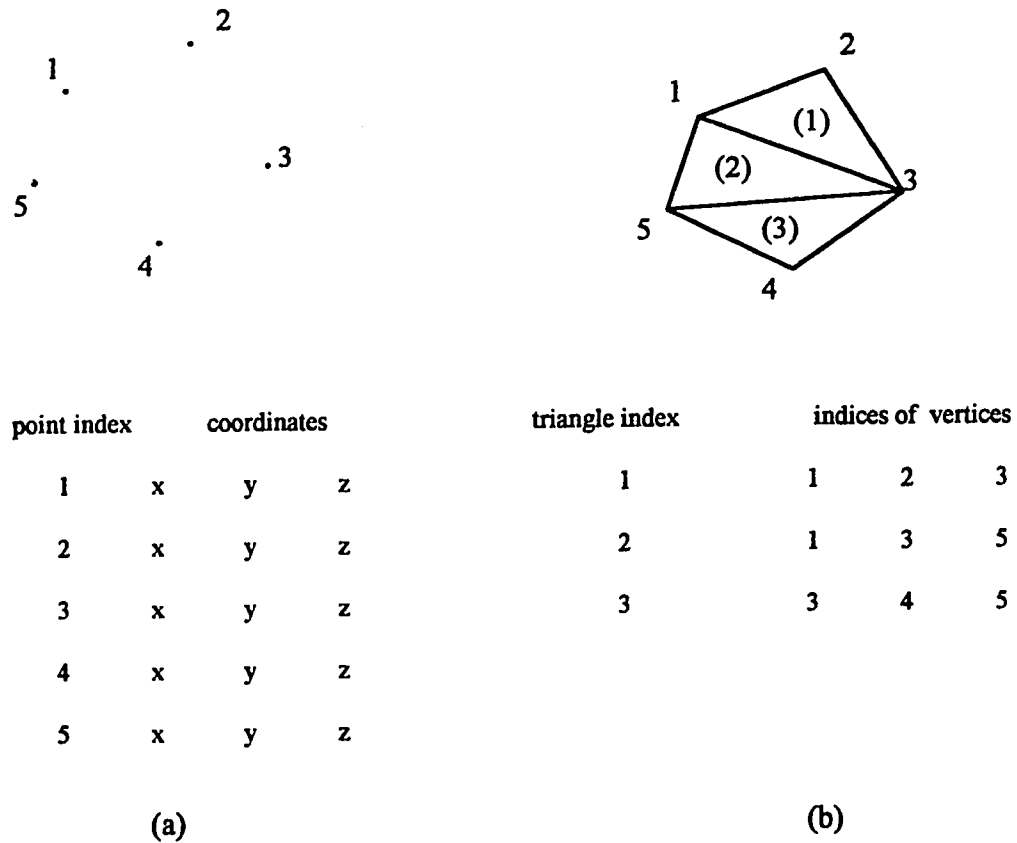


Figure 3.4 Input and output data. (a) input data. (b) output data.

## 2. User defined types

A user defined type "Point\_Type" is used to declare a point with three coordinates  $x$ ,  $y$ , and  $z$ . All data read from the input file are stored as an array of points. During the triangulation, circumcenters are of type Point\_Type which have  $x$ ,  $y$ , and  $z$  coordinates.

A line is defined by its two endpoints with its direction adjusted against the facing vertex of the triangle. A user defined type "Line\_Type" is used to hold line information, having fields for indices of the two endpoints and the index of the facing vertex of a line.

The valid lines of existing triangles are stored into a line list of type "List". Once a new triangle is connected, two new lines are constructed. The basic operations with the line list include: a) checking through list elements against new lines to determine whether the new lines are to be appended; b) removing the top element off the list when no points are found for a line. In the later case, the line just evaluated becomes the boundary line of the triangulated region, while the line just removed from the list is the new line to be evaluated for the next iteration of triangulation. The triangulation terminates when the list is empty. *Figure 3.5* shows the main data structures (in C programming language) described above.

### **3.3.2 Format of Data Points**

Data is acted upon via indices. Each point must have a distinct index, otherwise intersecting triangles may be generated. Ensuring the points to be distinct can be done by calculating the distances between each pair of points. If the distance is zero, the point set is not distinct. In this case an additional process for re-organizing the indices of points is necessary before the triangulation process begins.

```
typedef struct {  
    float    x, y, z;  
} Point_Type;  
  
typedef struct {  
    int      endpoint1, endpoint2,  
           facing_vertex;  
} Line_Type;  
  
typedef struct {  
    Line_Type    line;  
    Line_Type    *next_element;  
} List;
```

Figure 3.5 Data structures for triangulation

### 3.3.3 List Operations

As briefly mentioned in section 3.3.1, basic operations with a line list are `Create_List`, `List_Append`, `List_Delete`, `List_Pop`, `List_Empty`, and `List_Destroy`.

**Create\_List** creates an empty list with a list name and a pointer to the head of the list. This operation is performed before the triangulation begins.

**List\_Empty** checks whether the list is empty. The value returned is Boolean, either TRUE or FALSE. Triangulation is terminated if the list is empty (TRUE). Otherwise (FALSE), the top of the list can be popped for the next triangulation by operation **List\_Pop**.

**List\_Pop** retrieves the top element of the list. The line returned from the list is generally used for the next iteration of the triangulation process. This operation also updates the list by deleting the top element and moves the Top of the list to the next element.

**List\_Destroy** removes a line list by freeing the memory occupied by the list. It also frees the memory for the list pointer.

**List\_Delete** and **List\_Append** are used to update the list during triangulation depending on the status of the point found for this triangulation. At the beginning of the triangulation process, each point has the default status of *Unconnected*. Once a point is connected into a triangle, its status is changed to *Connected*. When forming a new triangle, the status of the latest point for the triangle is used to determine the corresponding list operations. The following situations must be considered in adding triangulated patches:

If the point is *Unconnected*, one of the two lines is appended into the list while the other is used for the next iteration of triangulation (*Figure 3.6*). Which of the two lines is chosen to be appended is not a problem as long as the order of the append is kept consistent.

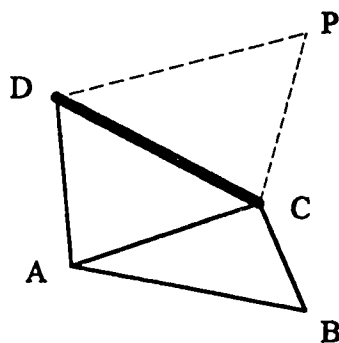


Figure 3.6 List operation when point (P) is *Unconnected*. New lines are CP and PD.

Line CP is appended into the list while PD is used for next triangulation.

If the point is *Connected*, the two lines that have the connected point as an endpoint must be checked against the list. If only one line is found, that list element is deleted (*List\_Delete*) and the other line is used for the next triangulation (*Figure 3.7-a*). If both lines are found, then both elements are deleted (*List\_Delete*) from the list and the two new lines are disregarded. However, the connection of this triangle is valid. Thus the triangle connection is written into the result file, which prevents generating a hole on the resultant surface (*Figure 3.7-b*). If none of the lines are found in the list, one of the lines is

appended onto the list (List\_Append), and the other is used for the next iteration of triangulation (*Figure 3.7-c*).

When checking for a line through the list, the two endpoints of the line are compared with that of each of the list elements, until they are found for a list element, or failing that, the Tail of the list is detected. The order of the endpoints for a line and the direction of the line do not need to be considered when comparing the line with a list element.

### 3.3.4 Direction of Triangulation

A triangle is a surface patch. The normal of the patch is used to determine the direction--either front or back facing of the patch. If the normals of patches are not consistent during triangulation, some patches will be flipped over producing an incorrect surface.

The problem of maintaining the correct patch direction is solved by keeping the order of triangulation consistent. Suppose there is an existing triangle  $\Delta ABC$  with line  $BC$  at the top of the list. Once the line is popped, line  $BC$  is taken out of the list for the next triangulation. The line must be changed to  $CB$  during this iteration of triangulation while its facing vertex remains unchanged, so the direction of the next triangle is  $\Delta BPC$  instead of  $\Delta BCP$  (*Figure 3.8*).

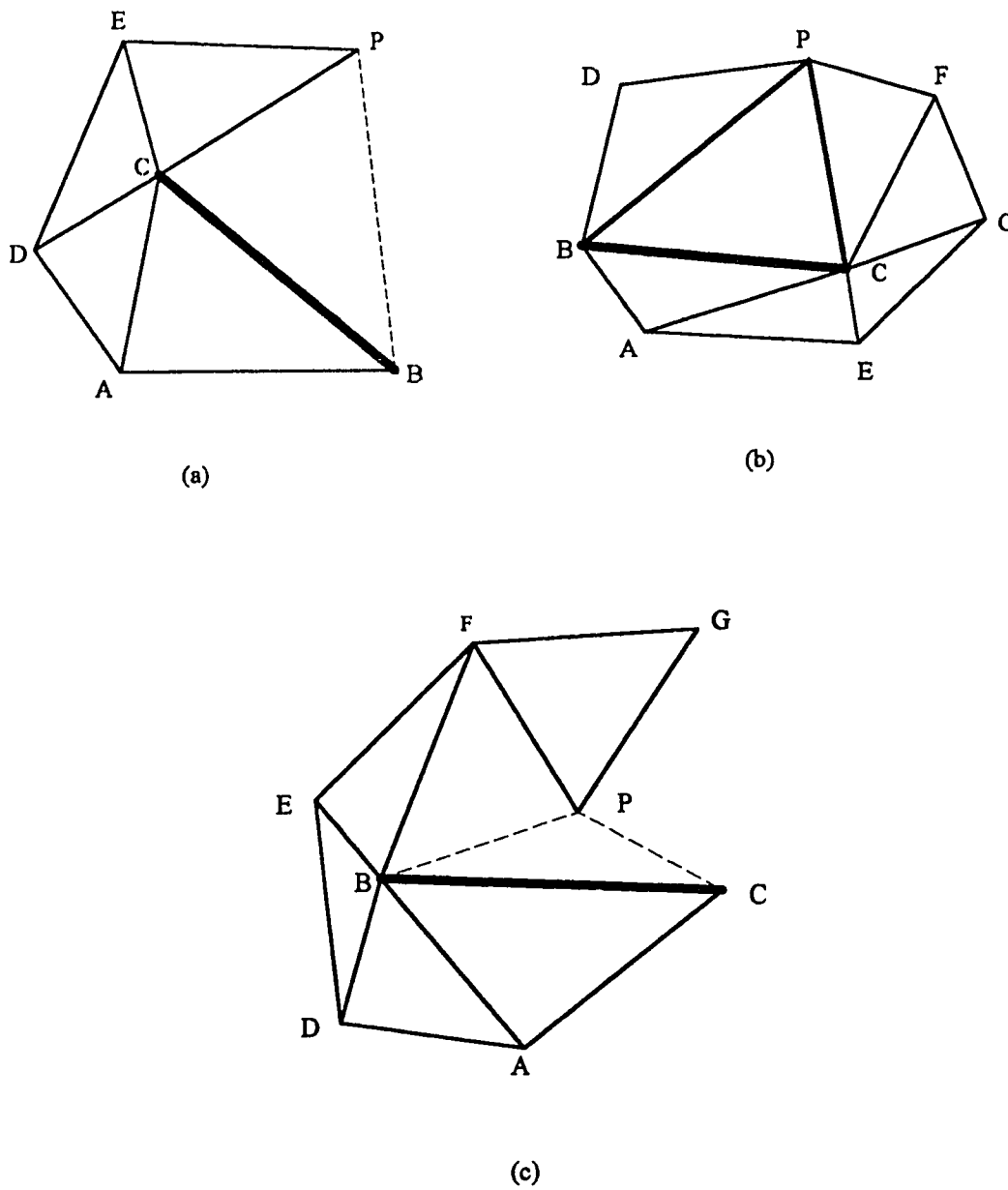


Figure 3.7 List operation when Point ( $P$ ) is *Connected*. (a)  $PC$  is deleted from the list while  $BP$  is used for next triangulation. (b) Both lines  $BP$  and  $PC$  are deleted from the list but  $\triangle BPC$  is valid. (c) No line in list, line ( $BP$ ) appended onto list, the other line ( $PC$ ) is used for next triangulation.



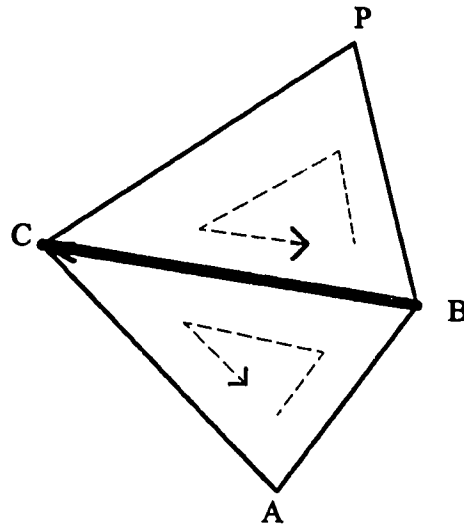


Figure 3.8 Change of direction of triangle. Line direction is B to C for  $\Delta ABC$ . When evaluating for the next triangle, the direction is changed to C to B for  $\Delta BPC$ . The normals of triangles are kept consistent.

### **3.4 Unconstrained Triangulations**

Appendix 1 lists the pseudo code for the main triangulation program and the pseudo code for the routine which is used in the main program to search for a point to triangulate. In the main program, the name of the file containing the input data points is given while assuming that the data in the file are already formatted and identical as described in Section 3.3.1.

The triangulation program triangulates a given set of discrete data points with an output file containing the triangle connection list. In three dimensional space, when rendered as a surface mesh, the planar triangles approximate the 3-D surface of the object.

If the region of points on the 2-D projection plane is convex, the modelling program can produce an accurate and smooth representation of the 3-D surface of the object. However, if the region is concave, the triangulation program, which continues until no points can be found for every valid line, produces some long thin triangles across the concave regions. These long thin triangles become surface artifacts when the triangular connections are rendered as surface patches in the 3-D space (*Figure 3.9*).

The situation in which triangulation algorithms always construct convex triangular representation of a data set is known as the "boundary effect" [73]. This is because triangulation algorithms continue until no points can be found for any valid lines, i.e., until the triangulated region is convex.

The trunk surface is normally a region of concavity. With the algorithm above, the surface model generated from a data set contains some long, thin triangles, which are typically across the boundary of the data and stretch between the shoulder and near the hip. *Figure 3.10* shows the boundary effect when a trunk surface model is displayed in the

3-D space, from which the artifacts obviously affect the visualization of the trunk model, and thus affect the clinical assessment of trunk deformity.

Therefore, the algorithm previously described can not be directly used to model a trunk surface. It is necessary to develop some constraints applied to the triangulation process to produce an accurate surface model without visual artifacts.

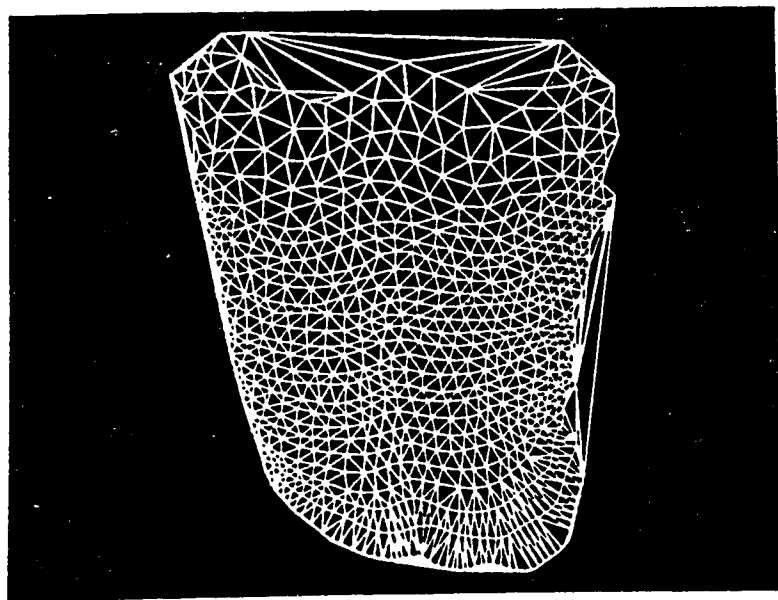


Figure 3.9 Triangulation of a concave region becomes a convex.

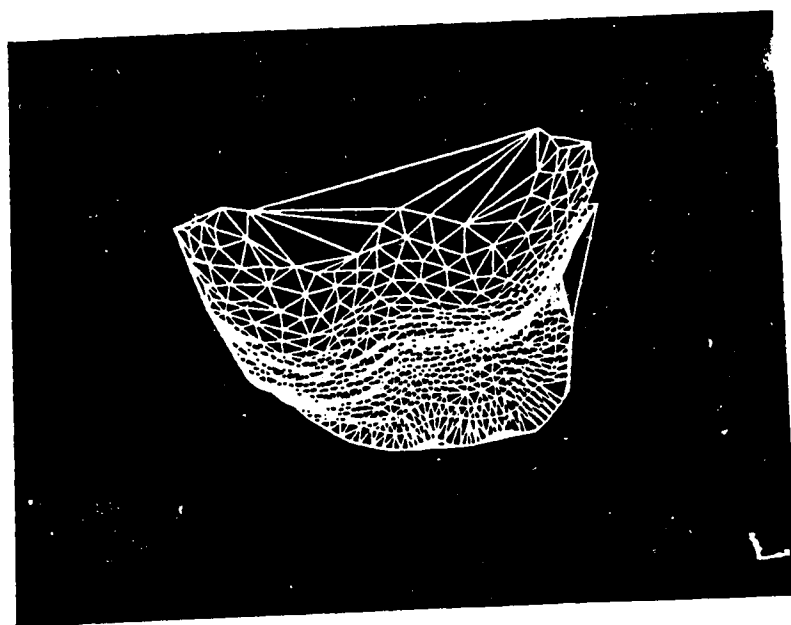


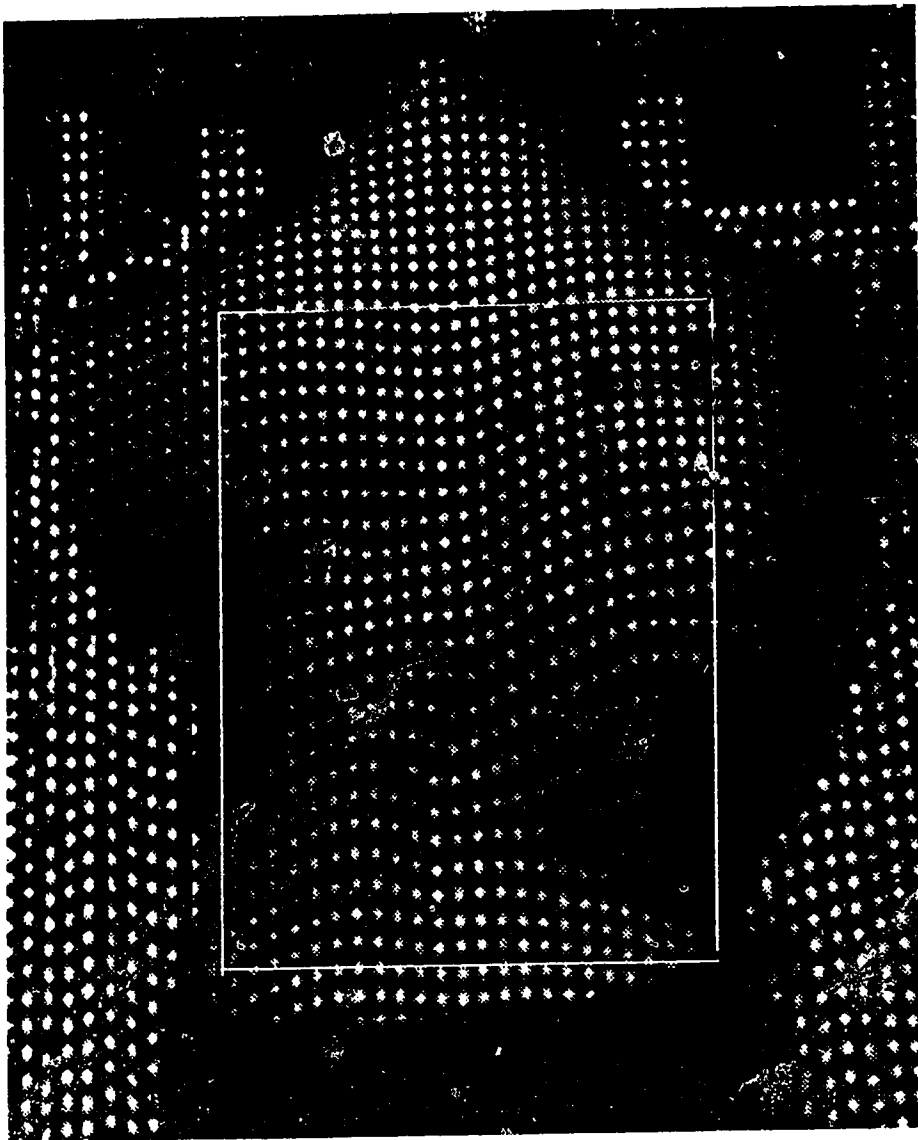
Figure 3.10 Artifacts in a trunk model

### **3.5 Boundary Processing**

#### **3.5.1 Introduction and Review of Literature**

The boundary effect causes the resultant model mis-represent the actual surface an object. With triangulation, this typically generates a convex hull representation of data set in a two-dimensional case. In three-dimensional space, however, the 3-D data set will be represented as convex polyhedrons with facets of the polyhedrons being convex polygons.

The solutions seem to be based on the principle of ignoring all information outside an arbitrary limit or window [73]. In this case, all data outside the window are disregarded when triangulation proceeds. Consequently, the 2-D triangulated region of the data must be totally contained in the window. When using this solution to model the trunk surface, some data points, especially those on the shoulders or near the hip, must be ignored to generate a trunk surface without the boundary effect (*Figure 3.11*). In *Figure 3.11*, the 3-D surface model is not complete for clinical assessment because 3-D trunk deformities are often indicated by a tilted pelvis, unbalanced shoulders, and twisted trunk. The concave side areas of a trunk are important for clinicians to analyze the degree of the deformity, the possible progression of the whole trunk, and to decide the treatment for the particular patient. Therefore, it is more desirable that most of the data be used during modelling for a more complete view of the 3-D deformity of the trunk.



Hoppe et. al. [74] reduced the boundary effect by using a "Marching Cube" contouring approach. During contour tracing, a function at vertices of a cubical lattice is sampled. Then the contour intersections with tetrahedral decomposition of the cubical cells are determined. To accurately estimate boundaries, the cube size must be set so that edges are of length less than  $\rho + \delta$ , where  $\rho$  is density of points and  $\delta$  is the sampling noise. In practice this value is often set a few times until the accurate boundaries are obtained. This method directly estimates the 3-D boundaries of a 3-D data set. However, the algorithm "can not guarantee the result to be correct". For surface modelling of trunk deformity to be used in clinical assessment, the above algorithm is not an appropriate method to reduce the boundary effect.

Some researchers [69-70] tried to decompose the polygon of the data set. In the two-dimensional case, the polygon is decomposed into convex and concave sub-polygons, with each of these sub-polygons being convex. In *Figure 3.12*, the original polygon  $P$  ( $ABCDEF$ ) is concave (*Figure 3.12-a*). This polygon is divided into two parts  $P_1$  ( $ACDEF$ ) and  $P_2$  ( $ACBA$ ). Each of these two sub-polygons itself is a convex polygon while  $P_2$  is the concave part of  $P$  (*Figure 3.12-b*). Then  $P = P_1 - P_2$  which is shown in *Figure 3.12-c*. With this approach, the reconstruction of the surface for the original data set in the concave polygon  $P$  is achieved by the reconstruction for the convex polygons  $P_1$  and  $P_2$  with the concave part of  $P$ , which is  $P_2$  in the figure, deleted.

This approach works well in some simple cases like that shown in *Figure 3.12*. Concave parts of polygons can be determined by traversing the boundary points and recording the changes of the traversing direction [69]. However, for complex structures, this does not work well because determining the concavity of a polygon usually involves human judgment.

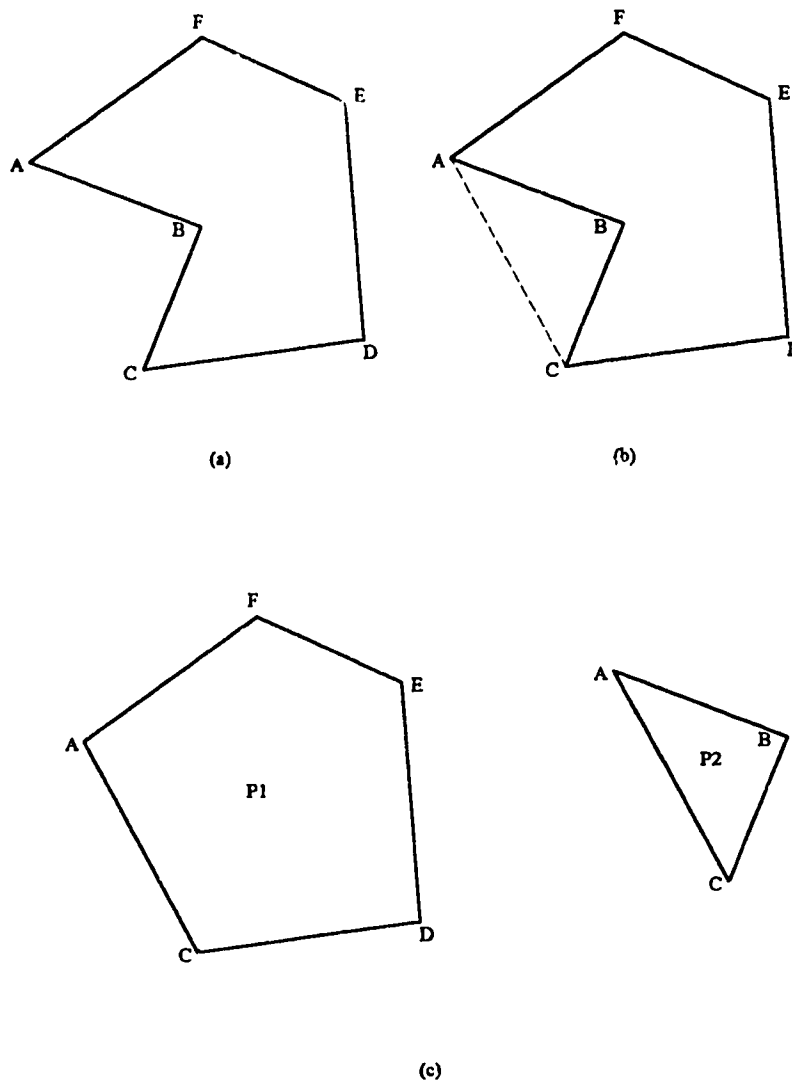


Figure 3.12 convex hull decomposition of a concave polygon. (a) original polygon  $P$   
 (b) possible decomposition. (c)  $P = P_1 - P_2$  while  $P_1$  and  $P_2$  are convex.

Another approach is using a pre-defined distance as a threshold for the length of each edge of triangles [48]. The assumption is that long edges produce undesirable



triangles. These long thin triangles are most likely to occur over gaps (see *Figure 3.13*). This pre-defined threshold is compared to the triangle edge lengths. If the length of an edge is greater than the threshold, the proposed triangle is rejected. This threshold is defined such that long thin triangles are not generated. In the polygon shown in *Figure 3.13*, the proposed triangle  $\Delta AMH$  is examined for each of the edges against the threshold. And so is triangle  $\Delta P_1P_2H$ . The threshold is defined so that these triangles are rejected.

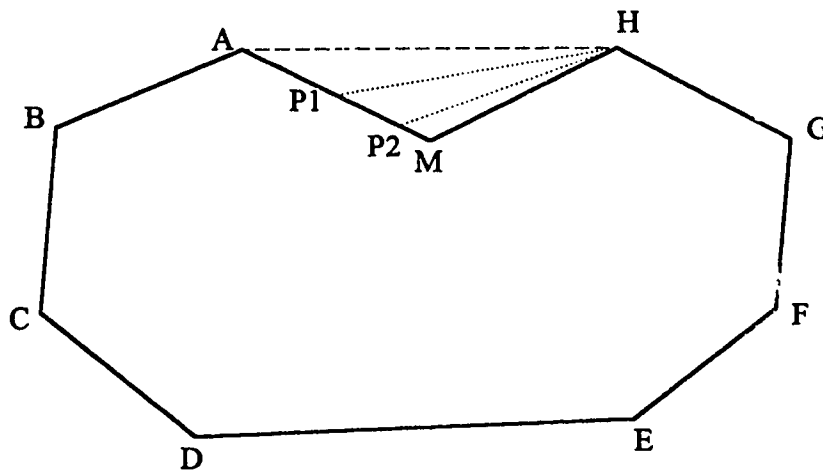
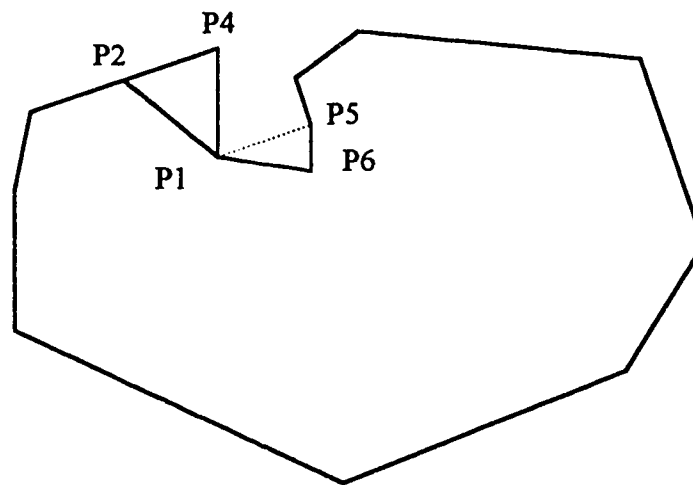


Figure 3.13 A pre-defined threshold is used to reject the long, thin triangles

$\Delta AHP_1$ ,  $\Delta P_1HP_2$  and  $\Delta P_2HM$ .

This approach can be used to reduce the boundary effect when the concavity of the region is not sharp because most possible long thin triangles are across the concave

boundary. However, there are several problems with this approach. The first is that if the region of the data set contains a sharp concave area, triangles generated across the boundary may not be rejected because the edge lengths of these triangles may be smaller than the pre-defined distance (*Figure 3.14*). The second is that the pre-defined threshold that works for one data set may not work for another, which causes the difficulty in defining the threshold for a set of data points. Finally the threshold is defined as global to the data set, which may cause a problem that the threshold works well in some regions of the data while it fails for other regions.



**Figure 3.14** A problem using pre-defined threshold to reject triangles. Case 1: triangle  $\Delta P_1 P_5 P_6$  can not be rejected because the threshold is not large enough to reject internal triangles such as  $\Delta P_1 P_4 P_2$ ; Case 2: across-boundary triangle  $\Delta P_1 P_3 P_6$  is rejected with threshold which will also reject internal triangles such as  $\Delta P_1 P_4 P_2$ .

### 3.5.2 Proposed Solution

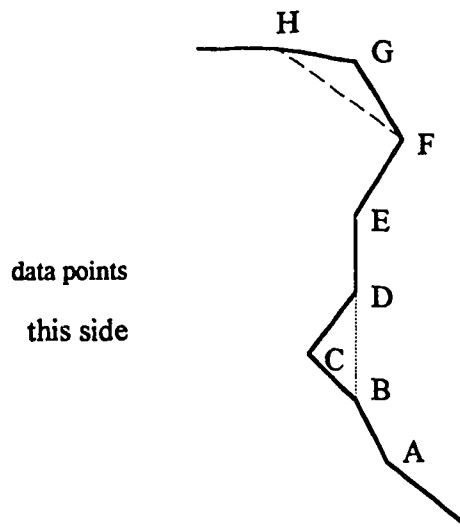
In this thesis, a method of reducing the boundary effect is developed. Unlike Boissonnat's approach, which may produce triangles across the boundary even if the boundary is pre-defined accurately, the proposed approach only generates triangles inside the boundary of the data set, as long as the boundary of the region is completely defined. Thus the generality of the whole surface modelling process is improved.

The proposed process applies the boundary constraint to the triangulation algorithm. By calculating the cross products of boundary position vectors, while keeping the direction of the boundary consistent, triangles outside the boundary of the region can be determined.

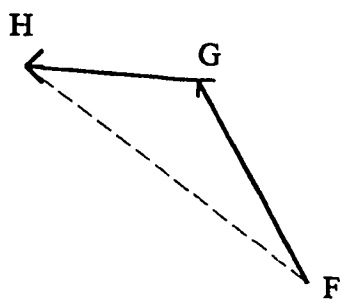
Suppose the boundary of a data set is given such that (1) the region supporting the data set can be defined by a closed polygon, and (2) the direction of the boundary is consistent. For every boundary triangle, the cross product of the two boundary position vectors is calculated and compared with the normal of the bounded region defined by the boundary list. If the direction of the cross product is different from that of the normal of the region, the triangle is rejected. Otherwise the boundary triangle is valid for the given data set with the boundary list.

This process is shown in *Figure 3.15* where (a) shows a possible example of data set taken from a trunk surface, with (b) and (c) indicating some possible boundary triangles with the data set. Assuming the boundary list is defined in a counter clock-wise

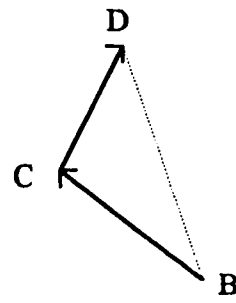
direction, i.e., as the direction as  $ABCDEFGH$ , each boundary triangle is examined according to this boundary list. For triangle  $\triangle BCD$ , the cross product of the boundary position vectors is  $\mathbf{V}_1 = \mathbf{BC} \times \mathbf{CD}$ , which has a direction pointing to the planar area from the viewer. This direction is different from that of the boundary list and thus the triangle  $\triangle BCD$  is rejected. On the other hand, for triangle  $\triangle FGH$ , the cross product of the boundary position vectors is  $\mathbf{V}_2 = \mathbf{FG} \times \mathbf{GH}$ , which has a direction pointing out to the viewer, the same as the direction of the boundary list. Therefore triangle  $\triangle FGH$  is valid for the data set with the boundary list and thus is generated as a triangular patch of the surface.



(a)



(b)



(c)

Figure 3.15 Detection of invalid triangles. (a) an example of data describing a trunk;  
(b) a valid boundary triangle; (c) invalid boundary triangle is rejected.

For surface modelling of trunk deformity, data taken from patients' backs contain no information other than the three coordinates of each point. Thus the boundary of a data set is not available for the triangulation of the data to reduce the boundary effect. Consequently, a boundary must be generated from a given data set, with which the above boundary process can be applied to the triangulation of the data set.

### **3.5.3 Boundary Generation**

#### **3.5.3.1 Boundary Generation Problem**

Unless the region of the data set can be considered as convex, constructing a boundary list from a set of discrete data points is an ill-defined problem. If the region is convex, there are some algorithms available for defining the convex hull of the region [56]. However, if the region is not convex, determining the concavity of the region always involves human intervention [69]. For a given set of discrete data points, different people may decide different boundary lists, depending on the distribution of the points, their interpretation of what the object looks like, and the acceptable accuracy requirement of the boundary for the data set. *Figure 3.16* shows an example of different boundaries for a same data set.

To generate a boundary list from a set of discrete data points by a computer, the challenge is to reduce human intervention to a minimum, while ensuring the ideal boundary. This also strongly depends on the distribution of the given data points.

Consequently, a priori knowledge about the distribution of a data set is useful in generating the boundary list for the data set.

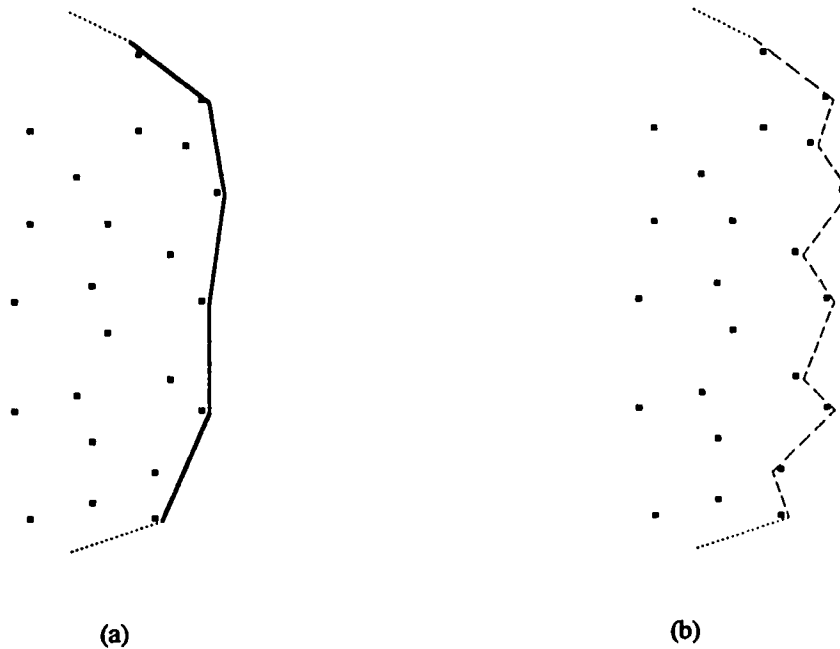


Figure 3.16 Different boundaries for the same data set. (a) and (b) both are considered a boundary for the same data set.

### 3.5.3.2 Preliminary Boundary List

When using the video-based technique as described in Chapter 1, the projected dot pattern is a grid of targets of constant and known spacing (*Figure 1.3*). The number of

targets upon the trunk is dependent primarily on the size of the trunk. The angle of the camera, the finite projection distance and the trunk curvature cause the distortion of the targets on the trunk surface. Therefore targets on the trunk have irregular spacing (*Figure 1.4*). Often the generated data points, after digitization, will remain rectangular in shape on the 2-D projection plane. This information provides valuable priori knowledge for the boundary generation. Suppose the region of the data points is defined in the projection plane  $x - y$  with depth of each point defined by its  $z$  value. The method used in this thesis to generate the boundary for a data set is:

(1) a rectangle is defined such that all points are contained inside this rectangle, i.e., with the rectangle being (maximum  $x$  - minimum  $x$ ) long and (maximum  $y$  - minimum  $y$ ) wide;

(2) Using a selectable resolution, the rectangle is divided into grids. Each point lies in only one grid although there may be more than one point in one grid. The resolution of the grid may be entered by the user according to the number, and the rough resolution of the data points. The resolution is selected so that on average each grid has at least one point inside. If the resolution is too small, however, intersecting boundary lines may be generated (*Figure 3.17*). In the current data acquisition system, the resolution of  $35 \times 40$  for a 2000 point set is reasonable and produces an acceptable boundary for the data. A two-dimensional array  $G(m, n)$  is appropriate to store the points within each grid.



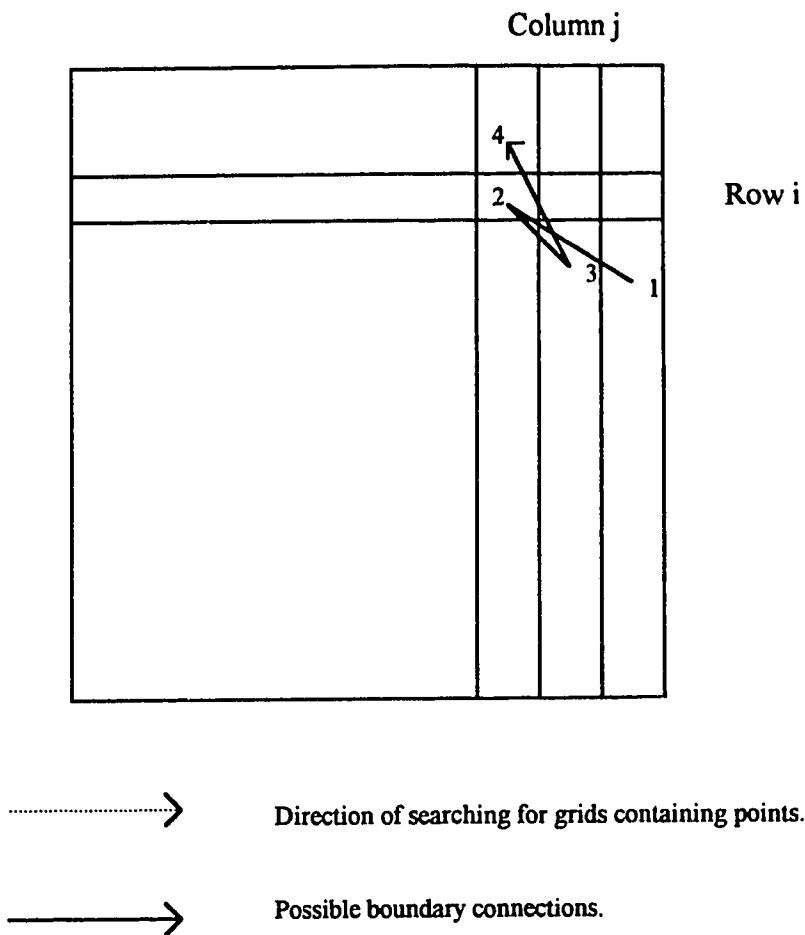


Figure 3.17 Too small resolution produces intersecting boundaries.

(3) For each of the outermost blocks, points having the extreme values are obtained. To get the extreme value, in the  $x$  direction, points are compared and the point(s) having the a maximum  $y$  (for top row) or a minimum  $y$  (for the bottom row) among points in the same block are stored, while in the  $y$  direction, points with a maximum  $x$  (for the right column) or a minimum  $x$  (for the left column) are recorded. This

process is performed in one direction, say counter clock-wise, starting from the bottom-left grid  $G(1, 1)$  and ending with the same grid point.

A special case occurs when there is no point in an outermost grid. The present method handles this situation by traversing towards the center of the region from the outermost block in either the  $x$  or  $y$  direction. Suppose there is no point in a right Grid  $G(i, n)$  where  $i$  is a variable. Then the one next to  $G(i, n)$  in the  $y$  direction towards the center, i.e., Grid  $G(i, n-1)$  is examined. This process continues until one point is found, which might be in Grid  $G(i, n-k)$  where  $k$  is normally a small number. If there is no point in a bottom Grid  $G(1, j)$  where  $j$  is a variable, Grid  $G(2, j)$  is examined. If no point is found in  $G(2, j)$ , Grid  $G(3, j)$  is examined, ... until one point in this direction is found. Similar processes apply to the situations for a left grid or a top grid if no point is found inside the grid.

(4) The points obtained from the above steps which have extreme values are then stored in a list. The neighboring points in the list are connected into boundary line segments in the same direction as the direction of traversing the outermost blocks. The above procedures are shown in *Figure 3.18*.

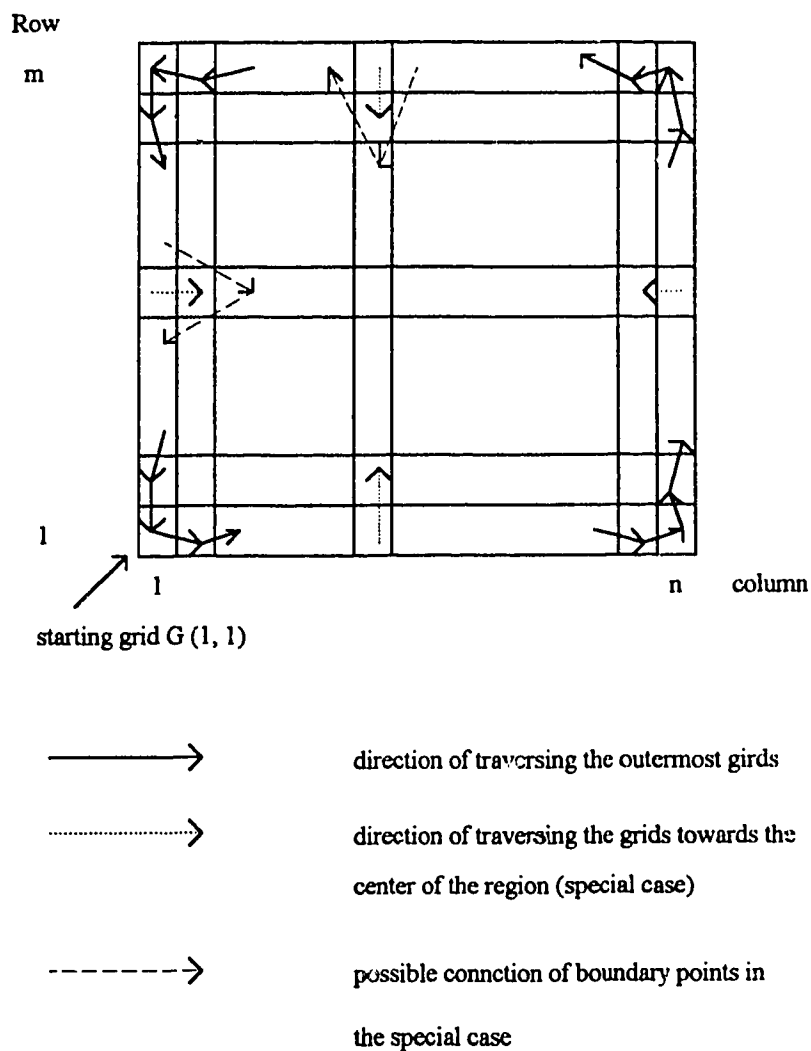


Figure 3.18 Boundary generation by gridding data points. Data are divided into  $m \times n$  grids.

(5) It is possible that some points which are not with extreme values fall outside, or on the boundary line segments. *Figure 3.19* shows one of these examples. To prevent this situation, every line segment is examined against the possible points between the two

endpoints of the line segment. In *Figure 3.19*, there is a line segment,  $CE$ , such that one point,  $D$ , between points  $C$  and  $E$  falls outside the line. Thus the line segment is modified to enclose the point  $D$  in the boundary line segment, i.e., line  $CE$  is modified into  $CD$  and  $DE$ . So is line  $MP$  which is modified into  $MN$  and  $NP$ .

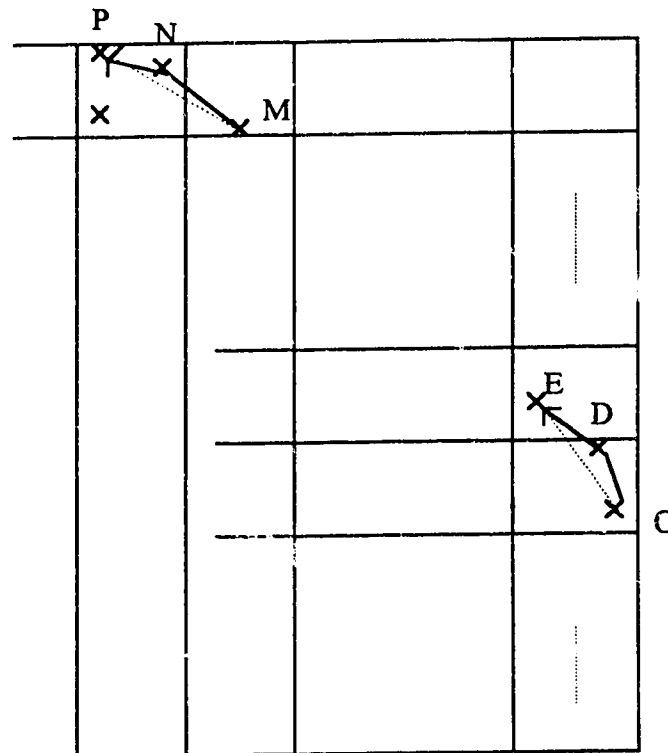


Figure 3.19 Modification of the boundary line segments.

If there is more than one point falling outside, or on a line segment, those points must be sorted according to the direction of the line segment. The point with a maximum value in the direction is then inserted into the line segment such that the line is modified to

form two line segments. Then both these two lines are examined separately in the same way against other points between the endpoints of each line segment, until all these points are inside the line segments. This process is implemented as a recursive procedure in the boundary generation program, which does not finish until all boundary line segments are examined and all points are enclosed inside the modified boundary or on the modified boundary.

In *Figure 3.20-a*, points  $C$ ,  $D$  and  $E$  are outside the boundary line segment  $AB$  in a right column. Therefore they are sorted into  $x$  decreasing order as  $D$ ,  $E$  and  $C$ . Line  $AB$  is split into line segments  $AD$  and  $DB$ . Then line  $AD$  is examined against point  $C$  which is between  $A$  and  $D$ , while line  $DB$  is examined against point  $E$  which is between  $D$  and  $B$ . Both points  $C$  and  $E$  are inside the line segments respectively and so lines  $AD$  and  $DB$  are the modified boundary line segments for line  $AB$ .

*Figure 3.22-b* shows a complicated case. line  $AB$  is first split into  $AC$  and  $CB$ . Then both lines  $AC$  and  $CB$  are examined.  $D$  is found outside line  $CB$ , so line  $CB$  is further split into  $CD$  and  $DB$ . After line  $AB$  is modified into lines  $AC$ ,  $CD$  and  $DB$ , all points are inside these boundary line segments. Therefore the recursive process is terminated.

The boundary line segments are considered a preliminary boundary list, which normally encloses all points inside this boundary (exceptional cases: see section 3.5.4). Applying this boundary list to the region of the data, the boundary of the region often contains jagged edges.

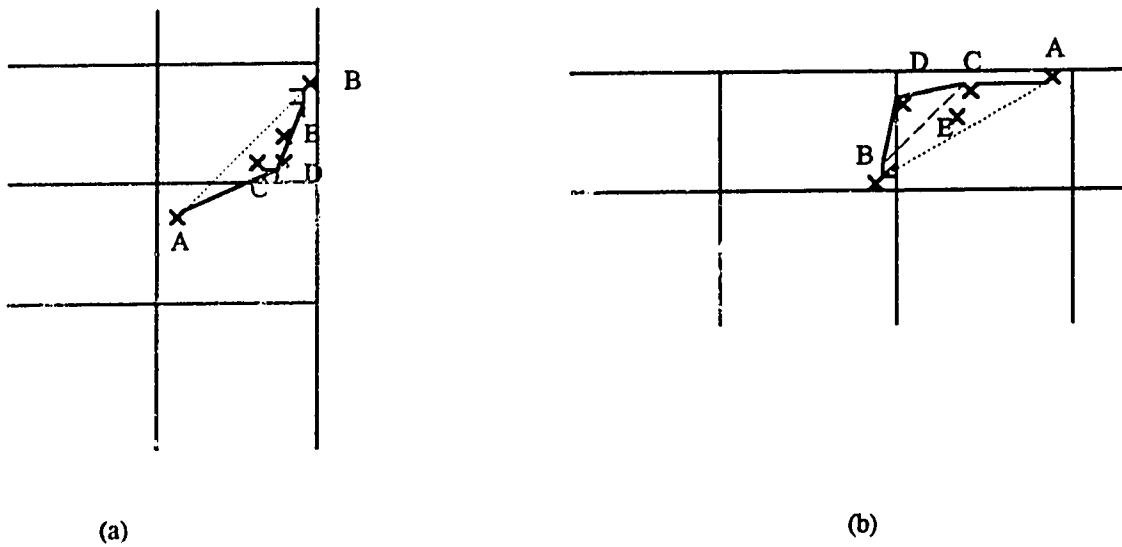


Figure 3.20 More than one point outside the boundary line segments.  
 (a) A simple case. (b) A complicated (recursive) case.

### 3.5.3.3 Improvement of Boundary List Using Angle Constraints

The boundary with jagged edges in 2-D will cause jagged edges in the surface model in 3-D. Even though this does not affect the correctness and accuracy of the surface model, it is more natural that the trunk surface have a smooth edge.

To solve this problem, an angle constraint is applied to the preliminary boundary list. In this work, the angle of  $90^\circ$  is applied as a threshold such that neighboring lines with an intersecting angle less than this threshold are not considered as boundary line segments; they will be straightened into one line. All the neighboring lines are checked through for the intersecting angles. If an angle less than  $90^\circ$  is found, the two neighboring line segments are not boundary lines. They are replaced by one line joining the two endpoints on opposite sides. In *Figure 3.23*, where the intersecting angle between lines  $BC$  and  $CD$  is less than  $90^\circ$ , so  $BC$  and  $CD$  are replaced by one line  $BD$ . However lines  $MN$  and  $NP$  are not changed because the angle  $\angle MNP$  is greater than  $90^\circ$ .

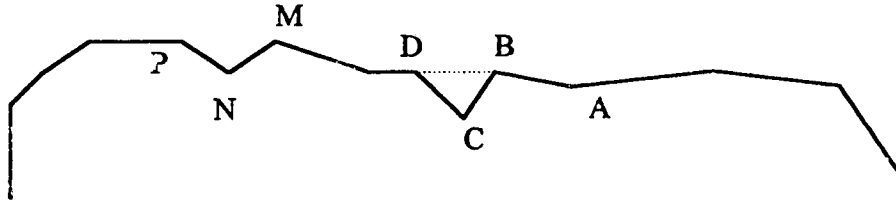


Figure 3.21 Angle constraint for boundary list

Applied to the preliminary boundary list, this process produces a secondary boundary list. The secondary boundary is smoother than the original one because the jagged areas are eliminated.

### 3.5.4 User Interface for Boundary Modification

#### 3.5.4.1 Why the User Interface?

The above boundary generation scheme considers basically only the four sides of the region of the data set. If the data given for surface modelling is virtually rectangular and each of the blocks in the four corners contains exactly one point, the boundary generation scheme can generate an exact boundary list for the given data set. In some cases in which any of the corner blocks contains no points, this scheme has a great potential to generate a boundary with some boundary line segments intersecting. This is because if there are no points in a corner block, say  $G(m, n)$ , when traversing towards the center of the region, the scheme does not know which way to traverse.

*Figure 3.22* shows an example of this case. There are no points in the corner block  $G(m, n)$ . The extreme point found in the  $x$  direction in  $G(m-1, n)$  is marked "1". For  $G(m, n)$ , because there is no point found, the next block in the  $x$  direction is examined, until point "2" is found. The block  $G(m, n)$ , however, is also an outermost block in the  $y$  direction, so it is examined in this direction, in which point "3" is found. Then for  $G(m, n-1)$ , the scheme goes down to  $G(m-1, n-1)$  and finds point marked "4". Therefore the boundary line segments in the corner is then "1-2-3-4". These lines are intersecting, which makes it difficult to determine which point in the data is inside, outside or on the boundary.



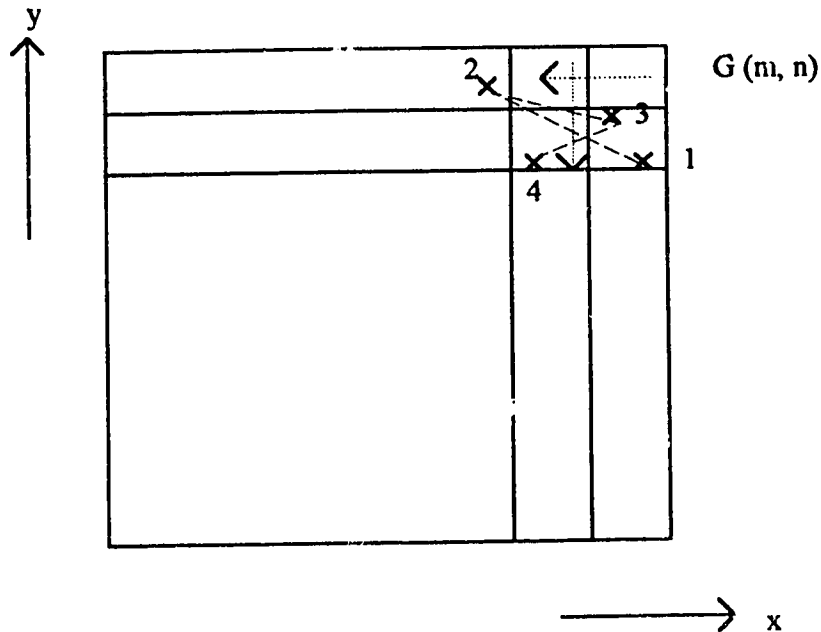


Figure 3.22 An example that causes boundary intersecting.

For some data sets, this difficulty can be overcome by decreasing the resolution of the grids of the data points so that the corners are ensured to contain points. This process, however, may cause some inaccurate boundary line segments which generates artifacts in the surface model. In the example shown in *Figure 3.23*, the resolution for gridding the data set is decreased, so the corner block  $G(m, n)$  contains some points. While gridding of points with this resolution prevents the problem of generating intersecting boundary line segments as above, it generates some boundary lines that are not accurate for the data set. In the figure, the possible boundary line segments are "1-6-7-8" which are obviously not as

accurate as "1-2-3-4-5-6-7-8" when resolution of gridding is small. Using the boundary of "1-6-7-8", possible triangles connected in this boundary area are shown in the figure as  $\Delta 132$ ,  $\Delta 143$ ,  $\Delta 154$ , and  $\Delta 165$ . These triangles may go across the region from one side to another, like  $\Delta 154$ , and  $\Delta 165$ , which become artifacts in resultant surface model when displayed in a 3-D space.

The problems vary case by case: the boundary generation depends on the distribution of data. An easy way to ensure the boundary to be correct is to permit user intervention.

In this thesis, the boundary is generated with the method described above. At the corners, the points that are farthest from the center of the region are considered boundary points and connected into boundary line segments. Then a user interface permits the user to modify this boundary list, until the boundary is satisfactory.

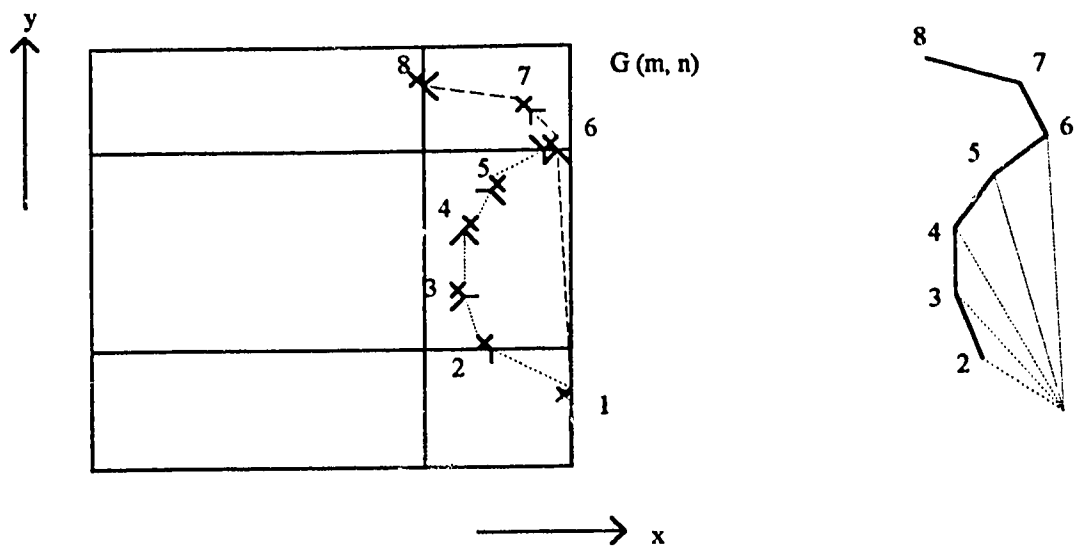


Figure 3.23 Reducing the resolution of gridding may cause triangles across a concave boundary

### 3.5.4.2 User Interface Overview

In the user interface, the data points and the secondary boundary list are displayed in an Object Window. For better software modularity and hierarchy, the modification of the boundary, and modification of the data points in some cases, are handled by applying some events to the object. The events are grouped into a menu in a Menu Window. A Rotation Window is designed for rotating the object in  $x$ ,  $y$ , or  $z$  directions, which enable the user to view the effect of rotation easily. In the Menu Window, there is an event for "zooming" of the object. Rotation and Zooming enable the user to have a better look at the data set and the boundary lines. A Help Window is also designed to make the user interface more user-friendly, which instructs the user how to select the right menu, and how to perform appropriate operations in a menu. The Help Window also prints messages

if the user makes wrong selections. *Figure 3.24* shows the layout of the user interface for boundary modification. The user interface in this thesis is written in *C* using IBM's graphics package *graPHIGS* [75].

The dimension of the Object Window is determined by the data. There are certain transformations which must be applied to the data points when displaying them at the center of the window. These transformations normally include scaling and translation.

The set of data points is considered a graphical structure in *graPHIGS*, and so are the secondary boundary line segments. Each of the points is regarded as an element of the point structure, while each of the boundary line segments is regarded as an element of the line structure. In the program, a List is used to store the boundary line segments. The modification of the boundary caused by the user's selection in the event menu will cause the modification to this list. When Exit Event is executed, the boundary list containing the final boundary line segments will be used in the triangulation program.

Normally the original data set is not supposed to be modified. But in some cases, due to the digitization of the data, a small number of points of the data set may be far away from the major part of the set. In this case, users may ignore these points because their main concern is the surface, while these distant points may cause more distortion to the surface than regular points. Therefore the user interface permits modification of points such that the user can delete points from the data set, or re-insert the points that have been deleted.

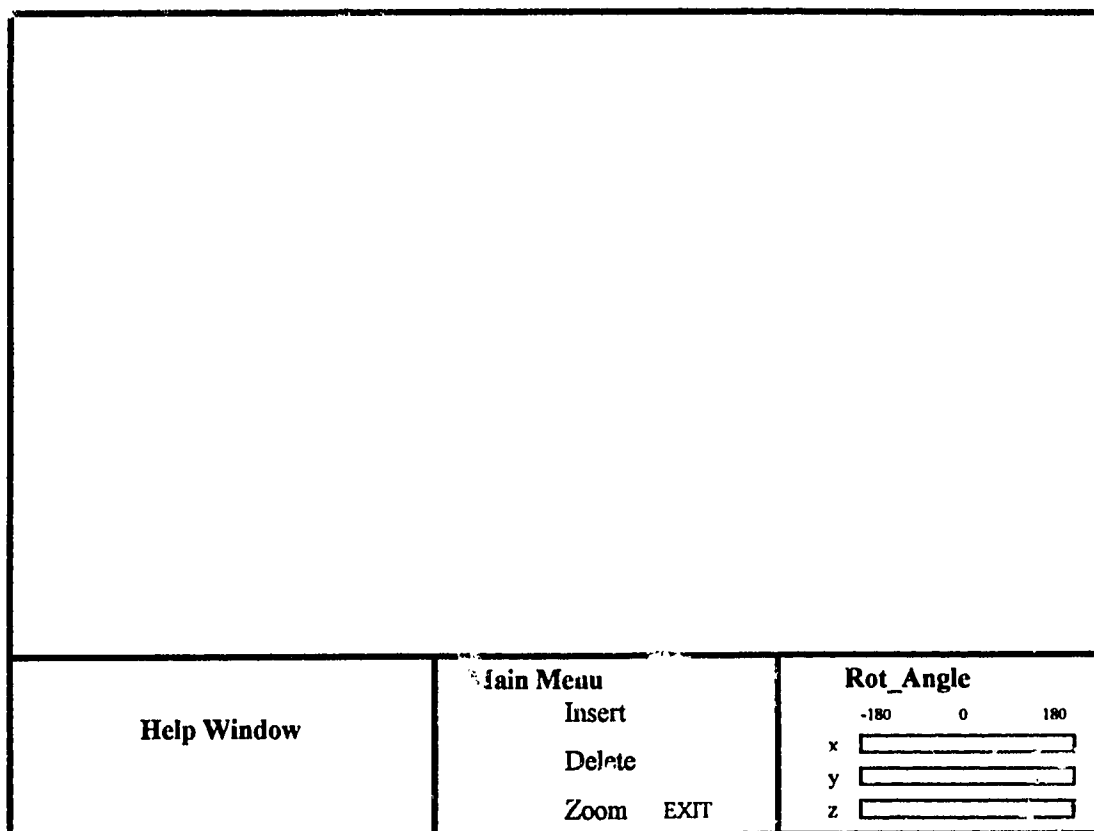


Figure 3.24 Window Layout

### 3.5.4.3 Event Handling

The basic events are Delete-Point, Delete-Line, Insert-Point, Insert-Line, Zooming, Rotation, Exit, and Return. The event hierarchy and the menu of the interface are designed in the way shown in *Figures 3.25, 3.26, and 3.27.*

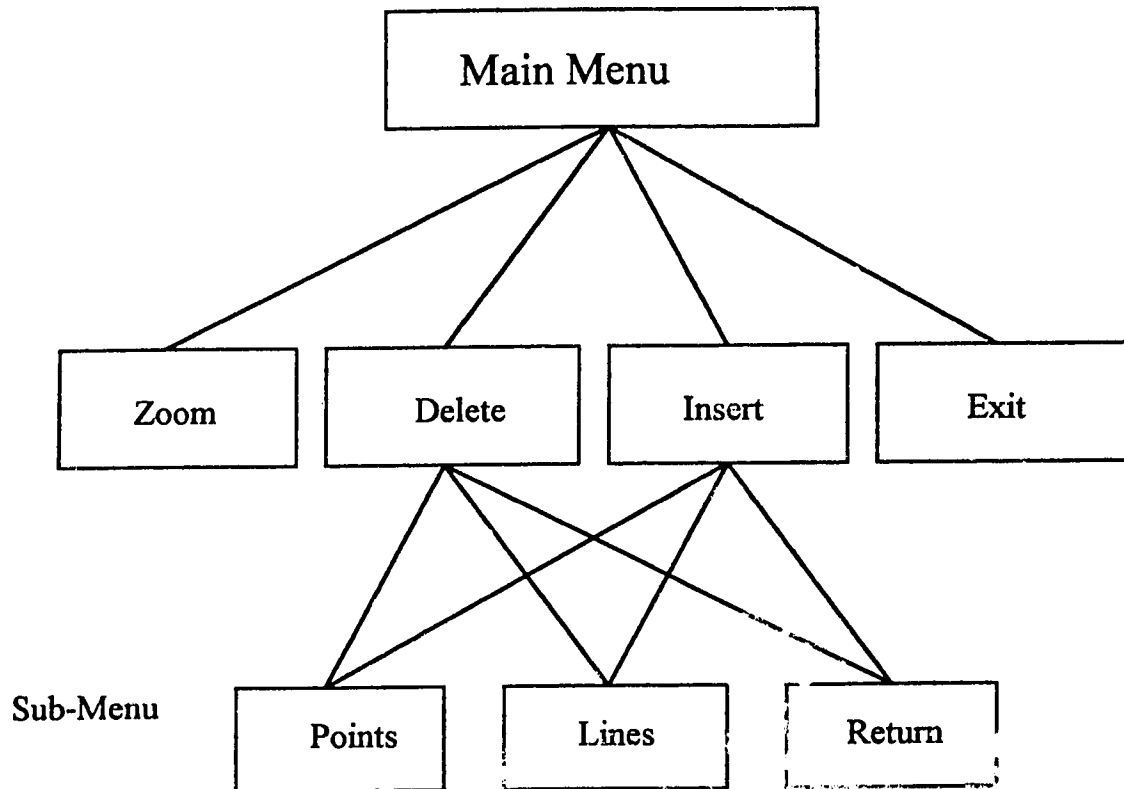


Figure 3.25 Event hierarchy

From each sub-menu, control can be returned to the upper-level menu. "Rotation Sub-menu" is a separate menu because once Rotation is selected, all graphical structures in the Object Window, including points and boundary lines, are rotated at the same angle selected by the user. The angle selection is implemented as a scroll bar driver and will be described in the following sub-section.

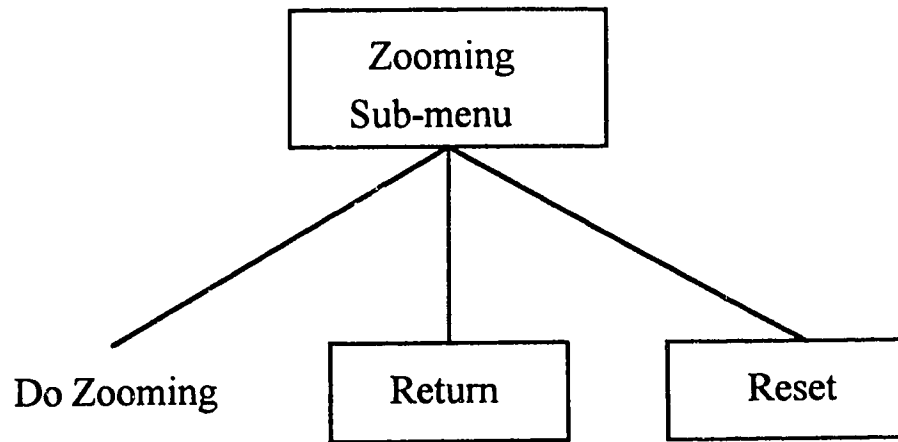


Figure 3.26 Zooming sub-menu

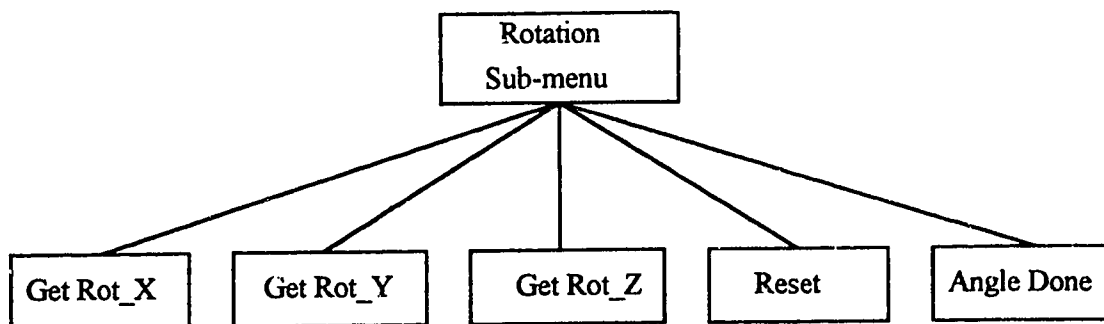


Figure 3.27 Rotation sub-menu

## 1. Point/Line Modification

For a better illustration of the modification, the deleted boundary line segments are erased in the Object Window, while lines inserted are displayed in a different color. A similar method is used to handle points. Points are first displayed in one color. If points are deleted by a Delete-Point Event, the color for these points is changed to another. Once the deleted points are re-inserted into the point set, the color of the points re-inserted is changed back to the original color.

### (a) Modification of Boundary Lines

The modification to a boundary is achieved by first deleting the lines segments between the two endpoints that have been selected by the user, then inserting the desired lines segments in between.

The two selected endpoints of boundary line segments are checked through the boundary list. If the two endpoints are found, the line segments in the boundary list between these two points are deleted, in which case the direction of the boundary must be considered (e.g. clock-wise). Otherwise, a corresponding message is displayed in the Help Window indicating that the points selected are not in the boundary list.

Once the line segments are deleted, the user can select the Insert-Line Event to insert a line, or lines, into the boundary list. The insertion is also executed in the same direction as that of the boundary. If more than one line segment is to be inserted into the boundary list, the user can select the points to be in the boundary list consecutively. The



boundary list modification is implemented in such a way that the second endpoint of a preceding line segment is the first endpoint of the next line segment, and the line segments inserted into the boundary list start with the first endpoint selected when deleting the lines, and end with the second endpoint of deletion.

#### (b) Modification of Points

There is a list used to store the indices of the points that have been deleted. The user can delete points with Delete\_Point Event by just clicking the mouse at the point desired to be deleted. The point which is nearest to the cursor is deleted, with the index added to the deleted point list. The deleted points can be re-inserted into the point set by using Insert\_Point Event. Once one of these points is selected, it is checked through the deleted point list. If the point to be re-inserted is not in this list, a corresponding message displays in the Help Window. Otherwise the point is re-inserted into the point set with its original color. The point index in the delete point list is removed.

## 2. Zooming

A zooming function is implemented so that the user can examine smaller areas of interest more clearly. The user must first select Zooming Event in the Main Menu. In the Zooming Sub-Menu, the user can go back to the Main by selecting the Return Event, reset the transformation of the object by selecting Reset Event, or zoom the object in the Object Window.

To zoom the object in the Object Window, the cursor must be positioned inside the Object Window. The first click on the mouse defines the position of the cursor as the center of zooming. The following clicks of the mouse will each time double the image appearance of the object in the Object Window with the center of zooming being the center of the Object Window. The transformations involved in the program are basically translations and scaling.

The modification of points and lines may be difficult if the image of the object is too small, or the number of points is big so that the connection of lines is difficult to see. For this reason, the event handling of the program is implemented in a way that the Zooming function can be handled at the same time as Deletion or Insertion. For example, it is possible to select Deletion/Insertion Events while the control is in Zooming; on the other hand, it is also possible to select Zooming while the control of the program is in Deletion/Insertion.

### 3. Rotation

The object in the Object Window can be rotated to any orientation. The user can select the desired angles in any of the  $x$ ,  $y$ , or  $z$  direction by moving the three scroll bars for each direction. Once the angle is selected, the object in the Object Window is rotated to the corresponding orientation. The rotation is performed about the original center of the object in Object Window.

The Reset Event in the Rotation Window resets the orientation of the object to its original state. The Done Event, while optional, is for examining the object at a combination angle. For example, if the user wants to see the object at  $45^\circ$  ( $x$ ),  $90^\circ$  ( $y$ ) and  $135^\circ$  ( $z$ ), he can just move the  $x$  scroll bar to  $45^\circ$ ,  $y$  to  $90^\circ$ , and  $z$  to  $135^\circ$ . Then selecting the Done Event will take the object to the desired orientation.

#### 4. Exit

This Event is executed only when the control of the program is in Main Menu. Once this Event is selected, the boundary list at this point will be regarded as the final boundary list, and the points in the deleted point list will not be considered in the triangulation program; the user interface is terminated immediately. The final boundary list is written into a Boundary file while the deleted point list is sent to a file Del\_Point file. These two files must be read at the beginning of the triangulation program to validate points and boundary lines for generating triangles.

#### **3.5.5 Result of Triangulation with Boundary Constraint**

The boundary processing described above generates an accurate boundary list for a given set of data points. By checking the cross products of boundary position vectors, this boundary list can be applied to the triangulation process to ensure triangles to be valid. The checking process has been described in Section 3.5.2.

Basic changes to the original triangulation program (Appendix A.1) in which the boundary is not taken into account, include checking point status for the boundary point (status "Bound"), and checking of proposed triangles for validity if the point is a boundary point. The modified pseudo code for boundary consideration is listed in Appendix A.2.

A surface display program is necessary to display the triangle connections generated from the triangulation program. This display program dynamically sets up the dimensions of the window for display when reading in the connection list and original data points. A position light source, a background light source and a diffusion light source are used in the display program for the necessary shading effect to illustrate the depth information. A user interface similar to the one in boundary modification program is used in the display program for users to perform necessary operations when analyzing the surface model.

The whole process for modelling trunk surfaces includes a boundary generation program (Boundary Generation), a boundary modification program (Boundary Modification), a triangulation program (Surface Generation), and a surface display program (Surface Display). These four programs are linked into a surface modelling package using the shell script program. The flowchart of the shell script program is in Appendix A.3. The shell script is designed in a user-friendly way in which the user is asked only to enter the name of the file containing data to be modelled. For the ease of management of original data and corresponding triangle connections, users can force the package to display a surface by skipping boundary processing programs and the triangulation program if the triangle connections for this data set have already been generated. The user can also ask the package to generate a surface for the same data set

which has already had the corresponding surface generated, with the new surface being another version for comparison. The final result of the package is a surface model representing the trunk described by the data set. This surface model is displayed on a computer screen which can be analyzed by the user (*Figure 3.28*).

Comparing the trunk surface in *Figure 3.28* with the surface generated from the original program (*Figure 3.10*), it is obvious that the boundary effect is reduced in the trunk surface generated from the modified program. With few visual artifacts, the modified surface model more accurately represents the 3-D trunk surface of the patient.

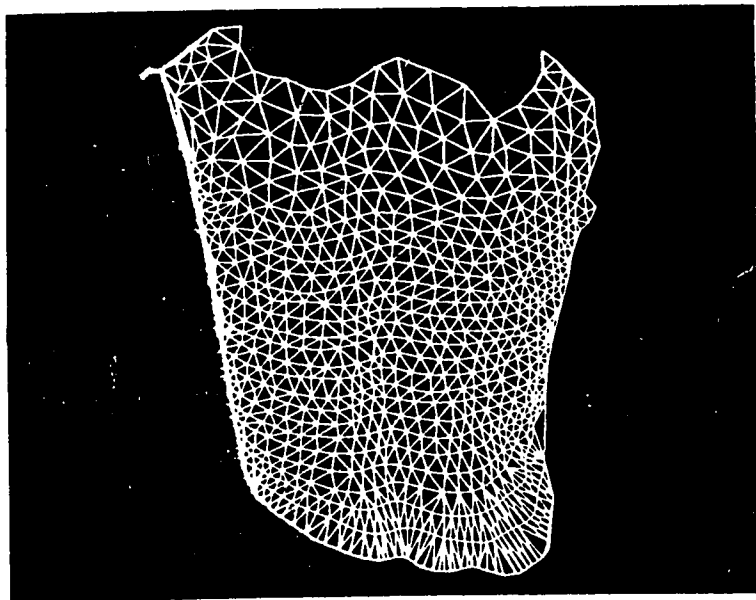


Figure 3.28-a The result of triangulation generated from the modified algorithm

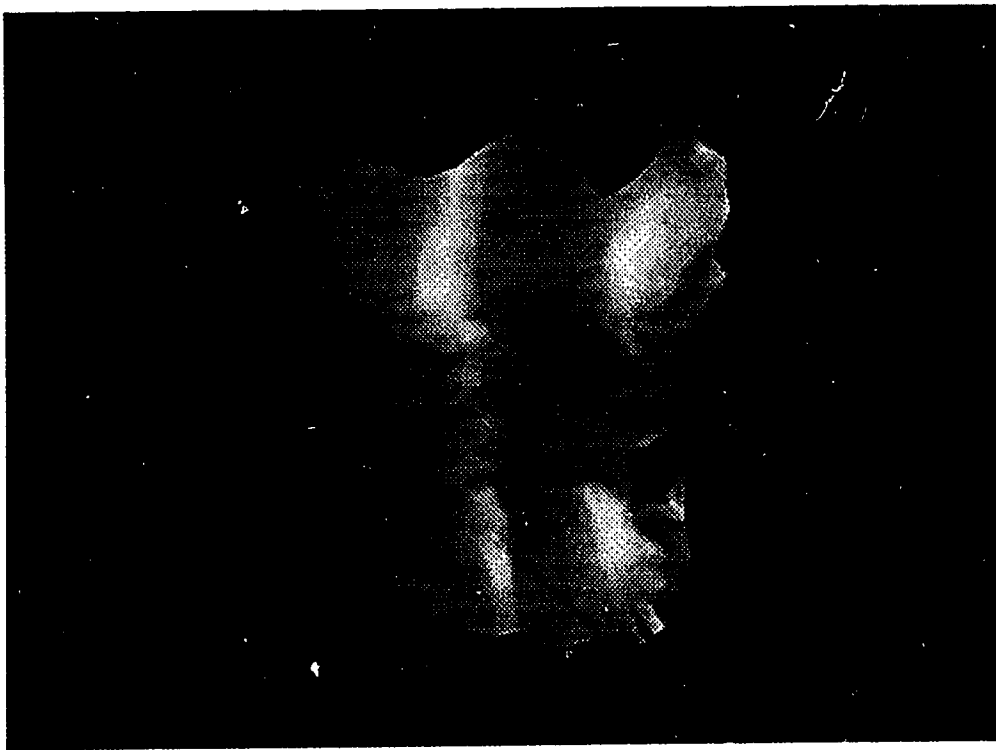


Figure 3.28-b A trunk surface generated from the modified program.

#### 4. ASSESSMENT OF THE SURFACE MODELLING METHOD

McLain's triangulation algorithm has been modified to handle more complex structures using a previously described boundary processing method. Instead of directly modelling a surface in 3-D space, the triangulation program constructs a surface model by triangulating data points in 2-D. The performance of the proposed surface modelling method can be assessed in three aspects: quality, speed and generality of the modelling method.

##### 4.1. Quality of Modelling

It has been mentioned that the triangulation method of McLain's algorithm is optimal in terms of the quality of triangles [47, 50-51, 61, 64, 76]. The algorithm uses the Delaunay Circle to search for a point to form a triangle, which ensures that the vertices of each triangle are nearest neighbors. This property of the triangulation method results in the smoothest surface when the generated triangular patches are displayed. Using an average of the weighted functions of the three vertices in a triangle can achieve the smoothest interpolation between triangles [72]. Other algorithms, such as Lewis' algorithm, do not satisfy Delaunay triangulation requirement [71]. Therefore they can not produce the best approximation of a surface when interpolated in the 3-D space.

Optimality of the proposed triangulation method can be assessed by examining the shape of resultant triangles. For the best interpolation, the best triangle must be an equilateral triangle [55, 72]. Given a set of randomly distributed data points, it is impossible

to generate a triangle connection in which all triangles are equilateral. However, one way to assess the performance of the triangulation process is to examine how close the resultant triangles are to equilateral triangles. If triangles produced by one scheme are more equilateral on average than triangles resulting from another, then the first algorithm is better than the second. An alternative yet more efficient method to examine the quality of a triangle is to calculate the ratio of the smallest angle to the largest angle in the triangle. The ratio for an equilateral triangle is 1 because an equilateral triangle is also equiangular. If the ratio for triangle *A* is higher than the ratio of triangle *B*, then *A* is better than *B* because *A* is closer to be an equilateral triangle. Therefore an average ratio can be calculated for all triangles generated by an algorithm to assess its performance.

The proposed surface modelling method has been tested with some data taken from patients at the Glenrose Rehabilitation Hospital. For each triangle list, an average ratio of the minimum angle to the maximum angle in all triangles was obtained. *Figure 4.1* shows the average ratios for 18 sets of data points which were randomly selected from the data base available in the Glenrose Rehabilitation Hospital. For a set of rectangular spaced data points, the ratio for each resultant triangle must be 0.5, and so is the average ratio. From *Figure 4.1*, all of the average ratios are close to 0.5, which reflects the regular spaced pattern used in the video-based technique to record trunk surfaces. The discrepancies come from: (1) the distortion of the projection, which results in irregular spaced data points from the rectangularly spaced dot-pattern for surface modelling; and (2) the boundary constraint used in the triangulation process, which produces some relatively long thin triangles along boundary.



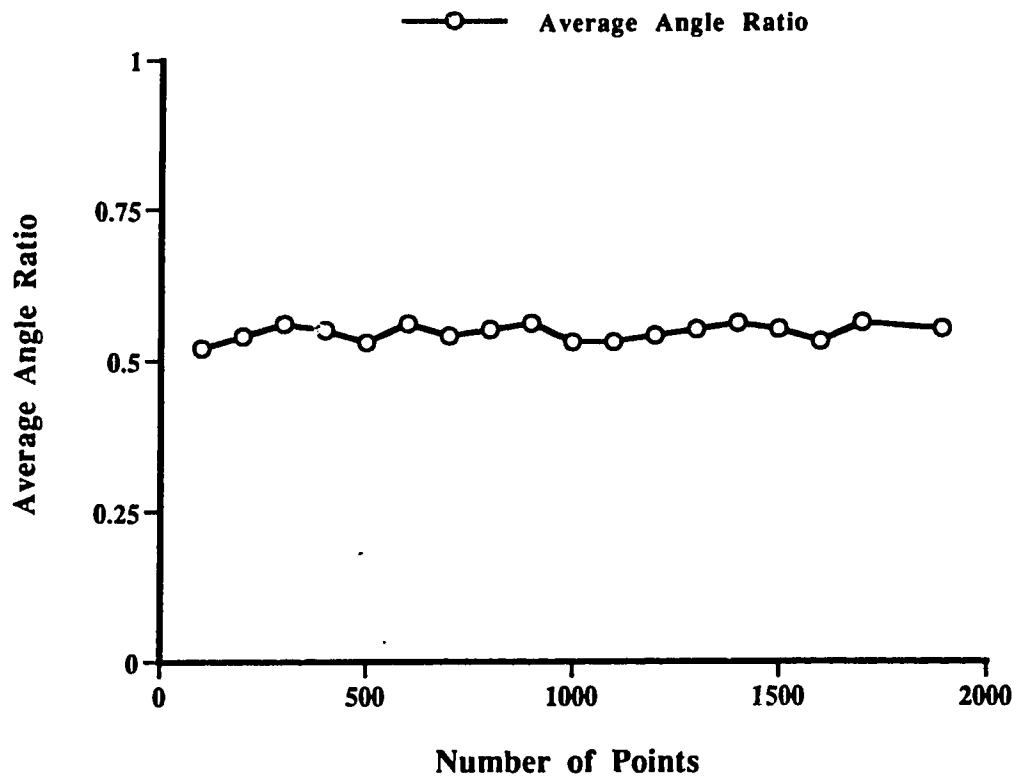


Figure 4.1 Average ratios for different data sets

## 4.2 Speed of Triangulation

The proposed surface modelling method starts from the center of the region supporting the data. With each triangulation iteration, a triangle is constructed and two new lines are connected. The basic operation of the modelling process is the calculation of circumcenters and the distances from the circumcenters to the line under evaluation. Suppose data points are randomly distributed inside a rectangular area as shown in *Figure 4.2*. Point *A* is the point nearest to the center of the region and is selected to start the triangulation process. Point *B* is the nearest neighbor of Point *A*. Points *A* and *B* form the first line to evaluate. The algorithm selects Point *C* to form the first triangle  $\triangle ABC$ . Following triangulation process continues from the three lines *AB*, *BC* and *CA*. If the top line in the line list is *BC*, the number of points to be evaluated is reduced to roughly a half because all points on the same side of *BC* as Point *A* are not evaluated. This iteration forms triangle  $\triangle BDC$  and generates two new lines *BD* and *DC*. If, for each of the new lines, the number of points to be examined is again reduced to roughly a half of that for line *BC*, then at each triangulation iteration, the number of points to be examined for each line in the list is reduced to a half, while the number of lines in the list increases at most by one. This results in a triangulation process running in the order of  $n \log n$ . However, to accurately define the complexity of the algorithm is difficult because the performance of the algorithm depends on the distribution of the data points.

McLain's algorithm runs at  $n \log n$  which is as fast as Lewis'. However, Lewis' was not selected for modelling trunk surfaces because it is not optimal for triangulation, which has been discussed in Chapter 2 and Section 4.1. *Figure 4.3* shows time requirements of the surface modelling program for 18 data sets.

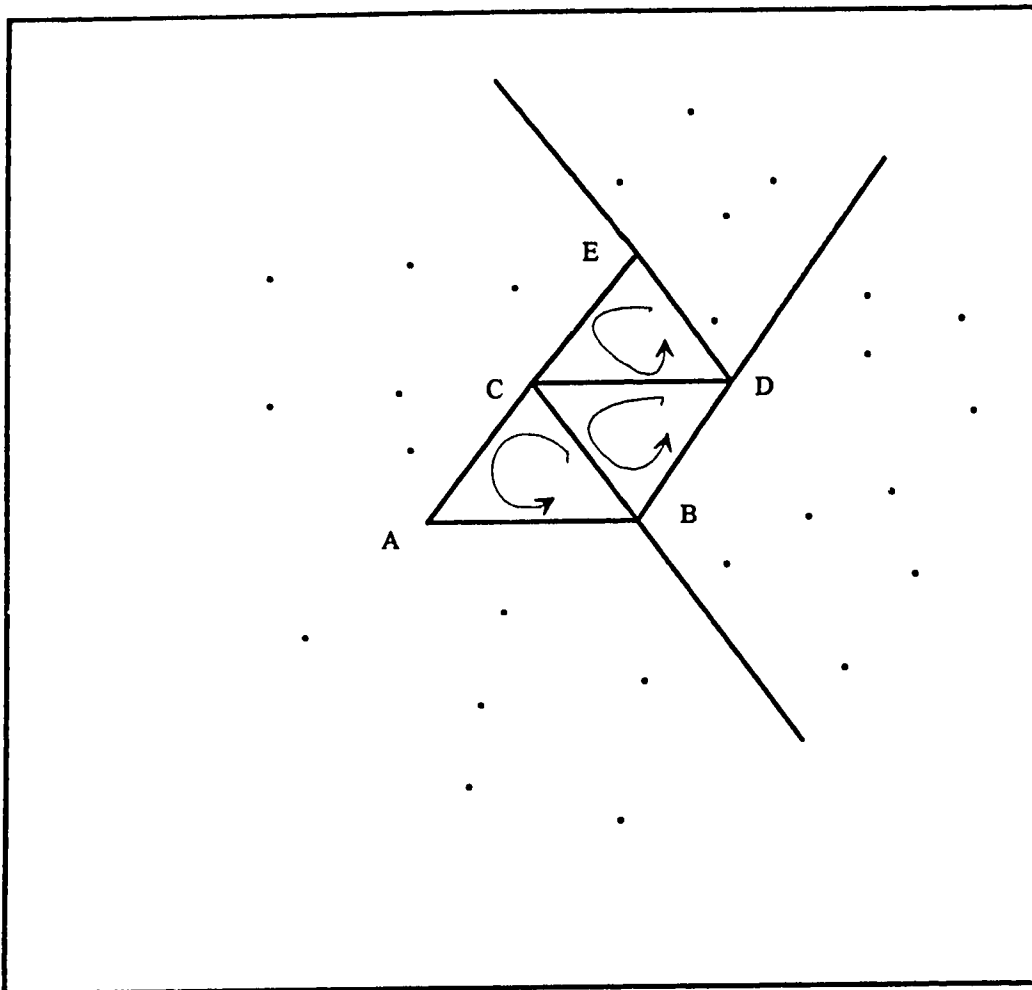


Figure 4.2 Number of points to be evaluated is reduced by a half at each triangulation iteration due to the consideration of signed distances.

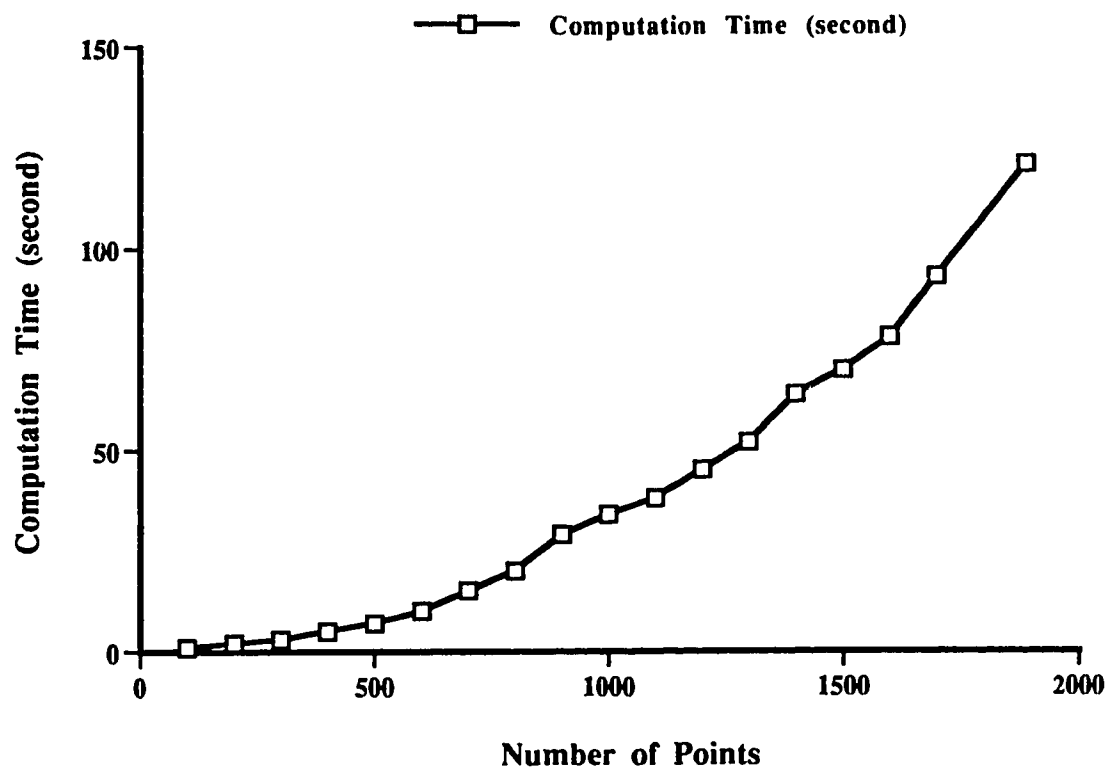


Figure 4.3 Computation time for different data sets.

### **4.3. Generality of the Modelling Method**

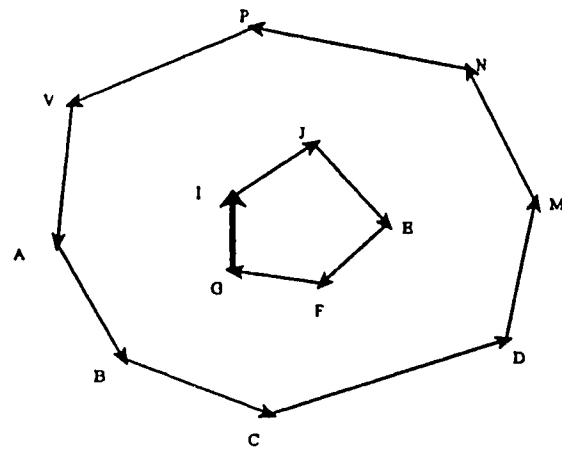
The algorithm is modified such that it can handle some complex structures while the boundary effect is reduced. Without this modification, the triangulation method always generates some triangles across the concave boundary which become the artifacts of the surface when displayed in the 3-D space. The reduction of the boundary effect relies on an exact boundary generated from the given data points. The modification for the boundary effect improves the generality of the surface modelling process.

As discussed in Section 3.5, many trunk surfaces have a concave appearance. Using the original algorithm will generate models with visual artifacts which limit the models to be used in clinical applications. The improvement of generality of the algorithm makes the algorithm produce more accurate surface models for assessment of 3-D trunk deformity.

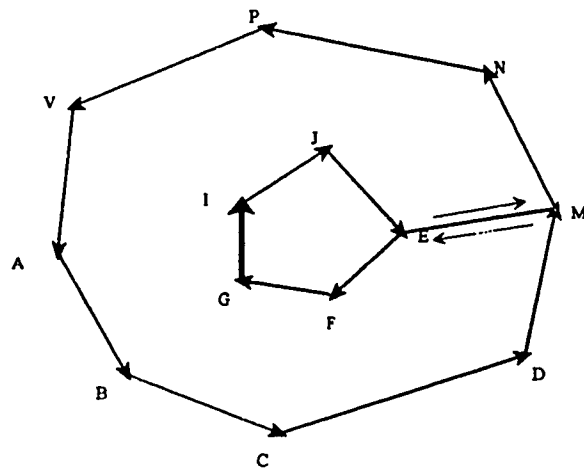
This improvement is based on an assumption that the density of data points is larger than the minimum density of boundary points. This assumption makes the algorithm able to detect triangles beyond a jagged boundary and then reject them. However, for more complex structures such as multiply connected data which may be used in finite element methods [56, 67], the proposed method may not work well because the assumption is not satisfied. In this case, Lewis' algorithm, which recursively splits the data based on boundaries until all sub-regions become triangles, is the best because points grouping is the result of boundaries reconstruction. In the example shown in *Figure 4.4*, the region supporting the data contains a hole. Given the boundary of the region as *ABCDMNPVA-GIJEFG (Figure 4.4-a)*, the region is not a simply closed polygon which can not be

handled by McLain's algorithm. However, Lewis' algorithm can handle this structure by only connecting the boundary as *ABCDMEFGIJIEMNPVA* as shown in *Figure 4.4-b*. With this boundary list, the data can be split into sub-regions until triangulation finishes.

For surface modelling to be used in the clinical assessment of trunk deformity, data obtained using video-based techniques are normally not multiply connected even though they are concave with jagged edges. McLain's algorithm, with the modification to reduce the boundary effect, is adequate to handle all data encountered in the Glenrose Scoliosis Program.



(a)



(b)

Figure 4.4 Lewis' algorithm handles more complex structures. (a) A multiply connected region to be triangulated. (b) Boundaries are combined into one which can be processed by Lewis' algorithm.

## **5. CONCLUSION AND RECOMMENDATIONS**

### **FOR FUTURE WORK**

#### **5.1 Conclusion**

The purpose of this research was to construct a surface model from discrete data points using an appropriate modelling approach. Trunk surfaces, which are described by the data points measured from people with scoliosis, can be represented by the surface models. The boundary generation program generates a boundary list for any given data set. This boundary list can be modified using a user interface to improve the boundary accuracy. McLain's triangulation algorithm has been modified to handle more complex structures and to produce a surface model with little boundary effect. Instead of directly modelling in 3-D space, the surface is constructed by triangulating data points on its 2-D projection plane, which reduces the computation time significantly. When displayed on a computer screen by a surface display program, the 3-D model can be used to represent the actual trunk surface. The comparison of images generated using the original method and the modified algorithm shows that the modified algorithm generates more accurate trunk models than the original one.

A user interface makes it convenient for the user to modify the boundary for a data set, as well as to portray the trunk surface for analysis of trunk deformity. The event handling of the user interface is designed with a hierarchical structure, which makes the implementation efficient and flexible.



Four programs, Boundary Generation, Boundary Modification, Surface Generation, and Surface Display, are linked into a surface modelling package using a shell script program. The package is designed to allow the user to enter only the name of the file containing the data to be modelled. While necessary during the generation of a boundary from discrete data points, human intervention is reduced to a minimum by using default values for the gridding process.

A 3-D surface model improves the understanding of the 3-D nature of trunk deformity. With these models, the clinician can investigate the effects of treatment by conveniently comparing 3-D surface models for data measured from a patient at different times. Combined with the internal spinal alignment information, the trunk surface model provides a tool for clinicians to study the possible relationships between spinal deformity and trunk deformity. Being reproducible and easy to operate, these surface models are useful in documenting patients' records and for educational purposes.

## **5.2 Limitations**

The surface modelling program generates a 3-D surface by triangulating a given data set on its 2-D projection plane. In cases of manifold data points, directly applying the program will, depending on the distribution of the data, either generate a jagged surface at the manifold area(s), or be unable to terminate. In the first case, the incorrect surface cannot be used to represent the 3-D surface of a trunk. The second case may cause a major software problem by generating many intersecting triangles and many more new lines, until the computer or computer system runs out of memory.

The above situation can be explained with *Figure 5.1*. Suppose data points are manifold and no points are lying at exactly the same location on the projection plane. The surface construction method constructs the surface by grouping the nearest neighbors on the 2-D projection plane while ignoring the coordinates in the direction of the projection. Instead of generating a folded surface, the method will generate a jagged surface which is illustrated in *Figure 5.1*. If there are some points lying at exactly the same position on the projection plane, data points become indistinct. This situation will cause the program to generate lines and triangles forever, which has been discussed in Chapter 3. Thus data points must be distinct before the surface generation program can begin. Although this condition is not always met in all children with trunk deformity--approximately 10-15% of children seen through the Glenrose Scoliosis Program have a waist crease or regions of sharp changes which cause manifolds, techniques used to obtain the 3-D data set are not able to measure the folds in the trunk.

Portability is another problem of the surface modelling package due to the use of IBM's graPHIGS programming language in user interface design and in surface display. Testing of the modelling package has been done on IBM's RISC-6000 computers running in an UNIX environment with the AIX operating system. If the modelling package is to be used on other machines, the user interface and the surface display program may have to be re-written according to the operating system and graphical packages running on the particular machine. The Surface Generation program, which generates the 3-D surface (the triangle connection list in the output file), is portable and will run on most UNIX machines

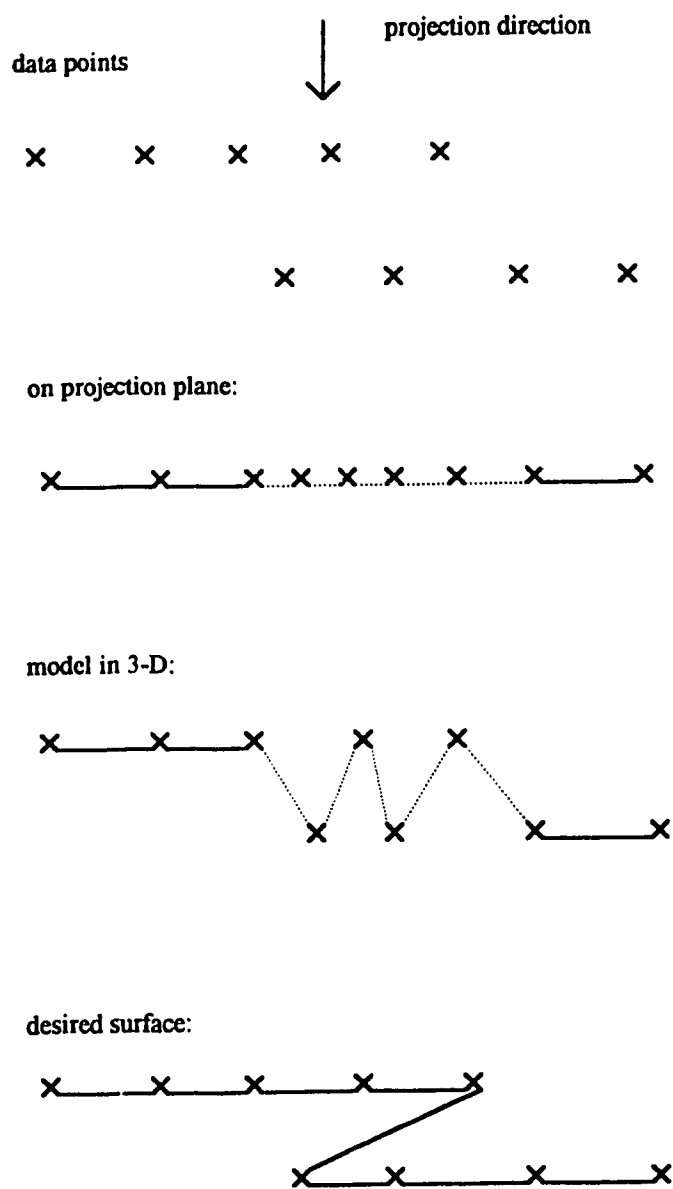


Figure 5.1 Jagged areas generated when modelling manifolded data.

The surface generation program reduces the boundary effect by rejecting the triangles beyond the boundary. The process of rejecting triangles, which involves calculating cross products of position vectors, contains a potential bug in determining the validity of triangles. In certain cases, some lines in which none of the endpoints is a boundary point can be connected into triangles with a non-boundary point on the other side of the boundary (*Figure 5.2*). In triangle  $\triangle MNP$ , none of the three vertices is a boundary point, so the triangle is not rejected by the program. This triangle is considered as valid for line  $MN$ . Thus two new lines  $MP$  and  $PN$  may be appended into the line list. Therefore, the triangulation program may generate more and more triangles and lines and cannot terminate. In this case the boundary processing of surface generation fails.

Whether the problem of boundary process failing occurs depends on the distribution of data points and how closely boundary points are selected and constructed as boundary line segments. If the density of data points is high while the boundary points are not close enough, this problem is most likely to occur. If the data points have a low density while the boundary is accurately defined with neighboring points being close, the proposed triangulation program can terminate properly. In the above example, the problem can be solved by modifying the boundary to that shown by the bold lines instead of using the original line segments (*Figure 5.2*). The new boundary ensures potential across-boundary-triangles to have boundary points as the vertices so that the boundary process works properly for these boundary line segments.

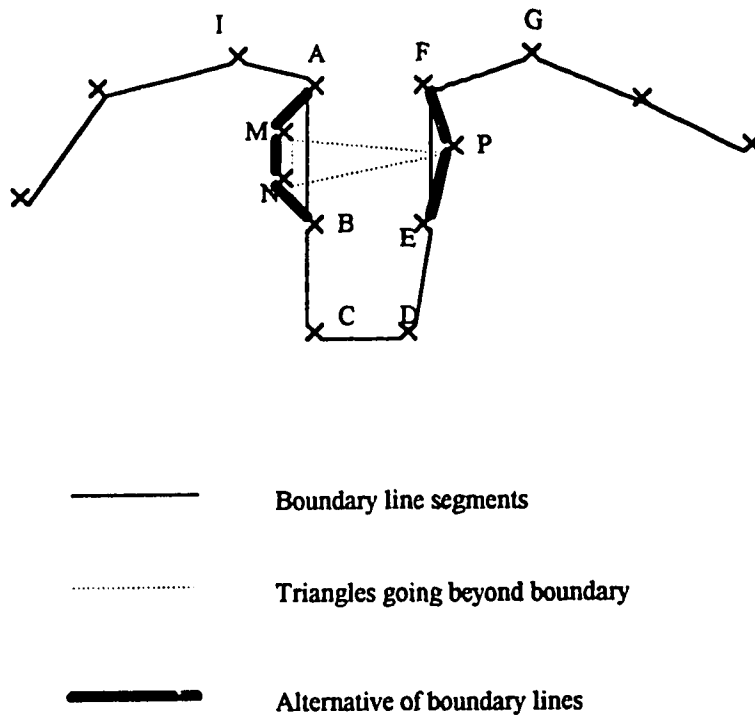


Figure 5.2 Problem with rejecting triangles beyond boundary

The triangulation program generates a triangle connection list which is displayed as triangular patches representing 3-D surfaces. The method is appropriate only for modelling 3-D surfaces from discrete data points. In clinical applications, however, Computer Tomography (CT) and Magnetic Resonance Image (MRI) are the most popular techniques used to record the shapes of anatomic objects. These two methods normally generate contours of objects directly. Therefore directly applying the proposed method to the data obtained from CT and MRI introduces unnecessary computation because the method never uses the valuable information of the contour lines.

### **5.3 Recommendations for Future Work**

Short-term development can emphasize improvement of the efficiency of the surface generation program, which is the most computationally intensive of the four programs used in the surface modelling package. There are two possible ways to improve the efficiency of the approach.

1. Many articles in the literature [77-78] have suggested that pre-sorting data may improve the performance of algorithms. In the surface modelling package, data is sorted into a two-dimensional array by gridding the points. However, this information is used for boundary generation only. In the triangulation process, if points are sorted properly, it may be possible that not all valid points, but only points "close" to the grids containing the endpoints of the line under evaluation, need to be examined for the line for the next triangle. Further investigation concerning how to utilize the sorting information is useful for improvement of the efficiency of the triangulation program.

2. In the current triangulation program, each valid point is calculated for the possible circumcenter of the point and the two endpoints of a line under evaluation. Then each of the corresponding distances from each circumcenter to the line is calculated. These distances are sorted to get the minimum. The calculations are computationally intensive. However, they may be repeated for some points with some lines. If the results of the calculation can be stored in appropriate data structures, they can be re-used for other

points or lines in the triangulation process, which would likely reduce required computation. However, further study of how to design appropriate data structures to process related calculations is required, along with how to make use of the already existing results for further triangulation iterations. It should be noticed that the possible improvement in efficiency is a result of the trade-off between computation time and memory requirements because storing the results of calculations needs a considerable amount of memory due to the huge number of combinations between points and lines in the data set.

It has been discussed in Section 5.2 that the triangulation program may not be able to terminate in some cases, which may cause some computer system problems because of the heavy use of memory. However, this kind of problem is hard to detect and solve because its occurrence depends on the distribution of data. Similar problems also occur in other computational geometric modelling applications which are discussed in [56, 80]. For better performance of the surface modelling package, further investigation is required to ensure that the triangulation process terminates properly. Checking the status of triangles may be a good alternative to checking the status of each point and line in the current program. For example, a proposed triangle can be checked through the triangle list. If the triangle is not found in the list, the triangulation is accepted and the triangle is appended into the list. Otherwise the triangulation is rejected and the triangulation program terminates. However, checking the status of triangles is more time-consuming than checking that of points and lines. To ensure the equality of two triangles, many vertex matches of two triangles must be checked because one triangle is defined by three vertices.

Long-term development should emphasize improvement of the boundary process. In the current surface modelling package, human intervention is necessary to ensure an accurate boundary. The boundary is then used in the triangulation program to reduce the boundary effect. The performance of the triangle rejection scheme relates to the distribution of data points. In some cases the scheme does not work well, which has been described in Section 5.2. Some interesting methods to reduce the boundary effect have been proposed in the literature [74, 79]. Even though the method can not guarantee the resultant surface to be correct, it does produce some 3-D surfaces without boundary effect and human intervention. Therefore long-term research and development in investigation of Hoppe's [71] surface modelling method and other possible alternatives would be valuable and useful in clinical applications of surface modelling.



**BIBLIOGRAPHY**

- [1] W.P. Bunnell, "An Objective Criterion for Scoliosis Screening," *Journal of Bone and Joint Surgery*, 66-A, pp. 1382-1387, 1984
- [2] G.C. Robin, "The Aetiology of Idiopathic Scoliosis," *Freund Publishing House*, 1990
- [3] G. De Giorgi, A. Gentile and G. Mantriota, "3-D Study of the Spine: Our Ten Year Experience," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 71-80, 1992
- [4] A. Merolli, "Surface Topography Should Replace Radiography in the Evaluation of Scoliotic Deformity," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 176-177, 1992
- [5] A. Merolli, P.T. Leali and L. Aulisa, "Surface Topography in the Evaluation of Scoliotic Deformity: Its Role in Comparison with Standard Radiography," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 449-455, 1992
- [6] V.J. Raso, B. Greenhill, M.J. Moreau and D. Budney, "The Combined Reconstruction of Spine and Surface Deformity," *Surface Topography and Spinal Deformity, Gustav Fischer Verlag*, pp. 13-20, 1986

- [7] B. Drerup, "3-D Acquisition, Reconstruction and Modelling Techniques Applied on Scoliotic Deformity," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 2-10, 1992
- [8] M.A. Edgar and M.A. Metha, "Long-Term Follow-Up of Fused and Unfused Idiopathic Scoliosis," *Journal of Bone and Joint Surgery*, 70-B, pp. 712-716, 1988
- [9] J.D. Pearson, P.H. Dangerfield, C.A. Hobson and Y. Li, "An Automated Visual System for the Assessment of the 3-D Deformity of Scoliosis," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 50-56, 1992
- [10] R.G. Burwell, "Standard Trunk Symmetry Scores," *Journal of Bone and Joint Surgery*, 65-B, pp. 452-460, 1983
- [11] H. Takasaki, "Moire-Topography," *Appl. Optics*: 9, pp. 1467, 1970
- [12] W. Frobin and E. Hierholzer, "A Stereophotogrammetric Method For the Measurement of Body Surfaces Using a Projected Grid," *Proc. SPIE*, Vol. 166, pp. 39-44, 1980
- [13] W. Frobin and E. Hierholzer, "Automatic Measurement of Body Surface Using Rasterstereography," *Photogrammetric Engineering and Remote Sensing*, Vol. 49, No. 3, pp. 377-384, 1983
- [14] E. Hierholzer, B. Drerup and W. Frobin, "Computerized Data Acquisition and Evaluation of Moire Topograms and Rasterstereographs," *Moire Fringer Topography and Spinal Deformity, Gustav Fischer Verlag*, pp. 233-240, 1982

- [15] G. Wind, V.K. Dvorak and J.A. Dvorak, "Computer Graphic Modelling in Surgery," *Orthopedic Clinics of North America*, Vol. 17, No. 4, pp. 657-668, 1986
- [16] E. Keppel, "Approximating Complex Surfaces by Triangulation of Contour Lines," *IBM J. Research and Development*, pp.2-11, January 1975
- [17] H. Fuches, Z.M. Kedem and S.P. Uselton, "Optimal Surface Reconstruction From Planar Contours," *Communications of ACM*, Vol.20, No.10, pp.693-702, October 1977
- [18] S. Ganapathy and T.G. Dennehy, "A New General Triangulation Method for Planar Contours," *Computer Graphics*, Vol. 16, No.3, pp. 69-75, 1982
- [19] J-D. Boissonnat, "Shape Reconstruction From Planar Cross-Sections," *Computer Vision, Graph and Image Processing*, Vol. 44, No. 1, pp. 1-29, 1988
- [20] D. Meyers, S. Skinner and K. Sloan, "Surfaces From Contours," *ACM Trans. on Graphics*, Vol.11, No.3, pp.228-258, July 1992
- [21] M. Girard and A.A. Maciejewski, "Computational Modelling for the Computer Animation of Legged Figures," *SIGGRAPH 85*, pp. 263-270, 1985
- [22] S.M. Goldwasser, R.A. Reynolds, D.A.Talton and E.S. Walsh, "Techniques for the Rapid Display and Manipulation of 3-D Boimedical Data," *Comp. Med. Imag. and Graphics*, Vol. 12, No. 1, pp. 1-24, 1988

- [23] S.B. Murphy, P.K. Kijewski, S.R. Simon, H.P. Chandler, P.P. Griffin, D.T. Reilly, B.L. Penenberg and M.M. Landy, "Computer-Aided Simulation, Analysis, and Design in Orthopedic Surgery," *Orthopedic Clinics of North America*, Vol. 17, No. 4, pp. 637-649, 1986
- [24] E. Hierholzer and G. Luxmann, "Three-Dimensional Shape Analysis of the Scoliotic Spine Using Invariant Shape Parameters," *J. Biomechanics* 15, pp. 583-598, 1982
- [25] C.E. Aubin, J. Dansereau, and H. Labelle, "Incorporation of Costo-Vertebral Joints Modelisation Into A Personalized Finite Element Model of The Scoliotic Spine: Application for Simulation of Bodton Brace Correction," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp.400-407, 1992
- [26] D. Hill, V.J. Raso, N.G. Durdle and A.E.Peterson, "Designing; a Video Based Technique For Trunk Measurement," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp.157-161, 1992
- [27] P.J.M. Scholten and A.G. Veldhuizen, "Computer Aided Spinal Analysis," *Int. Symposium on 3-D Scoliotic Deformity, Gustav Fisher Verlag (J. Dansereau Ed.)*, pp. 416-423, 1992
- [28] J. Borrell, "Solid Modelling Becomes the Driving Froce Behind Automation," *Digital Design*, pp. 88-100, November 1983
- [29] J.K. Krouse, "Engineering without Paper," *High Technology*, pp. 38-46, March 1986
- [30] M.E. Mortenson, "Geometric Modeling," *John Wiley & Sons*, 1985

- [31] D. Terzopoulos and K. Fleischer, "Modelling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," *SIGGRAPH 88*, pp. 269-278, 1988
- [32] G. Markowsky and M.A. Wesley, "Fleshing Out Wire Frames," *IBM J. of Research and Development*, Vol. 24, No. 5, pp. 582-597, 1980
- [33] J. Foley, A. van Dam, S. Feiner and J. Hughes, "Computer Graphics: Principles and Practices," 2nd Edition, *Addison-Wesley*, 1991
- [34] J.S. Hersh, "A Survey of Modelling Representations and Their Applications to Biomedical Visualization and Simulation," *Proc. 1st Conference on Visualization in Biomedical Computing, IEEE Press*, pp. 432-441, May 1990
- [35] B.A. Barsky and T. DeRose, "The Beta2-Spline: A Special Case of the Beta-Spline Curve and Surface Representation," *Computational Geometry and Applications*, Vol. 5, No. 9, pp. 46-58, 1985
- [36] B.A. Barskey, "Computer Graphics and Geometric Modelling Using Beta-Splines," *Springer-Verlag*, 1988
- [37] R.H. Bartels, J.C. Beatty and B.A. Barsky, "An Introduction to Splines for Use In Computer Graphics," *Morgan Kaufmann, Los Altos, Ca.*, 1987
- [38] S. Uptill, "The Renderman Companion," *Addison-Wesley*, 1990

- [39] M.J. Powell, "Piecewise Quadratic Surface Fitting for Contour Plotting," *Software for Numerical Mathematics, Academic Press (D.J.Evans Ed.)*, pp. 253-271, 1974
- [40] M.J. Powell and M.A. Sabin, "Piecewise Quadratic Approximations on Triangles," *ACM Transactions on Mathematical Software*, Vol. 3, No. 4, pp. 316-325, 1977
- [41] A.H. Barr, "Topics in Physically Based Modelling," *Addison-Wesley*, 1989
- [42] H. Edelsbrunner, T.S. Tan and R. Waupotitsch, "An  $O(n^2 \log n)$  Time Algorithm for the MinMax Angle Traingulation," *Proc. of 6th Annual Symposium on Computational Geometry, ACM Press*, pp.44-52, June 1990
- [43] G.M. Nielson and R. Franke, "Surface Construction Based Upon Triangulations," *Surface in CAGD, North-Holland Publishing Company (R.E.Barnhill and W. Boehm Ed.)*, pp.163-177, 1983,
- [44] B. Chazelle and L. Palios, "Triangulating a Nonconvex Polytope," *Proc. of 5th Annual Symposium on Computational Geometry, ACM Press*, pp.393-400, June 1989
- [45] D.G. Kirkpatrick, M.M. Klawe and R.E. Tarjan, "Polygon Triangulation in  $O(n \log n \log n)$  Time With Simple Data Structures," *Proc. of 6th Annual Symposium on Computational Geometry, ACM Press*, pp. 34-43, June 1990
- [46] J. Ruppert and R. Seidel, "On the Difficulty of Tetrahedralizing 3-Dimensional Non-Convex Polyhedra," *Proc. of 5th Annual Symposium on Computational Geometry, ACM Press*, pp.380-392, June 1989

- [47] C.A. Wang and L. Schubert, "An Optimal Algorithm for Constructing the Delaunay Triangulation of A Set of Line Segments," *Proc. 3rd Annual Symposium on Computational Geometry, ACM Press*, pp.223-232, 1987
- [48] J-D. Boissonnat, "Geometric Structures for Three-Dimensional Shape Representation," *ACM Trans. on Graphics*, Vol.3, No.4, pp.266-286, October, 1984
- [49] J-D. Boissonnat and M. Tellaud, "A Hierarchical Representation of Objects: The Delaunay Tree," *Proc. 2nd ACM Symposium on Computational Geometry, ACM Press*, pp. 260-268, 1986
- [50] D.F. Watson and G.M. Phillip, "Survey: Systematic Triangulations," *Computer Vision, Graphics, and Image Processing*, Vol.26, pp.217-223, 1984
- [51] D.T. Lee and A.K. Lin, "Generalized Delaunay Triangulation for Planar Graphs," *Discrete Computational Geometry*, Vol.1, pp. 201-217, 1986
- [52] D.A. Rhind, "Skeletal Overview of Spatial Interpolation Techniques," *Computer Applications 2*, pp. 293-309, 1973
- [53] C.M. Gold, "Triangular Element Data Structures," *The University of Alberta Computing Services Users Applications Symposium Proceedings*, Edmonton, Canada, pp. 43-54, 1976
- [54] C.M. Gold, T.D. Charters and J. Ramsden, "Automated Contour Mapping Using Triangular Element Data Structures and An Interpolant Over Irregular Triangular

Domain," *Geological and Mineralogical Associations of Canada Program With Abstracts 1*, pp.170-175, May 1976

[55] R. Sibson, "Locally Equiangular Triangulations," *The Computer Journal*, Vol. 21, No. 3, pp.224-233, 1978

[56] F.P. Preparata and M.I. Shamos, "Computational Geometry," *Springer-verlag*, 1985

[57] I. Babuska and A.K. Aziz, "On the Angle Condition in the Finite Element Method," *SIAM J. Numerical Analysis*, Vol. 13, pp. 214-226, 1976

[58] Y.T. Lee, A.D. Pennington and N.K. Shaw, "Automatic Finite-Element Mesh Generation from Geometric Models: A Point-Based Approach," *ACM Trans. on Graphics*, Vol.3, No.4, pp.287-311, October 1984

[59] R. Barrera and A.M. Vazques, "A Hierarchical Method for Representing Relief," *Proc. 4th Symposium on Spatial Information Technologies for Remote Sensing Today and Tomorrow*, South Dakota, pp. 87-92, October 1984

[60] L.D. DeFloriani, "A Hierarchical Structure for Surface Approximation," *Computer Graphics*, Vol. 8, No.2, pp. 475-484, 1984

[61] L.D. DeFloriani and E. Puppo, "Constrained Delaunay Triangulation for Multiresolution Surface Description," in *Proc. 9th IEEE Int'l Conference on Pattern Recognition*, Computer Science Press, 1988



- [62] L. D. DeFloriani, "A Pyramidal Data Structure for Triangle-Based Surface Description," *IEEE Trans. on Comp. Graphics and Applications*, Vol. 9, pp.67-78, March, 1989
- [63] J. Ponce and O. Faugeras, "An Object Centered Hierarchical Representation for 3-D Objects: The Prism Tree," *Computer Vision, Graphics and Image Processing*, Vol. 38, Nol. 1, pp. 1-28, April 1987
- [64] O. Palacios-Velez and B.C. Renaud, "A Dynamic Hierarchical Subdivision Algorithm for Computing Delaunay Triangulations and Other Cloest-Point Problems," *ACM Trans. on Math. Software*, Vol.16, No.3, pp.275-292, September 1990
- [65] L. Scarlatos and T. Pavlidis, "Hierarchical Triangulation Using Terrain Features," *Proc. of '90 Visualization, IEEE Press*, pp.168-175, October, 1990
- [66] L. Scarlatos and T. Pavlidis, "Hierarchical Triangulation Using Cartographic Coherence," *CVGIP: Graphical Models and Image Processing, Vol.54, No.2*, pp.147-161, March, 1992
- [67] B.A. Lewis and J.S. Robinson, "Triangulation of Planar Regions With Applications," *The Computer Journal*, Vol.21, No.4, pp.324-332,1978
- [68] B. Chazelle and J. Incerpi, "Triangulation and Shape Complexity," *ACM Trans. on Graphics*, Vol.3, No.2, pp.135-152, April 1984
- [69] S.E. Chen and R. E. Parent, "Shape Averaging and Its Applications to Industrial Design," *IEEE Trans. on Graphics and Applications*, pp.47-54, January 1989

[70] S.B. Tor and A.E. Middleditch, "Convex Decomposition of Simple Polygons," *ACM Trans. on Graphics*, Vol.3, No.4, pp.244-265, October 1984

[71] A. Oxley, "Surface Fitting by Triangulation," *The Computer Journal*, Vol. 28, No. 3, pp.335-339, 1985

[72] D.H. McLain, "Two Dimensional Interpolation From Random Data," *The Computer Journal*, Vol.21, No.4, pp.178-181, 1976

[73] D.F. Watson, "Computing The n-dimensional Delaunay Tessellation With Application to Voronoi Polytopes," *The Computer Journal*, Vol. 24, No. 2, pp.167-172, 1981

[74] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, "Surface Reconstruction From Unorganized Points," *SIGGRAPH 92*, pp.71-78, 1992

[75] "The graPHIGS Programming Interface," *Version 2, Release 2.0, IBM Corporation*, November, 1990

[76] R.J. Renka, "Interpolation of Data on the Surface of a Sphere," *ACM Trans. on Math. Software*, Vol.10, No.4, pp.417-436, December 1984

[77] J. Bentley, "Programming Pearls," *Addison-Wesley*, 1989

[78] R. Sedgwick, "Algorithms," 2nd Edition, *Addison-Wesley*, 1988

[79] W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3-D Surface Reconstruction Algorithm," *Computer Graphics*, Vol. 21, No. 4, pp. 161-169, 1987

## APPENDIX A.1. PSEUDO CODE FOR UNCONSTRAINED TRIANGULATION

This section provides the pseudo code of the main program, and the routine of searching for a point to form a triangle. For the main program, the file name containing the input data is given while data are assumed to be already formatted and points to be distinct, as described in section 3.3.1. The output of this program is a file containing a triangle list for the data set. Each triangle consists of the indices of the three vertices. Status of a point can be either "*Unconnected*" (NOT\_CONN) or "*Connected*" (CONN). Only those points which do not have any triangles connected with them are considered "*Unconnected*".

### 1. Main Program for Triangulation

```
{ input:    name of the input file containing the data to be modelled.  
  output:   connection list of triangles in an output file.  
  precondition: data are already formatted, and identical. }
```

**BEGIN**

```
open input file and read data into a point array  
initialize status for each point to NOT_CONN  
initialize a temp_line as a temporary line
```

```

create a line list
set Finish flag as "FALSE"
open an output file for writing triangle connections
find the point (P1) which is nearest to the center of the region
find the nearest neighbor (P2) of point P1
construct line P1-P2 and update status for P1, P2 to CONN
find point(s) for line P1-P2 for the first triangle
{here circles through P1, P2, and valid point P are examined and each circumcenter
is computed; the distance from each of this circumcenter to line P1-P2 is calculated.
The point(s) with the least distance are then found as P3 for 'optimal' triangulation}
    If one point (P3) Then
        update point status for P3 to CONN
        write P1-P2-P3 into outfile as the first triangle
        append lines P2-P3 and P3-P1 into the list
        set temp_line as line P1-P2 with facing vertex P3
    Else If more points Then
        put these points into a dynamic array More_Point[n]
        For each point More_Point[i] with i = 1 to n Do
            set point More_Point[i] status to CONN
            calculate angle between P2-P1 and P2-More_Point[i]
        sort More_Point such that their angles in decreasing order
        write triangle P1-P2-More_Point[1] into output file
        append line P2-More_Point[1] into line list
        For each point More_Point[i] with i = 2 to n Do
            write triangle P1-More[i-1]-More_Point[i] into output file

```

```

        append line More_Point[i-1]-More_Point[i] into line list
    append line More_Point[n]-P1 into line list
    set temp_line as line P1-P2 with facing vertex More_Point[1]
    free array More_Point

```

Else

```

    {no more points to connect--}EXIT

```

While not Finish Do

```

    find point(s) for line temp_line
    {similar way to find point(s) P as that for the first triangle}
    If point not found Then
        If list empty Then
            set Finish flag TRUE
        Else
            pop a line element from the list to temp_line
        endif {empty}

```

Else If one point (P) found Then

```

    write new triangle (endpoint2-P-endpoint1) into output file
    If P is NOT_CONN Then

```

```

        update status for P to CONN
        append line endpoint2-P into line list
        set temp_line as P-endpoint1

```

Else If P is CONN Then

```

    constrcut line1 as endpoint2-P and line2 as P-endpoint1
    check line1 and line2 through the line list
    if line1 found

```

```

delete line1 in the list
If line2 found Then
    delete line2 in the list
    If list not empty Then
        pop a line element from the list to temp_line
    Else
        set Finish flag as TRUE
    endif {not empty}
Else If line2 not found Then
    set temp_line as line2
endif {line2 found/not found}
Else If line1 not found
    If line2 found Then
        delete line2 in the list
        set temp_line as line1
    Else If line2 not found Then
        append line1 into line list
        set temp_line as line1
    endif {line2 found/not found}
endif {line1 found/not found}
endif {CONN/NOT_CONN}
Else If more points Then
    put these points into a dynamic array More_Point
    For each point More_Point[i] with i = 1 to n Do
        calculate the angles between lines P2-P1 and P2-More_Point[i]

```

```

sort More_Point such that their angles in decreasing order
write triangle endpoint2-More_Point[1]-endpoint1 into output file
If More_Point[1] NOT_CONN Then
    update status for More_Point[1] to CONN
    append line endpoint2-More_Point[1] into the line list
Else If More_Point[1] CONN Then
    If line endpoint2-More_Point[1] found in line list Then
        delete the line element from the list
    Else
        append the line into the list
    endif {line found/not found}
endif {CONN/NOT_CONN}
For each More_Point[i] with i = 2 to n Do
    write triangle More_Point[i-1]-More_Point[i]-endpoint1 to outfile
    If More_Point[i] NOT_CONN Then
        update status for More_Point[i] to CONN
        append line More_Point[i-1]-More_Point[i] into line list
    Else If More_Point[i] CONN Then
        If line More_Point[i-1]-More_Point[i] found in the list Then
            delete the line element from the list
        Else
            append the line into the list
        endif {line found/not found}
    endif {CONN/NOT_CONN}
endfor

```



```

    If line More_Point[n]-endpoint1 found in the line list Then
        delete the line element from the list

    Else
        append the line into the list
    endif {line found/not found}

    If list empty Then
        set Finish flag TRUE
    Else
        pop a line element from list to temp_line
    endif {list empty}

    free dynamic array More_Point
endif {one point/more point/no point}

endwhile

close input/output files

free memory for line list and point status

free memory for the point array

END {main}

```

## 2. Routine to Find Point(s) For A Line

```

{input:      a line with two endpoints: P1 and P2, and its facing vertex V;
output:     the index of the point that is optimal to points P1, P2 for triangulation, or
            the indices of 'm' points (more_point[m] that are optimal to points P1, P2
            for triangulation, or return No_Point if no points found for the line;
precondition: an array of 'n' points have been defined. }

```

```

BEGIN
construct the line equation P1-P2 with its direction adjusted by V
initialize num_of_point to '0'
initialize temp_dist to MAX
For each point (P) of the point array Do
    If the point is neither P1 nor P2 Then
        calculate the distance from point (P) to line P1-P2
        If the distance positive Then
            calculate circumcenter of P1, P2 and P
            calculate distance from the circumcenter to line P1-P2
            If distance less than temp_dist Then
                update the temp_dist to this distance
                set num_of_point to '0'
                set more_point[num_of_point] to the index of the point
                num_of_point incremented by '1'
            Else If the distance equal to temp_dist Then
                set more_point[num_of_point] to the index of the point
                num_of_point incremented by '1'
            endif {distance comparing with temp_dist}
        endif {distance positive}
    endif {point valid}
end {For}
If num_of_point greater than '1' Then
    set More_Point TRUE

```

Else If num\_of\_point equal to '1' Then

    set One\_Point True

Else

    set No\_Point True

Return

END {routine}

## APPENDIX A.2. PSEUDO CODE FOR TRIANGULATION WITH THE BOUNDARY CONSTRAINT

This program is modified from the original one in Appendix A.1. In this program the boundary constraint is considered. Points are classified into "*Unconnected*" (NOT\_CONN), "*Connected*" (CONN) and "*Boundary-point*" (BOUND). Input files include the file containing the data, the file containing the boundary list, and the file containing the deleted points. Once a triangle containing a boundary point as its vertex is proposed, a routine which examines the boundary constraint is called.

### 1. Main Program for Triangulation The Boundary Constraint

```
{ input:      names of the data file, the boundary file and the delete_points file.
  output:     connection list of triangles in an output file.
  precondition: data are already formatted, and points are distinct. }
```

BEGIN

open the data file to read data into a point array

initialize status for each point to NOT\_CONN

open the boundary file to read in boundary points

update status for boundary points to BOUND

open delete\_points file to read deleted points into an array

initialize a temp\_line as a temporary line  
 create a line list  
 set Finish flag as "FALSE"  
 open an output file for writing triangle connections  
 find the point (P1) which is nearest to the center of the region  
 find the nearest neighbor (P2) of point P1  
 construct line P1-P2 and update status for P1, P2 to CONN  
 find point(s) for line P1-P2 for the first triangle  
 {here circles through P1, P2, and valid point P are examined and each circumcenter  
 is computed; the distance from each of this circumcenter to line P1-P2 is calculated.  
 The point(s) with the least distance are then found as P3 for 'optimal' triangulation}

If one point (P3) Then

If P3 is BOUND Then

Boundary\_Condition\_Checking (P1, P2, P3)

Else

update point status for P3 to CONN

write P1-P2-P3 into outfile as the first triangle

append lines P2-P3 and P3-P1 into the list

set temp\_line as line P1-P2 with facing vertex P3

Else If more points Then {suppose  $n$  points}

put these points into a dynamic array More\_Point[ $n$ ]

For each point More\_Point[ $i$ ] with  $i = 1$  to  $n$  Do

calculate angle between P2-P1 and P2-More\_Point[ $i$ ]

sort More\_Point such that their angles in decreasing order

If More\_Point[1] is BOUND Then

```

Boundary_Condition_Checking (P1, P2, More_Point[1])
Else
    write triangle P1-P2-More_Point[1] into output file
    update status for More_Point[1] to CONN
    append line P2-More_Point[1] into line list
For each point More_Point[i] with i = 2 to n Do
    If More_Point[i] is BOUND Then
        Boundary_Condition_Checking(P1,More_Point[i-1],
                                    More_Point[i])
    Else
        write triangle P1-More_Point[i-1]-More_Point[i]
                                into output file
        Updata status for More_Point[i] to CONN
        append line More_Point[i-1]-More_Point[i] into line list
    append line More_Point[n]-P1 into line list
    set temp_line as line P1-P2 with facing vertex More_Point[1]
    free array More_Point
Else
    no more point to conect--EXIT
While not Finish Do
    find point(s) for line temp_line
    {similar way to find point(s) P as that for the first triangle}
    If point not found Then
        If list empty Then
            set Finish flag TRUE
        Else

```

```

        pop a line element from the list to temp_line
    endif { empty}
Else If one point (P) found Then
    If P is BOUND Then
        Boundary_Condition_Checking (endpoint1, endpoint2, P)
    Else If P is NOT_CONN Then
        write new triangle (endpoint2-P-endpoint1) into output file
        update status for P to CONN
        append line endpoint2-P into line list
        set temp_line as P-endpoint1
    Else If P is CONN Then
        constrcut line1 as endpoint2-P and line2 as P-endpoint1
        check line1 and line2 through the line list
        if line1 found
            delete line1 in the list
            If line2 found Then
                delete line2 in the list
                If list not empty Then
                    pop a line element from the list to temp_line
                Else
                    set Finish flag as TRUE
                endif {not empty}
            Else If line2 not found Then
                set temp_line as line2
            endif {line2 found/not found}
        endif {line1 found}
    endif {P is BOUND}
endif {one point (P) found}

```

Else If line1 not found

    If line2 found Then

        delete line2 in the list

        set temp\_line as line1

    Else If line2 not found Then

        append line1 into line list

        set temp\_line as line1

    endif {line2 found/not found}

    endif {line1 found/not found}

endif {BOUND/CONN/NOT\_CONN}

Else If more points

    put these points into a dynamic array More\_Point

    For each point More\_Point[i] with i = 1 to n Do

        calculate the angles between lines P2-P1 and P2-More\_Point[i]

    sort More\_Point such that their angles in decreasing order

    If More\_Point[1] is BOUND Then

        Boundary\_Condition\_Checking (endpoint2,  
                                          More\_Point[1], endpoint1)

    Else If More\_Point[1] NOT\_CONN Then

        write triangle endpoint2-More\_Point[1]-endpoint1 into output file

        update status for More\_Point[1] to CONN

        append line endpoint2-More\_Point[1] into the line list

    Else If More\_Point[1] CONN Then

        If line endpoint2-More\_Point[1] found in line list Then

            delete the line element from the list



```

Else
    append the line into the list
endif {line found/not found}
endif {BOUND/CONN/NOT_CONN}
For each More_Point[i] with i = 2 to n Do
    If More_Point[i] is BOUND Then
        Boundary_Condition_Checking(More_Point[i-1],
                                     More_Point[i], endpoint1)
    Else If More_Point[i] NOT_CONN Then
        write triangle (More_Point[i-1]-More_Point[i]-endpoint1)
        update status for More_Point[i] to CONN
        append line More_Point[i-1]-More_Point[i] into line list
    Else If More_Point[i] CONN Then
        write triangle (More_Point[i-1]-More_Point[i]-endpoint1)
        If line More_Point[i-1]-More_Point[i] found in the list Then
            delete the line element from the list
        Esle
            append the line into the list
        endif {line found/not found}
    endif {CONN/NOT_CONN}
endifor
If line More_Point[n]-endpoint1 found in the line list Then
    delete the line element from the list
Else
    append the line into the list
endif {line found/not found}

```

```

    If list empty Then
        set Finish flag TRUE
    Else
        pop a line element from list to temp_line
    endif {list empty}
    free dynamic array More_Point
endif {one point/more point/no point}
endwhile
close input/output files
free memory for line list and point status
free memory for the point array
END {main}

```

## 2. Boundary\_Condition\_Checking

```

{input:      three point indices for a proposed triangle, i.e., P1, P2, P3;
output:      the triangle connection with lines processed if the triangle is valid;
              otherwise returns nothing.
precondition: none}

```

```
BEGIN
```

```

If P1, P2, P3 are BOUND Then
    check through boundary list for the boundary line segments
    If line segments found Then

```

calculate the cross product (**CP1**) of position vectors of the line segments

calculate the cross product (**CP2**) of position vectors of the triangle

If **CP1** and **CP2** are in opposite directions Then

{triangle is invlaid, return}

If line list empty Then

set finish flag as TRUE

Else

pop a line into temp\_line

endif {empty}

endif {directions}

endif {lines found}

endif {BOUND}

calculate length for each edge of the triangle

If any of the lengthes greater than Threshold Then

{triangle invalid, return}

If line list empty Then

set finish flag as TRUE

Else

pop a line into temp\_line

endif {empty}

Else

write triangle into output file

construct lines P1-P2 and P2-P3

check line P1-P2 and P2-P3 in line list

If P1-P2 found Then

```
If P2-P3 found Then
    If line list empty Then
        set finish flag as TRUE
    Else
        pop a line into temp_line
    endif {empty}
Else
    set temp_line as P2-P3
endif {P2-P3 found}
Else
    If P2-P3 found Then
        set temp_line as P1-P2
    Else
        append line P2-P3 into list
        set temp_line as P1-P2
    endif {P2-P3 found}
endif {P1-P2 found}
endif {length greater than Threshold}

END
```

## **APPENDIX A.3. DISPLAY AND SHELL SCRIPT PROGRAM**

### **1. Surface Display Program**

The constrained triangulation program described in Appendix A.2 generates a triangle connection list which is stored in a file. With this connection list, and the original data points, the triangles can be displayed in a 3-D space.

The basic structure for the display program is a polygon (triangle). The Surface Display Program was implemented in the way similar to the user interface used in boundary modification. However, since a surface model can not be changed once constructed, the display program does not need the modification menu. Rotation of the model is the main process required in the Display Program. Due to the solid surface display, rotation of the object is not as fast as rotation of the data points and boundary. Therefore rotation was implemented using keyboard to drive the rotation instead of the mouse.

The Display Program first opens the data file to read in the original 3-D data points into an array. With this reading process, the dimensions of the Object Window (the view volume) is also defined by the maximum and minimum values of  $x$ ,  $y$  and  $z$ .

Then the program opens the triangle connection file to read in the connection list. During this process, each connection of the vertices by their indices are converted into the

actual point connection with three coordinates for each point. The actual triangle is added into the triangle structure. Once all connections are read and converted, the graphical structure is displayed in the Object Window.

Controls to the Display include "Exit" from the program, "Rotation" of the surface model, and "Erase/Add" edges of triangles, and "Reset" to the original orientation.

Exit mode terminates the display program. It is activated by pressing "E" or "e". Once this happens, all memory and graphical resources used in the display program are released.

Once the coordinate system is defined, rotation of the object can be performed about each of the three axes while the center of the object is kept unchanged. Keys "→" and "←" rotate the model about y axis. Keys "↓" and "↑" are used to rotate the model about x axis. "Shift ←" (or "Alt ←") and "Shift →" (or "Alt →") are for the rotations about z axis. The direction of rotation is in accordance with the definition of the coordinate system (right-handed system). The number of degrees for rotation can be incremented by pressing "+" or decremented by pressing "-". There are different number of degree intervals to be selected for the amount of increment or decrement. "R" or "r" is used to reset the surface to its original orientation.

When the surface is displayed, the edges of the triangles can be either displayed or eliminated. The default is set so that the original surface model is displayed with these

edges. If the user is not interested in the connections, the edges can be erased from the model by pressing "." on the keyboard. Pressing the same key again will add the edges into the existing model.

## **2. Shell Script Drive Program**

Four steps are necessary to produce a surface model from a given data set: (1) generation of the boundary from the data set; (2) modification to the boundary; (3) triangulation of data; and (4) display of the triangulated surface. Each of these steps is an independent program with input file(s) and output file(s)--except for the display program which generates a graphical object on the screen instead of an output file.

In most cases, users prefer to only enter name of the file containing the data to be modelled, and then expect the final surface model on a computer screen. Therefore these four programs are linked together as a package. The input and output of each program are shown in *Figure A.1*.

For better management of data and the programs, the files generated for each given data set are named according to the following conventions:

Suppose the file containing the data for modelling is "A". The input to Boundary Generation program is the name of this file "A". The file opened by this program to store

the boundary for the data set is named "A.boundary". And the output of the boundary modification program are two files: "A.final\_bound" for final boundary and "A.delete\_point" for the points deleted by the user. In the triangulation program, the output is the triangle connection list which is named "A.surface" indicating that this surface is derived from the original data set "A". These file naming conventions make it easy for users to recognize the file relationships, easy to add additional versions of surface model using different boundary lists.

If the surface of a given data set is already available, the user can either directly display the surface image without triangulating the data again, or chose to triangulate the data again for a comparison of the triangulation effect using different boundaries. The flow-chart in *Figure A.2* shows the linkage of the four programs. Miscellaneous information module lists names of the data files available for modelling. The system asks the user to respond for the proper action. In the Boundary Generation program, the user is asked to enter the resolution of griding of the original data set, i.e., numbers of rows and columns. At each query step, the system waits until the user enters a proper command or number.



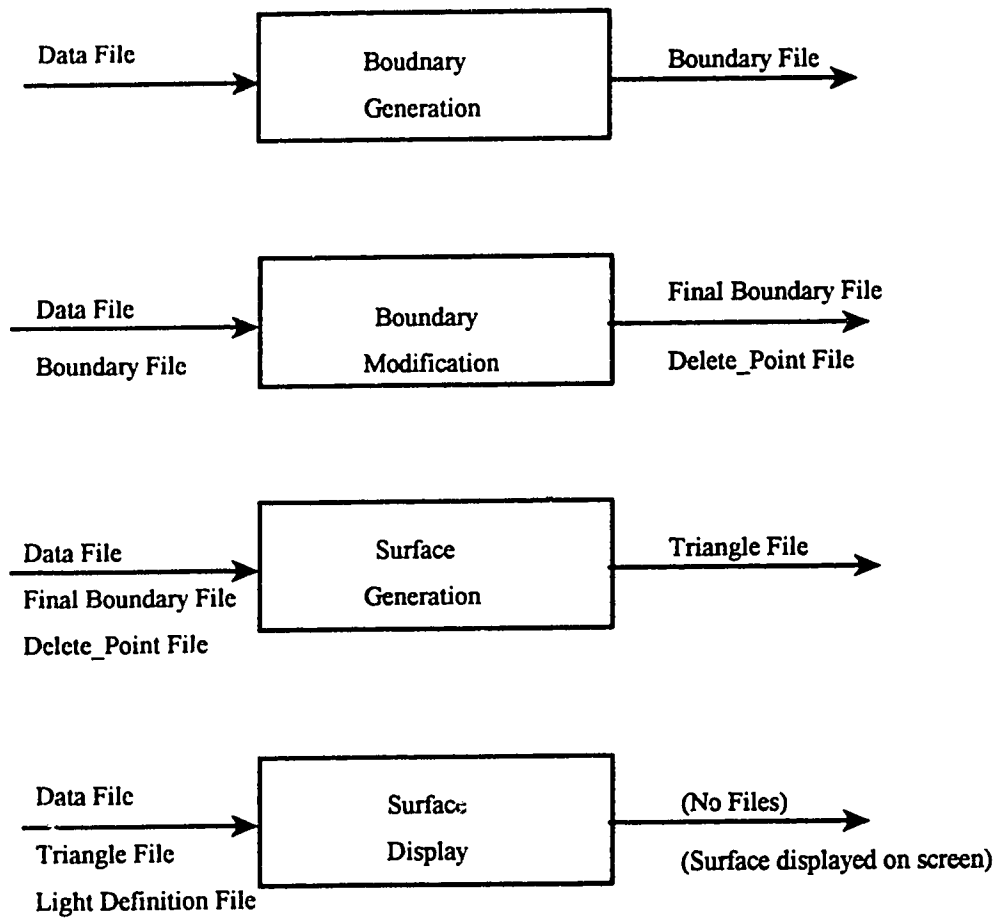


Figure A.1 Programs and their input/output.

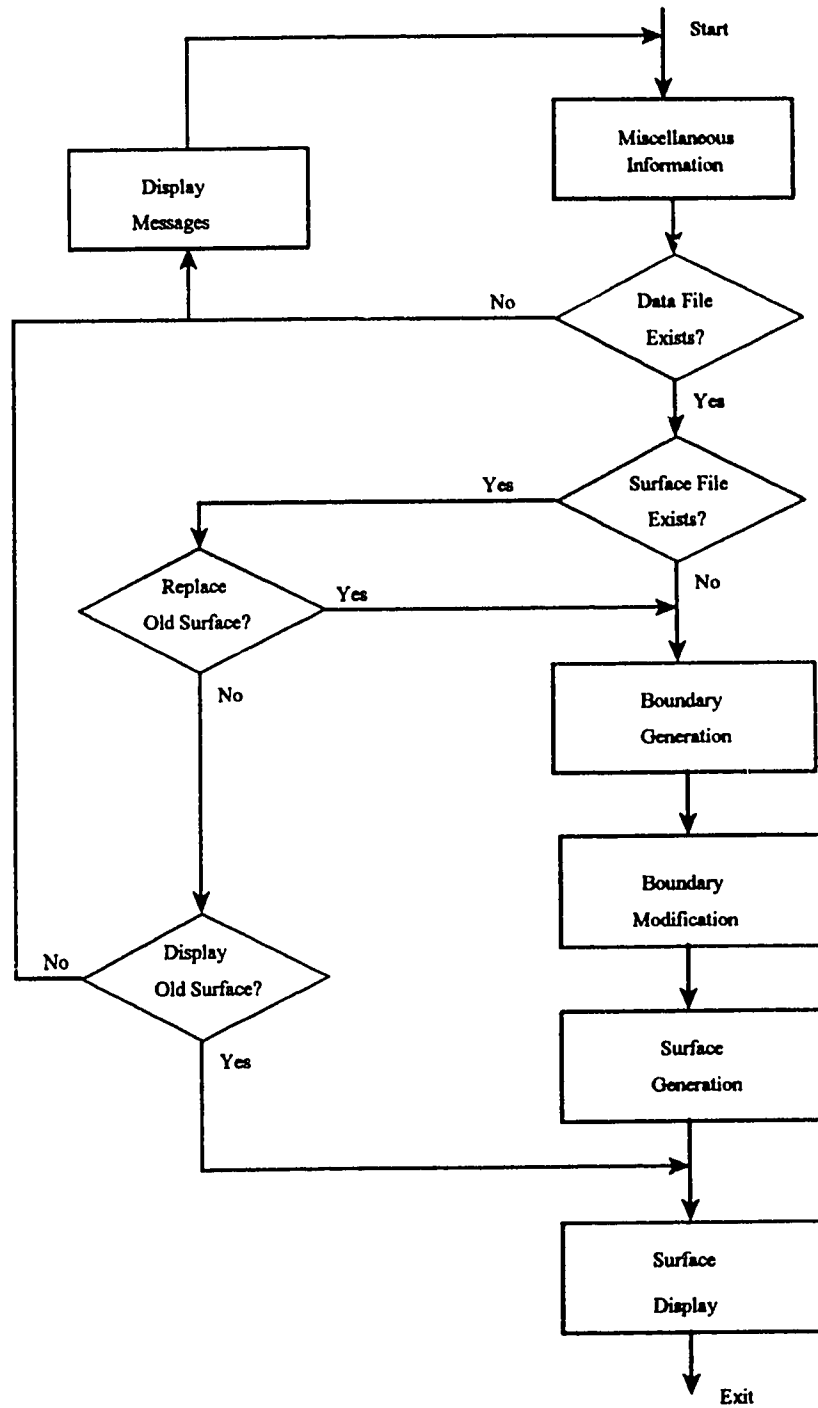


Figure A.2 Package drive program