The reconstruction of measured engineering components and their comparison with solid models

Cakir, Mustafa Cemal

Award date:
1989

Awarding institution:
University of Bath

THE RECONSTRUCTION OF MEASURED ENGINEERING COMPONENTS AND THEIR COMPARISON WITH SOLID MODELS

Submitted by Mustafa Cemal CAKIR

for the degree of PhD

of the University of Bath

1989

COPYRIGHT

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

When components have been made they must be inspected. In the research reported in this thesis a group of algorithms which form an automatic inspection method are described. Thus, a set of measurements of an engineering component - the coordinates of its surface points - taken by a coordinate measuring machine (particularly the non-contact type which is being developed at Bath University) [61] can be compared with a master solid model from a CAD system. Some of these algorithms exist in the literature on Stochastic Computational Geometry, but have not before been applied to this problem, some algorithms are new. Although the surface points can be gathered by using a measuring machine, an algorithm is also introduced which simulates its function.

The algorithms include:

a. methods for creating a tetrahedral packing with the measured points as vertices in which the surface of the component will be embedded as triangular facets,

b. methods for finding out which of the tetrahedra are *solid*, and form a solid body which represents the measured component,

c. methods for creating a surface model of the component by finding the surface triangles of each solid tetrahedron,

d. methods for finding the real faces of the measured component by finding the triangles lying on the same surface,

e. methods for matching the measured component to the solid model of the object created by CAD system under translation and rotation.

Once the two descriptions are matched, faces of the measured component may then be compared with the corresponding faces of the solid model and any out-of-tolerance differences can thereby automatically be reported. The way in which the matching is achieved is robust in the presence of errors in the component which cause it to differ slightly in shape from the solid model. As the whole purpose of measurement is to check for such errors, this is particularly important.

ACKNOWLEDGEMENTS

Firstly, I wish to express my deepest gratitude to my supervisor, Dr. Adrian Bowyer, who, being most generous with his time, has been a continual source of guidance, encouragement and inspiration. I am also grateful to him for reviewing the manuscript. I also wish to express my deep appreciation to Mr. Andrew Wallis for his advice and suggestions on various parts of my project.

Special thanks are due to: Mr. Nigel R. Phelan, Mr. Ian Walker and Mr Izzet Isik for many useful discussions and their helpful comments on my draft thesis. I also wish to thank Mr. Antony E. Carter for his help, support and friendship. Above all I am indebted to all my colleagues, my friends and all the people who have contributed directly or indirectly to this work.

Finally, I wish to thank my loving wife for her help, support, understanding and love throughout the project without which it would have never been completed.

LIST OF FIGURES

Page

Cont'd.

Page

Cont'd.

Page

Cont'd.

Page

Cont'd.

Page

CONTENTS

Page

Cont'd.

Page

Cont'd.

Page

Cont'd.

Page

Cont'd.

Page

Cont'd.

Page

CHAPTER 1

INTRODUCTION

1.1 Background To Solid Modelling

1.1.1 CAD

Computer-aided design (CAD) is the technology which is concerned with the use of computers to perform certain functions in design [11, 57]. It can also be defined as the use of computer systems to assist in the creation, modification, analysis or optimization of a design. Its hardware typically consists of a computer, one or more graphics display terminals and keyboards and other peripheral equipment; while its software consists of the computer programs to implement computer graphics on the system and the application programs such as those for stress-strain analysis of components, computing the dynamic response of mechanisms, heat transfer calculations, numerical control part programming and so on.

The various design-related tasks which are performed by a modern computer-aided design system can be grouped into five functional areas [54]:

1. Geometric modelling
2. Engineering analysis
3. Design review and evaluation
4. Automated drafting

5. Part classification and coding

In the research reported in this thesis the author dealt with geometric modelling and the checking of manufacturing errors by using a geometric modeller.

1.1.2 Geometric Modelling

Geometric modelling is the process in which a mathematical model is created to represent, store and manipulate geometric information about the size and the shape of physical objects in computer memory [4, 11, 54, 80]. In theory the class of physical objects is restricted to objects that are solid and rigid with a mathematically well-behaved surface. They can move about in space with the restriction that two physical objects cannot occupy the same space at the same time (in theory: not all systems prevent this).

Geometric modellers can be classified in three groups:

1. Wire-frame modellers
2. Surface modellers
3. Solid modellers

1.1.2.1 Wire-frame Modelling

Wire-frame geometry is the first order of complexity in the definition of geometric models [11, 31, 54, 57, 81]. Wire-frame pictures are the simplest to create to check results in the quickest way. They expend relatively little computer time and memory and provide accurate information on the location of edges on the part.

But, since they are line models (which means they only require 3D coordinates to define the end points of lines in space), they can only provide partial information about objects. They convey no information about the surfaces themselves; they do not differentiate between the inside and the outside of the object; and since only the vertices and connecting lines are present to interpret the model, several interpretations may arise from a single model. **Figure 1.1** shows an example of this type of ambiguity. Although fast wire-frame displays are still popular, because of the ambiguity of representing such quantities as the surface area and volume of the object wire-frame modelling has been replaced by solid modelling in the recent years.

1.1.2.2 Surface Modelling

A higher level of sophistication in geometric modelling is *surface modelling* [11, 31, 57, 69, 81]. This is one of the major techniques used in representing three-dimensional objects. A surface model can be built by defining the surfaces on the wire-frame model in a way which is analogous to stretching a thin piece of material over a framework. **Figure 1.2** shows the types of surfaces that the surface modellers may have.

Essentially, surface models are in the form of a mesh and the surface is a collection of facets (nearly all systems allow curved surfaces). They define part geometry such as surfaces and boundaries precisely and they help to produce smooth continuous surfaces in NC machining. They are used in aircraft and aerospace engineering, the automative industry and shipbuilding, as well as in

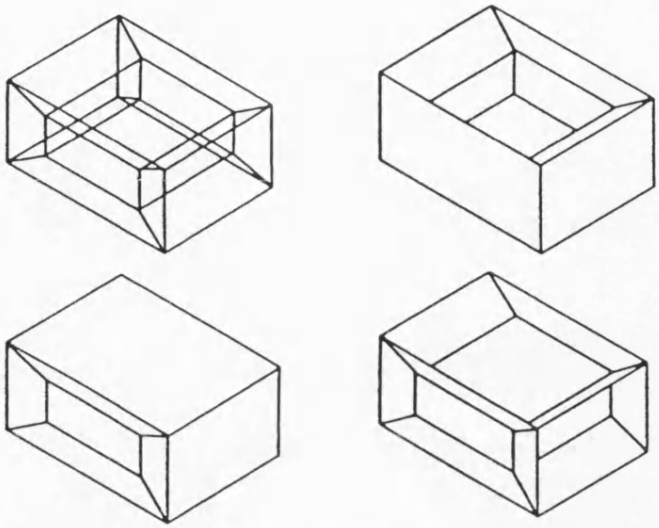


Figure 1.1 Possible interpretations of a wire-frame cuboid
(from Rooney et al. [96])

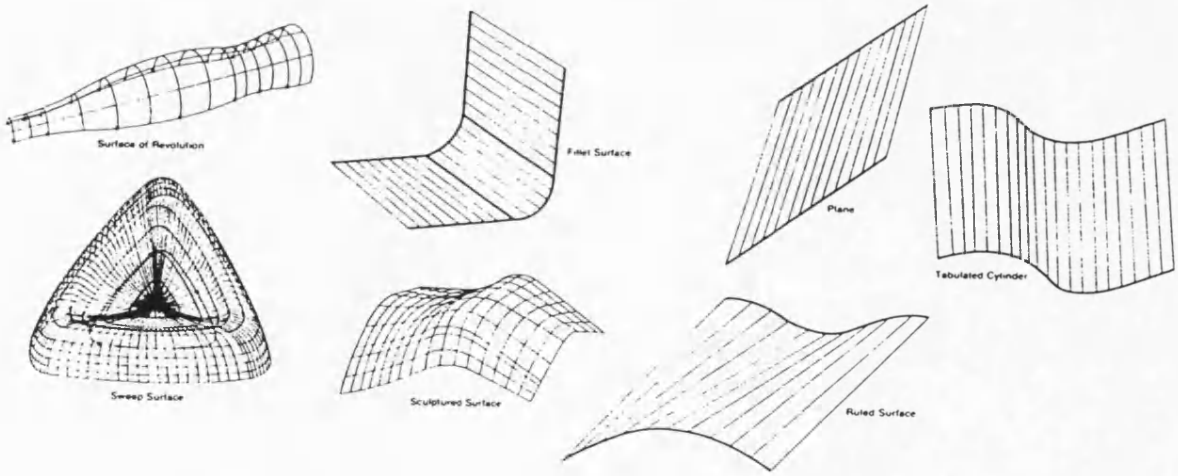


Figure 1.2 Types of surfaces (from Krouse [69])

medium sized companies manufacturing forgings, castings and moulded products. But they are mainly used in computer graphics. Many of the ambiguities of wire-frame models can easily be overcome by surface models.

However, surface models alone are not suitable as a general means of representing mechanical parts. For example, a face may be left off a part, there may be gaps between faces, or a face which occupies space inside a part may be defined. Since surface models do not prevent such errors, they cannot guarantee accuracy; and without accuracy highly-automated applications are impossible to achieve.

1.1.3 Solid Modelling

The highest level of sophistication in geometric modelling is *solid modelling* [11, 31, 54, 57, 81, 96]. The term *solid modelling* encompasses a body of theory, techniques and systems focused on informationally complete representations of solids which permit any (in theory at least) well-defined geometrical property of any represented solid to be calculated automatically. Given the appropriate data, these systems can in principle represent and calculate any kind of information about objects, for example their colours, volumes, centres of gravity, moments of inertia, surface areas, costs and so on. Since they can handle both surface and volume problems and they are able to deal with complicated geometries, they are probably the best current method for representing engineering components.

The solid model provides a single computer representation of an engineering component which, again in theory, can be used for all computer assisted

engineering and manufacturing tasks. Since it is complete and unambiguous, it can form the basis of a highly automated CAD/CAM application program.

The activities to which a solid model can be applied are as follows [89] :

a. *Design conception:* Component geometry can be defined by creating a solid model; new parts or components are most frequently created by modifying existing designs,

b. *Design calculation:* Characteristics of the solid model such as its enclosed volume, surface area, principal axes, moments around each axis and mass can be computed to test the initial design against the fundamental constraints imposed,

c. *Clearance studies:* The solid model can graphically be assembled with other solid models of the same product to ensure that the parts will fit together or move correctly,

d. *Structural analysis:* The distribution of temperature or mechanical loading within the solid model can be determined for a given set of conditions by using finite element mesh models. In this way the design may be checked against design requirements at an early stage without extensive prototype fabrication and testing,

e. *Kinetic analysis:* The solid model has a high potential for automated kinetic analysis where loads are calculated for an articulated mechanism. For example, the motion of a robot could be computed by simulating its response to the robot's program. Mass properties can automatically be computed from the

solid models of each component. These properties are then used to compute loads on the robot's joints at each point in time in the simulation,

f. *Kinematics:* If component is required to move during use, its working envelope can be checked on a graphics display screen and possible obstacles and collision courses can be detected (e.g. robot arm movements),

g. *Ergonomics:* If the product is to provide human comfort, the geometry produced can be checked using standard human body models (e.g. using endomorph and ectomorph models in say, the design of a driver's cab) [57],

h. *Jig and fixture design:* Jigs and fixtures are often redesigned for existing products to improve productivity or to make use of new machines. Solid modelling can be used to design this tooling correctly in relation to a model of the part,

i. *Tooling design:* Solid modelling is as productive for designing and laying-out tooling as it is for designing assemblies. If special tooling is required during manufacture, the tool can be designed around the solid model of the part to be machined or assembled,

j. *Machining simulation:* A sequence of machining can be planned and tool paths can be checked by driving a cutting tool around the solid model of the part, a model of the machined component can be created by subtracting the swept volumes and precise measurements of dimensions can be taken from the model, production planning sheets and NC tape can be created accordingly [109],

k. *Casting, forging and joining processes:* Having constructed a solid model for a casting process, the volume of an object can be calculated to determine the amount of material required. Even more sophisticated analyses such as that of the cooling rate in castings or material flow in injection mouldings can be achieved; the plastic deformation of objects in forging processes or the strength of the joints in welding process can be analysed [114],

l. *Machine vision:* By linking machine vision techniques to models, systems can be trained to recognise different components and their positions and orientations (e.g. parts on a conveyor can be identified by an overhead camera and mis-oriented parts classified) [114],

m. *Quality control:* The solid model can be used to determine how quality will be assessed and possibly improved upon,

n. *Detail drawings:* On acceptance of design, the 3D solid model can be manipulated to produce 2D orthographic views and dimensioned accordingly,

o. *Assembly and assembly planning drawings:* The solid model can be manipulated with its mating models to produce 2D orthographic assembly views; bills of materials and parts lists can be created accordingly; the order of assembly can be planned on the graphics display screen by using the solid model of the product; a series of exploded isometric views together with the necessary assembly instructions can then be produced,

p. *Production advertising:* Production presentation and appeal can be enhanced by using solid models with colour shading to provide graphical illustrations

for use in sales brochures,

q. *Technical publications:* The production of maintenance manuals and customer instruction manuals can be supported by using solid modellers.

In contrast to all these benefits, the main hinderance that solid modellers have is that they are relatively slow, and are greedy for computer processing power when compared to wire-frame or surface modellers. Problems of solid modelling systems and their solutions are discussed by Chiyokura [31].

All solid modelling systems provide facilities for creating, modifying and inspecting models of three-dimensional solid objects, but the way of representing such models in the computer differs. In general, we may classify the representation schemes in six groups [96]:

1. Pure primitive instancing
2. Generalised sweeps
3. Spatial occupancy enumeration
4. Cellular decomposition
5. Boundary representation (B-rep)
6. Set-theoretic representation

All these methods won't be discussed here; the reader is referred to [96] for full details. Set-theoretic representation and boundary representation are the most widely used representation schemes in commercial modellers.

1.1.3.1 Boundary Representation

In boundary representation, a solid is represented by its boundary elements [31, 81, 96]. These elements are classified in two ways: geometric elements (points, curves and surfaces) which form the object, and topological elements which define the relationships between the geometric elements. The boundary representation approach keeps a list of faces, edges and vertices of the object together with the topological and adjacency relationships between them. The simplest boundary representation model is the triangular-faced polyhedron which is stored as a list of triangles. Arbitrary surfaces are approximated to any desired degree of accuracy by utilising many triangles.

1.1.3.2 Set-theoretic Representation

In the set-theoretic representation, a model is constructed from basic, three-dimensional, volumetric solid primitives such as blocks, cylinders, cones, spheres, hexahedra, tori and tubes [31, 67, 81, 96]. These basic primitives are often combinations of even simpler entities known as *half spaces*. The relationships between primitives are defined using the Boolean operators *union, intersection* and *difference* [57]. Types of modelling primitives and the effects of Boolean operations on primitives are shown in **Figure 1.3.**

The data structure representing the complete object consists of a binary tree **(Figure 1.4)** in which the leaf nodes represent the primitive solids and the internal nodes represent the Boolean operators to combine these. The primitive solids which form the leaves may either be represented by simple functions which define

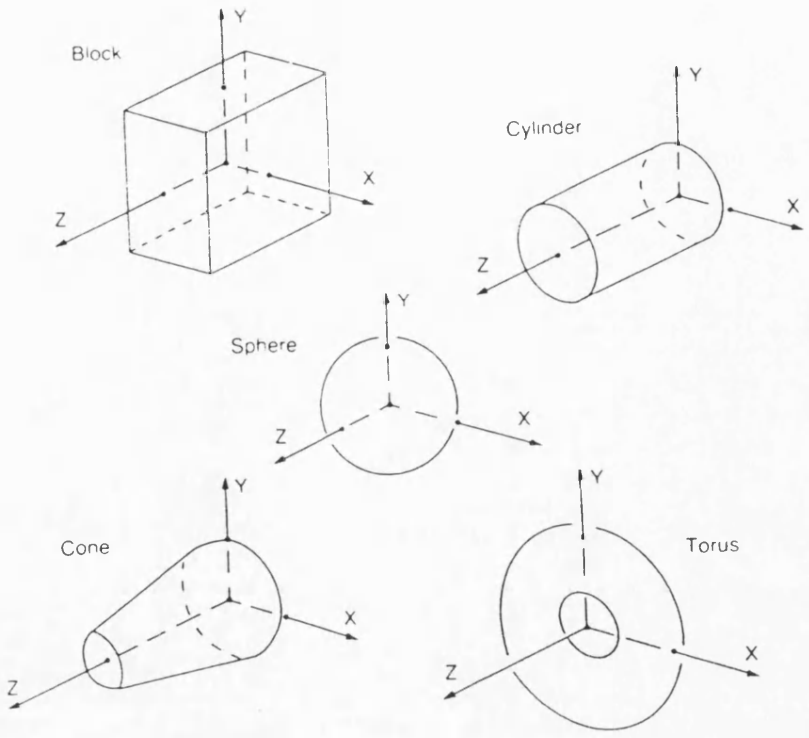


Figure 1.3 (a) Solid primitives

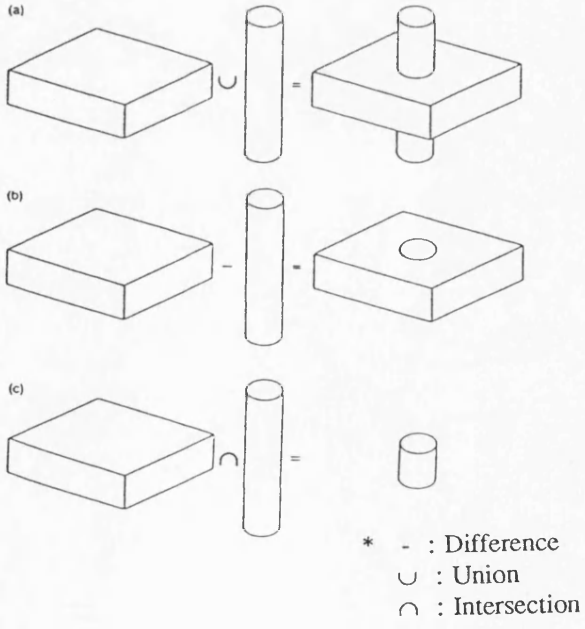


Figure 1.3 (b) Effects of Boolean operations on primitives
(a)Union (b)Difference (c)Intersection

a volume in space or as half spaces. A half space is a surface (usually infinite) which completely divides three-dimensional space into two or more regions: a solid region, a void region and other regions. Any given point is then either in the solid, in the void, or on the dividing surface. Any complicated solid and void subdivision of space (such as the enclosed volumes forming the solid primitives) can be produced as the combination of such half spaces.

Both boundary and set-theoretic representations have their relative advantages and disadvantages [31, 80].

The set-theoretic representation tree allows the calculation of the surface area and the volume of an object unambiguously. Set-theoretic representations construct a correct and precise model from the available library of primitives (as, indeed, do boundary representations). They are suitable for many planning and design problems such as rough-machining and collision checking (but not very suitable for graphics or finish-machining), they are easy to create, store and transmit, and they are guaranteed to be solid (though to the surprise of the user, they may sometimes be null), but they are slow at producing pictures. It is generally possible to convert set-theoretic representations to B-reps.

On the other hand, boundary representations are efficient sources of geometry for computer vision, graphics and NC finish-machining operations. They give more freedom to the designer in building complex models, but the validity of the model is more difficult to maintain. They are bulky, difficult to create and more expensive on memory since they are costly to store and transmit. **Figure 1.5** shows the comparison between B-rep and set-theoretic modelling techniques.

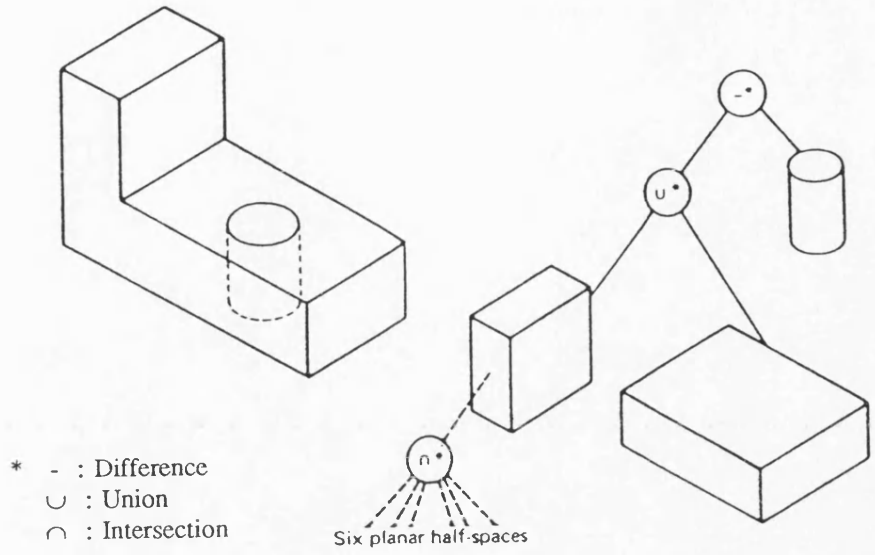


Figure 1.4 A solid object and its binary tree (from Rooney et al. [96])

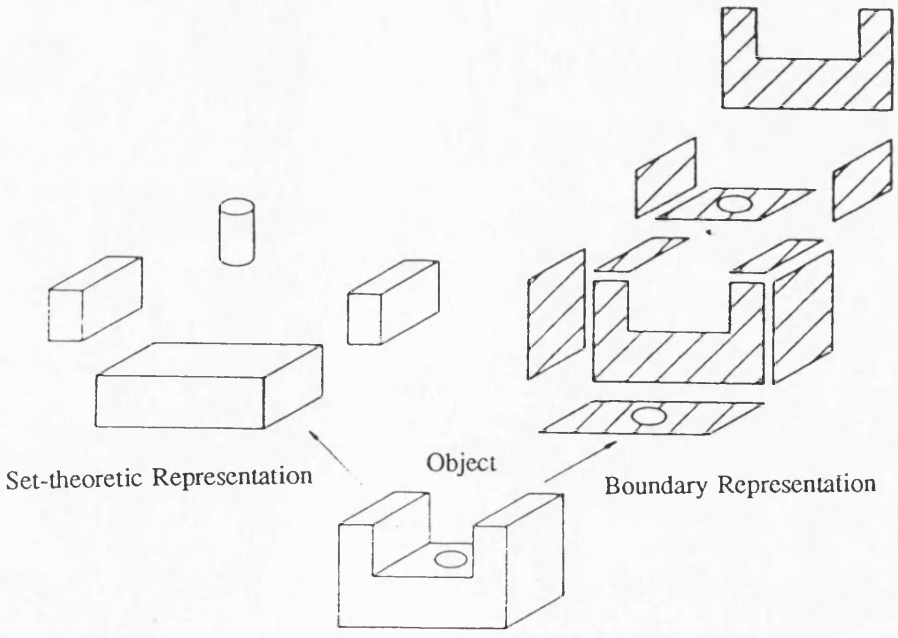


Figure 1.5 Comparison between B-rep and set-theoretic modelling techniques (from Meguid [80])

The solid modeller used in this research was a set-theoretic solid modeller.

1.2 Shape Reconstruction

Shape reconstruction is the problem of deducing a shape from a set of given or calculated data. These data - which are the only information about the solid objects - may be in different forms: they may be in the form of three-dimensional images made of voxels (volume elements) in which the shape is constructed by extracting and following the faces of voxels which are on the boundary of the objects; or they may be a finite number of planar contours which intersect the three-dimensional solid object; or, as in this research, the three-dimensional coordinates of surface points obtained by a laser range finder.

In the applications which use planar contours as data, a solid object is logically divided into slices of parallel cross-sections with finite thickness. Here, the only information about the solid consists of the intersections of its surface with the planes. The sequence of contours is used to construct a piecewise planar approximation to the original object surface or the volume of the object whose boundary is a polyhedron with triangular faces intersecting the cutting planes along the given contours [50, 51, 63, 68].

In some cases the data consist of range data. These are the cases where the only information about the solid is the geometrical position of each surface point, no topological information is available. Neighborhood relationships can be obtained by using the Voronoi tessellation [25] and the shape can be constructed by finding the triangular faces which lie on the boundary of a tetrahedral packing

obtained from the Delaunay triangulation. This was the method used in this research reported here.

Alternatively, the shape is also constructed by fitting of polynomial surface patches such as Coons patches, Cartesian product patches, Bezier and B-spline patches and so on to the range data [31, 49, 81, 82, 112]. Different surface patch types and a survey on surface fitting is given in Besl and Jain [12]. Research done on various types of shape reconstruction will be discussed in the literature survey in Chapter 2.

1.3 Aims of This Project

When an engineering component has been manufactured it must often be checked for defects. Ideally such checking should be done against the original design of the component and any out-of-tolerance differences between the two should be reported.

This thesis will describe a group of algorithms which allow a collection of points on the surface of a manufactured component (such as might be gathered using a coordinate measuring machine) to be matched automatically with a solid model of the component. This would allow any defects on gross features of engineering components to be detected. A diagram of the measurement problem is shown in **Figure 1.6**. Some of these algorithms are extant in the literature on Stochastic Computational Geometry, but have not before been applied to this problem; others are new.

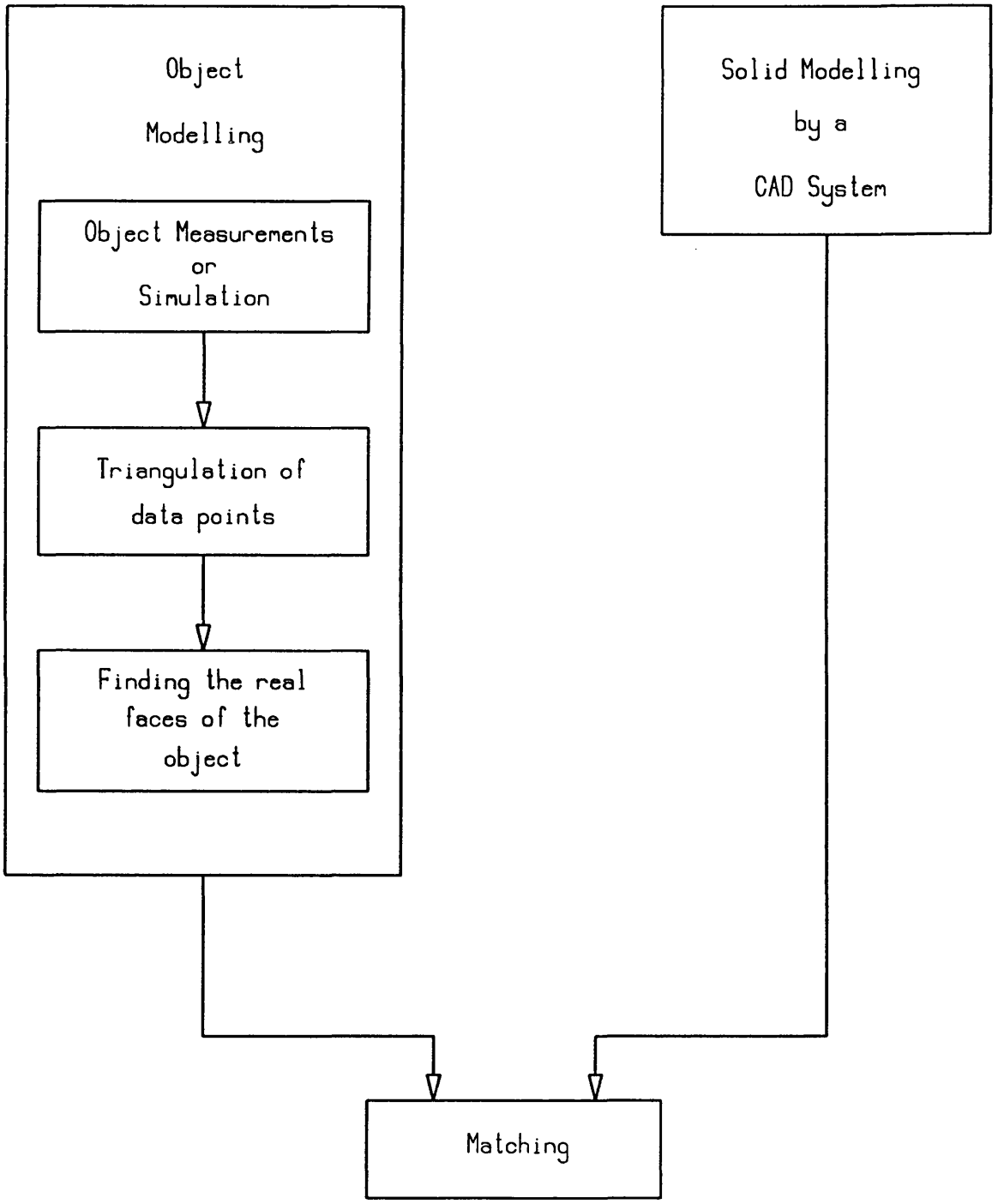


Figure 1.6 The measurement problem

The algorithms have been developed especially to handle the large numbers of surface points that may be gathered from a component using a laser non-contact measuring machine [27]. However, they would also be quite suitable for use with a conventional measuring machine. An application of the algorithms on a simple staircase model is shown in **Figure 1.7**. In figure 1.7 (a) the surface points of the model are shown. These surface points are processed to produce the model shown in figure 1.7 (b). This model is ready to be matched with the master solid model of the same component. Different colours in the figure correspond to different faces to be matched.

The solid modeller used for this research is called DORA. This is a set-theoretic solid modeller developed at Bath by John Woodwark [115]. The algorithms would work just as well with any other set-theoretic or B-rep modeller.

The system described in this thesis was implemented to deal with faceted components and solid models only, but it can be extented to work with curved components.

1.4 Thesis Outline

The first two chapters of this thesis contain background information about solid modelling and a review of the techniques available for triangulation, shape reconstruction and matching. This is followed in Chapter 3 by a review of coordinate measuring machines and some brief information about the laser non-contact measuring machine developed at Bath University. An algorithm which simulates the process of this measuring machine is also introduced in Chapter 3.

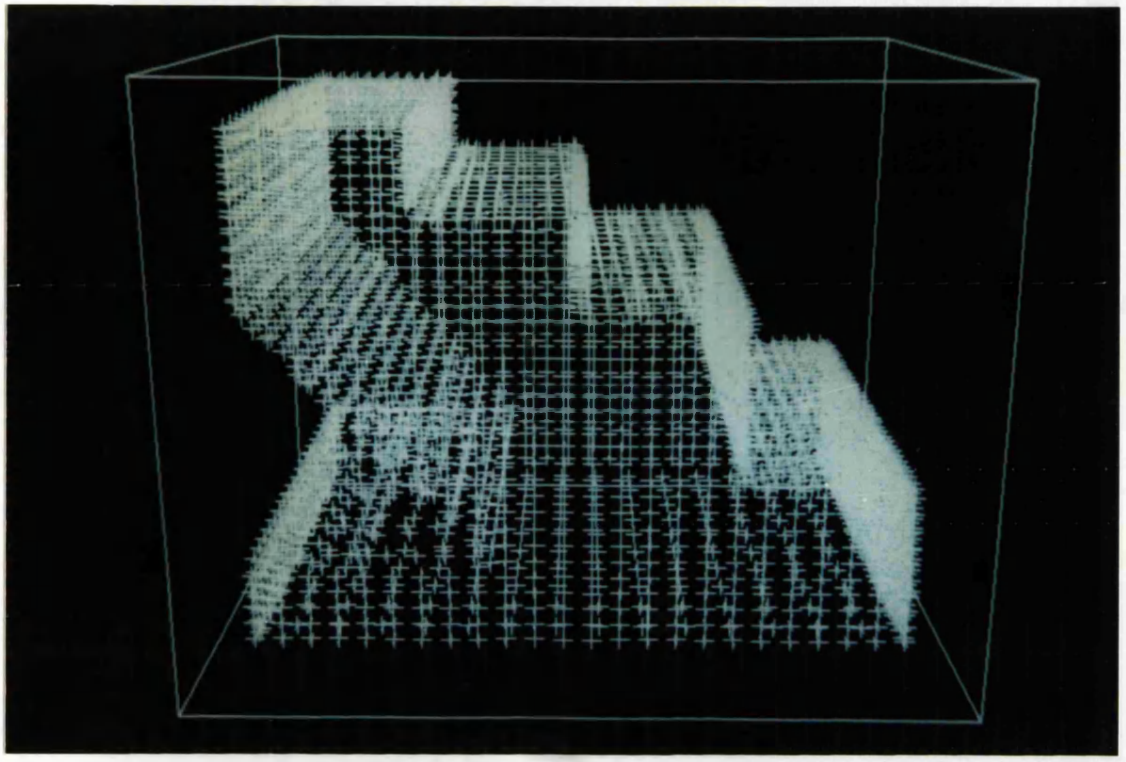


Figure 1.7 (a) 3D surface points

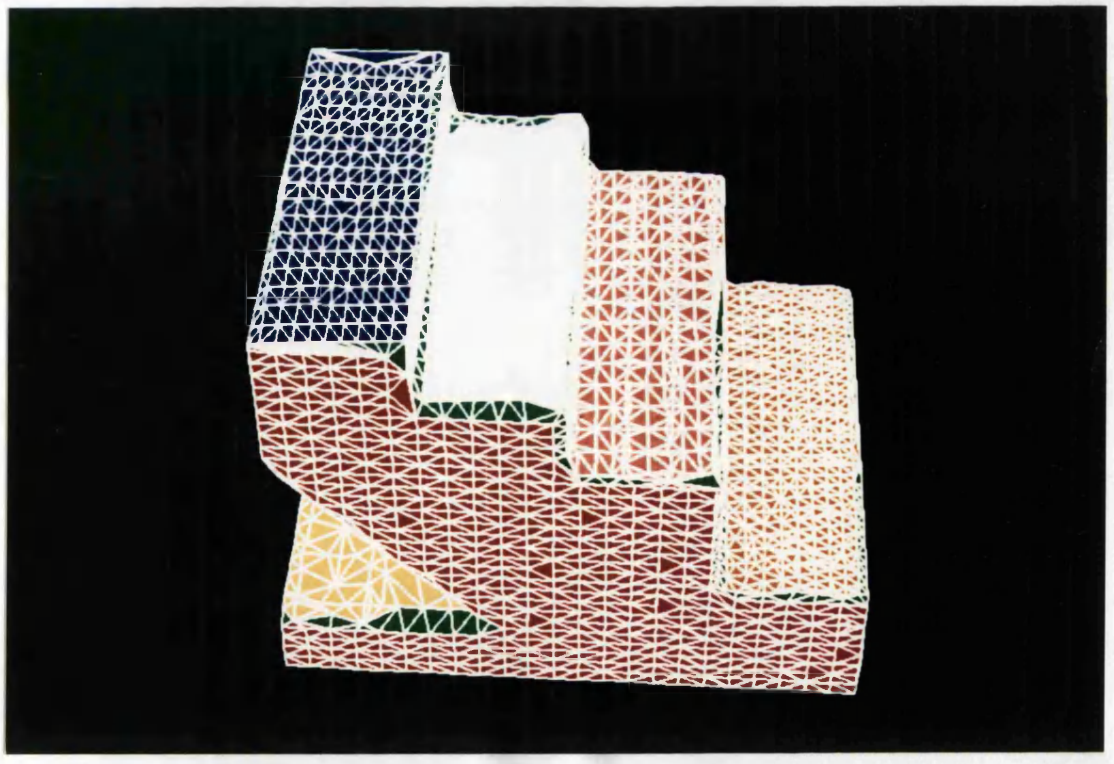


Figure 1.7 (b) The measured component to be matched with its solid model

Starting from Chapter 4 a group of algorithms is described for triangulation, classification of tetrahedra (which are the result of triangulation), clustering and matching. Chapter 4 introduces an efficient multi-dimensional algorithm [25] to construct the tessellation and triangulation to obtain the neighborhood relationship between the gathered surface points. An algorithm introduced in Chapter 5, which classifies the tetrahedra as air or solid, creates a solid model for the measured component and triangulates the surface of this component. This is followed by the description of an another algorithm in Chapter 6 which clusters the surface triangles and gathers them in collections, each collection representing a real facet of the solid model of the measured component. Chapter 7 then describes an algorithm which compares the real faces of the model with the faces of the component and matches them under translation and rotation. The final chapter consists of suggestions for further research and conclusions.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

This chapter is a review of the literature on shape reconstruction, polyhedral approximation, triangulation and matching. Although research on shape reconstruction in scene analysis and image processing are not directly relevant to the research described here, they are worth mentioning to give an idea about different types of shape reconstruction. Most of these surveyed literature use range data (which mostly consist of the three-dimensional surface points) and polyhedral approximation.

2.2 Shape Reconstruction

The problem of reconstructing a shape is that of finding an interpolation over a set of points lying on its boundary. A lot of research has been done on the problem of fitting a function to data given at a set of points scattered throughout a domain in the plane but this will not be reviewed in this survey. Instead, the relatively small amount of literature concerned specifically with shape reconstruction using a set of points scattered all over the closed surface bounding a solid object or a set of finite planar cross-sections shall be considered (this will be explained below). Such problems arise in robotics, shape analysis, computer graphics and image processing. Good surveys on shape reconstruction in image processing are given in Besl and Jain [12], Ahuja et al. [2] and Henry [61]. Some of the techniques used to build a shape in image processing are quite different from the techniques mentioned in this research, but because of some similarities (such as the same sort of input - range data - and planar approximation), they are worth mentioning.

Researchers who use range data as their input reconstruct shape in many different ways. Dane and Bajcsy [37] proposed an object-centered three-dimensional model builder which uses 3D surface point information obtained from many views. Their technique forms subgroups of data points (for each view according to the information about their location and orientation) and determines surface primitives to represent them. It fits planar or quadratic surface primitives onto them via a least squares technique. It then transforms the surface primitives from a local coordinate system to a common global coordinate system by using the known transformations, and identifies the identical surfaces and builds a surface

model. Clustering and region growing algorithms are different from the author's but their subgroups are similar to his face clusters (see Chapter 6). The data points that they used in their technique are drawn from a limited area of the local surface area because of the conservative nature of region growing process. This fact makes the accurate estimation of the surface parameters more difficult. When fitting surface primitives onto subgroups via least squares technique no method has been mentioned to eliminate points with gross inaccuracies (which lie well away from the surface to be fitted due to experimental inaccuracies and seriously affect the accuracy to fit).

In another approach Vemuri and Aggarwal [107] described an algorithm for the reconstruction of three-dimensional objects using range data obtained by a laser range finder from a single view. Their algorithm fits surface patches to square neighbourhoods by computing the standard deviation of the Euclidean distance between consecutive points. If the standard deviation is less than a threshold, a surface patch is created to fit the square using splines. This technique avoids the fitting of surfaces to squares of high deviation, such as at edge discontinuities. However their technique suffers from an inability always to be able to represent re-entrant surfaces and undercuts as any method based on collection of 2D height grids. The data acquisition system they use works on the principle of light sheet triangulation [61].

Henderson [59] developed a technique of extracting planar faces from range data obtained by a laser range finder. His technique depends on a sequential region growing algorithm called the *Three-point Seed Method* [60]. In this method

three-dimensional surface points - which are obtained from various sides of an object - are first stored randomly in a list with no topological connectivity information. Planar convex faces are then determined by sequentially choosing three very close non-collinear points and investigating the set of points lying in the plane of these three points. Provided the points which are found that lie close to the plane satisfy narrowness conditions, they are labelled as a plane and removed from the range data. This procedure is repeated until the range data is exhausted or no more planes can be found. The shape is reconstructed from the union of these planar faces. This technique is a useful technique since it is not restricted to single view range data images and is robust even for noisy data but problems were observed in detecting the edges.

Potmesil [90] proposed a method for constructing surface models of arbitrarily-shaped solid objects by matching three-dimensional surface segments onto the range data obtained by a triangulation-based range finder. Range data for an object's surface are fitted using a sheet of parametric bicubic rectangular surface patches which are recursively merged into a quadtree hierarchical structure where each object is represented by a tree of surfaces. Ray-casting technique is used to obtain the surface information as in this thesis, but his technique evaluates not only surface point information but other surface informations such as the surface-normal vector and the surface-normal curvature at the intersection points between the ray and the surface description. This information is used in matching and merging algorithms (this information cannot, in general, be obtained in real measuring, so this simulation is not very realistic). Surface segments of the object are

transformed into a common 3D space by a matching procedure and then their overlapping sections are eliminated by a merging procedure. This match-and-merge process is iteratively repeated until a complete model of the object is generated. **Figure 2.1** shows the last two stages of Potmesil's method. Thirty-six views are used to define the object. First, 18 three-dimensional segments are reconstructed from the views, which are then merged into six new segments. These six segments are matched into a single model and the shape is reconstructed.

Boissonnat [19], Boissonnat and Faugeras [21] and Faugeras et al. [47] developed an algorithm for building a polyhedral approximation of three-dimensional surface points obtained from triangulation-based laser range finder. In another two papers Boissonnat [18, 20] proposed to use the Delaunay triangulation to construct shape by triangulating the surface of the object. He used the same triangulation-based laser range finder to obtain the 3D surface points. The polyhedral approximation technique he used will be explained in section 2.3 and Boissonnat's triangulation technique (which is similar to the one described in Chapter 3) in section 2.3.1.

Many more examples which use range data as input to construct shape can be added to these. On the other hand, some of the shape reconstruction techniques use planar contours as data. These techniques construct surface contours by intersecting a 3D solid with a finite number of specified parallel planes and then connect contours on consecutive slices with triangles. These are the methods where the only information about the solid consists of the intersections of its surface with planes. Here, the problem is not a detection problem but an interpolation problem.

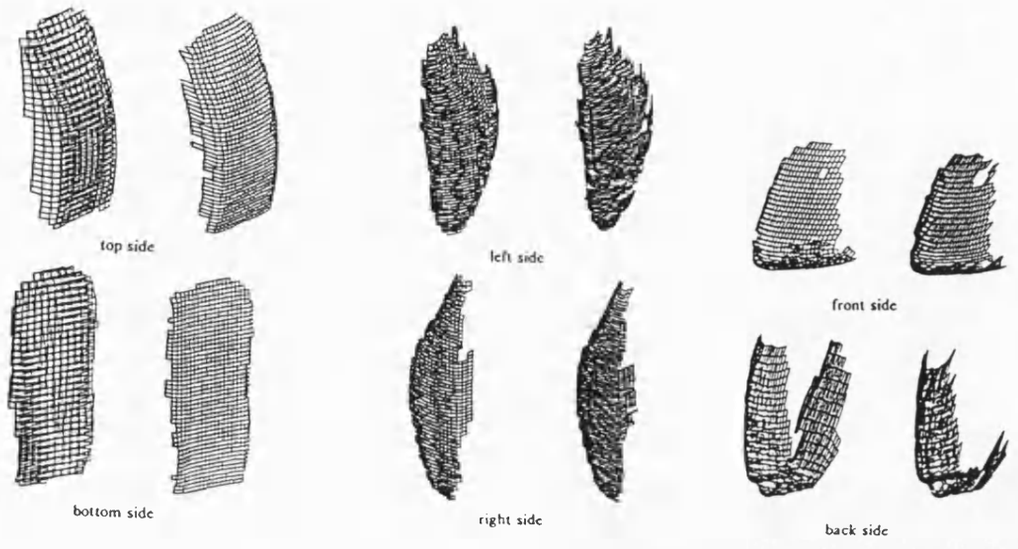


Figure 2.1 (a) 18 surface segments to be merged into six new segments (from Potmesil [90])

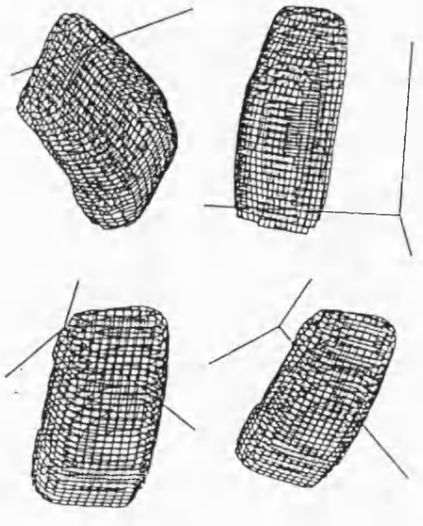


Figure 2.1 (b) Surface segments to be matched into a single model (from Potmesil [90])

Kepel [68]; and Fuchs, Kedem and Uselton [50] reduce this interpolation problem to constructing a sequence of surfaces, one between each pair of adjacent contours. These surfaces are constructed from elementary triangular tiles, each defined between two consecutive points on the same contour and a single point on an adjacent contour. **Figure 2.2** shows a set of contours on a given shape and the individual triangular tiles defined over these contours.

Ganapathy and Dennehy [51] described an heuristic method for triangulating the three-dimensional surface formed by spanning a set of planar contours. Their method (which imposes no restriction on the orientation of contours) is claimed to be superior to Fuchs's or Kepel's since Kepel's method is dependent not only on the number of points approximating a contour but on the shape of that contour as well, while Fuchs's method requires a large number of steps for triangulation and is not suitable for applications where speed is of primary importance.

Boissonnat [17] worked on a similar problem as well. Unlike Fuchs, Kedem and Uselton's method which constructs the surface of the object, his method constructs a volume whose boundary is a polyhedron with triangular faces intersecting the cutting planes along the given contours. If there are L cross-sections, his method constructs a sequence of L-1 partial shapes instead of constructing a shape over these cross-sections, where each of the partial shapes connects two cross-sections lying on adjacent planes. He then computes the L-1 Delaunay triangulations of the vertices of the L-1 couples of adjacent cross-sections and reconstructs the shape and obtains the volume, slice by slice, by pruning these L-1 Delaunay triangulations (since the boundary of the object can be obtain without the need of

computing the Delaunay triangulation, the reason for this computation is not very clear). The surface of the object can be produced by looking at the surface of the obtained volume. **Figure 2.3** shows a set of contours and the partial surfaces. For clarity, only boundary faces of the non-eliminated tetrahedra (which are not lying in a cutting plane) are shown.

Other shape reconstruction techniques have also been introduced in the literature. Some of these methods use intensity images as input: Baker [6] proposed a shape building method which uses many intensity images taken from different known rotated views; Shapira and Freeman [100] described an algorithm for constructing three-dimensional objects bounded by planar or quadratic surfaces from a set of photographs of scene taken from different views; Bocquet and Tichkiewitch [16] proposed a method which uses digitised standard mechanical drawings from three orthogonal views as input to reconstruct a shape and so on. These techniques use different sort of data as input than the data used in this thesis, and therefore will not be described in this section. In some other techniques such as Schmitt, Barsky and Du's [98] (which is based on an adaptive subdivision approach) shape is reconstructed by fitting bicubic Bernstein-Bezier patches meeting with G^1 geometric continuity onto the three-dimensional data; or in Little's [74] technique convex polyhedral object models are reconstructed by using their corresponding *extended Gaussian images* (EGI), as is Ikeuchi's [64], which also reconstructs the original shape of a convex polyhedron from its EGI. A volume-based approach (in which the shape is obtained by the Delaunay triangulation - see section 4.2) was used to reconstruct a shape in this thesis. Thus neither of these works is directly

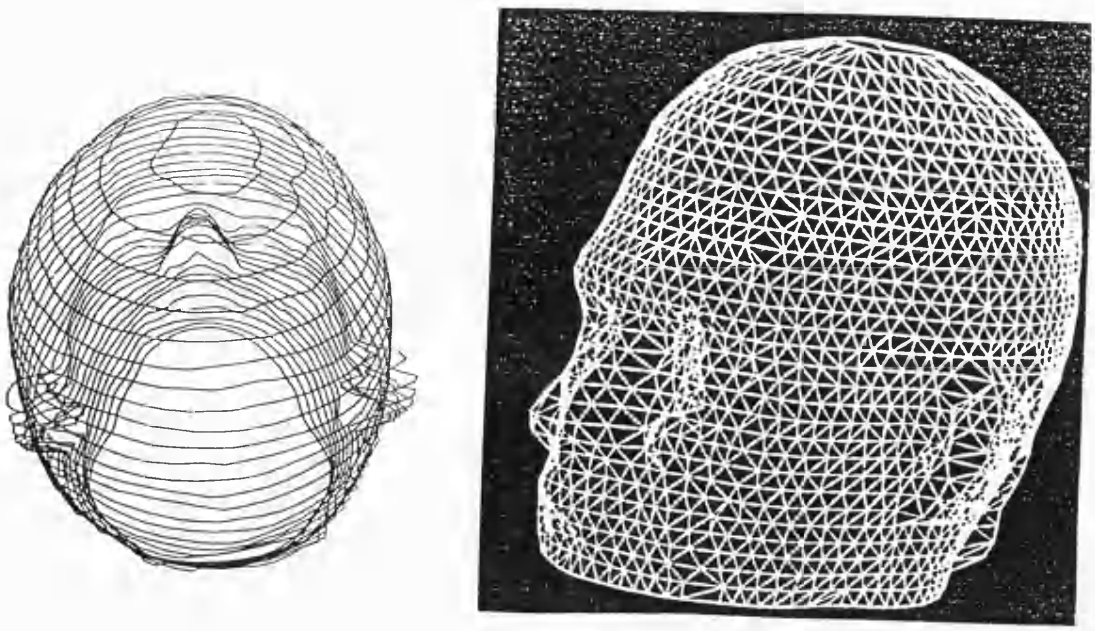


Figure 2.2 Contours and triangular tiles (from Fuchs et al. [50])

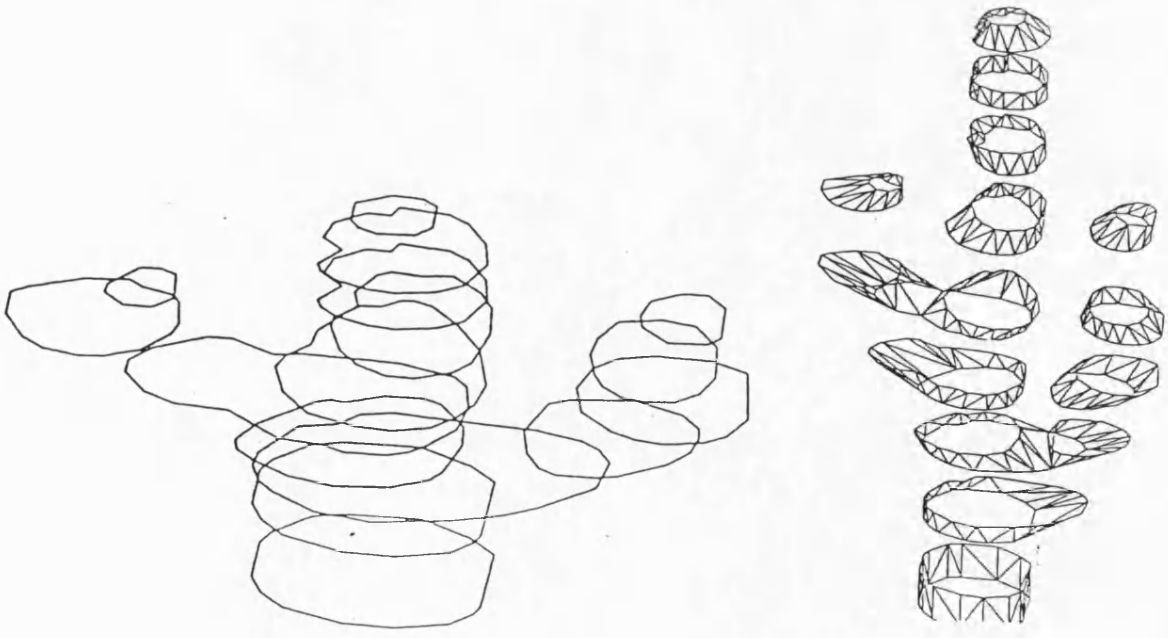


Figure 2.3 Contours and partial surfaces (from Boissonnat [17])

relevant. However, extended Gaussian images were used in matching the reconstructed shape onto the solid model primitives. Detailed information will be given about extented Gaussian images in Chapter 7.

In the applications of computer graphics, voxels (volume elements) have been used as data to construct shape as well as parallel contours. The literature related to these applications and a list of references about the applications of shape reconstruction in medicine is given by Lorensen and Cline [77].

2.3 Polyhedral Approximation

In many applications of pattern recognition, robotics, and computer vision, objects are initially represented by the coordinates of points lying on their boundaries. Since no relation between these points is known, this is a poor representation of the object. To make this representation complete and thus allow the shape to be analysed, the three-dimensional surface points should be approximated by three-dimensional surfaces. This is a *polyhedral approximation* of the surface of the object. Such approximations have been used to solve many problems such as: definition of the shape of the object; controlling the automatic machining of surfaces [33, 41]; smooth interpolation between the points [32, 48, 88]; or, on the contrary, reduction of the points without damaging greatly the actual shape of the object [47]; calculations of geometric properties such as volume, surface area, axes of inertia; definition of the surface normals at the points; extraction of elementary shapes and so on. Literature surveys on the problem of creating a triangulation over the object's surface will be covered in the next section.

Polyhedral approximations have been used widely in shape reconstruction and shape analysis. Criteria which make polyhedral approximations useful as object representations are given in Henderson [58]. Ganapathy and Dennehy [51] used polyhedral approximations in their triangulation algorithm, and Henderson and Bhanu [60] created a three-dimensional model in terms of planar faces approximated by polygons. These two methods have been described in the previous section.

Faugeras and Ponce [45] proposed a hierarchical structure for describing 3D objects. They use a recursive polyhedral approximation algorithm to construct a binary tree structure called a *prism tree*. In another paper, Faugeras and Hebert [46] segmented the surface of the object into planar regions by polyhedral approximation in their matching procedure.

Boissonnat [19], Boissonnat and Faugeras [21] and Faugeras et al. [47] proposed an efficient algorithm for building a polyhedral approximation of a set of 3D surface points. As mentioned earlier, they obtained the surface points by using a laser range finder which is similar to the one developed at Bath University [61]. Their algorithm is presented as a generalization of an existing algorithm for polygonal approximation of a two-dimensional curve. The basic approach is a graph-guided, divide-and-conquer procedure and the steps of the algorithm are given in [19],[21] and [47]. The net result of the algorithm is an approximation of the original surface by a polyhedron whose faces are triangles. Results of two fairly complicated objects are shown in **Figure 2.4**. In spite of the difference in processing algorithms, the input and output for this approach are quite similar to

Henderson's [59].

Faugeras [44] proposed a method for producing a complete polyhedral representation of objects based on geometric matching between primitive surfaces in order to construct the models in his 3D object recognition algorithm. This algorithm will be explained in section 2.4. **Figure 2.5** shows the object he used for modelling (which is an automobile part) and the surface triangulation of this object.

Several alternative approaches to polyhedral approximation are also introduced [58], including region growing, local feature clustering and divide-and-conquer techniques; though most of the time these techniques work together with polyhedral approximations like Henderson and Bhanu's [60] sequential region growing algorithm. Region growing techniques fit surface patches to small seed areas which consist of a small group of points on the surface, and then expand the patches by adding and testing the neighbouring areas (or points) to see whether or not they satisfy the necessary conditions to join the expanding region. This is done until the equation of the patch no longer provides a good fit to the points added. This occurs when the patches reach the boundary of a different surface.

Local feature clustering techniques, different from region growing techniques, measure some feature at each point of the data (e.g., the surface normal at that point), and apply standard clustering techniques to the resulting vectors. Application of standard clustering techniques to local features of the range data has a tremendous potential. Clustering methods are required to be fast, to take neighbour constraints into account, and to allow the use of supplementary information. A good review of various methods and applications is given by Diday et al. [39].

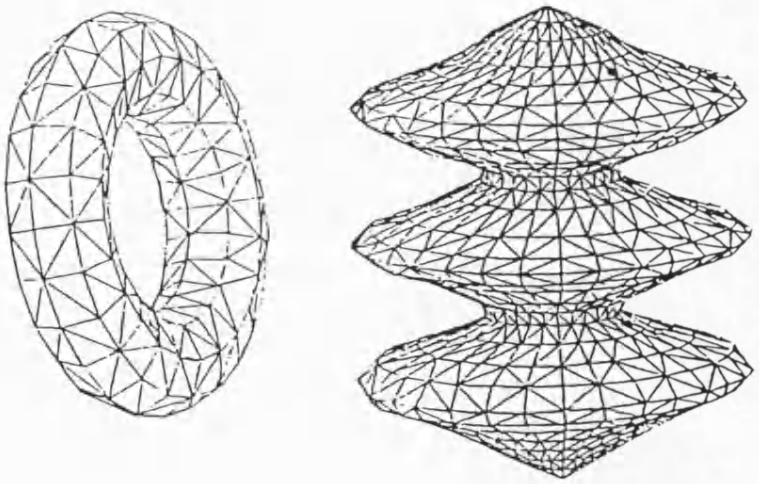


Figure 2.4 Objects reconstructed by polyhedral approximation (from Boissonnat [19])

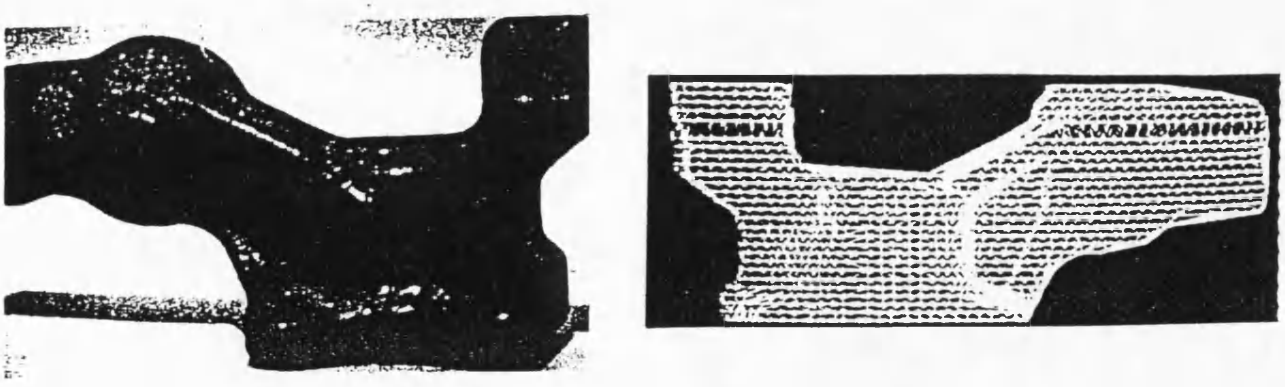


Figure 2.5 An automobile part and its 3D triangulation (from Faugeras [44])

Finally, the divide-and-conquer technique applies a recursive algorithm to process the data in two halves. This is a fast method. Boissonnat and Faugeras's [21] algorithm is a graph guided divide-and-conquer approach to polyhedral approximation of a set of 3D surface points as mentioned earlier. Divide-and-conquer techniques have been applied to the polyhedral approximation problems in multidimensional spaces successfully [9, 10, 92].

2.3.1 Triangulation

The problem of approximating the surface spanning a given set of 3D points as a polyhedron of triangular faces is that of creating a *triangulation* of the surface.

Triangulations have received a lot of attention in the past and have many areas of application [73]. They have been widely used in finite elements mesh generation [22, 23, 34, 97, 116], in computer graphics and in computer vision. In some applications such as Wordenweber's [117], a triangulation technique has been used to produce pictures in geometric modelling.

In some others, it is used to construct smooth surfaces from scattered data in 3D [48, 88]. Choong [33] proposed a novel heuristic triangulation technique to model a measured shape (the measurements being taken by a 3-axis measuring machine) which he used in an automated polishing process of die cast components by a robot. The three-dimensional geometry required was defined in the form of $z = f(x,y)$ where z is represented by an array of height ordinates of the surface for discrete values of x and y. The technique is based on the creation of two-dimensional mesh of equilateral triangles on a plan view of the component. It

distorts this mesh in the boundary region so that adjacent nodes are forced to coincide with points defining the boundary curve. .Nodes lying outside the boundary are rejected. Heights for each node are calculated by interpolating the z values from the original height ordinate array. The laying of the initial triangulation lattice is shown in **Figure 2.6**. As is seen from the isometric view in figure 2.6, the triangle elements in curved areas are highly distorted. The surface is then subjected to a triangulation process which attempts to create a structure of equilateral triangles. The result of triangulation is shown in **Figure 2.7**. This technique also suffers from same problem that Vemuri and Aggarwal's technique suffers and it only works with single-valued surfaces.

Choi et al. [32], unlike Choong's technique [33], proposed a technique which is concerned with the triangulation of 3D points (which are scattered on a 3D surface of complex shape) to construct smooth surfaces. Their algorithm, which is based on Thiessen/Voronoi polygonization [94], first triangulates 3D points and obtains a proper triangular grid. It then improves the grid by applying smooth triangular interpolants onto the points. To improve the smoothness of the surface, a smoothness criterion and a grid improvement algorithm (different from the Lawson's max-min angle criterion [70] and the circle criterion [70] to obtain an optimal triangulation grid) have been introduced by them. It is reported that his triangulation algorithm may cause some numerical problems when the 3D points on a closed surface are triangulated (i.e. when the apex angle of a convex cone exceeded 89.92 degrees).

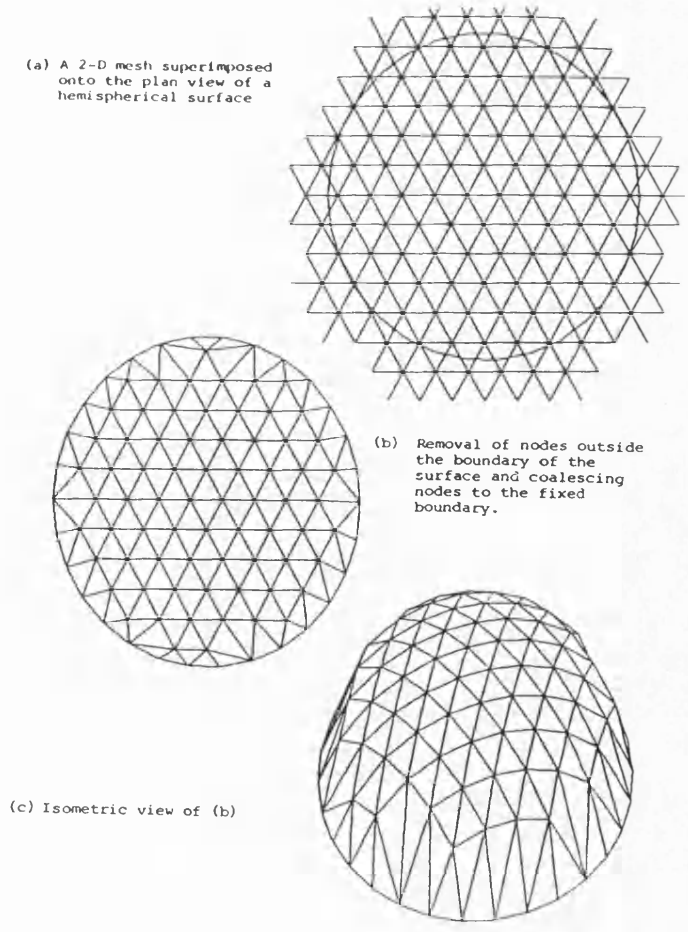


Figure 2.6 Laying of the initial triangulation (from Choong [33])

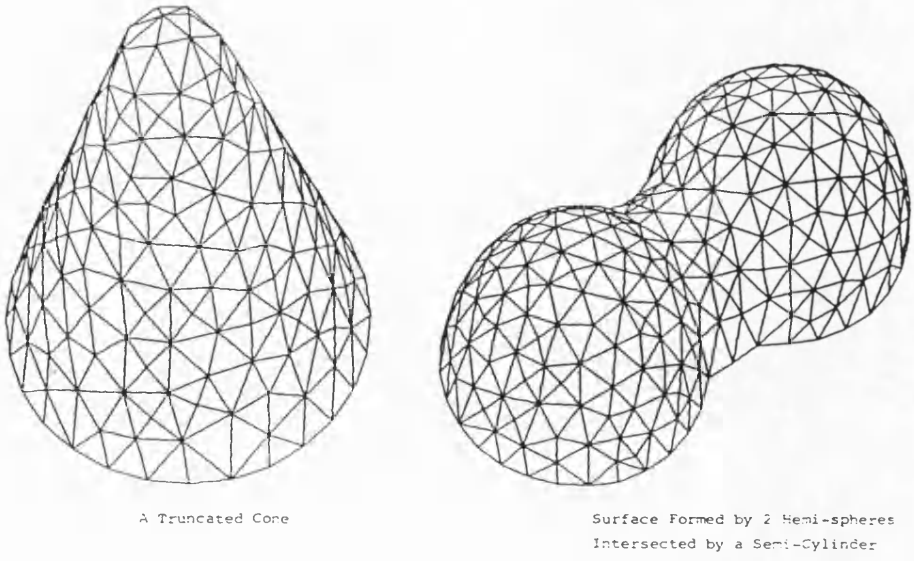


Figure 2.7 The results of triangulation (from Choong [33])

Oxley [84] used triangulation in surface fitting. His algorithm recursively splits an area up into one or two triangles and other areas in an initial triangulation stage. After applying this recursive process to the whole surface, the areas which are not triangles are stored on a stack. The algorithm proceeds by repeatedly removing an area from the stack, splitting it up and adding areas to the stack until the stack is exhausted.

Among the possible solutions of finding minimal structures (i.e. those with the fewest edges and which preserve the topology of the surface) to represent or to construct a shape, O'Rourke [83] suggested minimizing the surface area when triangulating a set of 3D points. His algorithm starts with the convex hull of a given set and shrinks this hull onto the internal points. In order to do this, it chooses a point internal to the hull, systematically modifies the current polyhedron to include this point, makes local adjustments to the polyhedron in the neighbourhood of the modified region and repeats this for all internal points until the internal points are exhausted. This algorithm is not guaranteed to reduce the residual error to get closer to the minimal polyhedron and has been claimed to yield strange results (see [20]).

Boissonnat and Faugeras's graph guided algorithm [21] - as explained earlier - also approximates the surface with triangles. It uses an underlying graph to structure the data for association with approximating triangles. Whenever a triangle does not approximate the data associated with it very well, graph methods are used to divide up the data for a better approximation.

One of the most well known triangulations is the Delaunay triangulation [25, 53, 106, 111]. The Delaunay triangulation is the geometric dual of the Voronoi tessellation [53, 106, 111] and is obtained by linking the points whose Voronoi polyhedra are adjacent across a common face. To analyse a shape and to compute some of its geometrical properties neighbourhood relationship information between its points needs to be obtained. The Delaunay triangulation obtains this information and produces a good polyhedral approximation of the shape. Delaunay triangulations and Voronoi tessellations will be explained in detail in Chapter 4. A good survey on defining the neighbourhood relationship of a point and applications of Voronoi tessellation can be found in Ahuja [1].

Delaunay triangulations have many different fields of application. Some of these applications are mentioned in [25], [72] and [111]. They have also been used by many finite elements mesh generators [30, 36, 78]. In one of his papers Boissonnat [18] proposed to use a Delaunay triangulation to represent a two or three dimensional shape which is defined by a finite number of surface points and constructed the shape by pruning Delaunay triangles between adjacent cross-sections in the other paper [17]. To generate the triangulation [18, 20], he used the efficient multi-dimensional algorithm [25] which was also used in this research. Since the Delaunay triangulation fills the interior of the convex hull of the points with tetrahedra, it is the volumetric representation of the object. The object is represented by a set of tetrahedra and the boundary of the set (which is the convex hull) is a polyhedral approximation of the surface of the object (if all the points are in the convex hull). If the convex hull does not contain all the points, some

tetrahedra need to be eliminated until all the points are on the boundary of the polyhedral shape.

Boissonnat [20] introduced four rules to eliminate redundant tetrahedra until all the points are on the boundary of a polyhedral shape. He eliminates the tetrahedra one after another by checking their associated value. This value (which is used to sort the tetrahedra) is defined as:

If A_I is the area of faces of the tetrahedron interior to the boundary, A_B is the area of faces of the tetrahedron on the boundary and A is the area of faces of the tetrahedron, then the value associated to each tetrahedron is

$$s = \frac{(\Sigma A_I - \Sigma A_B)}{\Sigma A}$$

He eliminates the tetrahedra with the smallest values first. He suggests that this criterion tries to minimise the modifications of the boundary by eliminating the less regular tetrahedra. This method, which tends to eliminate sharp projections (which may really be there) is different from the methods described in this thesis. The result of his elimination is shown in **Figure 2.8.** The triangulation algorithm used in this thesis will be explained in Chapter 4 and the method of eliminating the redundant tetrahedra in Chapter 5.

2.4 Matching

After reconstructing the shape the next problem to be solved is that of *matching* the reconstructed shape to a geometric model. Different matching techniques have been introduced in the literature. Most of these techniques have been used exten-

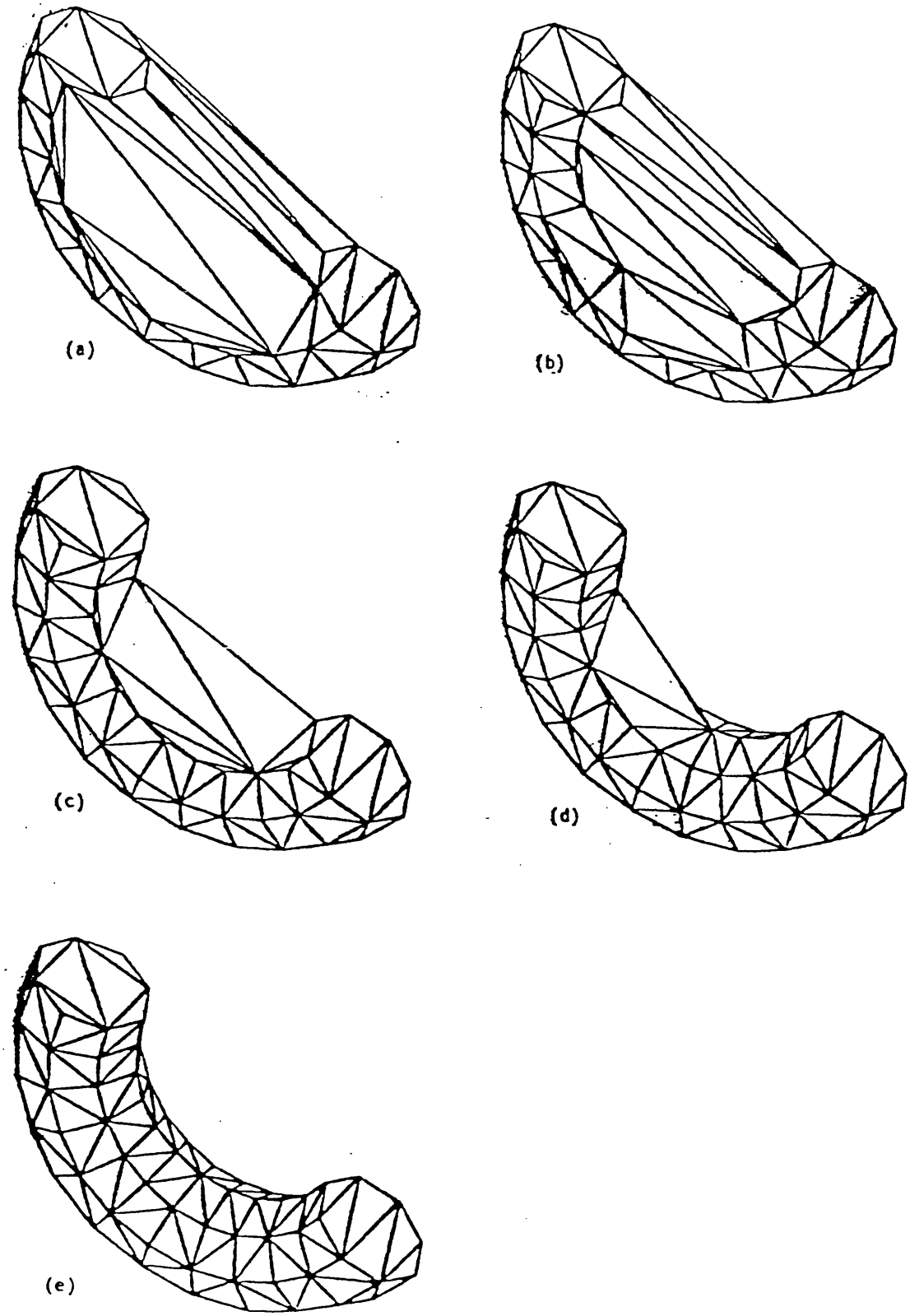


Figure 2.8 Elimination of redundant tetrahedra (from Boissonat [20])

sively in scene analysis for matching model and scene descriptions and in pattern recognition. Good reviews on some of the main techniques such as relaxation, Hough transform, Fourier descriptors and moments and so on are given in [44] and in [86]. It is not the intention here to give a complete survey of these techniques. Instead, a few of them will be mentioned to draw attention to the differences between these techniques and the technique used in this research.

Bhanu and Faugeras [14] proposed a technique for shape matching in 2D which they based on a relaxation scheme called stochastic labelling. Bhanu [13] then extented this technique to be used in 3D. His matching technique in three-dimension uses planar faces as primitives as explained earlier, and matches an unknown view with the structural 3D model by using a stochastic face labelling technique. The face features of area, perimeter, length of maximum, minimum and average radius vectors from the centroid of a face, number of vertices in the polygonal approximation of the boundary of a face, angle between the maximum and the minimum radius vectors and ratio of area/perimeter^2 of a face are used as well as a feature-weighting vector to compute the initial face-labelling probabilities. The compatibility of a face of an unknown view with a face in the model is obtained by finding transformations, applying them and computing the error in feature values. The compatibility functions used in the first and second stage iteration use weighted quantities of: the distance between neighbouring face centroids, the ratio of the areas of the neighbouring faces, the difference in face orientations, and rotation angles for the maximum intersection area of coplanar faces. Transformations are computed at the end of second iteration stage. This method handles arbitrary

view-points. However, it relies too heavily on the consistency of the output from the face-finding algorithm. Perhaps this is justified, but no evidence is given. All face-adjacency information is not utilised. A block diagram of this matching algorithm is given in **Figure 2.9.**

Davis [38] investigated representing a 2D shape as a spring-loaded template and searching for matches to this template with a relaxation-like graph processes. He used a hierarchical matching process: the low level of his hierarchy is based on the association graph where the nodes represent simple angle matches and the edges represent ordering and distance constraints; and the higher level is based on the line graph of the association graph where the nodes represent more complicated structures (such as pairs of simple matches) and edges represent more complex constraints based on comparing pairs of similarity transformations. An experimental study of his matching algorithm which uses the coastlines of several islands as the input shapes is given in [38].

Faugeras [44], Faugeras and Hebert [46] and the group in INRIA introduced a 3D object recognition and positioning algorithm which is based on geometrical matching between primitive surfaces. Their algorithm uses a segmentation of the surfaces to be identified into geometrical primitives and matches the scene primitive list to the model primitive list using an approach that minimises the mean square-error criterion over all plane-to-plane transformation matches. The translation and rotation matching are separated into two independent least-squares problems. The translation permits a standard linear least-squares solution but the rotation does not. Since classical least-squares methods cannot be directly applied to

the rotation, *quaternions* are used to convert the nonlinear 3D rotation problem into four-dimensional eigenvalue problem which can be solved directly. The quantitative measure of goodness of the match between the data and the model is calculated by combining the rotation and translation matching errors.

This technique is similar to the matching technique mentioned in this thesis in the sense that they both use least-squares technique to find the best transformation. In both techniques the estimation of the translation and of the rotation are independent. However, the major difference between the two techniques is the order of transformation. INRIA's matching procedure does the rotation first. It rotates the model planes until they become as parallel as possible to the scene planes and then translates them to minimise the sum of the differences of the distances to the origin. The technique described in this research, on the contrary, does the translation first. It translates the configurations to have centroids (or in other words *plane centres* which are minimum distance away from the faces) at the origin and does the rotation next to minimise the residual sum of squares between the planes.

Horn [62], Horn and Ikeuchi [63], Ikeuchi [64] used extended Gaussian images (EGI) to match the object models to the scene models. They compute the prototype surface-normal-vector orientation histograms for various shapes by using three-dimensional object models and compare the scene object histogram and prototype object histogram to compute the best match. The best match determines the orientation of the object in space. This approach is quite similar to the INRIA approach and to the approach used in this research since they all use surface-normal matching procedure. **Figure 2.10** shows the surface normals of some

samples and the extended Gaussian image of the normals.

Ballard and Sabbah [8] used the generalised Hough transformations to compute the 3D transformations (translation, rotation and scaling) between a given view and a given object; Zahn and Roskies [118] used Fourier shape descriptors; Dudani et al. [40] used moment invariants for the same purpose. Since a new method of matching was devised for this work, these literature and the literature about the other types of shape matching will not be reviewed in this thesis.

For the purpose of matching, the *Procrustean technique* [29] is introduced by the author. This technique and a matching algorithm using this technique will be described in detail in Chapter 7.

2.5 Concluding Remarks

In this chapter different techniques for shape reconstruction, polyhedral approximation, triangulation and matching have been covered. Although some of these techniques are quite different from the techniques used in this research, they have been described precisely to give some idea about those differences.

As mentioned in the first chapter, the aim of this project was to form a model from the measured data and then match this model to the master solid model generated by a CAD system. Two problems arise here: the two differing descriptions of the same (or nearly same - the purpose of inspection is to find manufacturing errors) component and the different coordinate systems to which the measured component and the master solid model belong. The algorithms which deal with

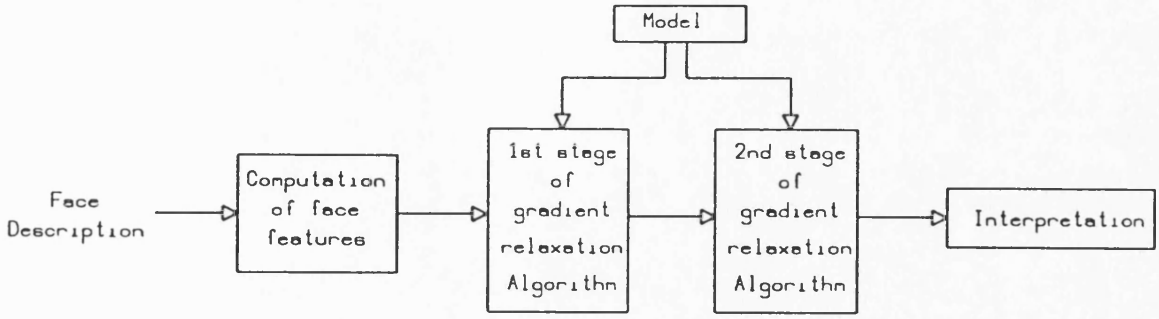
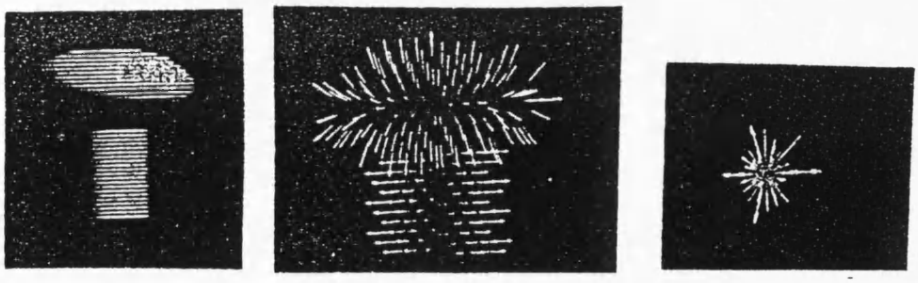


Figure 2.9 Block diagram of 3D shape-matching algorithm (from Bhanu [13])



A sphere on a cylinder Surface normals Extented Gaussian image

Figure 2.10 Surface normals and their extended Gaussian images

these problems and match the objects under translation and rotation by using the Procrustean algorithm will be explained in the next chapters.

CHAPTER 3

GATHERING THE DATA

3.1 Introduction

In a number of applications objects are initially described by a set of coordinates (x,y,z) of surface points scattered all over the object. Such data are in general irregularly located in 3D space and can be generated by any coordinate measuring machine.

In this chapter a laser coordinate measuring machine is described. This is a non-contact type measuring machine, and was developed at Bath University [61]. In the latter sections, an algorithm which simulates the process of this measuring machine is introduced.

3.2 Co-ordinate Measuring Machines

The introduction of the new range machine tools - transfer lines, NC and CNC - has drawn attention to the need for rapid measurements of components. Since extremely complex components have started to be produced on machining centres much faster than they can be inspected, it has become obvious that conventional inspection methods are both wasteful of time and effort. So, in order to improve the accuracy, ease and speed of measurements, to reduce the time taken to record and analyse the results, and to improve the communication of measurement results, co-ordinate measuring machines (CMM) have been introduced [52, 91].

In modern production, time is critical. For this reason, improving floor-to-floor time is one of the important tasks to be achieved in modern tooling technology. Since one of the time-consuming operations which prevents the movement of components after machining is inspection, inspection time should be reduced. The principal benefit of computer-controlled co-ordinate measuring machines is a reduction in inspection time.

Past inspection methods relied heavily upon manual methods, with a human operator sampling the production line in order to make time consuming measurements. Automated inspection techniques have significantly improved quality control by providing an inspection capability which is far more reliable than human methods. During inspection, a large amount of numeric data is generated. These data can be processed and programmed so that component errors can be displayed visually or printed.

Ideally, the inspection of a product should occur whilst the tool is cutting (any errors found during the machining can then be corrected instantly). This is called *adaptive* control or *in-process* control. In-process control is performed in real-time and is significant in providing feedback for applications such as the adaptive control of machine tools, the monitoring of toolwear and machine malfunctioning, and the control of assembly processes (i.e. in the adaptive control of machine tools the measurement results at the cutting point are fed back to the machine controls and automatic adjustments of the machine speed or tool tip position are made). An application of in-process control is mentioned by Choong [33]. This type of control is quite difficult in some methods of production.

However, this is not generally the requirement; inspection may be done with the machine stopped and the cutting tool withdrawn. This is called *off-line* inspection and may take place in-between the manufacturing steps or during the final inspection. Errors when found are corrected. The inspection technique described in this thesis is off-line.

The intention here is not to give detailed information about the co-ordinate measuring machine or inspection techniques. Instead, a particular type of measuring machine from which the data used in this thesis were gathered will be described. This is a non-contact *laser measuring machine.* However, some information on CMMs and their development is mentioned by Gilheany and Treywin [52], and their utilisation is mentioned by Black [15]. A very recent survey of three-dimensional co-ordinate measuring machines is given in [35].

3.3 The Laser Measuring Machine

Many different range-finding techniques used for measuring the shapes of three-dimensional objects have been introduced in the literature. These techniques, as shown in **Figure 3.1**, may be divided into two major categories: active techniques and passive techniques. In this section a laser range-finder [27, 44, 47, 61, 66, 79] based on triangulation will be described. Good surveys on different types of 3D data acquisition techniques are given by Jarvis [66] and Henry [61].

3.3.1 The Design Concept

The laser measuring machine [27, 61] from which the data were gathered was built for Rolls Royce Ltd. to measure turbine blades in three-dimensions as a final quality check. **Figure 3.2** shows the general layout of the machine, which is controlled by a small computer. The laser beam is directed anywhere on a component by the two galvanometer mirrors which rotate on perpendicular axes. The currents driving the mirror galvanometers are derived from the outputs of digital-to-analogue converters. The component to be measured (a turbine blade in the figure) is mounted on a rotary table which is also mounted on a vertical slideway to increase the range of the machine. The rotary table and slideway are both driven by stepper motors which are controlled by the computer. The slideway provided over 300mm of travel, enabling the objects to be measured in a cylinder of diameter 100mm and height 300mm. By use of the rotary table, the component may be inspected from many different directions.

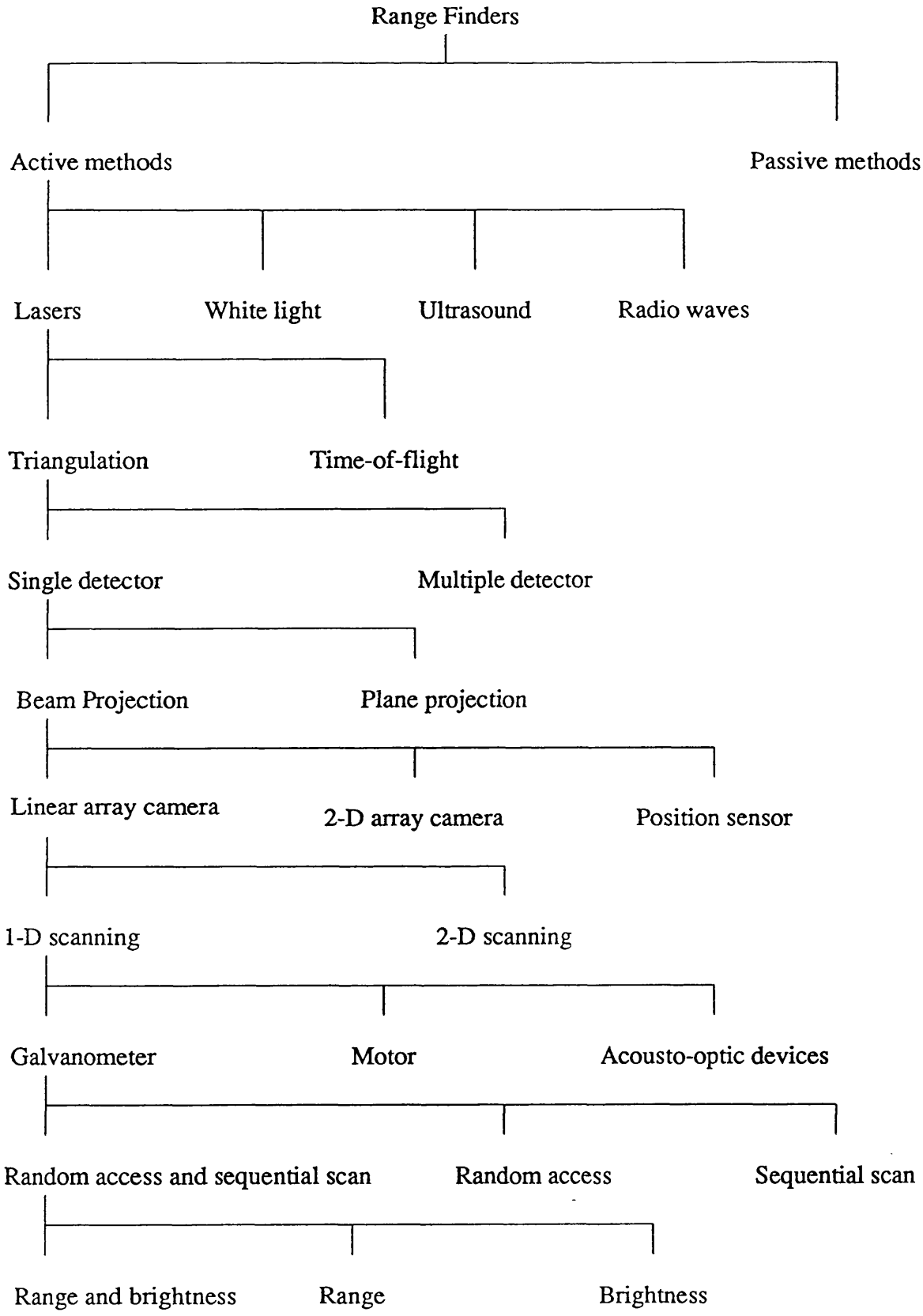


Figure 3.1 Types of range-finding techniques (from Parthasarathy [85])

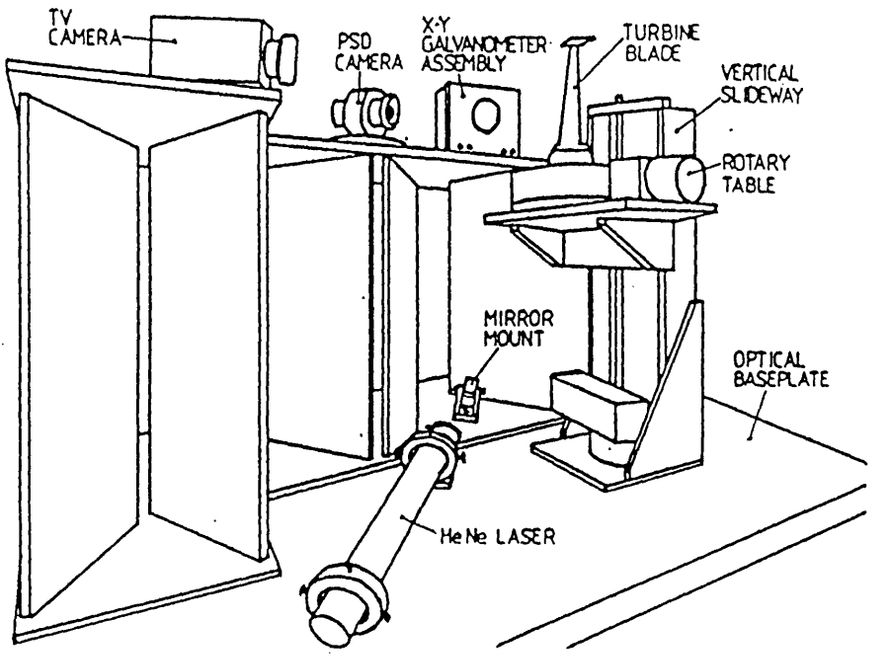


Figure 3.2 General layout of the laser measuring machine (from Henry [61])

The scattered light from the component is received by two Position Sensitive Detector (PSD) cameras mounted either side of the galvanometer assembly and angled at 45 degrees to each other. The signals produced by these cameras are amplified and sent to an analogue-to-digital converter; thus the coordinates of the light spot in three-dimensions are calculated. PSD cameras and their advantages over conventional cameras are mentioned in [27] and [61]. A plan view of the measuring geometry is shown in **Figure 3.3.**

As the whole apparatus is closed off with thick black curtaining for safety reasons, a TV camera is used to provide remote monitoring of the measurement process which allows the operator to position the target ready for the scanning. Once a view is selected by the TV camera, the illumination is switched off and the target is rotated to the scanning position and the scan starts. The target rotations between the scans and TV camera positions are controlled by the computer.

The light source used was a 5mW He-Ne laser, though this was replaced by a infra-red laser diode later on. The working volume in which the machine may take measurements was 100 x 100 x 200 mm, and a measurement accuracy of +/- 0.1 mm was achieved.

3.3.2 Taking Measurements

The laser measuring machine is based on the *laser triangulation* technique [61, 66, 79, 105]. In order to take a single measurement the computer generates the voltages for the mirror galvanometers needed to deflect the laser beam to the desired point on the component's surface. The image of the bright, small spot of the beam

is then picked up by two two-dimensional PSD cameras. These cameras each produce two voltages proportional to the position of spot's centroid focused on a rectangular photosensitive area. The four voltages resulting from the two images of the spot are read using analogue-to-digital converters and the computer calculates the three coordinates of the spot in space by using the values of these voltages together with the distance between the detectors. This is the triangulation technique.

The main alternative to the triangulation technique - as seen from figure 3.1 - is the *time-of-flight* technique [61, 66, 79, 105]. In this technique, a laser beam (or acoustic energy) is directed towards the component to be measured and the path-length of the beam that strikes the component is calculated, either by measuring the short time delay between transmitted and received pulses, or by examining the interference of the beam with its reflection back along its path. The advantage that time-of-flight techniques have over triangulation is that, as long as the beam strikes the component, a measurement can always be made. Since the transmitted and received beams are co-axial, occlusion problems (which occasionally occur in the triangulation technique when the light spot on component's surface is occluded from the view of one or both cameras by other projections of the component) do not exist in time-of-flight technique. However, the time-of-flight technique requires high-frequency electronics to make the measurements and sometimes to modulate the beam. Both techniques suffer when the beam strikes the component very obliquely and insufficient light is reflected into the detectors to make a measurement (this is a particular problem with shiny components).

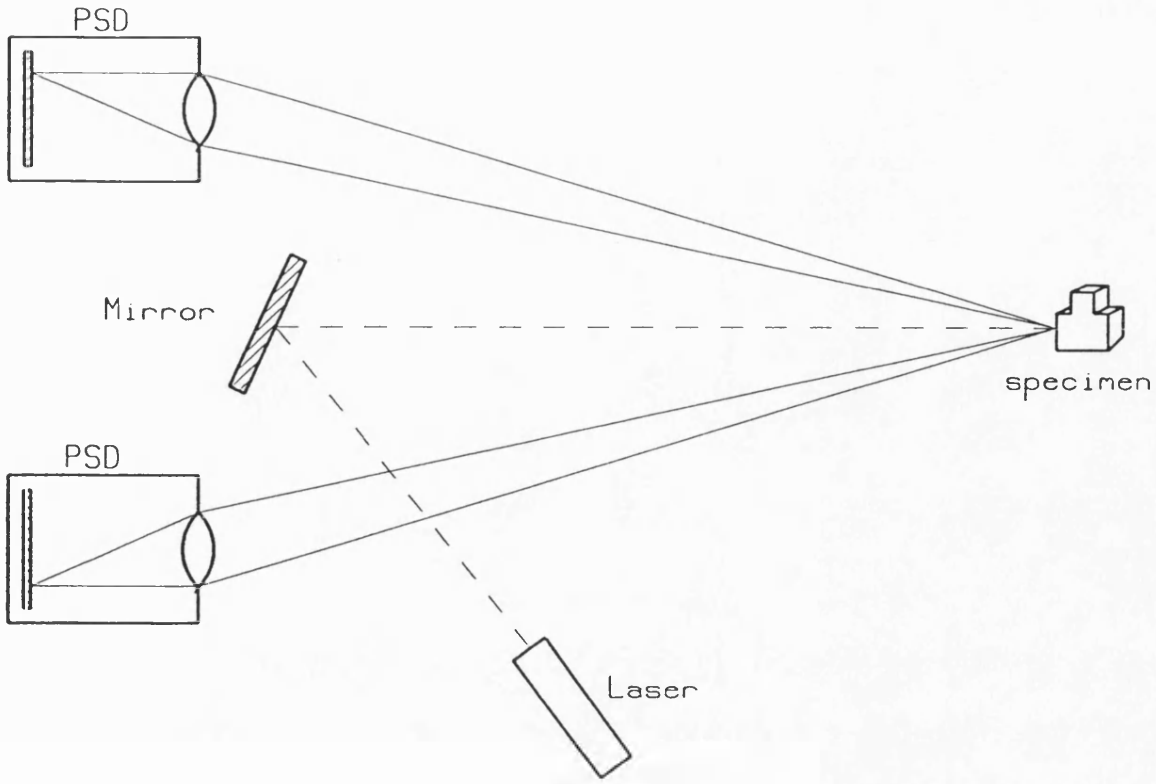


Figure 3.3 Plan view of the measuring geometry (from Bowyer et al. [27])

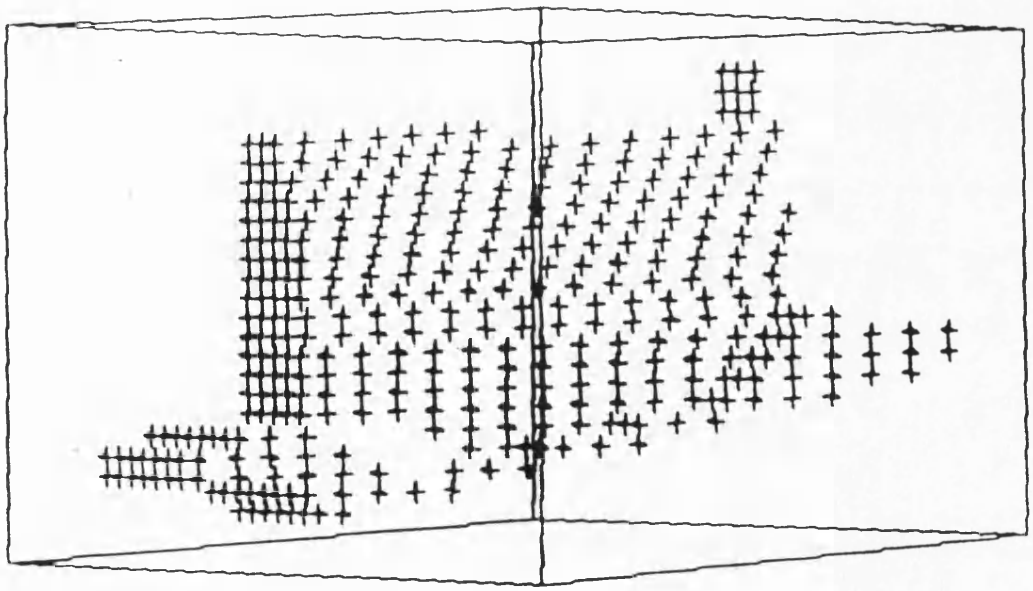


Figure 3.4 Surface points of the turbine blade (from Bowyer et al. [27])

The occlusion problem in triangulation can be eliminated by inspecting the component from several directions. Thus the scan of the component's surface was regular. Of course, the fact that the object's faces were not all at the same orientation meant that a given scan would lead to different densities of points on each. To keep the electronics simple and cheap, triangulation was preferred in [61]. Surveys on triangulation and time-of-flight techniques are given in [61], [66] and [79].

3.3.3 Analysing the Measurements

For inspection applications, measured data need to be compared with reference data produced from a model. For this reason, the data - which consist of the coordinates of surface points of the measured component - need to be analysed and then compared with reference data (a master solid model in this research) to find any manufacturing errors resulting from mismachining or any other reasons. This is the aim of this research. The methods used to analyse the data and to compare the model derived from the measured points with the master solid model of the component will be explained in the next chapters. As an example of input data, the surface points of a turbine blade are shown in **Figure 3.4**.

The laser measuring machine is capable of taking measurements at the rate of about three hundred measurements per second (however, to obtain the accuracy of +/- 0.1 mm this rate drops to thirty measurements per second). The measuring machine is used to measure engineering components automatically after teaching the machine from a master model of the components [27]. The teaching process is done by using a television camera.

A method by Henry [61] which is different from the methods described in this research is proposed for comparing measured data with reference data derived from a model. His method, which is based on the Delaunay triangulation, generates range images from the irregularly sampled 'z' range values and uses two-dimensional interpolation. It then subtracts the measured range image from the reference range image (which is generated either by ray-tracing a master model or by measuring it by a vision system) and calculates an error image which produces information on the types of defects that may be present.

The laser measuring machine is a physical implementation of the ray-casting algorithm. In the next section, simulation of the data gathering process using laser measuring machine will be explained. A ray-tracing algorithm, DORA [115], has been used for the simulation.

3.4 Simulation of Data Gathering

As was mentioned earlier, the data to be analysed and matched to the reference data consist of points in space. These data may either be gathered by using a laser measuring machine or by using an algorithm which simulates the process of this measuring machine (simulation of the laser measuring machine was needed because the measuring machine was unserviceable most of the time during the preparation of this research). A set-theoretic solid modeller, DORA (Divided Object Ray-casting Algorithm) which was developed at Bath University to produce pictures has been slightly modified to be used for the simulation purpose.

3.4.1 DORA - The Solid Modeller

DORA is based on the technique of ray-casting. Ray-casting [113] is a computer graphics technique which produces pictures by tracing a ray of light back from the viewer into a scene being depicted. A picture is produced by generating a pattern of straight lines or *rays* with a viewer's line of sight through each pixel on a raster-scan graphics display. The ray is traced until it strikes a surface of the object and the colour of that surface is transferred to the pixel. **Figure 3.5** shows the ray-casting process.

The ray-casting process in DORA compares each ray with a division structure - DORA divides the object space into sub-spaces to increase the speed of picture production over a number of pictures - and determines the first surface that the ray met. By comparing each primitive (half-space) with the ray and testing whether any of them represents a real surface and doing this for every ray, a picture is generated.

DORA uses a model consisting of a set-theoretic combination of half-spaces to produce a picture on a raster graphics display. The model (which must be suplied as a list of half-space definitions and the set-theoretic operator tree which relates the half-spaces) can be prepared by an Algol-like language called SID (Set-theoretic Input to Dora) [24]. Part of the SID code that generates a hex-headed bolt is given in **Figure 3.6**.

Obviously, the principle of ray-tracers is very similar to the principle of the laser measuring machine. A ray is traced until it strikes to a surface of the object

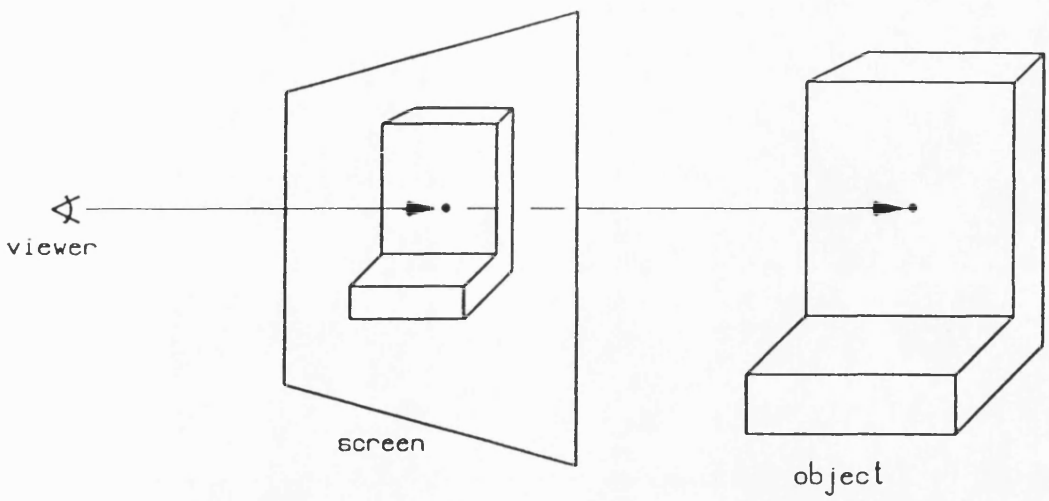


Figure 3.5 Ray-tracing

The SID model building language

```
; Build the bolt's shaft
  z_point :=pt(0,0,1)
  axis :=ln(z_point,z_point)                ; ln returns a line through a point
                                            ; in a given direction
  radius :=diam*0.5
  blt_ shaft :=cylinder(axis, radius,n_ facets)
; Now the hex head
  root_ 3 :=sqrt(3.0)
  radius :=diam*0.5*root_ 3
  face_ point :=pt(0,radius,0)
  face :=space(face_ point,face_ point)     ; space returns a planar
                                            ; face through a point
                                            ; with a given surface
                                            ; normal
  head :=face
  FOR count :=1 TO 5 DO
  { angle :=count*3.1415926/3
    head :=head & spin(face,axis,angle)     ; & means intersection
  }
```

Figure 3.6 The SID code that generates a hex-headed bolt

like a laser beam which is directed at the same surface. The only difference here is, although the laser measuring machine finds the coordinates of the point that the beam struck, the ray-tracer paints the corresponding pixel into the colour of the surface. By making some modifications to the ray-tracer this difference can be removed and the process of the laser measuring machine is simulated.

3.4.2 Modifications on DORA

DORA was modified to record the coordinates of the surface points that the rays struck. Each ray to be traced is defined in parametric form:

$$x = x_0 + ft$$
$$y = y_0 + gt$$
$$z = z_0 + ht$$

where (x_0, y_0, z_0) is the ray's starting point - a view-point which is defined in DORA's command file -; f, g and h are the ray coefficients generated by the ray-tracer; t is the parameter of the ray and (x,y,z) is the intersection point if there is any intersection. Rays are cast from all around the object (6 or 8 different view-points are defined for this purpose) and the coordinates of the intersected points are found and written into a file to be used as data in the future processes.

To control the number of rays used to scan the object's surface, two different angles are defined for horizontal and vertical directions. From these angles, increment angles α and β of horizontal and vertical directions respectively can be calculated as :

$$\alpha = \frac{\textit{Given angle in horizontal direction}}{\textit{desired number of rays in horizontal direction}}$$

$$\beta = \frac{\textit{Given angle in vertical direction}}{\textit{desired number of rays in vertical direction}}$$

These increment angles and the viewing pyramid are shown in **Figure 3.7.** By controlling the increment angles, complicated surfaces can be scanned with more rays than usual and more detailed information can be obtained to generate the model.

To take the accuracy of the laser measuring machine into consideration and to avoid degeneracy problems at the triangulation stage (which will be explained in the next chapter), the surface points were randomly perturbed by a small amount (between the range of 10μ and 100μ - which is the accuracy of laser measuring machine). This was not intended to simulate errors in planarities of the surfaces themselves, or their roughness. Some suggestions about these cases will be given in Chapter 8. **Figure 3.8** shows the surface points of two models. These surface points are obtained by the ray-tracer.

3.4.3 Limitations

A limitation that DORA has is that its models are *faceted.* Curved surfaces are approximated by a combination of planes and only infinite planar half-spaces are used as primitives by the modeller. This limitation has been accepted to generate fast running software. Since DORA has been used in simulation, this limitation in DORA affects the data gathering process.

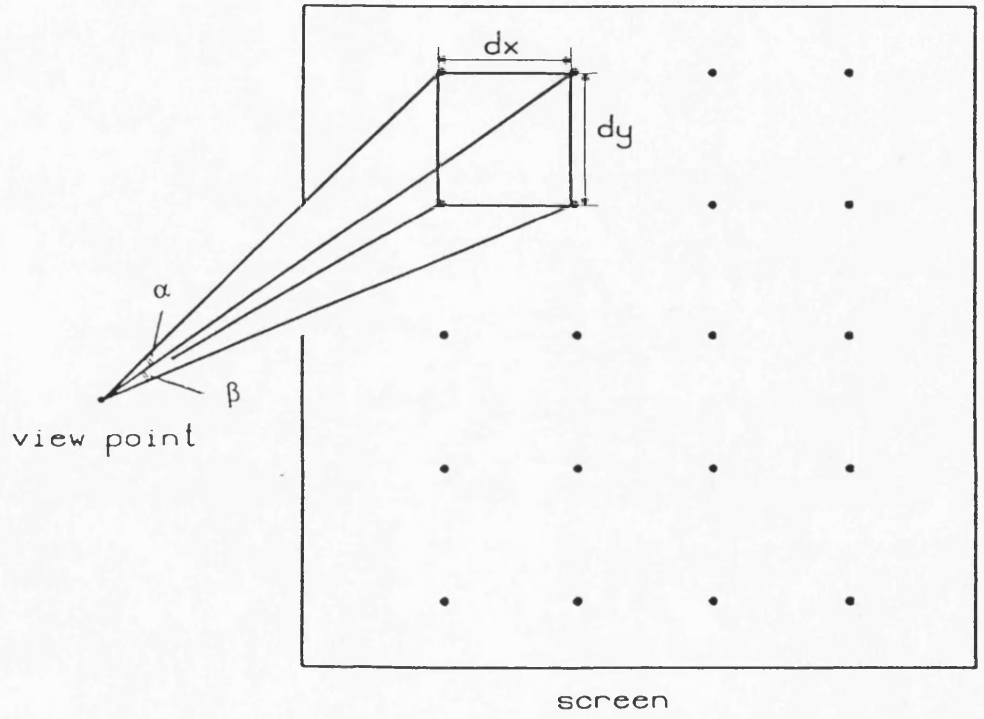


Figure 3.7 Increment angles and the viewing pyramid

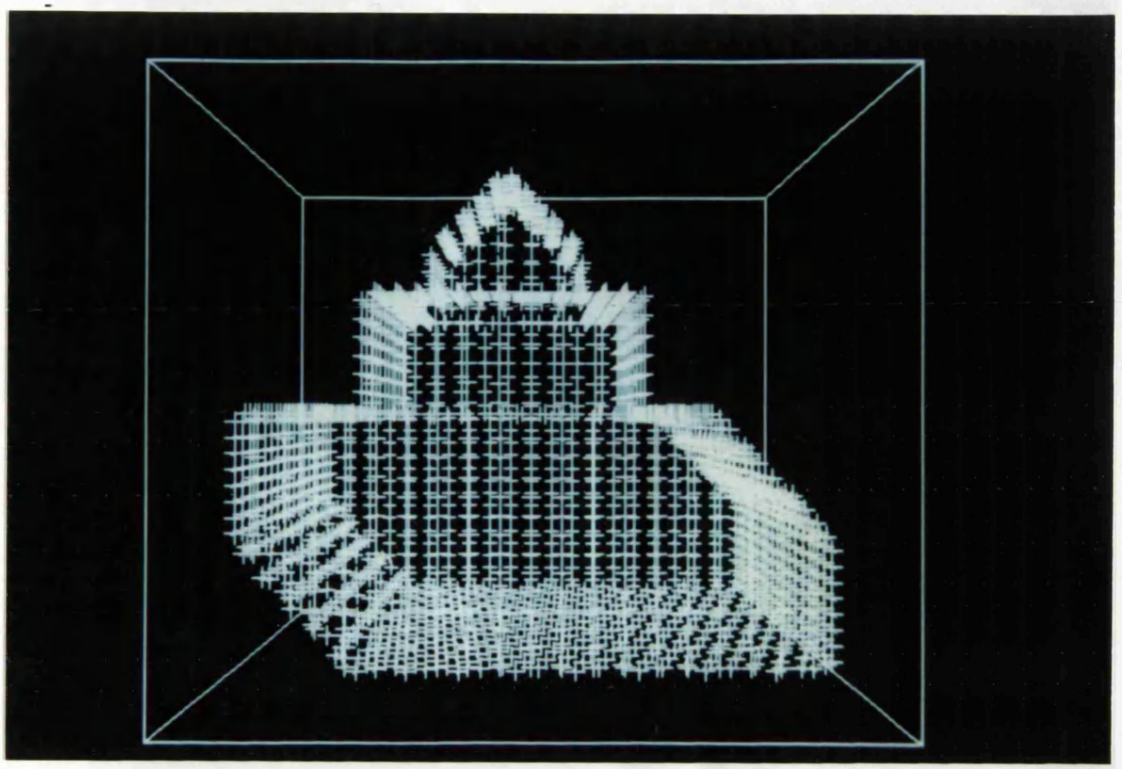


Figure 3.8 (a) Surface points

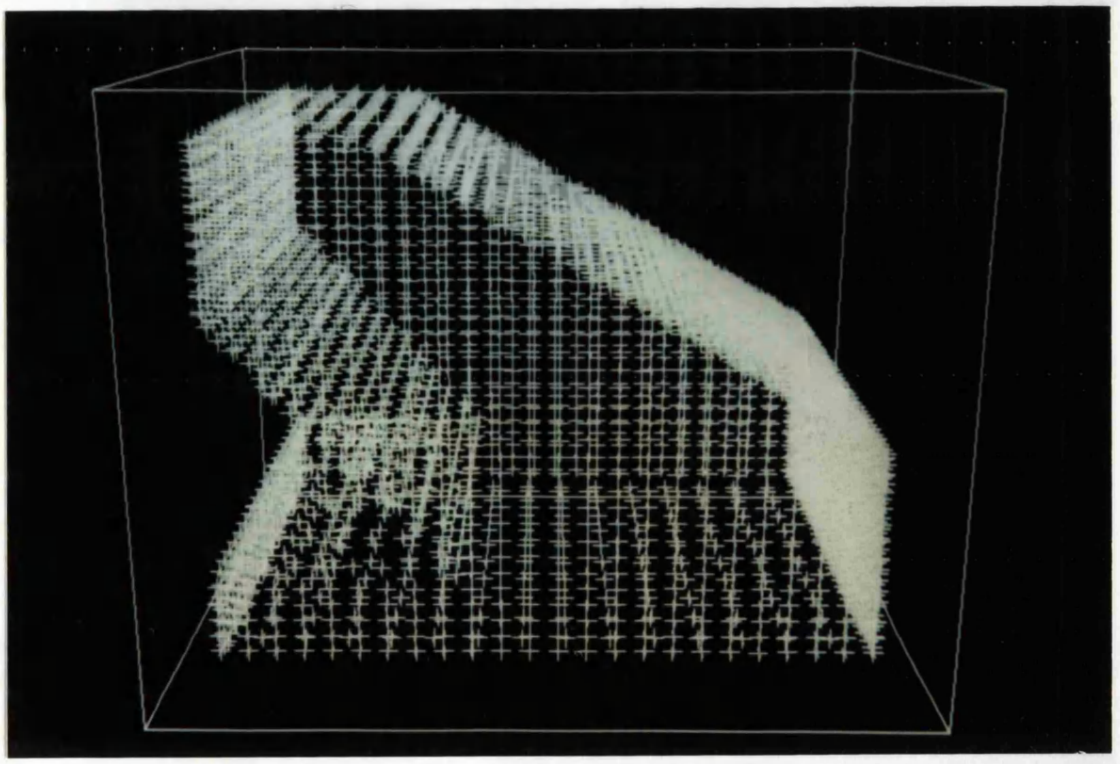


Figure 3.8 (b) Surface points

3.5 Concluding Remarks

In this chapter the laser measuring machine and simulation of data gathering process is described. At this stage, the data which have been obtained either by this measuring machine or by the simulation are ready to be processed. In the next chapter, an efficient algorithm is introduced for processing the data. Although a laser measuring machine has been described in detail in this chapter, all the algorithms described in the next chapters would work just as well for a mechanical probe.

CHAPTER 4

PROCESSING THE DATA

4.1 Introduction

The gathered data consist of a set of co-ordinates (x,y,z) representing the surfaces of an engineering component. This information needs to be processed further to form a shape which will then be compared with the master solid model of the same component to find any defects.

This chapter introduces an efficient multi-dimensional algorithm [25] which uses the *Delaunay triangulation* to process the measured points and to find the neighbourhood relationships between them.

4.2 The Delaunay Triangulation

The data to be matched to a collection of solid model primitives consist of points in space. Only the positions of points lying on the boundary of the measured objects are known; no topological information is available. Obviously, this information is insufficient to form or to analyse the shape of the measured object; some additional information such as the neighbourhood relationship between the points is also needed. To obtain this information the Voronoi diagram (also referred to as the Dirichlet tessellation amongs mathematicians or Thiessen polygons in geography) of the measured points is constructed.

The geometric dual of the Voronoi diagram is the Delaunay triangulation. Literature about the Voronoi diagram and Delaunay triangulation have already been mentioned in Chapter 2. In this section the definition and the properties of the Delaunay triangulation will be given. But first, the formal definition of a triangulation of a closed surface needs to be defined [19, 42]. A collection T of triangles (in Euclidean space) is a triangulation iff:

i. Any two triangles are either disjoint, or have a vertex in common, or have two vertices and consequently the entire edge joining them in common. That is, the triangles do not intersect and are not coincident except at vertices or edges. Also, all edges are simple in the sense that they have exactly two incident triangles. This is the *triangulation* requirement.

ii. All edges within a triangulated surface are connected, that is, there is a path of edges connecting any two of them. This requirement is called the *connect-*

edness condition.

iii. For every vertex of a triangle of T, its link is a simple closed polygon; that is, every edge of a triangle is adjacent to exactly two triangles. This requirement is called the *link* condition.

From this definition, by reformulating the criteria above, some more definitions [42] can be derived such as:

i. Some subset of triangles, that pairwise share common edges, can be combined and this combination is called a *face.* In usual practice a face is chosen to be that subset of edge-sharing triangles that lie on a surface defined by a single equation or parametric representation. A triangle belongs to exactly one face.

ii. Faces may wholly bound to other faces and there is a path from one face to any other face by crossing shared edges. Faces can intersect only at shared vertices and edges.

The aim of triangulation is to find the surfaces. But at this point, a question arises: What is a good triangulation? A triangulation is regarded as *good* for *interpolation* purposes if its triangles are nearly equiangular. It was shown by Sibson [101] that there is only one locally equiangular triangulation of the convex hull of a 2D finite data set and that is the Delaunay triangulation.

The Delaunay triangulation is the geometric dual of its Voronoi diagram. In 2D the Voronoi diagram is a pattern of packed convex polygons covering the whole plane, and is determined by a finite set of distinct points: each point is associated with a territory that is that area of the plane nearer to it than to any other

data point. The formal definition of Voronoi diagram [53] is given as: for a given set $P = \{ p_1, \ldots, p_n \}$, $N \geq 3$ of points in the Euclidean plane where the points are not all colinear and where no four points are cocircular, the region of P_i is the set T_i defined by

$$T_i = \{x: d(x, P_i) < d(x, P_j) \text{ for } all \quad i \neq j\}$$

where d is Euclidean distance.

Lee and Schachter [72] explained the structure of the Voronoi diagram by assuming the structure is the cells of a growth process. Voronoi polygons which have a boundary segment in common are said to be contiguous as are their generating points and are called *Voronoi neighbours*. In *two-dimensions* polygons meet in threes (except in degenerate cases, see section 4.2.2) at *Voronoi vertices* so the lines joining contiguous generating points define triangles. These triangles triangulate the whole area within the convex hull of the generating points. The perpendicular bisectors of the edges of this triangulation give the boundaries of the polygons and circumcentres of the triangles are vertices of the polygons. This triangulation is called the Delaunay triangulation. **Figure 4.1** shows the Voronoi diagram and the Delaunay triangulation for 16 points. Bold lines represent the tessellation where the dotted lines represent the Delaunay triangulation.

In three-dimensions the territory of each data point becomes a convex polyhedron: the region of space nearer to the point than to any other. The faces of the polyhedra are convex polygons and each convex polygon lies in the plane which bisects an edge of a Delaunay tetrahedron. **Figure 4.2** shows a three-dimensional Delaunay vertex and its associated Delaunay tetrahedron.

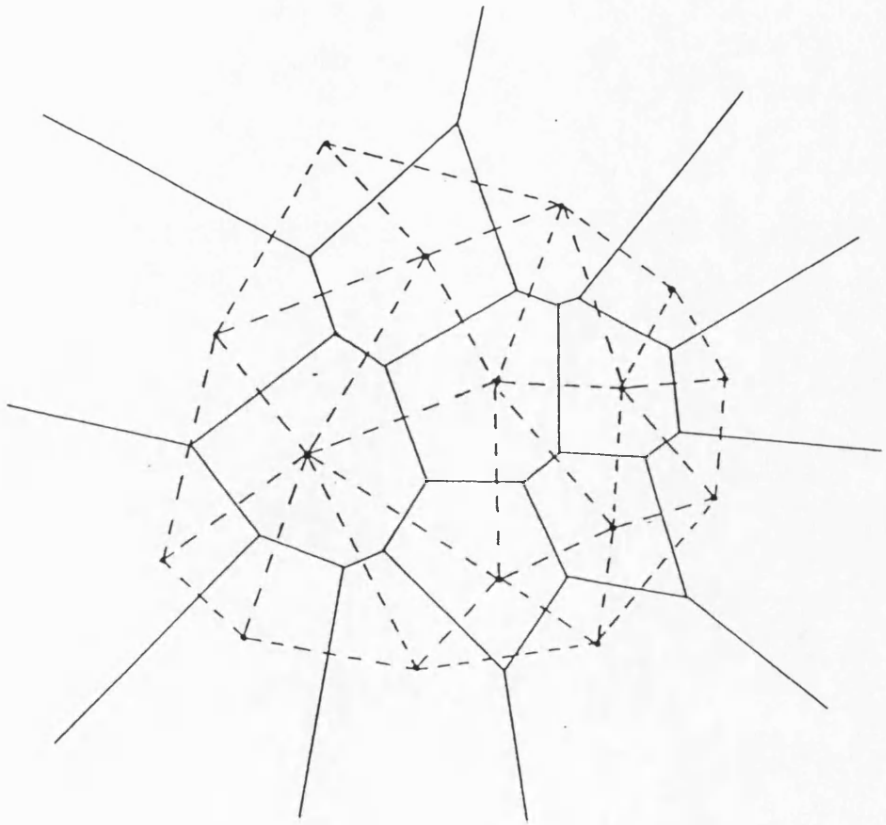


Figure 4.1 Voroni diagram and Delaunay triangulation

4.2.1 Properties of the Delaunay Triangulation

Properties of the Delaunay triangulation and their proofs are given by Lee and Schachter [72]. In this section some of these properties will be described without giving any proof.

Delaunay triangulations have several nice properties. First of all, they are *locally equiangular* (in fact they are the only locally equiangular triangulation of a finite data set as shown by Sibson [101]; for a definition of locally equiangular triangulation also see Sibson [101]). The Delaunay triangulation of a set of points is shown to satisfy the *max-min angle criterion* [70, 72, 101]. Lawson [70] suggested this criterion, which requires that the diagonal of every convex quadrilateral occurring in the triangulation should be well chosen to be able to make the resultant triangles as nearly equiangular as possible. **Figure 4.3** shows this criterion on an example. Since the triangulation that maximises the minimum interior angle of the two resulting triangles needs to be chosen, triangulation (b) in figure 4.3 is preferred.

Lawson [70] also used the max-min criterion to describe a local optimization procedure for constructing a triangulation. The edges of the Delaunay triangulation of a finite set of points are locally optimal. Since the Delaunay triangulation can be constructed by means of a local optimization criterion which ensures global optimality, it is the unique and the optimal triangulation of the convex hull of a set of points.

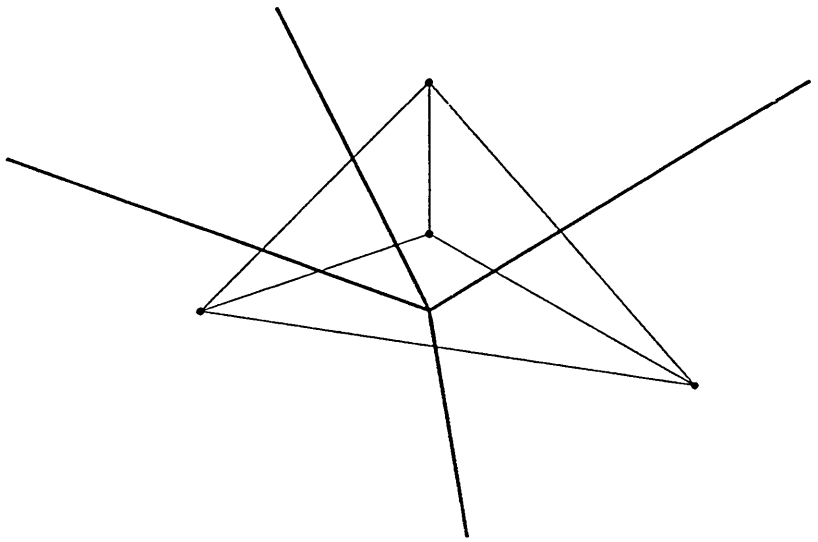
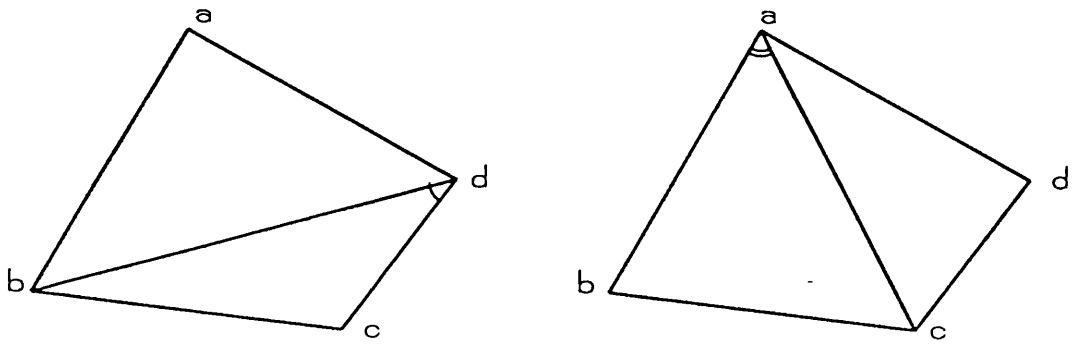


Figure 4.2 A 3-D Delaunay vertex and its associated Delaunay tetrahedron (from Bowyer [25])



(a) (b)

Figure 4.3 The max-min angle criterion

The second criterion which is used to construct the Delaunay triangulation is the *circle criterion.* This criterion [72] is defined as: "For a given set $P = \{ p_1, \ldots, p_n \}$ of points, $\Delta\ p_i p_j p_k$ is a Delaunay triangle if and only if its circumcircle does not contain any other point of P in its interior". In three-dimensions the circle criterion becomes the sphere criterion. The four vertices of each Delaunay tetrahedron lie on the surface of a sphere and no other vertex lies within that sphere. This property characterizes the Delaunay triangulation [17] in k dimensions: "if an hypersphere circumscribing $k+1$ points of P does not contain any other point of P in its interior, this sphere is a *Delaunay sphere* and the corresponding $k+1$ points belong to a simplex of the Delaunay triangulation." The sphere criterion may also be regarded as a smoothness criterion (see [32]).

Apart from these properties, the Delaunay triangulation is also claimed to be a minimum edge length triangulation (or minimum-weight triangulation [106]) by Shamos and Hoey [99] but Lawson [70] and Lloyd [76] proved by counterexample that this is not the case.

All these properties make the Delaunay triangulation one of the most useful constructs associated with the interpolation of a given data set as well as constructing and analysing three-dimensional shapes.

4.2.2 Degeneracies

During the construction of the tessellation in two-dimensions four or more territories may happen to meet at a vertex. Such a vertex is said to be *degenerate* [53]. In general, for k-dimensions, a degeneracy occurs when more than $k+1$ Voronoi k-

dimensional domains share a vertex.

Two types of problems with degeneracies are encountered:

i. when truncation error in the computer causes an error on a near degeneracy, for example an algorithm might make the point P_i contiguous to point P_j but not make P_j contiguous to P_i,

ii. or when $k+1$ (or more) points lie in a hyperplane and are cyclic [25].

The first type of degeneracy occurs whenever the distance from the new point to its neighbouring points is within the expected accumulated truncation error bounds or the new point coincides with an existing point. The second type includes the case when the forming points are lying on the corners of a regular square grid which determines a tessellation in which every vertex is degenerate. However, this type is highly unlikely unless the data points are intentionally placed on the grid. The algorithm described in section 4.3 overcomes these degeneracies and has run successfully on highly degenerate point patterns. The solutions to degeneracy problems are given in [25].

4.2.3 Applications of the Delaunay Triangulation

Apart from the applications mentioned in Chapter 2, as discussed by Boissonnat [20] Delaunay triangulations are quite useful in: automatic modelling of three-dimensional objects (see Chapter 2 for the references), applications in higher dimensions [25, 111], computing the skeleton of a polyhedron [71] and the definition of a shape hull of a dot pattern. Amongst these applications only some

applications in automatic modelling of 3D objects will be mentioned in this section. These applications are:

1. Calculation of the mass properties such as volume, centre of mass, moments of inertia and so on by looking at the mass properties of the set of the interior tetrahedra. The volume is the sum of these elementary volumes, the centre of mass is the centre of gravity of the centres of mass of the different tetrahedra weighted by their volume, etc.

2. Calculation of equilibrium positions by finding the normal projection of the centre of mass onto the faces of the convex hull of the object which forms the boundary of the Delaunay triangulation.

3. Mesh generation to be used to perform stress and thermal analyses by using finite elements techniques (see Chapter 2 for the references). This mesh can be improved by adding new points in the interior of the object in order to obtain more regular tetrahedra.

4. Cruder polyhedral approximations by eliminating points producing elongated tetrahedra. This can be achieved easily if (as in the case of the algorithm described in the next section and in [111]) the algorithm computing the Delaunay triangulation is implemented as an iterative procedure which inserts the points into structure one after another and updates the triangulated structure after each insertion.

The application of the Delaunay triangulation to gathered data to form a structure of tetrahedra with the measured surface points as vertices will be explained in sec-

tion 4.5.

4.3 The Triangulation Algorithm

Different Delaunay triangulation algorithms in 2D or higher dimensions have been proposed in the literature [53, 106, 111] and the efficiency of some of these algorithms has been reviewed by Boissonnat [17]. An algorithm which uses a 3D Delaunay triangulation will be described in this section. It will be described in some detail (though it has been published in [25]) because the data structure which it uses will be needed for subsequent chapters in this thesis. A 3D Delaunay triangulation (which is an assemblage of space-filling, disjoint, irregular tetrahedra) implies a topology among the three-dimensional points. By setting the topological relationship among the 3D points, a polyhedron that consists of triangular faces is constructed.

The triangulation algorithm described in this section is used to obtain neighbourhood relationship information for the three-dimensional surface points of the measured component gathered either by a measuring machine or the simulation. More precisely the problem here is to find the polyhedron (which consists of triangular faces) whose vertices are the measured points.

This algorithm has some similarities with Watson's [111]. Watson also proposed an algorithm to compute k-dimensional Delaunay tessellations. His algorithm checks k-dimensional hyperspheres (which are the Delaunay spheres with the Delaunay vertices in the centre) and observes which circumspheres are intersected by the new point after each insertion. This process is the same as checking each

new point to see whether it is closer to the vertex than its forming points are. If the circumsphere is intersected, the structure is modified. His algorithm adds the points into the structure in an 'advancing forward' sequence which means the Delaunay k-simplices behind the advancing front are in their final configuration while those ahead of the front are subject to alteration. The major difference of this algorithm from the one described in this chapter is, Watson does not use any technique such as finding the nearest neighbour to eliminate some of the vertices and checks all of the vertices to find the closest vertex which leads his execution time to be $O(N^{\frac{(2k-1)}{k}})$ for N data points. The execution time of the algorithm described in this chapter is $O(N^{(1+\frac{1}{k})})$.

4.3.1 Data structure

Before starting to describe how the algorithm works, the data structure used needs to be explained. Green and Sibson's [53] two-dimensional algorithm (which has some similarities with the algorithm described here) uses an additive method to compute the contiguities of Delaunay triangles and stores the triangulation in the form of lists of contiguous points for each point. It adds the points in turn and modifies the contiguities as each point is added. In the two-dimensional case the contiguity lists can be arranged in cyclic order. For the points this cyclic order has no starting point: it is a ring structure, that is to say a list which can be broken arbitrarily. To insert the points into the structure one by one, the algorithm uses the fact that in 2D contiguity lists can be stored cyclically. This sort of ordering is

not possible in higher dimensions. So, how can the vertex or point structure be stored in higher dimensions?

To answer this question consider the 2D structure in **Figure 4.4.** Two lists can be constructed for each vertex in the structure: a list of points which form each vertex (each vertex is the circumcentre of its three forming points), and a list which contains the neighbouring vertices of each vertex, each one opposite one of the vertex's forming point. Territorial boundaries that extend to infinity can be considered as terminating in a vertex labelled zero. **Figure 4.5** shows the data structure for the six vertices in figure 4.4.

There is no cyclic order of the points around a vertex and the order of forming points in figure 4.5 is deliberately arbitrary. In k dimensions each vertex has $k+1$ forming points and $k+1$ neighbouring vertices opposite them.

4.3.2 Inserting a point into the structure

Triangulation of a data set starts with a simple structure (the most obvious starting pattern is the Delaunay simplex formed by the first $k+1$ points) and builds the triangulation upon this simple structure by adding each point to the structure one by one and modifying the structure after each insertion, starting the triangulation with the Delaunay simplex forming a tessellation containing one real vertex all of whose neighbouring vertices are 0. The only limitation which needs to be considered is that the first $k+1$ points must not all lie in a hyperplane in the k dimensional space in which the triangulation takes place. The flow-chart of the triangulation algorithm is given in **Figure 4.6.** Generating a convex hull from $k+1$ data points will

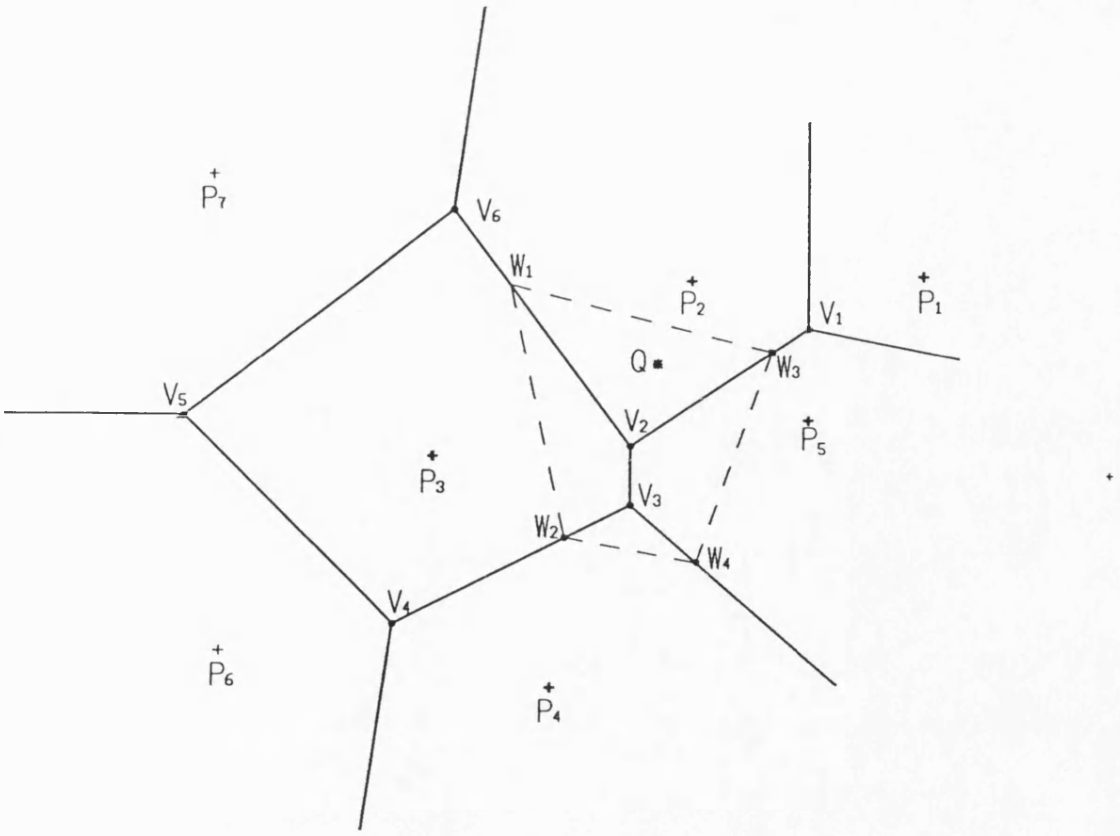


Figure 4.4 The newly inserted point Q - finding its territory (after Bowyer [25])

Vertex	Forming Points			Neighbouring Vertices		
V_1	P_1	P_2	P_5	V_2	0	0
V_2	P_3	P_5	P_2	V_1	V_6	V_3
V_3	P_3	P_5	P_4	0	V_4	V_2
V_4	P_6	P_3	P_4	V_3	0	V_5
V_5	P_6	P_3	P_7	V_6	0	V_4
V_6	P_3	P_7	P_2	0	V_2	V_5

Figure 4.5 Data structure the Delaunay triangulation

be explained in section 4.5.

To explain how the data points are inserted, suppose the new point Q needs to be inserted within the current convex hull of data points of the structure in figure 4.4. The new territory which will be formed by the insertion of this new point is indicated by the dotted lines. The algorithm which finds the territory of the new point can be outlined as follows:

1. Find the first vertex in the structure which will be deleted by the new point (say V_2). This vertex is any vertex which is nearer to the new point than to its forming points. There will always be at least one such vertex, as the vertex corresponding to the Delaunay simplex in which the new point lies will always be deleted and Delaunay simplices completely fill the convex hull of the currently included points.

2. Look for the other vertices which will also be deleted by starting from the first deleted vertex and performing a tree search through the vertex structure. This is not difficult if the data are stored as indicated in figure 4.5. The result will be a list of all vertices deleted by the new inserted point Q. In this case (as seen from figure 4.4) the list will be: $\{ V_2, V_3 \}$.

3. The points contiguous to the point Q are all the forming points of the deleted vertices: $\{ P_4, P_5, P_2, P_3 \}$.

4. Remove the old contiguities between the pairs of those forming points if all their vertices are in the list of deleted vertices (e.g. since the vertices V_2 and V_3 are deleted, remove the contiguity of P_3 - P_5)

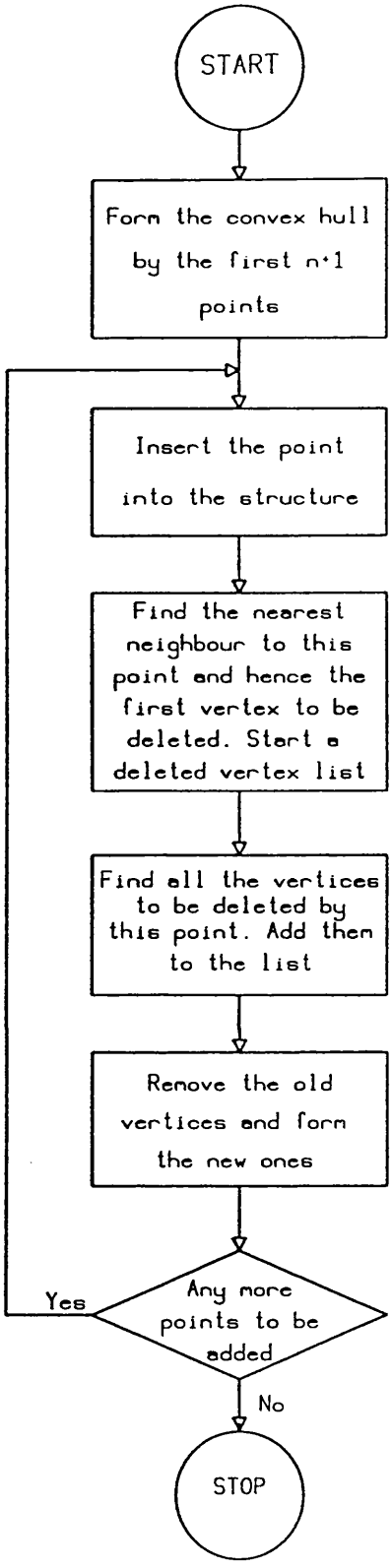


Figure 4.6 The flowchart of the triangulation algorithm

5. In this case four new vertices will be formed for the new point: $\{ W_1, W_2, W_3, W_4 \}$. Compute their forming points and neighbouring vertices. The forming points for each new vertex will be point Q and k of the points contiguous to Q. Each line in the tessellation is shared by k points around it (e.g. the line V_2 - V_6 is formed and shared by P_2 and P_3). The forming points of new vertices and their neighbouring vertices may be found by considering vertices pointed to by members of deleted vertex list that are not themselves deleted, and finding the rings of points around them. Thus W_1 points outwards to V_6 from Q and is formed by $\{ P_2, P_3, Q \}$.

6. Finally, overwrite the entries of deleted vertices with some of the new ones to save space.

All these operations considered above are of a local nature (except step 1). Therefore, the amount of work to be done is independent of the number of points currently in the structure and is roughly proportional to the number of new vertices created.

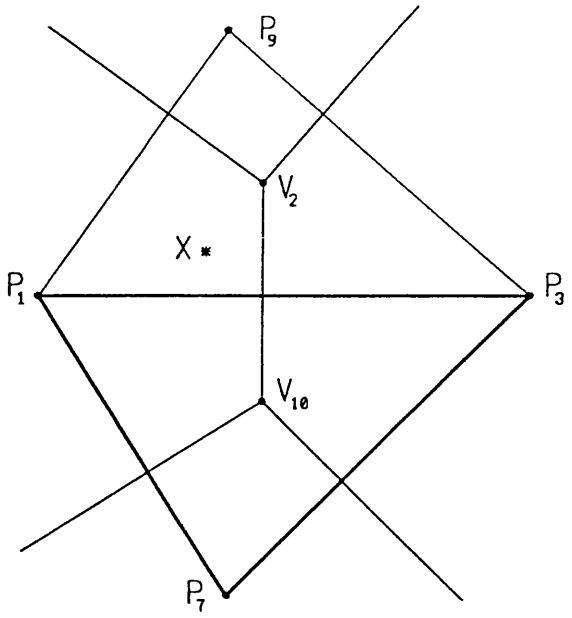
The problem to be solved in step 1 is to identify the first vertex which will be deleted by the point that is about to be inserted. One solution of this problem is easy: examine each vertex in the structure to see if it is nearer to the new point than to its forming points and find a vertex which satisfies the condition. But this would be very time consuming process (especially with large number of points in higher dimensions) and would destroy the benefit of the local nature of the insertion algorithm. Therefore, to overcome these disadvantages, a vertex needs to be identified without the need to examine most of the vertices in the structure. This

problem can be solved by *finding the nearest neighbour* of the new inserted point.

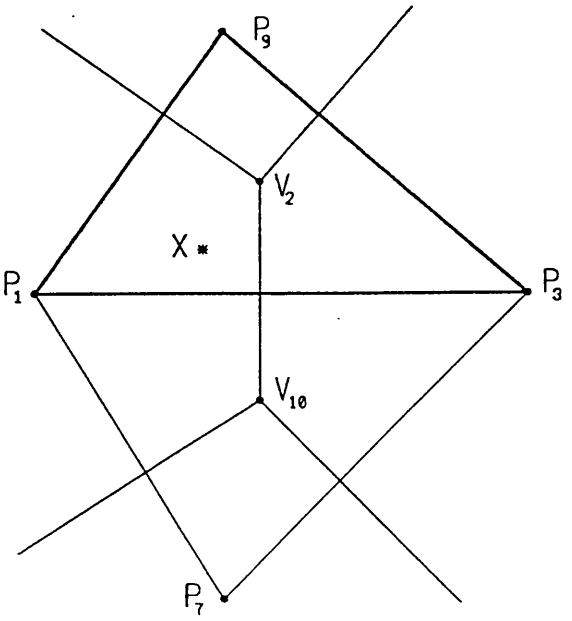
4.3.3 Finding the nearest neighbour

The solution as proposed in Green and Sibson's [53] two-dimensional algorithm is to start looking for the nearest neighbouring point at an arbitrary point and walk from neighbour to neighbour across the Delaunay triangulation, always approaching the new point until the point nearest to it is found. In applications in which the list of points is automatically in systematic order, the new point is likely to be near the one that had just been inserted and that last point would be the obvious place to start the walk. This will result in an almost negligible computational load. If nothing systematic is known about the position of points the obvious place to start this walk is at a point near to the centroid of the currently accepted points; in k dimensions a walk for N points starting from the centroid of the configuration should take $O(N^{\frac{1}{k}})$.

The routine which performs the walk starts with the last accepted point as the neighbouring point and checks if the new point is inside the Delaunay simplex which has the last accepted point as one of its forming points. To understand how the check is done consider the structure in **Figure 4.7**. The algorithm inspects the faces formed by omitting points in turn. It uses the result of vector products to make the decision. It first calculates the vector product of $P_7\vec{P}_1 \times P_7\vec{P}_3$ and checks the sign of the result. The sign of the result gives the side on which the opposing corner point lies. Then it does the same calculation for X (the new point). Since



(a) The inserted point is outside of the simplex



(b) The inserted point is inside the simplex

Figure 4.7 Finding the nearest neighbour

the sign of the result of $X\vec{P}_1 \times X\vec{P}_3$ is different from the previous sign as in figure 4.7 (a), this means the new point X is not on the same side as the opposing corner P_7 and, in fact, is not lying in the associated Delaunay simplex. In this case, it flags the path so that it is not taken again and finds the next simplex to examine. It does this by finding the vertex opposite to the point which causes the different sign (V_2 in figure 4.7 (a)). It repeats this until the simplex containing X is found, which it checks in the following manner: consider the simplex $\Delta P_1 P_9 P_3$ and the new point X in figure 4.7 (b). In this case the resultant sign of vector products of $P_1\vec{P}_9 \times P_1 P_3$ and $X\vec{P}_9 \times X\vec{P}_3$ are the same (as are the signs of $P_9\vec{P}_1 \times P_9 P_3$ and $X\vec{P}_1 \times X\vec{P}_3$, and $P_3\vec{P}_9 \times P_3 P_1$ and $X\vec{P}_1 \times X\vec{P}_9$) which means the new point is in the simplex. This ends the neighbour finding process.

4.3.4 Modifying the structure

Once the nearest neighbour of the new point has been found it is a simple matter to find a deleted vertex as explained in section 4.3.2. The new point must delete at least one point on the boundary of its nearest neighbour's territory.

The routine takes the neighbouring vertices list of the deleted vertex, calculates the squared distance between the new point and each vertex on the list (D_s) and the squared radius associated with the hypersphere in which the neighbouring vertex is in the centre (D_v), compares D_s and D_v, and adds the vertex into the deleted vertex list if the new point is closer to the vertex than its forming points are (which means $D_s < D_v$), otherwise it ignores the vertex. After deleting the old vertices, old contiguities are also removed and the structure is modified to construct

new vertices and new contiguities. The whole process is repeated after each insertion.

Finally, the algorithm should modify the structure only if the point inserted is within the current convex hull. If a new point is outside the convex hull, it should not delete any vertices and should be treated differently. This can easily be flagged, as none of the vertices of the new point's nearest neighbour are deleted, and overcome by setting up the initial simplex and the vertex on which the algorithm builds such that the $k+1$ points on the corners of the simplex remain the convex hull throughout the entire process. As the whole range of floating point numbers is available this is not difficult. These first $k+1$ points would, almost always, not be data values, but would be artificially generated to bound the problem. The generation of these points for the application of the triangulation algorithm to the measured or generated data is described in section 4.5.

4.4 Implementation of Details

4.4.1 Programming

The triangulation algorithm consists of a set of ISO FORTRAN subroutines which are callable from a simple main program that feeds the points to them one by one. The data structure of vertex lists and Delaunay simplexes are available at any stage of the processing as well as the lists of contiguities. Moreover, the lists of vertices around a point's territory or common to a pair of points (the vertices associated with a contiguity) are also produced within the processing.

It is possible to make the usual compromises between storage space and execution time (e.g., either to store the position and squared radius associated with each vertex in the structure or to compute these values when they are needed).

In only two subroutines do floating point calculations take place: the subroutine which calculates the squared distance between any two points in the k dimensional space in which the tessellation is being constructed, and the subroutine which calculates the circumcentre and squared radius associated with the hypersphere which passes through the $k+1$ point at the corners of a simplex. Since the radii of the circumspheres are used only in comparison, the squared values are compared to save the time of root extraction.

4.5 Application of the Algorithm to the Gathered Data

The aim of applying the Delaunay triangulation algorithm is to form a three-dimensional structure (a solid structure) which is the aggregation of a set of packed tetrahedra with the measured points as the vertices. This can be achieved by fitting the tetrahedra to the surface points and finding which of these are solid. Some of the triangles that form the surfaces of these tetrahedra will form a complete triangulation of the measure object's surface.

Because of the properties listed in section 4.2.1 the Delaunay triangulation is suitable for structuring the measured data. Since the only information about the gathered data is the position of the surface points lying on the boundary of the object, the adjacency relationship between the points needs also to be provided. This information (which can then be processed to determine which of them are

lying on the same surface) is obtained from the Delaunay triangulation.

The surface points are added to the structure one by one. Before feeding the points into the structure a convex hull of the measured points is generated by forming a Delaunay simplex from *k+1* points. These *k+1* points are not data values but are artificially generated to bound the data set and the Delaunay simplex formed by these points encompasses the data points. The position of these *k+1* points are calculated as follows:

1. Scan the measurements file and find the maximum and minimum values of *(x,y,z)*. Let $p = (x_{min}, y_{min}, z_{min})$ and $q = (x_{max}, y_{max}, z_{max})$.

2. Move the minimum and maximum points a given distance away in negative and positive directions respectively so that the convex hull includes the minimum and maximum points as well. The new p and q are now:

$$p' = p - (1,1,1)$$

$$q' = q + (1,1,1)$$

$$d = q' \cdot \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$$

   where d is the projection of q' onto the unit vector which is perpendicular to plane 1. **Figure 4.8** shows two-dimensional representation of this. The triangle enclosing the data set corresponds to the Delaunay tetrahedra formed by the first four points in 3D and line 1 corresponds to plane 1. The d given above is obviously the distance of plane 1 from the origin as well.

3. From these definitions, the equation of plane 1 in **Figure 4.9** can be written as:

$$\frac{x}{\sqrt{3}} + \frac{y}{\sqrt{3}} + \frac{z}{\sqrt{3}} - d = 0$$

and the first $k+1$ points (4 points for three-dimensional case) which are the vertices of the convex hull as:

$$\textit{Point 1} : (x_{p'}, y_{p'}, z_{p'})$$

$$\textit{Point 2} : (\dot{x}, y_{p'}, z_{p'})$$

$$\textit{Point 3} : (x_{p'}, \dot{y}, z_{p'})$$

$$\textit{Point 4} : (x_{p'}, y_{p'}, \dot{z})$$

where $\dot{x} = \sqrt{3} \cdot (d - \frac{y_{p'}}{\sqrt{3}} - \frac{z_{p'}}{\sqrt{3}})$, $\dot{y} = \sqrt{3} \cdot (d - \frac{x_{p'}}{\sqrt{3}} - \frac{z_{p'}}{\sqrt{3}})$

and $\dot{z} = \sqrt{3} \cdot (d - \frac{x_{p'}}{\sqrt{3}} - \frac{y_{p'}}{\sqrt{3}})$.

Placing the points in this way guarantees that all the data points will be inside them: they remain the convex hull throughout. The vertex data structure is modified after each point insertion. Since the data points are distinct there is no possibility of coming across the first type of degeneracy. To avoid the second type of degeneracy data points which are generated by simulation are randomly perturbed.

4.6 Concluding Remarks

In this chapter a Delaunay triangulation algorithm has been described to process the measured or generated data. At the end of such processing, a three-dimensional structure which is the aggregation of a set of packed tetrahedra with the measured points as the vertices is formed. Some of the triangles that form the

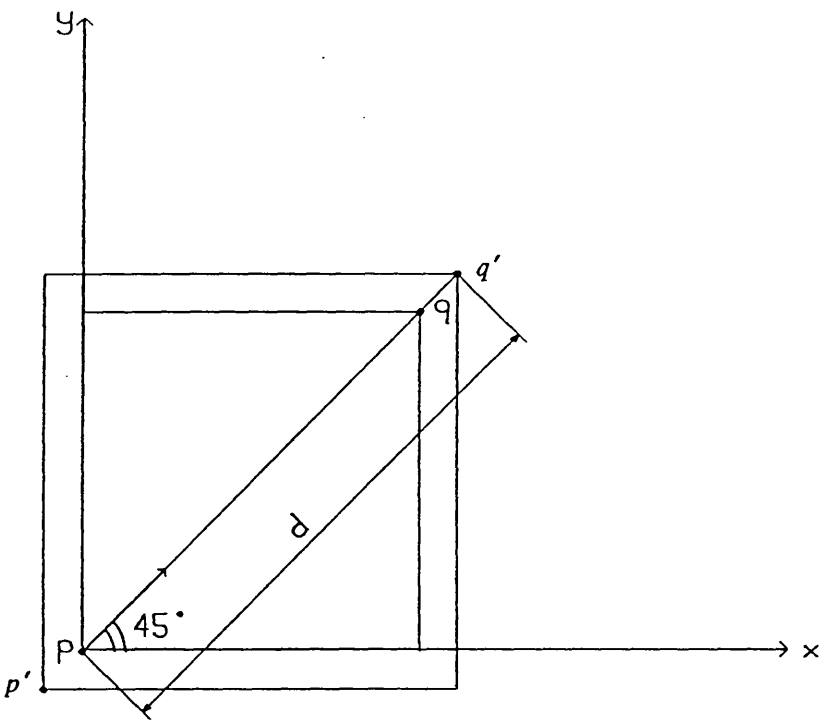


Figure 4.8 Projection of q' onto the unit vector in 2D

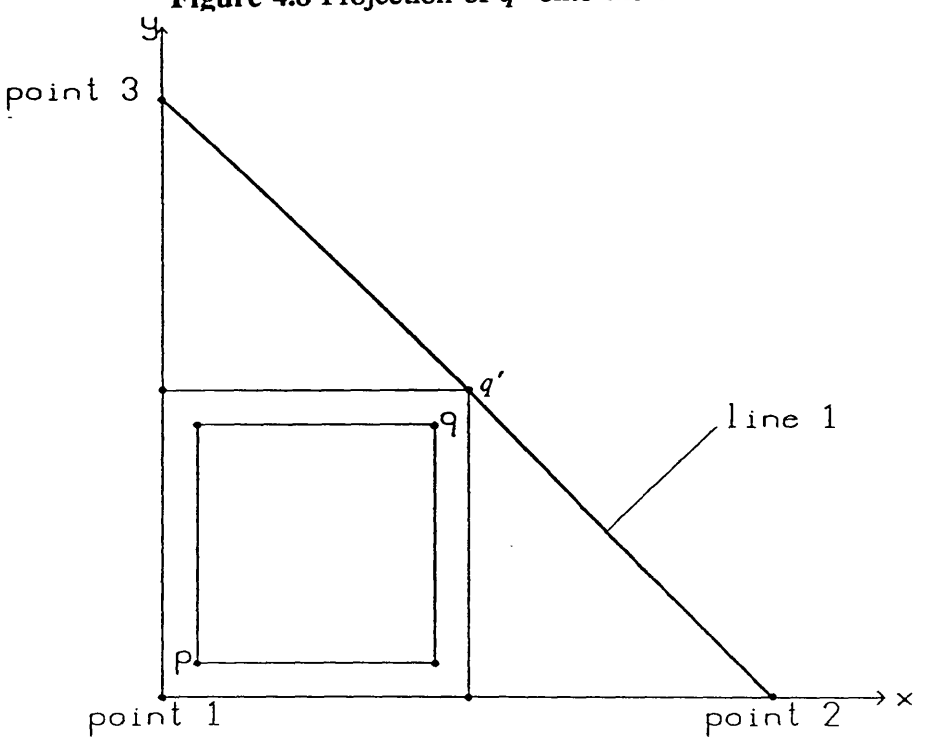


Figure 4.9 Convex hull in two-dimensions

surfaces of these tetrahedra will form a complete triangulation of the measure object's surface. But the problem here is to find which ones. A method of solving this problem will be explained in the next chapter.

CHAPTER 5

FINDING THE OBJECT'S SURFACE

5.1 Introduction

The Delaunay triangulation forms a tetrahedral packing with the measured (or generated) points as vertices. This is a volumetric representation of the object. Since the faces of this model will be matched to a collection of solid model primitives, its surface needs to be found first.

Some of the triangular faces of the tetrahedra will form a complete triangulation of the surface. In this chapter a method of finding these tetrahedra (and the surface of the model as a result of this) is introduced [29]. In addition, methods of eliminating the *redundant* tetrahedra (that is the tetrahedra which the algorithm initially categorises as solid but which are, in fact, air) is also described.

5.2 Classification of Tetrahedra

In order to find the surface of the model the tetrahedra that form the model need to be classified. After the triangulation has been computed some Delaunay tetrahedra will lie within the measured object and will thus be *solid*, whereas some will lie outside it and will be *air*. If the tetrahedra may be so classified, then any of their triangular faces that form a boundary between a solid tetrahedron and an air tetrahedron will be part of the component's surface. But how may the tetrahedra be classified in such a way?

As explained in Chapter 3, in order to measure each point a ray of light must have been directed at it or a measuring probe must have touched it. If the path taken by this is recorded then any Delaunay tetrahedra which it passes through must be air.

The algorithm which does the classification takes the surface points one by one, traces the path backwards from each point (by negating the coefficients of the ray which was used to detect each point) and classifies the tetrahedra which the ray passes through as air. Since the points are on the surface and the path is traced backwards all the tetrahedra which the path intersects on its way should be air. The flow chart of the algorithm is given in **Figure 5.1.**

This research was particularly concerned to deal with data gathered by the laser coordinate measuring machine mentioned earlier [61]. Henceforth the path will be considered to be a ray of light, but all the algorithms would work just as well with a mechanical probe path.

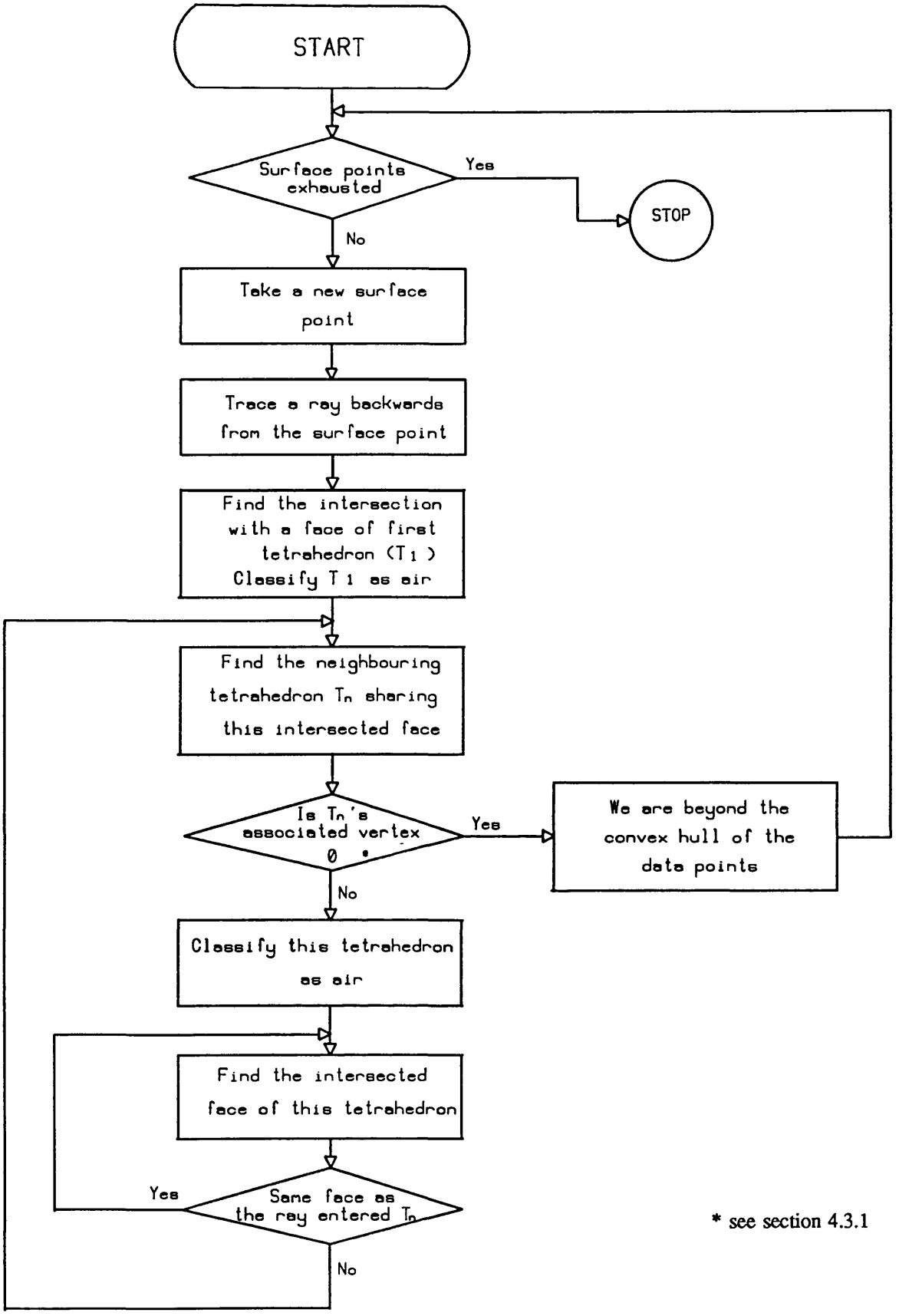


Figure 5.1 Classification of tetrahedra

The algorithm takes the first surface point and starts tracing its ray of light backwards. When it finds the intersection between a triangular face of the first tetrahedron and the ray (since the starting point is one of the forming points of this Delaunay tetrahedron, there is always one intersection with the first tetrahedron) it classifies this tetrahedron as air and continues tracing the ray into the tetrahedron which shares the intersected face with the first tetrahedron. After this there are always two intersections with the ray and each tetrahedron (unless the ray intersects an edge or a corner).

The ray-tracing algorithm uses parametric rays, as did the simulation algorithm (see Chapter 3). The next tetrahedron which the ray will pass through is determined in two ways: by using the ray parameters at the intersection points or by using the intersected face information. First, consider the ray parameter information.

Since rays are parametric, the parameter of ray at the intersection point is calculated as follows [28, 108]:

1. The equation of the plane in which a triangular face lies is given by

$$ax + by + cz + d = 0$$

The plane coefficients can be calculated from the coordinates of the points that form this triangle (see **Figure 5.2**). The implicit equation of a plane through these three points can be stated as a determinant:

$$\begin{bmatrix} x - x_J & y - y_J & z - z_J \\ x_K - x_J & y_K - y_J & z_K - z_J \\ x_L - x_J & y_L - y_J & z_L - z_J \end{bmatrix} = 0$$

which contains the three independent variables, x, y and z. This determinant states a vector formed by J and any point in the plane must be perpendicular to the vector product of the vectors from J to K and from J to L. This vector product is obviously normal to the plane. If the determinant is multiplied out it gives the usual form of plane equation indicated above and the coefficients of this plane are calculated from the co-ordinates of the three points which lie on this plane.

2. Since the rays are parametric and their equations are given as:

$$x = x_0 + ft$$
$$y = y_0 + gt$$
$$z = z_0 + ht$$

the parameter of the ray where it intersects the plane is then calculated as:

$$t = -\frac{(ax_0 + by_0 + cz_0 + d)}{(af + bg + ch)}$$

In order to determine the next tetrahedron which the ray passes through, the intersected face which has the bigger ray parameter value of the two ray parameter values at the intersection points is found and the ray is traced towards the neighbouring tetrahedron which shares this face with the intersected tetrahedron. The ray parameters of each ray at the intersection points are determined and the next tetrahedron that each ray will follow is found. The method of finding whether the ray pierces the triangular face or not will be explained below.

If the ray parameters are too close to each other (because of the rounding error of the computer) the algorithm fails to chose the bigger ray parameter and fails to find the next tetrahedron to follow. In order to avoid this problem, the

intersected faces can be checked as an alternative way of making the decision. Since the ray first intersects the face which is shared with the previous tetrahedron that the ray is coming from, the ray is traced towards the tetrahedron which has the other intersected face in common. **Figure 5.3** shows the ray-tracing algorithm in two-dimensions.

Since the ray intersects the line P_1P_4 (which corresponds a triangular face in three-dimensions), the next tetrahedron to be visited (corresponding to the vertex V_{10}) is the one which shares the line P_1P_4 with the tetrahedron of V_8. After the second intersection with the tetrahedron corresponding to the vertex V_{10} has been found, the next vertex is V_5 which has the common edge P_4P_7 with V_{10} and so on.

As explained in the section on the triangulation algorithm (section 4.3) the forming points of each Delaunay tetrahedron and the neighbouring vertices opposite to the forming points are kept in two separate lists. When an intersection between the ray and one of the faces of a Delaunay tetrahedron is found, the next tetrahedron to be visited is determined by checking the neighbouring vertices list and finding a Delaunay vertex that is opposite to the forming point of the tetrahedron which is not in the intersected face. For instance, in figure 5.3, for the tetrahedron corresponding to the vertex V_{10}, the next vertex to be visited is vertex V_5 which is opposite to P_1.

In order to determine if a ray pierces a triangular face (see **Figure 5.4**), the following calculations are performed:

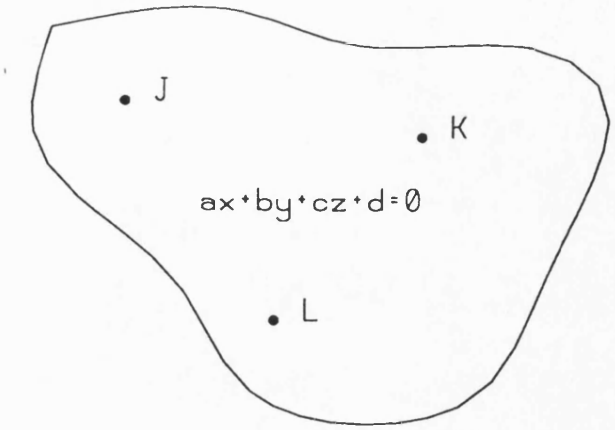


Figure 5.2 A plane through three points

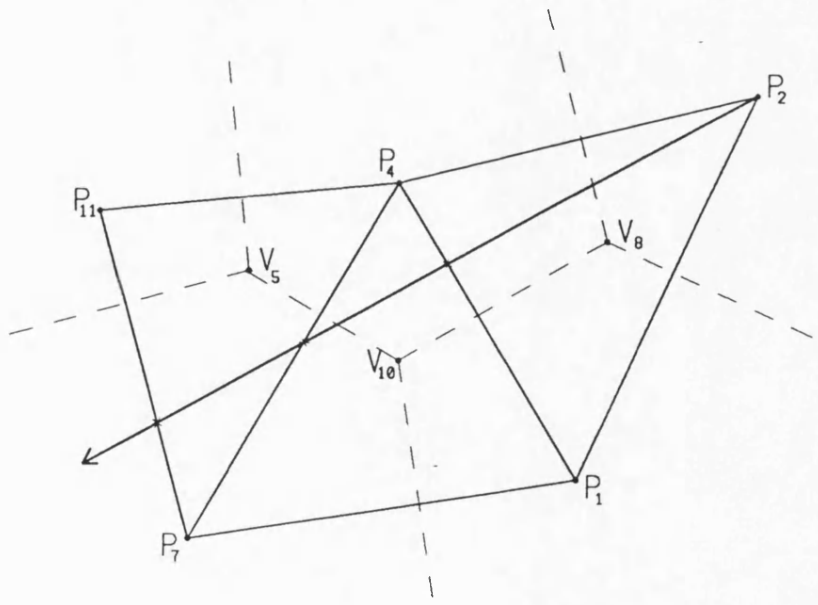
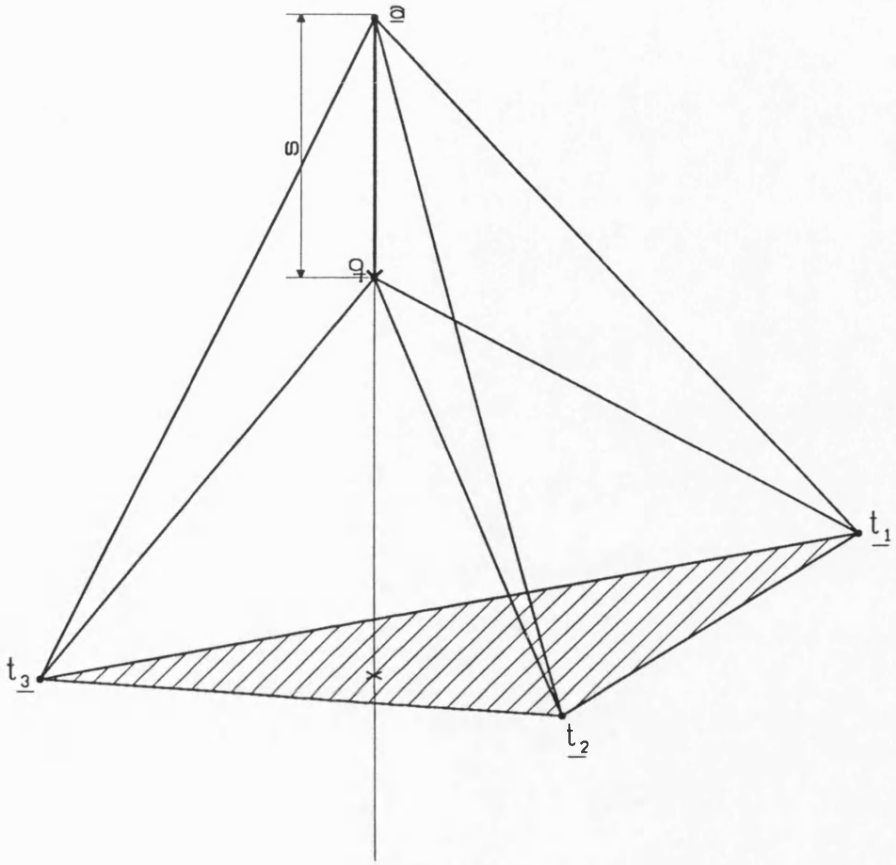


Figure 5.3 Ray-tracing in 2D. P_2 is the starting surface point and dotted lines represent the Voroni polyhedra.



$$V_1 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_1 - \underline{q}) \; (\underline{t}_2 - \underline{q}) \right]$$

$$V_2 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_2 - \underline{q}) \; (\underline{t}_3 - \underline{q}) \right]$$

$$V_3 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_3 - \underline{q}) \; (\underline{t}_1 - \underline{q}) \right]$$

Figure 5.4 Finding the intersection between the ray and the triangular face

1. Calculate the length, s, between $\underline{a}$ and the tetrahedron's centroid. $\underline{a}$ is the starting point of the ray (surface point).

$$s = \sqrt{(\bar{x} - x_0)^2 + (\bar{y} - y_0)^2 + (\bar{z} - z_0)^2}$$

$\bar{x}, \bar{y}, \bar{z}$ are the coordinates of the the centroid and x_0, y_0, z_0 are the coordinates of the starting point of the ray, $\underline{a}$, (origin of the line).

2. If s is equal to zero, then, the starting point is already on the triangular face, so return to the main program. Otherwise, set $\underline{q}$ to be a point s away from $\underline{a}$ on the line.

$$\begin{aligned} x &= x_0 + sf \\ y &= y_0 + sg \\ z &= z_0 + sh \end{aligned}$$

3. Subtract $\underline{q}$ from the triangle and $\underline{a}$ and calculate the determinants of:

$$V_1 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_1 - \underline{q}) \; (\underline{t}_2 - \underline{q}) \right]$$

$$V_2 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_2 - \underline{q}) \; (\underline{t}_3 - \underline{q}) \right]$$

$$V_3 = \left[(\underline{a} - \underline{q}) \; (\underline{t}_3 - \underline{q}) \; (\underline{t}_1 - \underline{q}) \right]$$

The V s are the signed volumes of tetrahedra calculated by vector products. If all the values of V are the same sign, the ray pierces the triangular face; if they are not, it does not.

There are always two intersections after the intersection with the first tetrahedron when the ray was traced backwards from the surface point to classify the tetrahedra. This statement is true unless the ray intersects an edge or a corner on its way. If the ray intersects an edge or a corner, the algorithm stops ray

tracing, skips this surface point after giving a warning of this sort of intersection and takes the next surface point for the next ray to be traced. This case is rare and skipping the surface point does not cause much problem in classification. Air tetrahedra which fail to be found by stopping the ray tracing can be found from the other surface points and the next algorithm to be described, which is capable of eliminating the remaining misclassified solid tetrahedra.

All the tetrahedra that the ray intersects are classified as air until the ray is beyond the convex hull. This process is repeated for every surface point and the majority of tetrahedra are thereby classified.

5.3 Eliminating the redundant tetrahedra

The classification algorithm, dependent on the complexity of the shape, categorises some of the tetrahedra as solid which are, in fact, air. This might happen when some tetrahedra are not visited by any of the rays re-traced from the surface points. These are the redundant tetrahedra and need to be eliminated to construct the actual shape of the object. In the measuring process any remaining ambiguities may be resolved by having the measuring machine (which is most useful if it is on line) take extra measurements which pass through the tetrahedra about which there is still doubt.

In the next sections two methods of eliminating these redundant tetrahedra will be described.

5.3.1 Eliminating the Long Flat Tetrahedra

The first method used in the author's research to eliminate the redundant tetrahedra was to find the tetrahedra (classified as solid by the previous classifying algorithm) which are long and flat and to eliminate them by categorising them as air. A flat tetrahedron is a tetrahedron whose volume is small when compared with its circumsphere. **Figure 5.5** shows this type of tetrahedron. This sort of situation is unlikely to happen in a real physical object and anyway, such an object would be impossible to manufacture. In order to find these tetrahedra, the triangular faces of each tetrahedron are checked to see whether any of them are long and flat. The algorithm which finds these faces calculates the area of the circle that passes through the vertices of a triangular face and compares this area with the area of the triangular face. The constant to be used in the comparison is calculated as (see **Figure 5.6**):

The area of an equilateral triangle is:

$$A_\Delta = (\frac{3\sqrt{3}}{4}) R^2$$

where $h = R + R\sin 30 = \frac{3}{2}R$ and $b = 2R\cos 30 = \sqrt{3}R$ and the area is $A_\Delta = \frac{bh}{2}$

The area of circumcircle is: $A_o = \pi R^2$

and the constant as calculated from these areas is:

$$x \cdot \frac{A_\Delta}{A_o} = 1 \qquad x = \frac{A_o}{A_\Delta} = \frac{\pi R^2}{\frac{3\sqrt{3}}{4} R^2}$$

$$x = \frac{4\pi}{3\sqrt{3}}$$

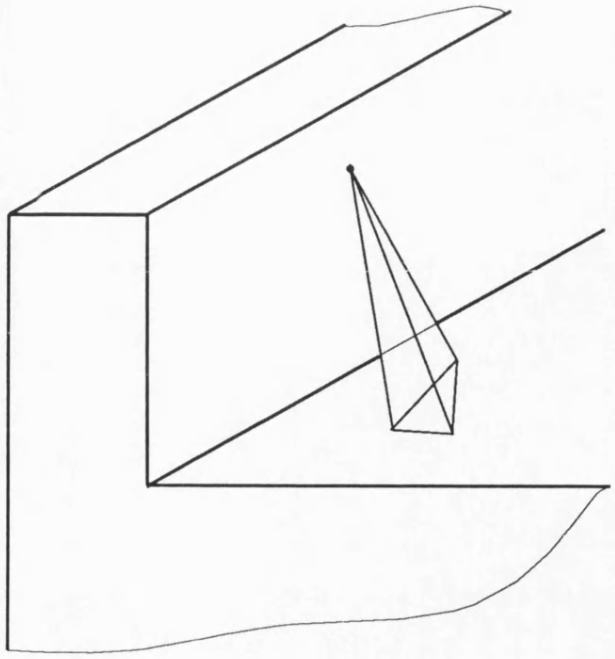


Figure 5.5 A long flat tetrahedron

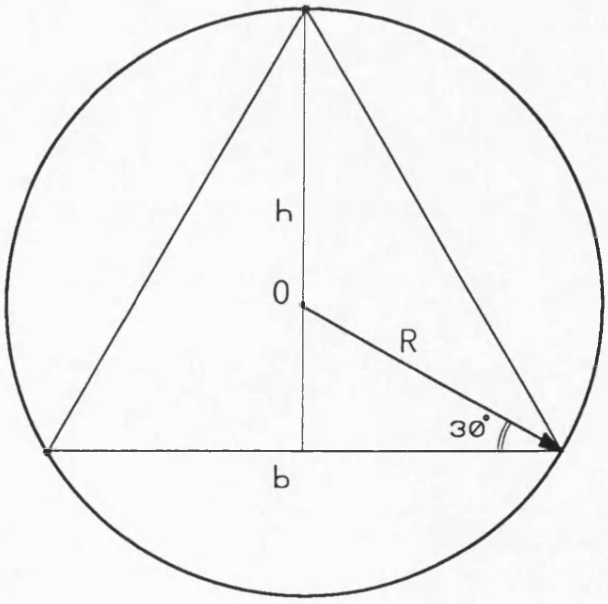


Figure 5.6 A triangular face and its circumcircle

The range of the constant in this formula is between 0 and 1. As the value gets closer to 0, the triangular face becomes much thinner and longer and as it gets closer to 1, the triangular face becomes more equilateral. The redundant tetrahedron has naturally three long and flat triangular faces. The algorithm finds the tetrahedra whose three of its four corresponding constants are close to 0 and one close 1 and eliminates them by classifying them as air.

Although this method excludes most of the superfluous tetrahedra, the second method described in the next section has been found to be more general and more suitable for the purpose.

5.3.2 The General Solution

The method explained in the previous section only eliminates the tetrahedra which are long and thin. But in some cases, there might be some tetrahedra with which this technique cannot cope (especially, tetrahedra on the edges). This sort of tetrahedron is not the long-thin type (see **Figure 5.7**), so, cannot be excluded by the technique described previously. In order to handle all sorts of redundant tetrahedra, a more general technique was developed.

This technique calculates the centroid of each solid tetrahedron, moves away at a given distance in positive and negative x,y,z directions consecutively, sends a ray of light from that distance onto the centroid and calculates the ray parameter at the intersection point. If the surface of the object is beyond the centroid, the tetrahedron is classified as air, otherwise it is solid. **Figure 5.8** shows this process on a simple example.

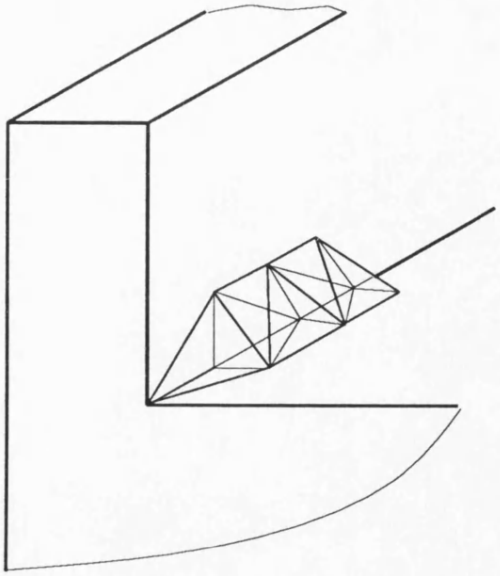


Figure 5.7 Redundant tetrahedra

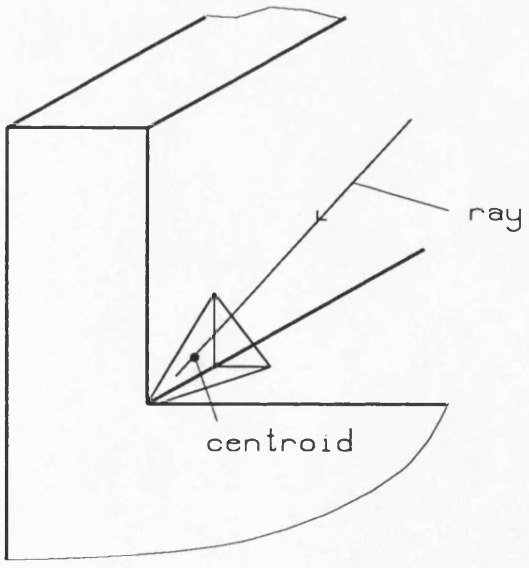


Figure 5.8 Elimination of redundant tetrahedron

This is actually what the measuring machine would do in real life. It takes some extra measurements either by a measuring probe or a laser beam which pass through the tetrahedra about which there is still doubt.

After the calculation of the centroid, a point at the given distance away in the x direction is chosen as the first starting point. A ray is sent onto the centroid, and the ray parameter at the intersection point of the ray with the surface of the object is calculated by running the solid modeller - DORA - for the second time (since the equations of the half-spaces which form the model are known, the determination of the ray parameter is very easy). If an intersection occurs before the centroid of the tetrahedron, this tetrahedron may be solid, if it does not, the tetrahedron is definitely air. In the case where the tetrahedron may be solid other measurements are made to try to classify its status. These are done in the remaining coordinate directions. If any of the intersection points are beyond the centroid of the tetrahedron, this tetrahedron is immediately categorised as air and the next solid tetrahedron is taken to be investigated. The flow chart of this algorithm is given in **Figure 5.9**.

5.4 Finding the Surface of the Object

The classification of the tetrahedra as solid or air allows the surface of the object to be found by finding the triangular faces of solid tetrahedra which form a boundary with air tetrahedra. In other words it facilitates the triangulation of the measured component's surface.

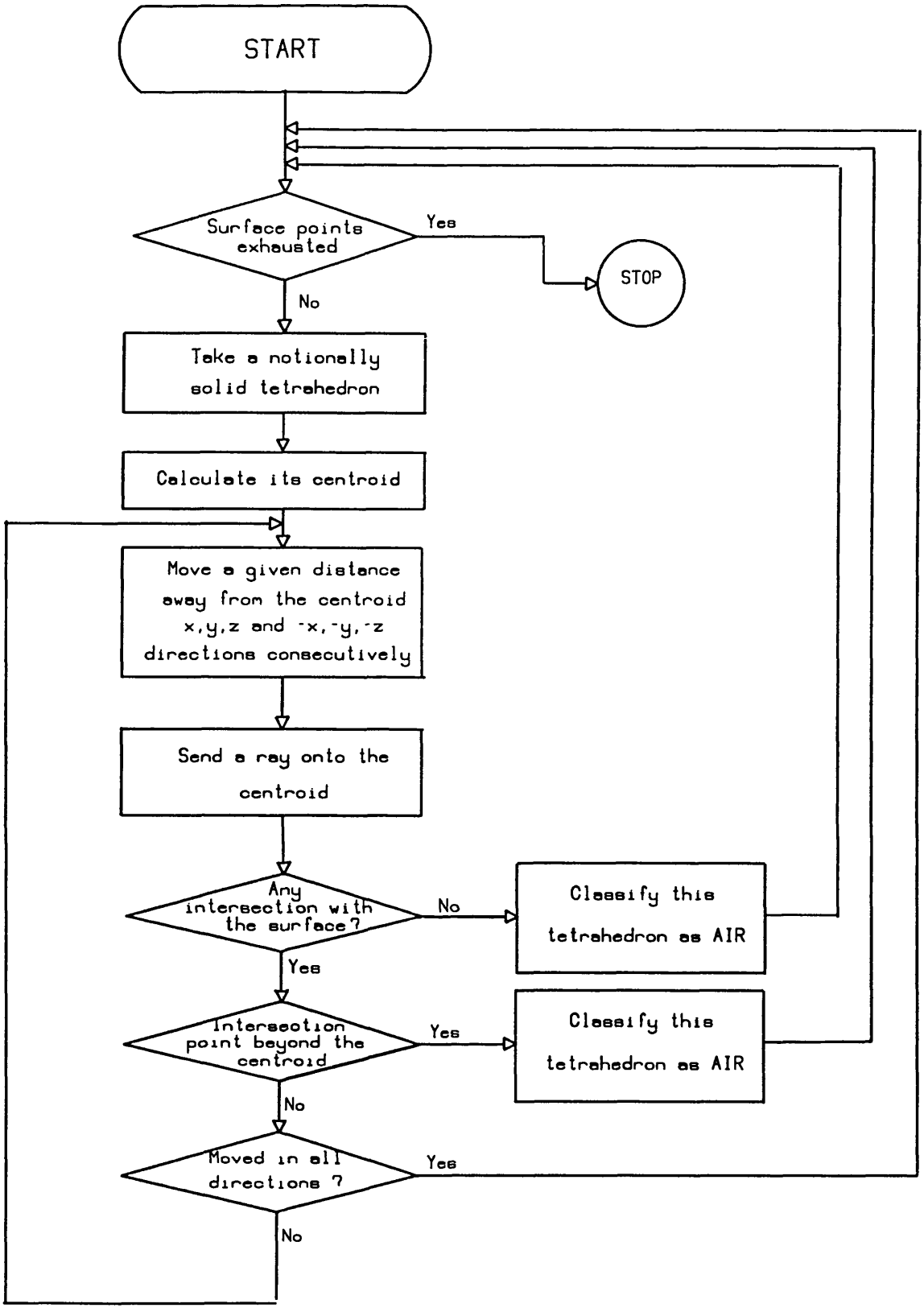


Figure 5.9 The method of eliminating the redundant tetrahedra

In order to find the neighbouring tetrahedra, the algorithm uses the neighbouring vertex information for each Delaunay vertex corresponding to each Delaunay tetrahedron. After the neighbouring tetrahedra which are air are found, the triangular faces of each tetrahedron that are shared by its air neighbours are added to the list of triangles which form the surface of the object.

The surface normal of each triangular face (which is needed for the algorithm to be described in Chapter 6) is calculated by using the vertex position information of each tetrahedron. As given in the definition of the Delaunay triangulation, Delaunay triangles are the perpendicular bisectors of Voronoi polygons. This means the boundaries joining the Voronoi vertices are perpendicular to the triangular faces of the Delaunay tetrahedra and represent the normals of the faces (see **Figure 5.10**). Since the position of each vertex has already been recorded in the triangulation process, the surface normals are calculated from the difference between the vertex position of the tetrahedron and its air neighbour's.

If the neighbouring vertex corresponds to a territorial boundary that extends to infinity (a zero-labelled vertex, see Chapter 3), since the neighbouring vertex is outside the convex hull no vertex position information is available. In this case the coefficients of the plane that the triangular face lies in give the normal of the triangular face and the plane coefficients are calculated as in section 5.2.

Furthermore, the algorithm finds the neighbouring triangles of each surface triangle. This is a very useful piece of information for the clustering process that will be described in Chapter 6. The algorithm checks the edges of each surface triangle to see which one of the other surface triangles shares each edge, and stores

the three neighbours of each triangle. The storage structure of the neighbouring triangles is shown in **Figure 5.11.**

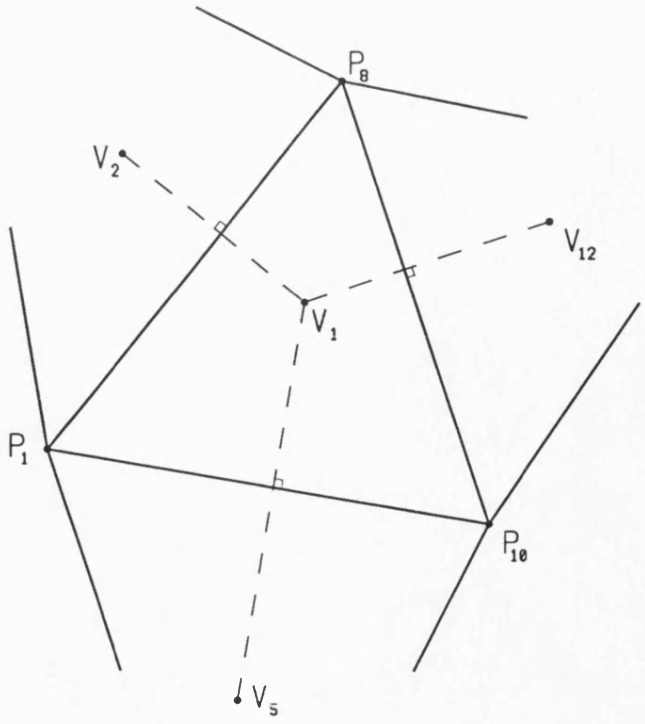
At the end of the process, the triangles lying on the boundary of the object have been obtained. The data structure (which will be used in the next process) contains the position of the forming points of the surface triangles, their normals, their neighbouring triangles and the vertex number of the corresponding tetrahedron to which they belong. This information is sufficient to determine the real faces of the object.

5.5 Limitation

As the number of surface points which the model is generated from are increased the structure is divided into smaller and smaller triangles and a better and better approximation of the surface is achieved. However, the increase in the number of data points increases computation time. For this reason, a compromise should be made between the approximation of the surface and the execution time. For instance, to process 1300 points took 13 minutes on a VAX 11/730, and this time was increased up to 39 minutes for 3200 points. Of course, running the software on a modern machine - such as a Sun 4 - would reduce these times radically.

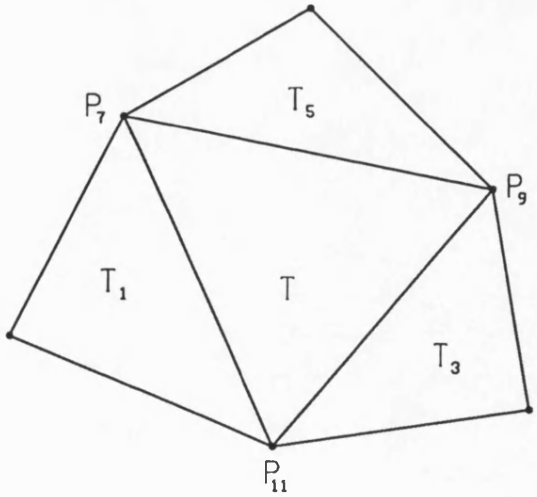
5.6 Concluding Remarks

In this chapter a technique to classify the tetrahedra as solid or air tetrahedra was introduced. This sort of classification allows not only the determination of the object's surface but all sorts of different calculations about the object as well, such



The normal to $\overline{P_1P_8}$ is: The position of V_2 - the position of V_1

Figure 5.10 Finding the normals of triangular faces



Forming points:
P_7 , P_9 , P_{11}

Neighbour triangles:
T_3 , T_1 , T_5

Neighbour triangles are stored opposite to the forming points

Figure 5.11 The neighbouring surface triangles of T

as its surface area, volume, centre of mass, moments of inertia and so on.

One of the aims in this project was to find the surfaces of the object. These would then be clustered into different sub-clusters each representing a real face of the object. For this reason, the three-dimensional structure (which is the aggregation of solid tetrahedra) is used only for finding the surface, but it is obviously available for different sorts of applications. Information about how to use both the triangulation algorithm and the classification algorithm can be found in Appendix A. The clustering technique will be explained in the next chapter.

CHAPTER 6

FINDING THE REAL FACES

6.1 Introduction

Once the surface of the object is determined, the next process is to find the real faces of the object. These will then be matched to the solid model primitives. As mentioned in Chapter 1, two problems are encountered in matching the measured component to the solid model primitives. The first problem arises from the fact that two differing descriptions of the same (or nearly the same) component must be compared. In order to make a comparison these two differing descriptions (the measured or generated data are in the form of the positions of surface points whereas the solid model primitives are in the form of half-spaces) should be in the same form.

In the previous chapters, the techniques used for processing the position of the surface point information were described. The aim of this processing was to form a volumetric model from the surface point information whose surface would provide the half-spaces to be used in the matching process. In this chapter some techniques for finding these half-spaces (only the ones which represent the real faces of the object generated from the measured data) will be described.

The second problem: that of matching of these faces with the solid model primitives, will be explained in the next chapter.

6.2 Finding the Faces

After carrying out the procedure described in Chapter 5, the surface of the object consists of the triangles lying on its boundary with the measured points as vertices. Since the faces of the object will be matched to the solid primitives, this surface information should be processed to find the faces. In order to find the real faces of the object, the surface triangles are gathered in collections, each collection representing a face of the component.

Each triangle forms a little plane in space. Even in one face all the triangles will not be exactly co-planar because of measurement errors. The triangles are subjected to cluster analysis to gather them together in collections representing faces. The aim is, indeed, to gather the surface points in collections and find the faces (half-spaces) that they are lying on. Since the surface points are the vertices of surface triangles, clustering the surface triangles is the same thing.

Two different types of clustering techniques have been applied to the surface triangles. The first clustering technique, SLINK [102], is an efficient clustering algorithm which is based on single-link or nearest neighbour cluster analysis. Before describing this technique some information should be given about cluster analysis in general [43, 56, 65, 93].

6.3 Cluster Analysis

As described by Hartigan [56] clustering is the grouping of similar objects. A clustering of a set is a partition of its elements that is chosen to minimise some

measure of dissimilarity and cluster analysis is a generic term for a group of techniques which produce classifications from initially unclassified data. For this reason, as observed from the definition, clustering techniques are ideal tools for the purpose of finding the object's faces. They may also be used in some other areas [7] such as finding the true typology, model fitting, prediction based on groups, data exploring to search for natural groupings in the data, data reduction to simplify the description of a large data set, generating hypotheses to be tested on future samples and so on.

Different types of clustering techniques have been introduced in the literature. It is not the intention here to give a review of cluster analysis techniques. Several attempts have been made at this (which is, in fact, a difficult task since the vast literature of the subject is scattered throughout journals from many different fields). Detailed information about these reviews is given in [43]. Since SLINK is based on the single-link method, amongst the clustering techniques only the single-link method (or the nearest neighbour method) will be described briefly in the next section.

The majority of clustering techniques uses a matrix of similarities or distances between the entities as an input for clustering. Therefore careful consideration is needed of the possible ways of defining these quantities. A similarity coefficient measures the relationship between two individuals, given the values of a set of p variates common to both. In general, similarity coefficients take values between 0 and 1. On the other hand, distance measures, which are different from similarity measures (though transformations between a set of distance function values and a

set of similarity function values is possible), can take any positive value.

As given in [43] a distance function $d(x,y)$ of pairs of points of a set E is said to be a metric for E if it satisfies the following conditions:

i. $d(x,y) \geq 0$; $d(x,y) = 0$ if $x = y$;
ii. $d(x,y) = d(y,x)$;
iii. $d(x,z) + d(y,z) \geq d(x,y)$

The third condition (which is the one which differentiates most between distance measures and similarity measures) is referred to as the *triangular inequality.*

The most widely used and the most familiar distance measure in clustering techniques is the Euclidean metric where the distance between points i and j denoted by d_{ij} is defined as

$$d_{ij} = \left\{ \sum_{k=1}^{p} (X_{ik} - X_{jk})^2 \right\}^{\frac{1}{2}}$$

where X_{ik} is the value of the kth variable for the ith entity.

In some cases Euclidean distance might be very unsatisfactory. This is because Euclidean distance is effected badly by scale changes in the variables. Some other possible metrics can be used in clustering. Examples are the absolute metric or the Minkowsky metrics. Their definitions are given in [43]. The distance measure used in this research will be explained in later sections.

6.3.1 The Single Link Clustering Method

The single link or nearest neighbour method is a type of hierarchical clustering method [43, 65]. For this reason, before explaining the structure of the single link

clustering method, some further information needs also to be given on hierarchical clustering techniques.

Hierarchical techniques are divided into two main groups: agglomerative methods and divisive methods [43]. Agglomerative methods proceed by a series of successive fusions of the N entities into groups where divisive methods partition the entire set of data into N groups each containing a single entity. The single link method is a type of agglomerative hierarchical method.

Agglomerative methods build a tree from leaves to root. They start clustering with the computation of a similarity or distance matrix between the entities, and end with a dendrogram showing the successive fusions of individuals which culminates at the stage where all the individuals are in one group. Different types of agglomerative methods have been described in [43]. Differences between the methods arise because of the different ways of defining similarity or distance between the groups of individuals.

The method of single link cluster analysis is the simplest of all hierarchical techniques. It may be applied with any associated similarity measure or distance measures. At each stage, after p and q have been merged, the similarity between the new cluster (which is labelled t) and some other cluster r is calculated as [3]:

1. If s_{ij} is a distance-like measure

$$s_{tr} = \min (s_{pr}, s_{qr})$$

s_{tr} is the distance between the two closest members of clusters t and r. If clusters t and r were to be merged, then for any entity in the resulting cluster the distance to

its nearest neighbour would be at most s_{tr}.

2. If s_{ij} is a similarity-like measure

$$s_{tr} = \max (s_{pr}, s_{qr})$$

s_{tr} is the similarity between the two most similar entities in clusters t and r.

The method is known as *single link* because groups are joined at each stage by the single shortest link between them. Although single link clustering is the simplest technique, it is incapable of delineating poorly separated clusters. However, if two clusters are moved farther apart then the method will distinguish between them quite well. **Figure 6.1** shows this where two clusters have their mutually closest members linked. The other problem in the single link method is that it is implemented with a stored data matrix and the storage requirement for the similarity matrix grows rapidly with the number of entities. **Figure 6.2** shows storage requirements for similarity matrices.

6.3.2 SLINK: An Efficient Single Link Clustering Algorithm

SLINK [102] which is an optimally efficient algorithm for the single link cluster method has been applied to the problem of clustering the surface triangles.

The SLINK algorithm carries out single link cluster analysis on an arbitrary symmetric non-negative dissimilarity coefficient (DC) read in value-by-value from an input stream and produces a representation of the resultant dendrogram.

As a result of the algorithm the pointer representation (see [102]) of N objects is converted into the packed form of a dendrogram. In general a dendrogram is a

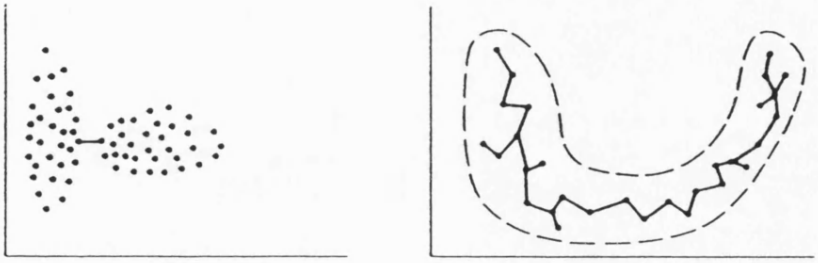


Figure 6.1 Single link clustering examples (from Anderberg [3])

STORAGE REQUIREMENTS FOR SIMILARITY MATRICES

Number of entities	Storage required	Number of entities	Storage required
50	1225	300	44,850
100	4950	350	61,075
150	11,175	400	79,800
200	19,900	450	101,025
250	31,125	500	124,750

Figure 6.2 Storage requirements for similarity matrices (from Anderberg [3])

nested sequence of partitions with associated numerical levels (which will be referred to *heights* from here on), the partition at a high enough height being the whole set. **Figure 6.3** shows a single link dendrogram and **Figure 6.4** shows the output of SLINK algorithm on a simple example which has only 6 surface triangles.

6.4 Application of the SLINK Algorithm to Surface Triangles

6.4.1 The Calculation of Dissimilarities

The surface of the object is the surface triangles with the measured points as vertices. In order to cluster these triangles to find the real faces of the object the SLINK algorithm is applied to them.

Each surface triangle lies on a little plane in space. If the coefficients of these planes are known (they can be calculated from the positions of the vertex points), then the dissimilarity coefficients of the surface triangles can be calculated. The aim here is to represent the planes as points on a hypercylinder in such a way that dissimilarity distances can be defined between them. The calculation of dissimilarities is as follows:

1. A normalised plane equation is represented as a point on a unit radius hypercylinder. The equations of the planes where the triangles lie are

$$a_i x + b_i y + c_i z + d_i = 0 \qquad i=1, n$$

where n is the number of surface triangles. Since the plane coefficients are normalised (which means $a^2 + b^2 + c^2 = 1$) each surface normal is a point on a

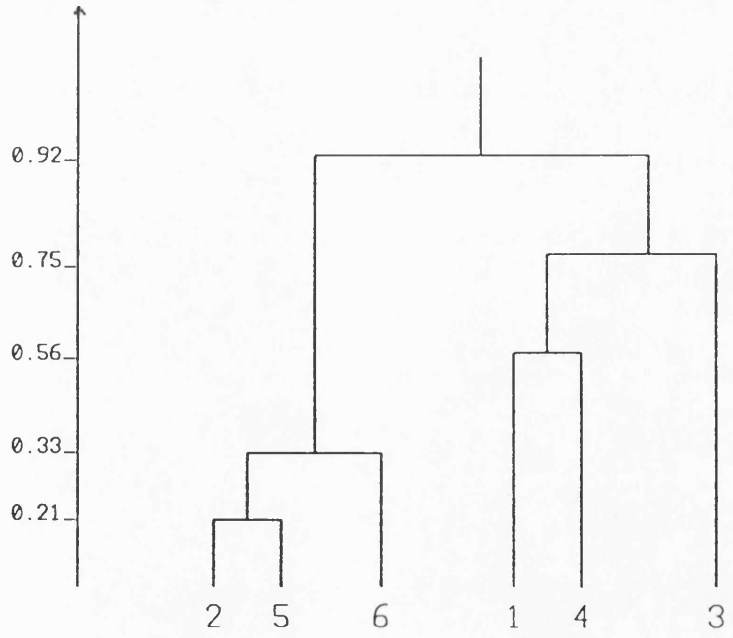


Figure 6.3 A single link dendrogram

```
Point 2   2.227E-01 -----+
                         |
Point 5   2.731E+00 ---------------------------------------------------------------------+
                         |                                                               |
Point 1   1.287E+00 ---------------------------------+                                   |
                         |                           |                                   |
Point 4   1.744E+00 ---------------------------------------------------+                 |
                         |                           |                 |                 |
Point 3   8.859E-01 ----------------+                |                 |                 |
                         |          |                |                 |                 |
Point 6   INFINITY  ---------------------------------------------------------------------------------
                         |          |                |                 |                 |
Links:                   |          |                |                 |                 |
                         |          |                |                 |                 |
P     2 - P     1   2.731E+00        |                |                 |                 |
P     3 - P     1   1.744E+00        |                |                 |                 |
P     4 - P     1   1.287E+00        |                |                 |                 |
P     5 - P     2   2.227E-01        |                |                 |                 |
P     5 - P     4   2.731E+00        |                |                 |                 |
P     6 - P     3   8.859E-01        |                |                 |                 |
P     6 - P     4   1.744E+00        |                |                 |                 |
```

Figure 6.4 Output of the SLINK algorithm

unit sphere which is the projection of a four-dimensional unit hypercylinder. The distance along the axis of this represents the d term in the equation above. Since drawing a hypercylinder is not possible, to understand the calculation of dissimilarities, consider a unit cylinder in 3D on which the lines $ax + by + d = 0$ are represented as points and a circle which is the projection of the cylinder (see **Figure 6.5**). This unit cylinder is analogous to the unit hypercylinder. **Figure 6.6** shows the unit sphere and two points (which are, actually, two planes in which the two surface triangles lie) p and q on it.

2. Cut the unit cylinder along the dotted line in figure 6.5, unwrap it and map the distances into a square whose sides are equal to π. The distances will then be

$$d_i' = (d_i - d_{\min}) \cdot \frac{\pi}{D}$$

where $d_{\max}$ and $d_{\min}$ are the maximum and minimum perpendicular distance of planes from the origin, $D = d_{\max} - d_{\min}$ and d_is are the distances before the mapping.

3. As seen from figure 6.5 the squared distance between the points p and q is equal to

$$\Delta_{pq}^2 = \Delta_{pt}^2 + \Delta_{tq}^2 \qquad \text{(Pythagoras theorem)}$$

$$\Delta_{pt} = \Theta \qquad \text{(as seen from the projection on the circle)}$$

and Θ is calculated from the scalar products between the points (which are, in fact, the planes in which the triangles lie) p and q as

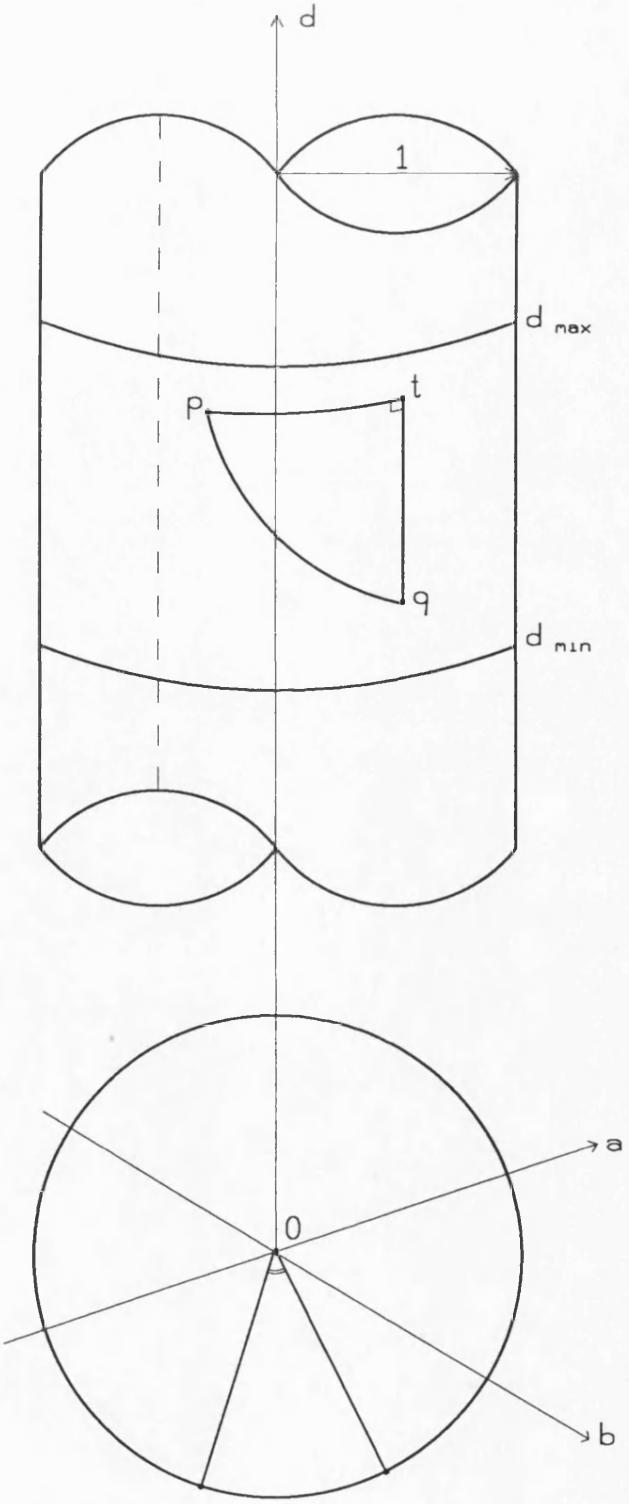


Figure 6.5 A unit cylinder and its projection

$$\Theta = \cos^{-1} ((a_p, b_p, c_p) \cdot (a_q, b_q, c_q))$$

If $\Theta > \pi$ then $\Theta = 2\pi - \Theta$. The squared distance between the points p and q is then equal to

$$\Delta_{pq}^2 = \Theta^2 + w \cdot (d_p^{'} - d_q^{'})^2$$

where w is the weighting factor to stretch the points if the distances are too close to each other.

The distances between the points on unit hypercylinder are used as the dissimilarity coefficients.

6.4.2 Determining the Clusters

The application of the SLINK algorithm to the calculated dissimilarity coefficients produces a resultant dendrogram with associated numerical levels for each cluster which can easily be converted into the usual tree-diagram. This information needs to be processed in order to determine the clusters each representing a real face of the object.

There are two ways of finding these clusters; either by defining a height (see section 6.3.2) on the tree-diagram or defining the number of clusters (which is easy since the shape and the number of the faces of the object is known). But first, some simple modifications need to be done to the output of the SLINK algorithm in order to form tree-diagrams.

The algorithm which deals with this problem sorts the heights first. It takes the smallest height value and finds the tag associated with this height. The tags

associated with each height value consist of the triangle numbers (surface triangles to be clustered) and this information is already provided by the SLINK algorithm. The algorithm links this tag with the next one (the next tag and the order of the tags are also known from the SLINK algorithm) and flags it by negating its sign in order not to use the same triangle again. The tag which has the second smallest height value is considered next, its next tag is checked to see whether its sign is negative or not; if it is not, the two surface triangles are linked together and so on. **Figure 6.7** shows the output of this sorting algorithm for the dendrogram given in figure 6.2. For the simplicity only six surface triangles were given as an input into the SLINK algorithm for this example.

The number of clusters and the height of the dendrogram (or tree-diagram) are related to each other. This makes two types of calculation possible; either the number of clusters can be determined by choosing the height in the dendrogram (the number of branches above the chosen height gives the number of clusters), or the height in dendrogram can be found by defining the desired number of clusters (which means a given number of branches determines the height). In order to understand these cases more clearly, consider the output of a cuboid model with 22 surface triangles and its tree-diagram in **Figure 6.8**. In this case a given height of 0.9 (which lies between 0.891 and 0.959) gives the desired number of clusters (which is 6 for a cuboid), or for the given 6 clusters, the value of 0.959 is encountered as the minimum height by the algorithm.

The algorithm which does this uses the link information between the triangles. As shown in figure 6.8, two different lists are formed after the sorting algorithm: a

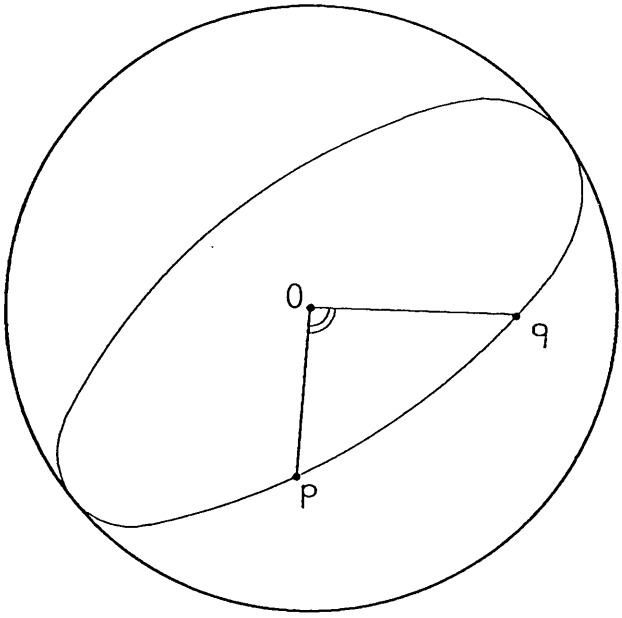


Figure 6.6 Unit sphere and two surface normals

```
Number of items :    6

    5      6      2.7309999
    4      6      1.7440000
    1      4      1.2869999
    3      6      0.8859000
    2      5      0.2227000
    6      0      0.0000000
```

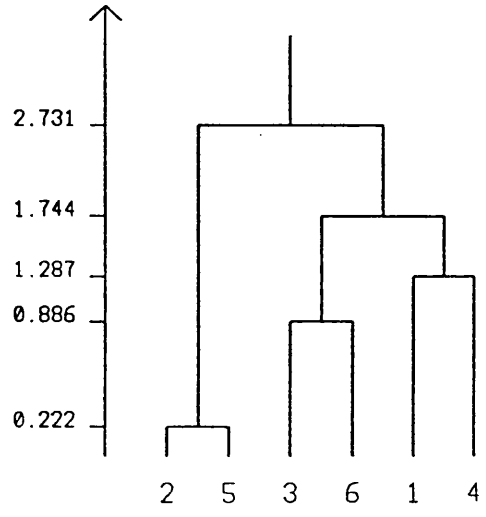


Figure 6.7 Output of the sorting algorithm and its tree-diagram

```
Number of items :   22

   14     22      1.2800000
    8     22      1.2390000
   12     22      1.2350000
   18     22      0.9654000
   16     18      0.9592000
   21     16      0.8915000
   20     21      0.7469000
   13     12      0.7305000
    5     20      0.6951000
   15     22      0.6903000
   10     13      0.6842000
    1     20      0.6832000
    2     20      0.6385000
    6      8      0.5658000
   11     14      0.2278000
    9     13      0.0201600
   17     18      0.0002427
    7     14      0.0001039
    4     10      0.0000731
    3      6      0.0000567
   19     22      0.0000312
   22      0      0.0000000
```

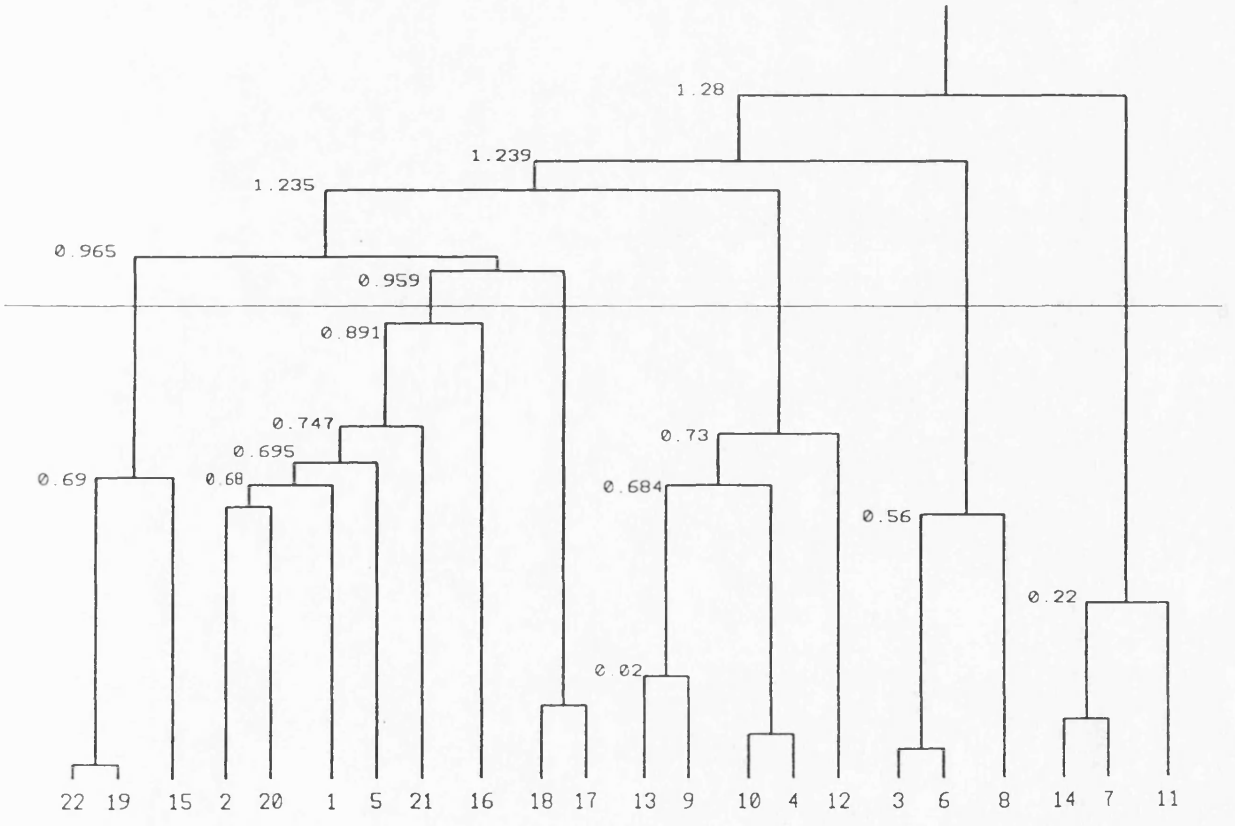


Figure 6.8 Tree-diagram of a model with 22 surface triangles

list of pair of surface triangles which are the elements of a two-dimensional array and a list of associated height values of each pair. The algorithm first asks the user to choose whether to find the height value or the number of clusters and according to the choice it branches.

Suppose that the height value is given and the number of clusters needs to be found. The algorithm determines the number of clusters by finding the number of surface triangles which are above the given height. Each triangle above the limit height forms a cluster (these clusters will be called main clusters). It then checks second elements of each pair below the limit height to find the triangles which have links with the main clusters. It takes the first main cluster and searches the second elements (triangles) of pairs to find the triangles which have links with this cluster. Whenever it finds a link which means the second element of the pair is the same as the triangle in the main cluster, it clusters the first element of the pair with the main cluster, negates the sign of the pair and does this until it finds the last linked triangle.

When the algorithm finds the last linked triangle (this means there are no more pairs whose second elements are the same as the triangle in the main cluster), it makes the same search for this triangle. This time it searches the second elements which are equal to the last linked triangle. After finding all the links of the last linked triangle, it searchs the lists backwards and finds the last negative signed pair, finds all the links of the first element of this pair, again searchs list backwards, finds the next last negative signed pair and so on. After clustering all the triangles which are linked to the first main cluster, the same process is repeated for

the other main clusters and all the surface triangles are clustered.

More or less the same process is done if the number of clusters are given. The main clusters and the triangles belonging to these clusters are determined according to the given number of clusters and the same process is repeated for the rest of the triangles. The flowchart of the algorithm is given in **Figure 6.9**.

At the end of the process the clusters of surface triangles (and the surface points obviously) each representing a real face of the object is found. Since the position of surface points in each cluster is known, the planes (half-spaces) which the surface points lie are found by using *principle component analysis* [61] to fit the planes into the surface points in each cluster. Principle component analysis is a technique which takes a cloud of points and finds the three axes of an ellipsoid which closely matches the cloud shape. The two longest axes of the ellipsoid determines the plane in which the points lie and the shortest axis determines the normal to the plane. This technique minimises the sum of the squares between the points and the plane. At this stage the measured object and its solid model are in the same form and ready to be matched.

Although the SLINK algorithm is an efficient algorithm for clustering the surface points and produces very good results for a small number of points, large numbers of surface points exceeded the memory capacity of the computer at the preparation stage of the dissimilarity matrix. For this reason, a different type of clustering algorithm has been developed. This algorithm will be explained in the next section.

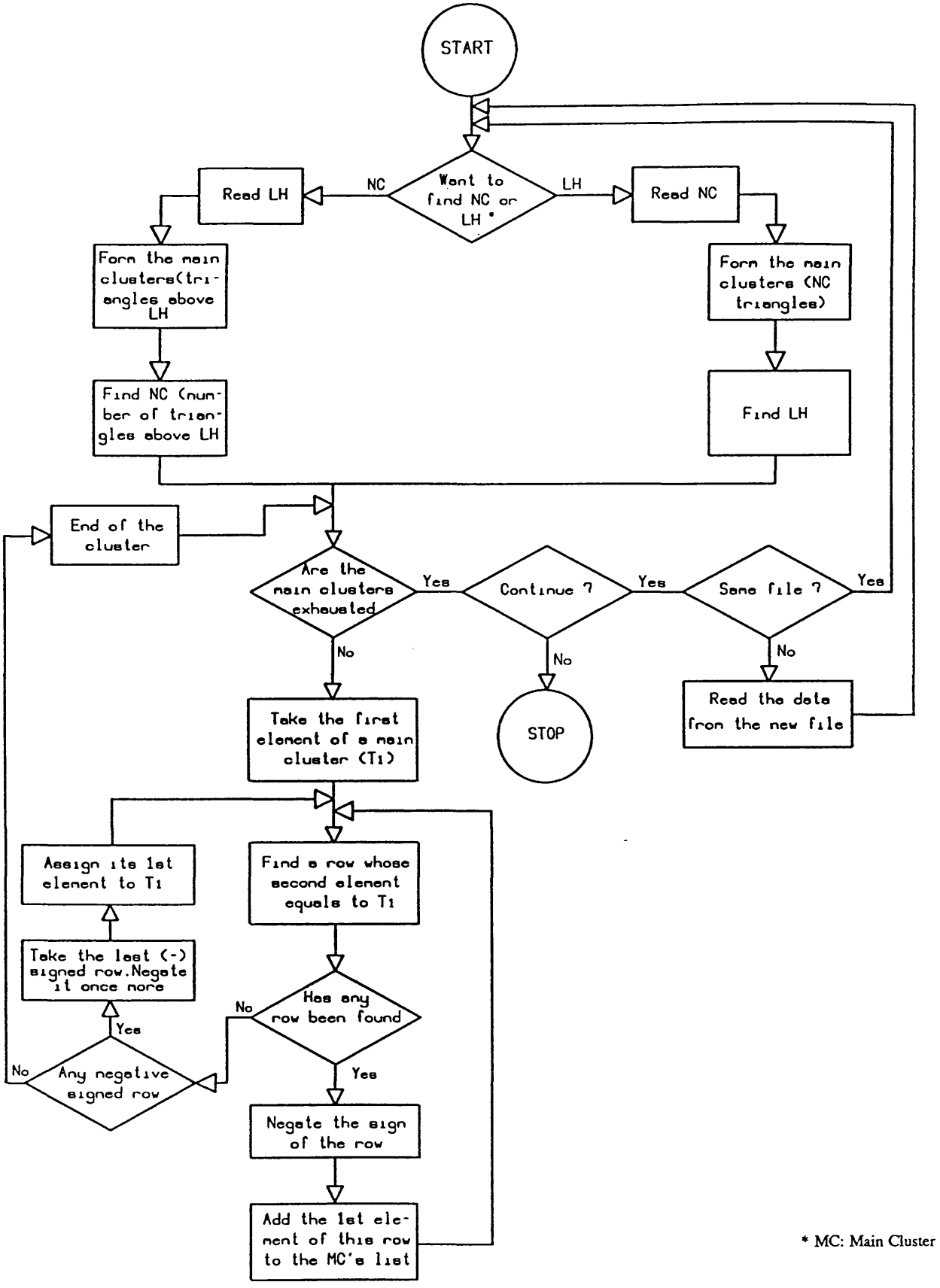


Figure 6.9 Finding the clusters

6.5 Clustering by Using the Surface Normals

In this technique surface normals are used for cluster analysis (as mentioned earlier each surface triangle forms a little plane in space). The algorithm reads in the surface normal information, calculates the scalar product of surface normals of two triangles and checks the result. If the result is greater than a number which is close to 1, the two triangles are put into the same cluster. Since the surface normals are normalised the scalar product equals the cosine of the angle between the planes in which the triangles or the surface points lie. If the result is close to 1, this means that the angle is close to 0 and the triangles lie on the same plane or on parallel ones.

The algorithm distinguishes the triangles lying on parallel planes by checking the perpendicular distance from their plane to the origin (the d term in their implicit plane equation, $ax + by + cz + d = 0$). It defines a mid-plane between the two parallel planes and clusters the triangles which are more distant from origin than the mid-plane in one cluster and the ones which are less distant in another. In the case of more than two parallel planes, first the parallel planes are split into two clusters, each cluster is checked to see whether they contain more than one plane, if they do, they are split again and checked again and the same process is repeated recursively until the triangles lying in each parallel plane are clustered in separate clusters. **Figure 6.10** shows two parallel planes and their mid-plane.

Since the neighbourhood relationship between the surface triangles (that is to say, for any given triangle, its three neighbouring triangles are known) is also provided (see section 5.4), any mis-clustered triangle is corrected by checking its

neighbouring triangles. If all three neighbours are in the same cluster but the triangle is not, it is put into the same cluster with its neighbours. If two neighbours are in the same cluster then the surface normals of all three neighbours are inspected and weighting is used to decide if the central triangle ought to be clustered with the pair or not. Now, consider the structure in **Figure 6.11.** If two neighbouring triangles are in the same cluster (say cluster 4) and the third one is the different one (say cluster 6), the decision on classifying the central triangle is made as follows :

1. Take the average of surface normals of two neighbouring triangles which are in the same cluster (N_{AV} in figure 6.11) and calculate the angle (Θ) between the average normal and the surface normal of the central triangle. Since the surface normals are normalised, the angle Θ is

$$\Theta = \cos^{-1} ((a_{AV}, b_{AV}, c_{AV}) \cdot (a_{CT}, b_{CT}, c_{CT}))$$

where a_{AV}, b_{AV}, c_{AV} are the coefficients of the average normal and a_{CT}, b_{CT}, c_{CT} are coefficients surface normal of the central triangle. The angle (Φ) between the third neighbour and the central triangle is

$$\Phi = \cos^{-1} ((a_{TN}, b_{TN}, c_{TN}) \cdot (a_{CT}, b_{CT}, c_{CT}))$$

where a_{TN}, b_{TN}, c_{TN} are the coefficients of the third neighbour's surface normal.

2. If the ratio of Θ to Φ is smaller than a weighting factor (which is 2 at the start) then the central triangle is clustered with its two neighbours, otherwise with the third one. As the weighting factor becomes larger, the surface normal of the central triangle becomes closer to the average surface normal.

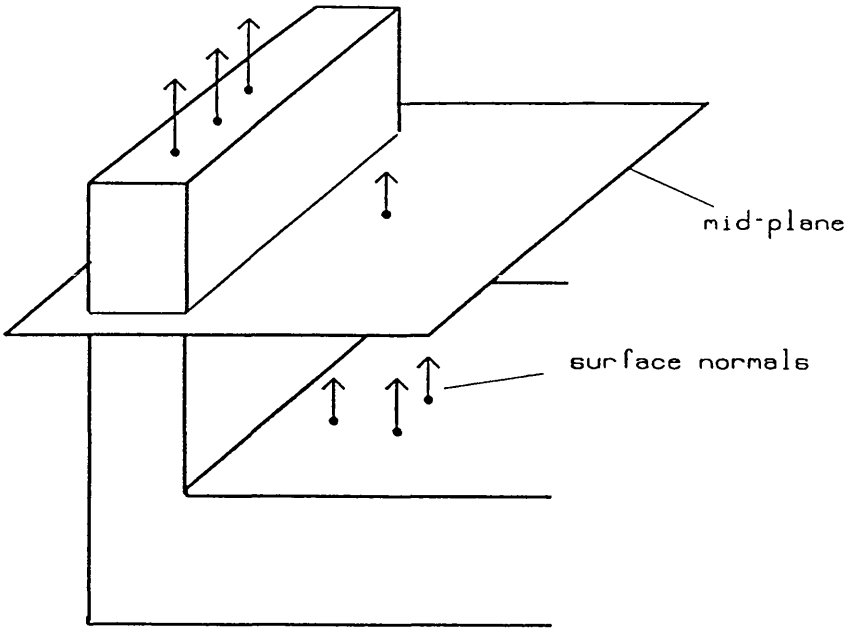


Figure 6.10 Two parallel planes and their mid-plane

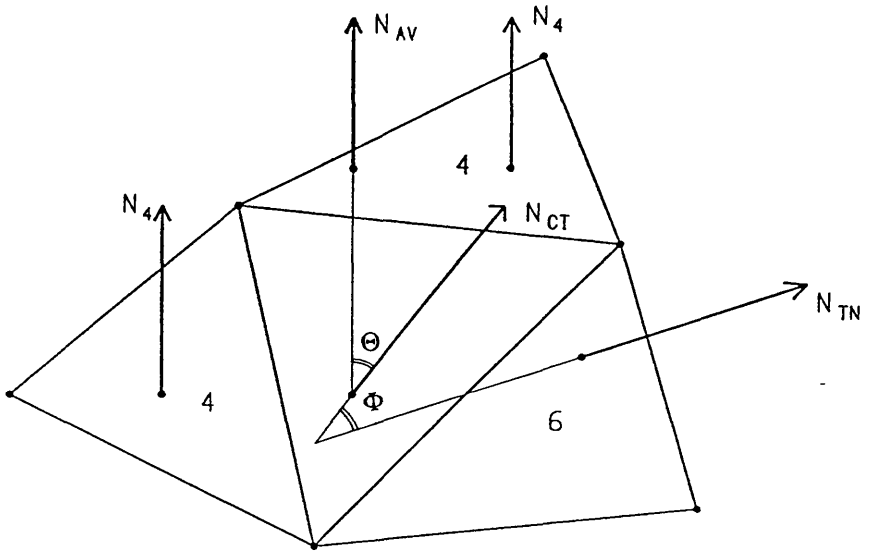


Figure 6.11 A surface triangle and its neighbours

If three neighbours of the central triangle are in different clusters, no change is made in central triangle's status.

Next, the algorithm handles *false chamfers*. False chamfers are artificial features which occur because of the lack of the ability of measuring machine to generate points exactly on the measured object's edges (**Figure 6.12**). They are a product of the measuring system and need to be found to be got rid of. The algorithm checks all the clusters and finds their boundary by checking the surface triangles in the cluster. If two of the neighbours of a triangle are in the same cluster as the triangle but one neighbour is in a different cluster, this means two forming points (surface points) of this triangle are on the boundary. Since the false chamfers should be one triangle wide and all of the surface points in a false chamfer are on its boundary (see **Figure 6.13**), the algorithm finds the clusters which are one triangle wide, classifies these as false chamfers and does not consider them to be real faces in the matching process. The numbers on the triangles and on the faces in figure 6.13 are the cluster numbers that the triangles belong to. For simplicity only the triangles on the front face are shown.

After correcting the misclustered triangles and finding the false chamfers, the algorithm re-organises each cluster. At this stage the algorithm is capable of producing a topology between the clusters. It takes each cluster in turn, finds the false chamfers to which each cluster is adjacent and finds the clusters which share these false chamfers. Since the false chamfers are effectively represent the edges of the measured object the clusters which share the false chamfers should be neighbours. By finding the neighbours of each cluster the algorithm produces the

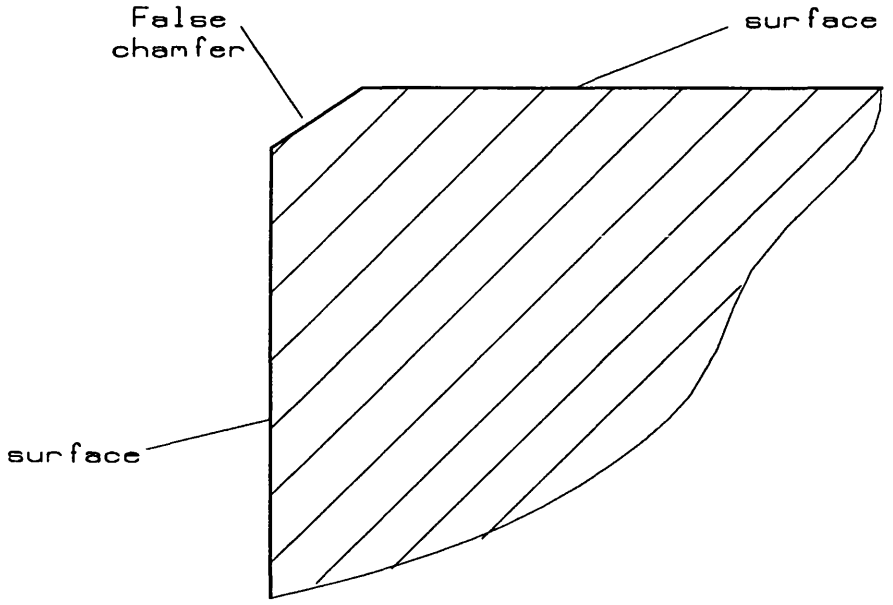


Figure 6.12 A false chamfer

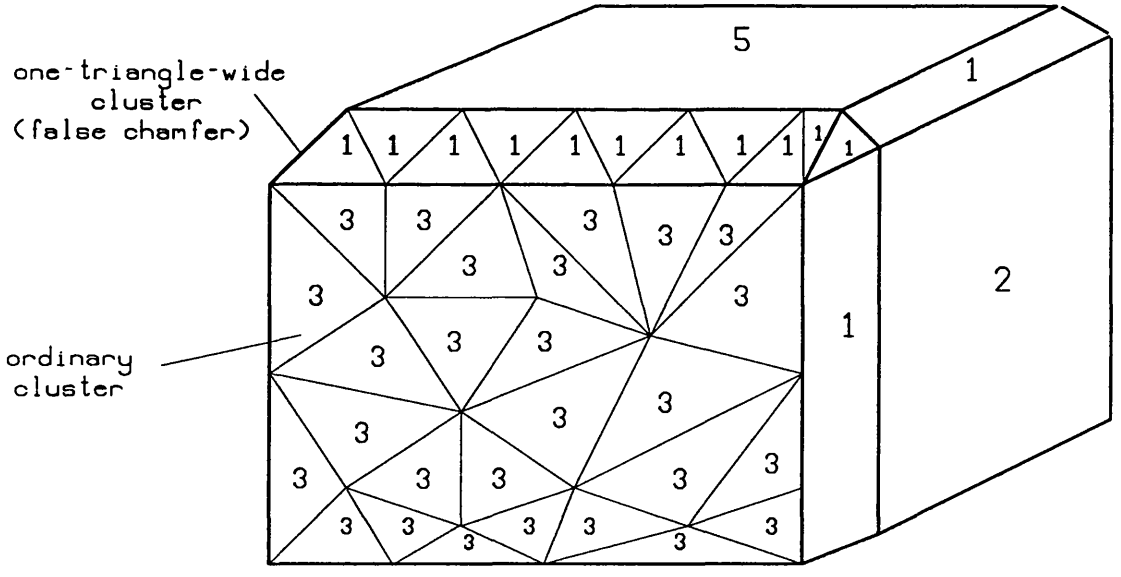
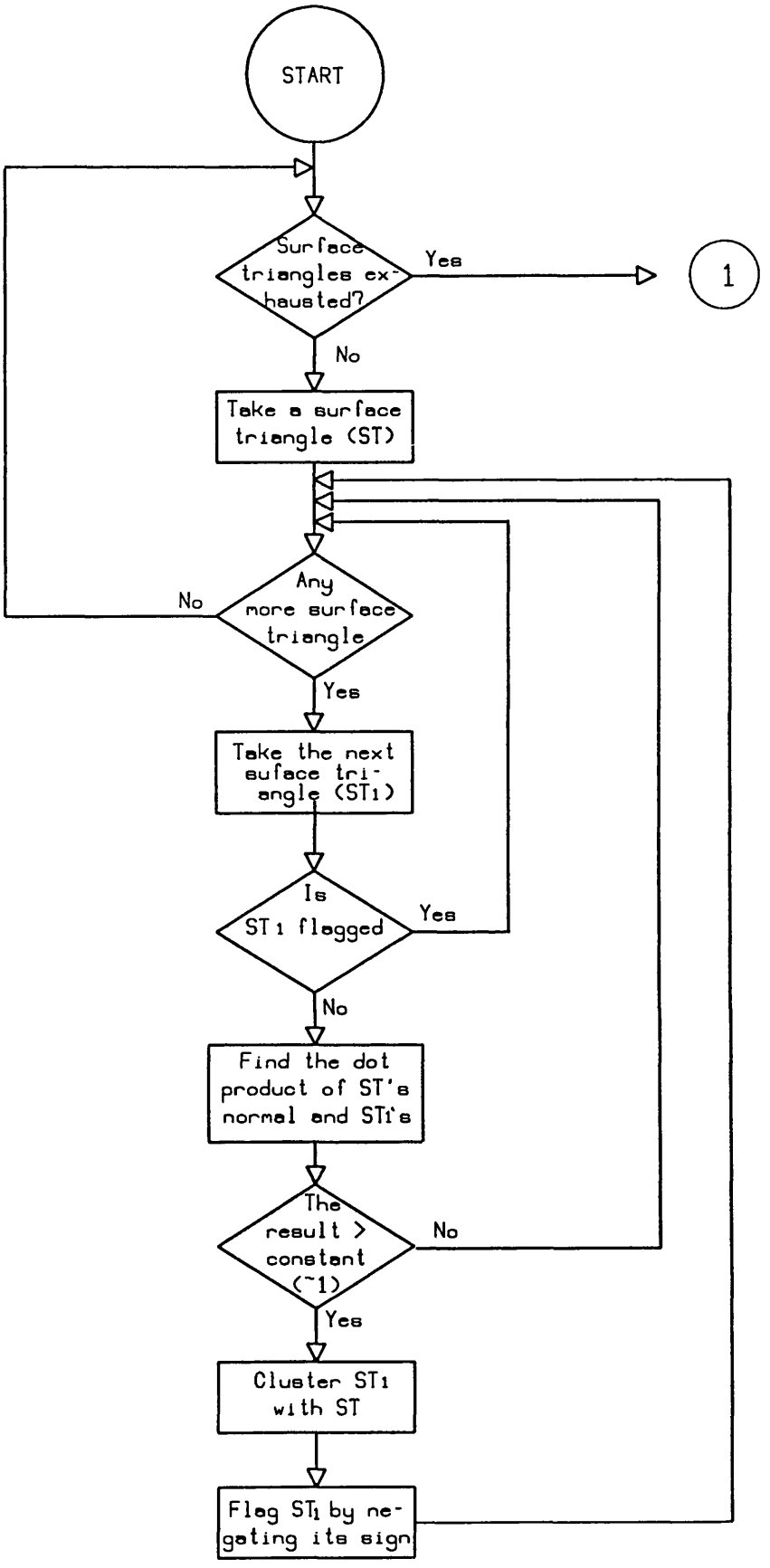


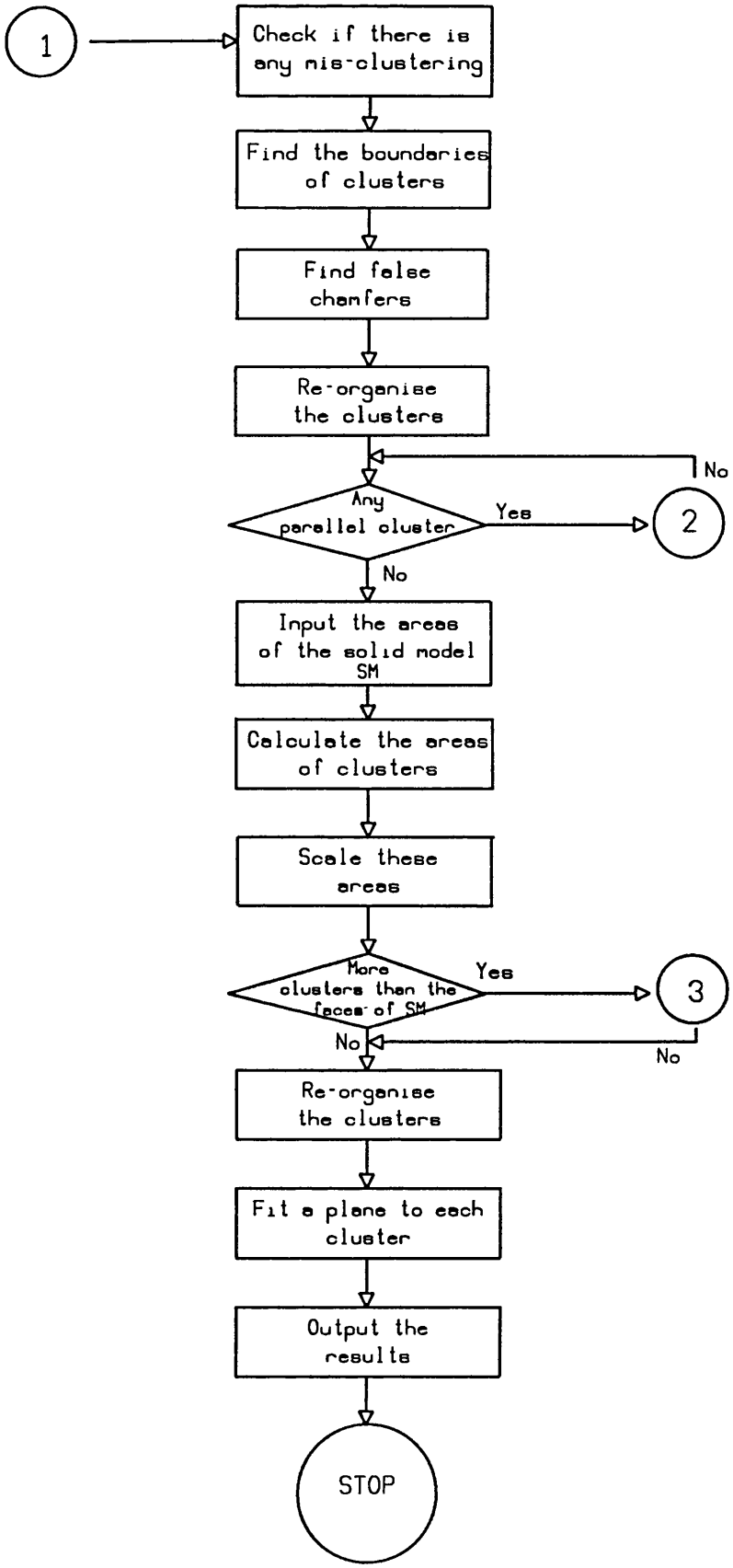
Figure 6.13 An ordinary cluster and a one-triangle-wide cluster

neighbourhood relationship between the clusters which allows a boundary model of the measured component to be generated. The flowchart of the clustering algorithm is given in **Figure 6.14.**

Once the clusters are formed their surface area (each cluster represents a face of the measured component) can easily be calculated (the surface area of each face is the sum of areas of triangles that form the cluster). This surface area information will be used in the matching algorithm to be described in Chapter 7. Since the number of faces of the solid model and their surface areas are known, any cluster which has much smaller area than the minimum of the surface areas of the solid model is doubtful. The algorithm checks these sorts of clusters and investigates whether they are false chamfers, or part of some other clusters, or to see if the face is not well-represented because of an unsufficient number of surface points. If the number of clusters found by the algorithm is more than the number of faces of the solid model, it is obvious that some of these clusters are doubtful. In this case the algorithm fits a plane to each cluster, calculates the angle between the plane which the doubtful cluster lies and any other planes, if the result of the scalar product - which is equal to the cosine of the angle - is greater than or equal to the constant used to cluster the surface triangles, it merges the doubtful cluster with the other cluster. After trying all the other clusters, if no merging is possible (which means the result of scalar product is smaller than the constant for all other clusters), the doubtful cluster is classified as false chamfer. The algorithm re-organises the clusters and calculates the number of clusters. The number of clusters is now obviously equal to the number of the faces of the solid model.


START
Surface triangles exhausted?
Yes
1
No
Take a surface triangle (ST)
Any more surface triangle
No
Yes
Take the next suface triangle (ST1)
Is ST1 flagged
Yes
No
Find the dot product of ST's normal and ST1's
The result > constant (~1)
No
Yes
Cluster ST1 with ST
Flag ST1 by negating its sign

Cont'd...


1
Check if there is any mis-clustering
Find the boundaries of clusters
Find false chamfers
Re-organise the clusters
Any parallel cluster
Yes
No
2
Input the areas of the solid model SM
Calculate the areas of clusters
Scale these areas
More clusters than the faces of SM
Yes
No
3
No
Re-organise the clusters
Fit a plane to each cluster
Output the results
STOP

Cont'd...

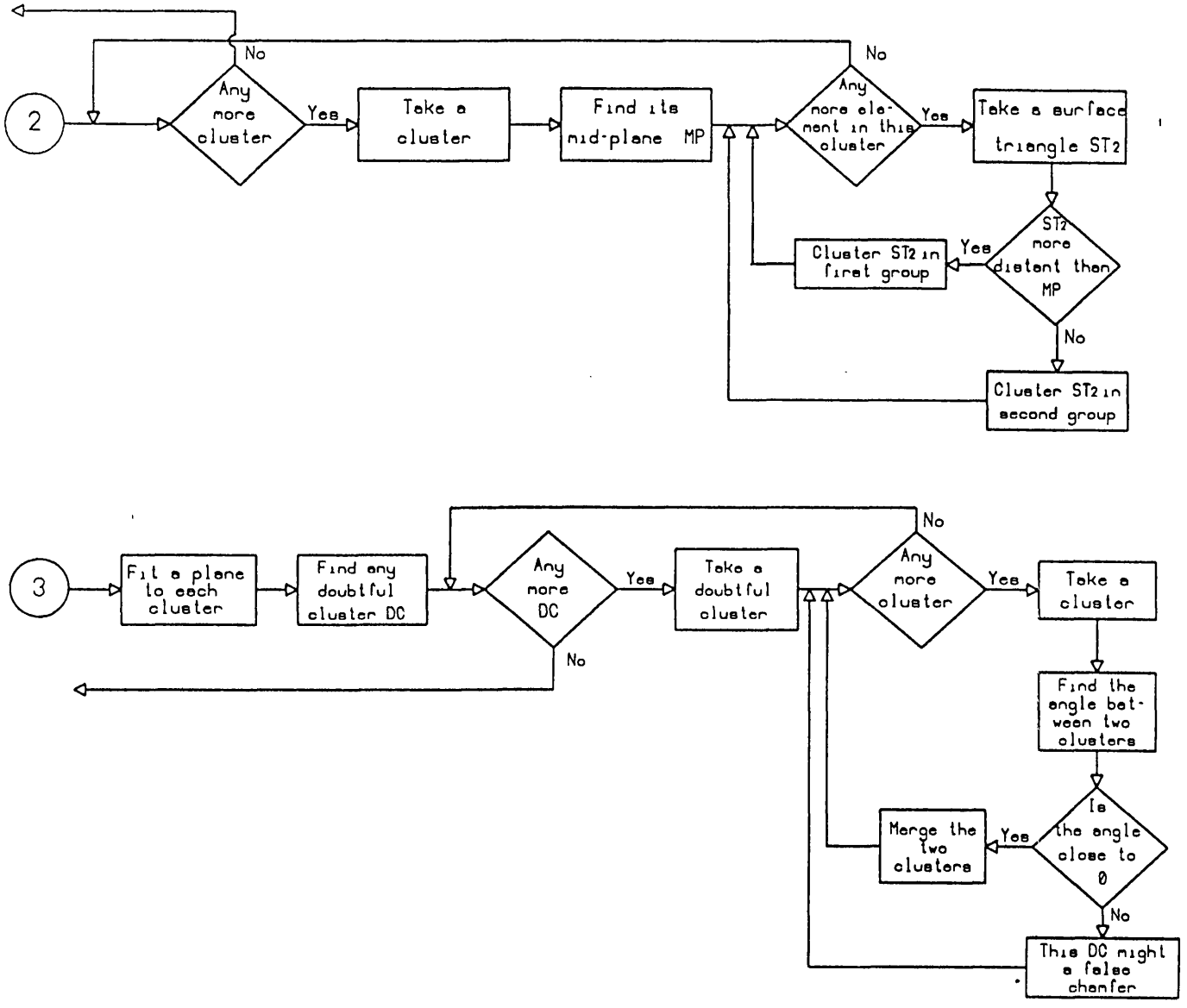


Figure 6.14 The flowchart of the clustering algorithm

As a result of all the processes the surface triangles (and obviously the surface points) are clustered together in different clusters, each cluster representing a face. Principle components analysis (see section 6.4.2) is then used on the measured points making up the triangle vertices in each collection to obtain a best-fit plane through them.

To see how to use the clustering algorithm see Appendix A. The input and the output of the clustering algorithm is given in **Figure 6.15**. The input consists of a list the positions of surface points forming each surface triangle, the normals of the planes in which each surface triangle lies, the neighboring triangles of each surface triangle and the number of the Delaunay vertex to which each triangle belongs. After the clustering only the cluster numbers of the surface triangles are added to this input to produce an output for plotting the surface triangulation and clustering. The second output consists of a list of neighbouring clusters of each clusters and the equations of planes which pass through the surface points in each cluster. For plotting an implementation of the painter's algorithm (or Newell-Newell-Sancha algorithm) [55, 95, 110] was used. **Figure 6.16** and **Figure 6.17** shows the result of surface triangulation and clustering on two simple examples. Different colours represent different clusters, which are, in fact, the faces of the object. The green colour represents the false chamfers. In figure 6.16 (b) and figure 6.17 (b) samples are shown from four different views.

Forming Point 1			Forming Point 2			Forming Point 3			Surface Normal			Neighbour Triangles			Assoc. Vertex
0.016	-0.001	5.499	0.018	0.000	5.697	0.016	0.000	5.900	4.764	-12.432	0.024	4	2	37	15810
0.000	0.007	5.982	0.016	-0.001	5.499	0.016	0.000	5.900	-2.186	-5.309	0.016	1	8	38	15810
0.363	0.000	5.699	0.363	0.000	5.899	0.016	0.000	5.900	-0.008	-23.460	-0.008	8	4	11	43
0.363	0.000	5.699	0.018	0.000	5.697	0.016	0.000	5.900	-0.024	-23.256	-0.035	1	3	39	43
-0.001	0.224	5.815	0.000	0.007	5.982	0.000	0.322	5.979	-26.877	0.020	0.196	7	102	6	20821
-0.001	0.224	5.815	0.000	0.017	5.791	0.000	0.007	5.982	-24.119	0.151	-0.044	38	5	41	13809
0.080	0.379	6.000	0.000	0.007	5.982	0.000	0.322	5.979	-2.527	0.086	9.482	5	99	12	20753
0.000	0.007	5.982	0.363	0.000	5.899	0.016	0.000	5.900	-0.008	-61.091	5.173	3	2	10	20752
0.709	0.000	5.699	0.709	0.000	5.900	0.363	0.000	5.899	0.010	-24.776	0.004	10	11	14	20797
0.000	0.007	5.982	0.709	0.000	5.900	0.363	0.000	5.899	0.006	-51.212	4.394	9	8	16	1824
0.709	0.000	5.699	0.363	0.000	5.699	0.363	0.000	5.899	0.003	-24.687	-0.008	3	9	43	20798
0.514	0.032	5.999	0.080	0.379	6.000	0.000	0.007	5.982	-0.296	-0.388	9.335	7	17	13	1614
0.501	0.413	6.000	0.514	0.032	5.999	0.080	0.379	6.000	0.004	-0.013	9.504	12	700	18	20801

Figure 6.15 (a) The input to the clustering algorithm

```
Neighbouring clusters :

Cluster   1 :   False chamfer
Cluster   2 :  12,   5,   3,  15,  11,   9,   6,  10,  14,  13,   4,   8,
Cluster   3 :   2,   6,   4,   8,   7,  12,
Cluster   4 :   7,   3,   2,   6,
Cluster   5 :   2,  12,  15,   7,
Cluster   6 :   3,   2,   4,   7,
Cluster   7 :   4,  12,  15,  11,   9,  10,  14,  13,   5,   3,   6,   8,
Cluster   8 :   2,   3,  13,   7,
Cluster   9 :  11,   2,   7,  10,
Cluster  10 :   9,   2,  14,   7,
Cluster  11 :   2,   9,  15,   7,
Cluster  12 :   2,   5,   7,   3,
Cluster  13 :  14,   2,   8,   7,
Cluster  14 :  10,   2,  13,   7,
Cluster  15 :   2,   5,  11,   7,

Plane Coefficients of each cluster :      Ax + By + Cz + D = 0

                      A          B          C          D
                   -------    -------    -------    -------
Cluster   2 :       0.0000     1.0000    -0.0001    -0.0001
Cluster   3 :       1.0000     0.0001     0.0000    -0.0005
Cluster   4 :      -0.0001     0.0000     1.0000    -0.9997
Cluster   5 :       1.0000     0.0000     0.0000    -2.0000
Cluster   6 :       0.7084     0.0000     0.7058    -2.8253
Cluster   7 :       0.0000    -1.0000     0.0000     6.0001
Cluster   8 :      -0.0001     0.0065     1.0000    -0.0241
Cluster   9 :      -0.0001     0.0000     1.0000    -3.9997
Cluster  10 :       1.0000     0.0000     0.0000    -5.9999
Cluster  11 :       1.0000     0.0000     0.0009    -4.0042
Cluster  12 :      -0.0015     0.0000     1.0000    -5.9983
Cluster  13 :       1.0000    -0.0001     0.0001    -7.9999
Cluster  14 :       0.0000     0.0000     1.0000    -2.0000
Cluster  15 :       0.0002     0.0000     1.0000    -5.0006
```

Figure 6.15 (b) The output of the clustering algorithm

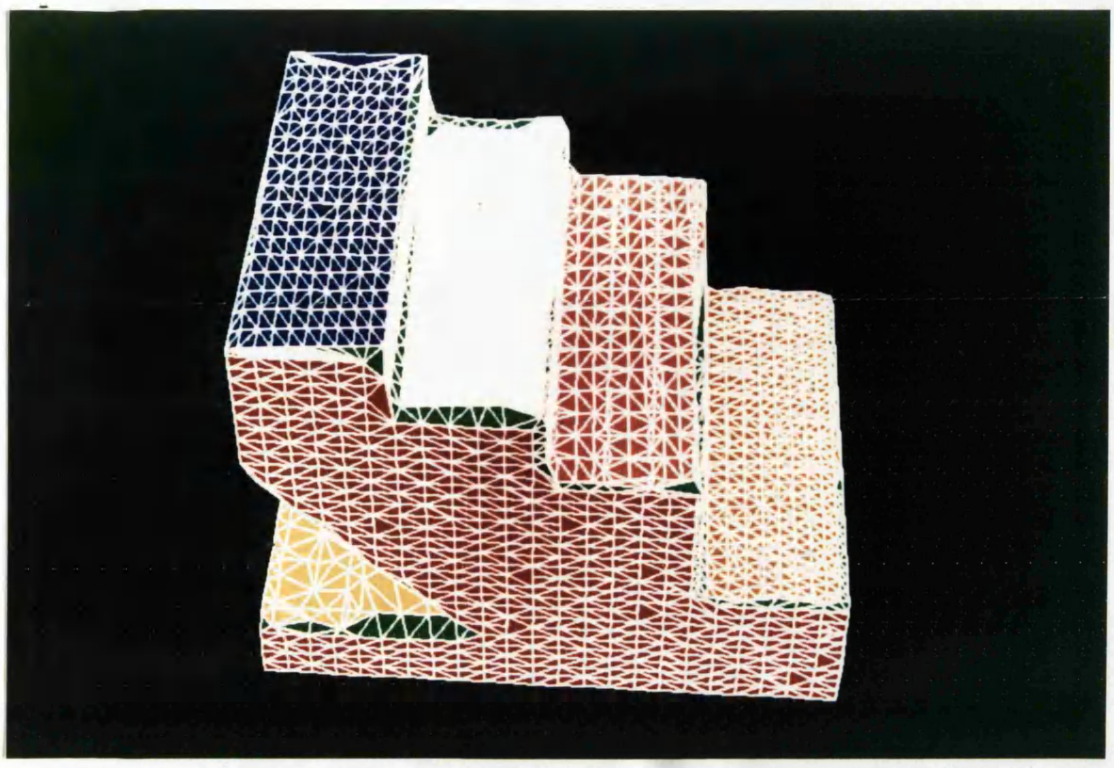


Figure 6.16 (a) Surface triangulation and clustering

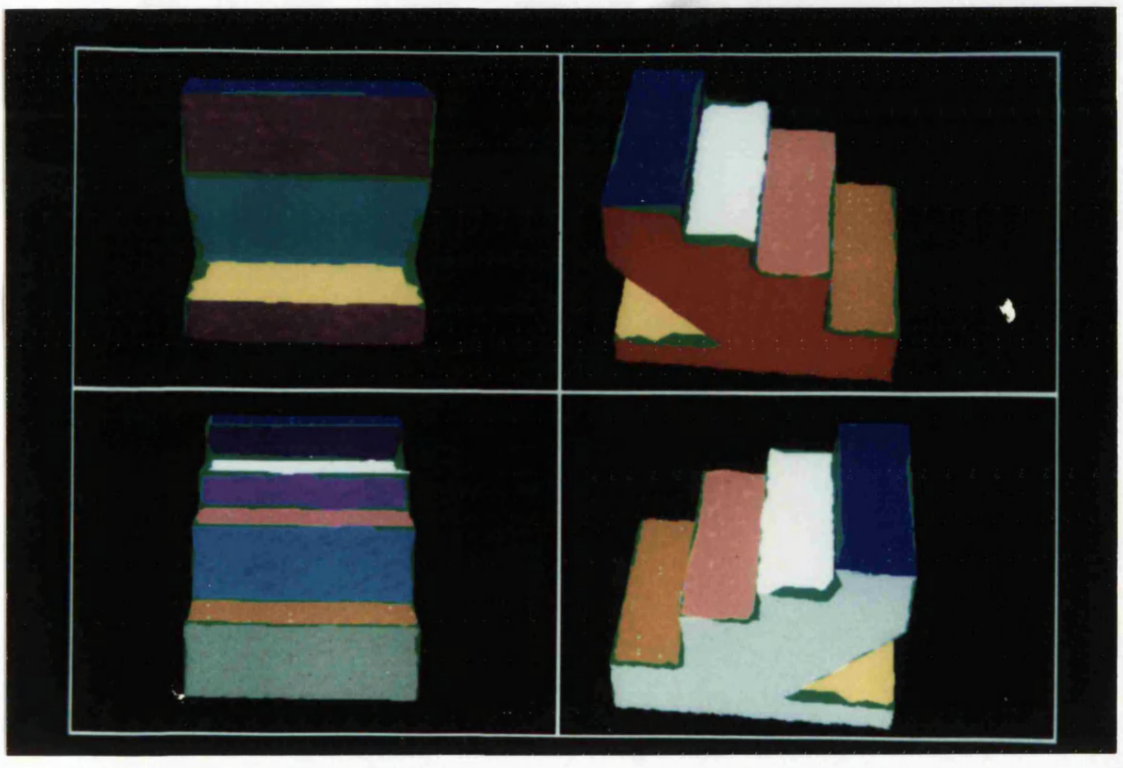


Figure 6.16 (b) Clustering the surface triangles

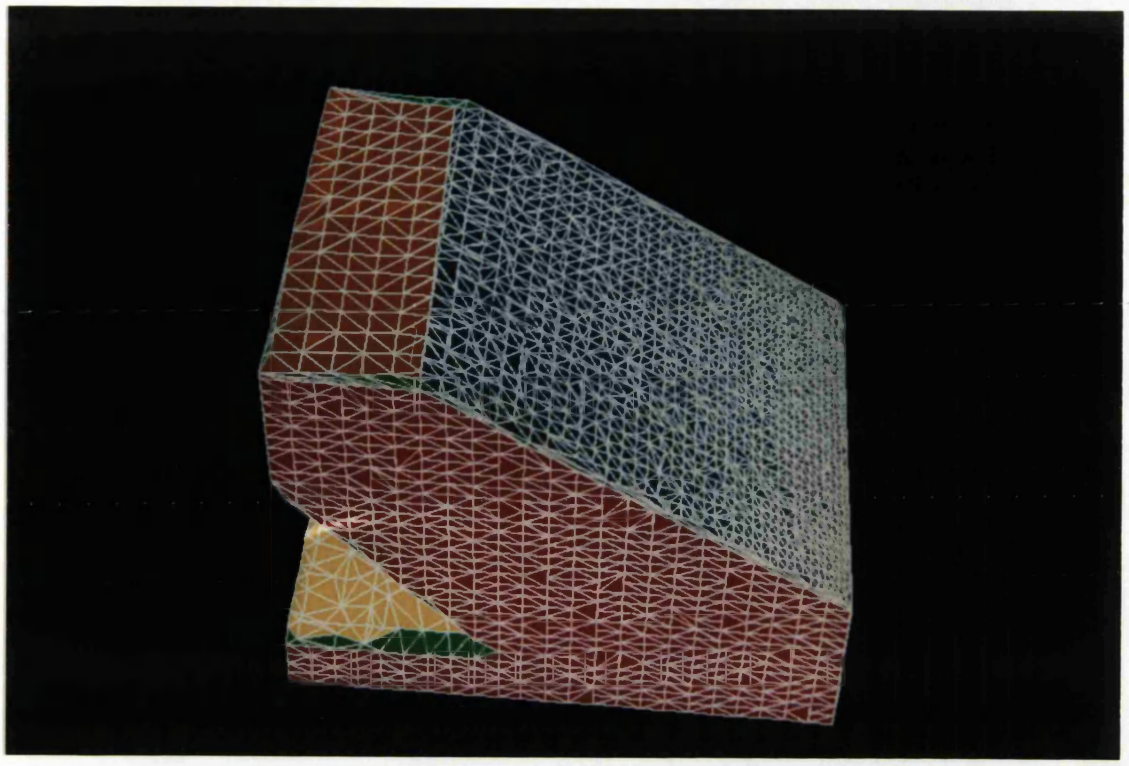


Figure 6.17 (a) Surface triangulation and clustering

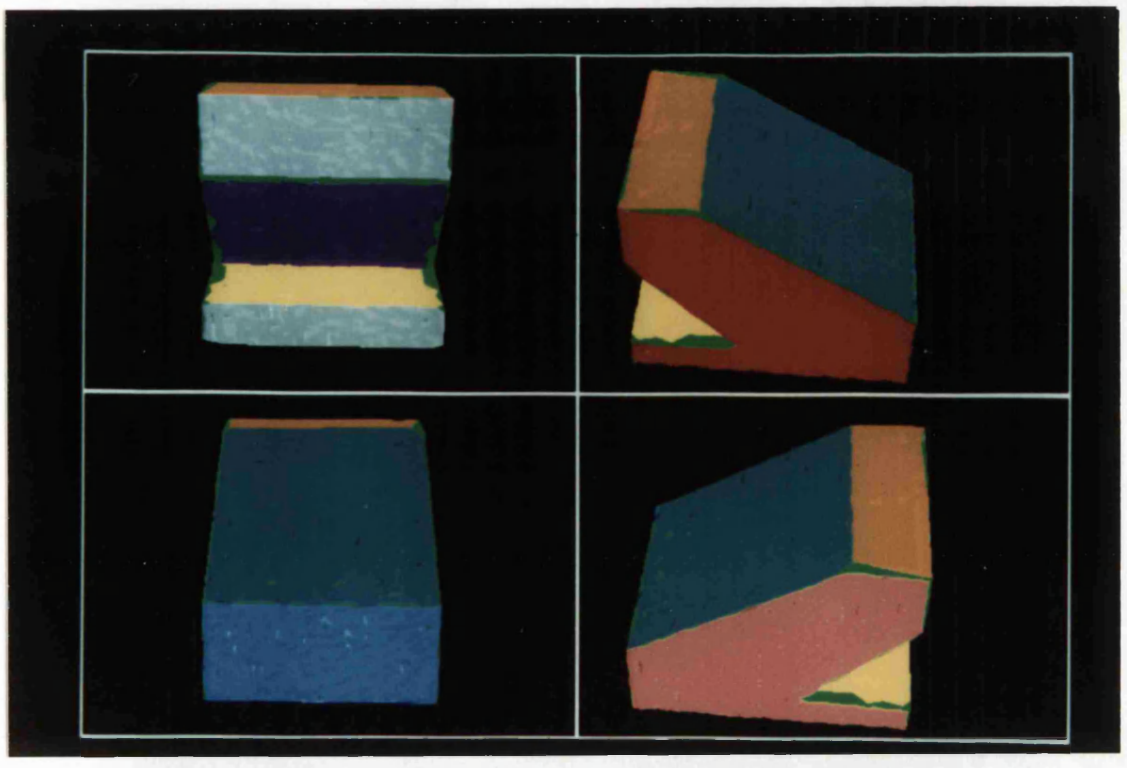


Figure 6.17 (b) Clustering the surface triangles

6.6 Limitation

The algorithms described in this research can handle models with flat surfaces only (including the clustering algorithms). No work on clustering cylindrical parts has been done by the author. The extension of the research to include the cylinders and cones will be mentioned in the last chapter, which will give some suggestions for future work.

6.7 Concluding Remarks

The aim of clustering the surface triangles is to find the faces of the object which are represented by a group of clusters. In this chapter two different techniques to cluster surface triangles are described. The first one, which is an efficient single-linkage algorithm [102], clusters the surface triangles by using their dissimilarity coefficients, whereas the second type clusters them according to the scalar product of their surface normals. After the surface triangles are clustered, a plane is fitted to each cluster. Thus each plane represents a face of the object.

The next problem to be solved at this point is to match the description of measured component to the solid model primitives. Since these two descriptions are now more or less in the same form, they are ready to be matched. The matching process will be explained in the next chapter.

CHAPTER 7

MATCHING

7.1 Introduction

The measured data and the solid model are now more or less in the same form: two collections of plane equations. Unfortunately they will, in general, be referred to different coordinate frames, so, in order to compare them, it is necessary to translate and to rotate them to a best-fit with each other.

For this purpose the technique of *Procrustean matching* [29, 103] has been extended to allow this to be done. In this chapter this technique and its application to the problem will be described in detail.

7.2 Matching the Two Descriptions

Finally, the surface description of the measured component needs to be matched with the surface description of the solid model generated by a CAD system, but this is not easy. Two problems might be encountered in matching the two descriptions: the first one - which was already covered in the previous chapters - is that there might be a difference between the types of the descriptions (the measured data is in the form of the positions of surface points where the solid model primitives are half-spaces); and the second one is that the measured component and the solid model might be related to different coordinate systems. In order to solve the second problem and to match the two descriptions under translation and rotation, *Procrustes analysis* was applied.

In some cases the translation and rotation (see Appendix B) between the measured component and the solid model are known (especially in the case of a measuring machine where the component is mounted in a fixture of known geometry and position before being inspected), but this is not the case for a general automatic inspection method. A good automatic inspection method must be capable of accommodating the general case where no information about the translations and rotations is available (i.e. in order to avoid rejecting perfectly good components that have been measured in slightly the wrong place).

The matching technique described in this thesis is capable of matching the surface descriptions of a measured component to the surface descriptions of a solid model, computing the difference between the two and reporting any out-of-tolerance differences deduced from them. After finding these differences it would

be possible to discard the faces within tolerance, and carry out the whole process again on the bits that don't match. This allows parts of the component which are of the right shape, but which are in the wrong place, to be identified and *their* position and orientation to be computed (e.g. if a bolt is missing or a hole is misdrilled on a manufactured component, this can easily be spotted).

7.3 Procrustes Analysis

Procrustes analysis is a technique which is used for assessing the goodness-of-fit between two configurations. In this section a summary of Procrustes analysis will be given. More detailed information about this technique can be found in Sibson [103].

Consider two configurations of points, **X** and **Y**, each of N points in *k*-dimensional space, neither of the configurations are in the same position nor orientation as each other. Each configuration will be represented by a $k \times N$ matrix, so **X**, for example, will be:

$$X = \left[x^{(1)}, \cdots, x^{(N)} \right]$$

One to one correspondence (which means $x^{(1)}$ corresponds to $y^{(1)}$ and so on) exists at this stage (the need for this will be removed later). If two such configurations, **X**, **Y** are given, they can simply be compared by the sum of their squared positional differences, G(X,Y):

$$G(X,Y) = \sum_{n=1}^{N} (x^{(n)} - y^{(n)})^T \cdot (x^{(n)} - y^{(n)}) = trace\ (X-Y)^T \cdot (X-Y).$$

G(X,Y) is called the *Procrustes statistic* [103]. The aim is to match the configurations such that the value of G is minimum under the Euclidean group of transformations such as translation and rotation.

As described by Sibson [103], the optimal matching under translation is obtained by keeping the **X** fixed and matching **Y** to it by sliding **Y** until the centroids of the two configurations coincide. An alternative to this is to translate both **X** and **Y** so that their centroids are both at the origin. This is simple because in order to perform the translations one to one correspondence between the points is not needed.

It is also shown by Sibson that the best match under rotation (that is the rotation which makes the value of G minimum) is obtained by applying a $k \times k$ orthogonal matrix **P** onto **Y** where **P** is:

$$P = XY^T (YX^TXY^T)^{-\frac{1}{2}}$$

Since applying **P** to **Y** does not change its centroid (if this is at the origin), the rotation does not effect the translation which was done previously; in other words it is independent of the translation.

The matching of point patterns under scale change is also discussed by Sibson [103]. In this thesis only translation and rotation will be taken into consideration when matching the two configurations.

7.4 The Procrustean Matching Algorithm

The only limitation in Procrustean matching is the need for the one to one correspondence between the points in each configuration in order to find the orthogonal matrix **P**. This is a serious limitation when considering the applicability of the technique to general pattern matching problems. In some cases the number of points in each configuration might also be different.

As mentioned in the previous section, in order to match the centroids of two point patterns no correspondence information between the points is necessary. Obviously this could easily be done even if there were different number of points in each configuration.

One thing that remains invariant under rotation is radius. Once **Y** has been translated so that its centroid coincides with that of **X** at the origin, no rotation of **Y** alters the radial distance of each of Y's points from the origin. For this reason the points in each configuration are first matched according to their radial distance.

The algorithm first sorts the radial distances of points in each point pattern into two lists, R_x and R_y. It finds both the nearest entry in R_y to each entry in R_x and the nearest entry in R_x to each entry in R_y. If the relationship between the corresponding radial distances is symmetrical the two points having these radii are accepted as matched, the links are kept and the linked points are removed from further consideration. Otherwise, the links are broken and the process is repeated until all points are linked up to a *nearest neighbour*. The symmetrical and un-symmetrical links in matching the sorted lists of radii is shown in **Figure 7.1**. This

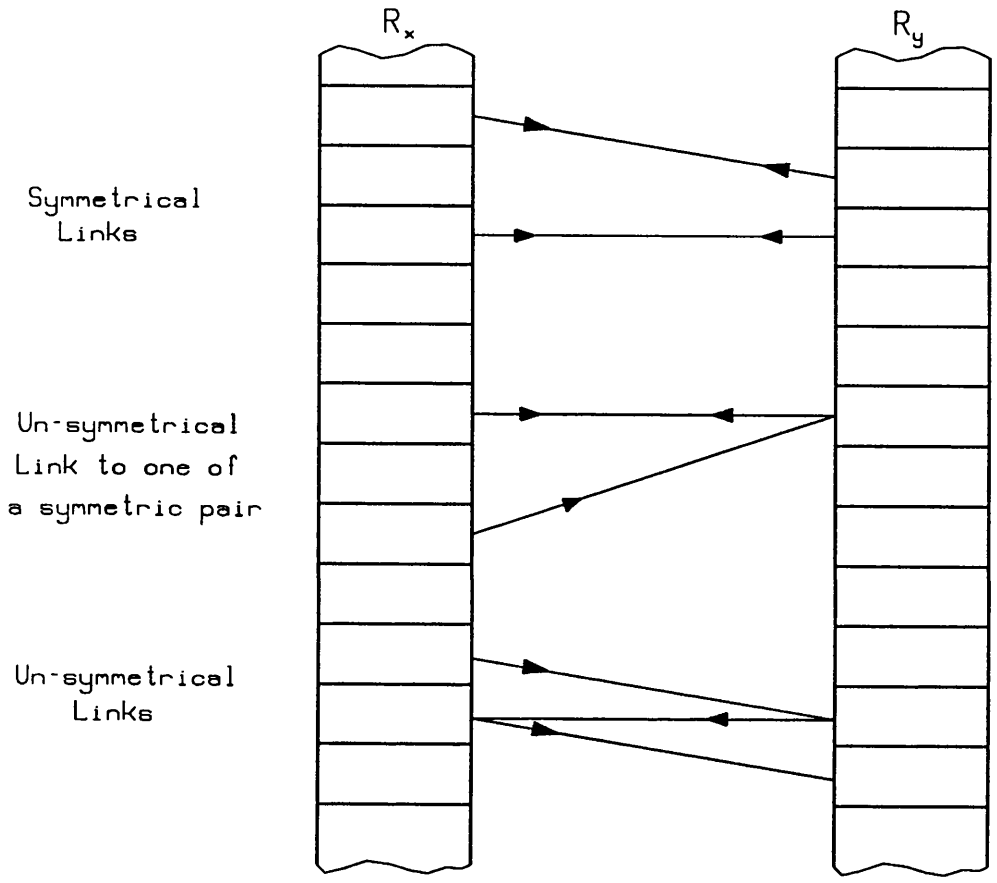


Figure 7.1 Matching the sorted lists of radii (from Cakir et al. [29])

process terminates quite quickly without links crossing one another as shown in the figure. In the case of differing number of points, in the larger configuration, obviously, there will have some unmatched *noise* points but this does not cause much problem.

By matching the points pattern according to the radial distance the rotation matrix **P** can be estimated and **Y** is rotated to roughly the correct orientation. Once this has been done most points will be correctly matched, but there will be some mis-matches because of near-coincidences in radii (**Figure 7.2**). Now there are two sorts of links between the point patterns: links that are short in length which represent correct matches and links that are longer (usually they form a small group) representing mis-matches.

In order to resolve this and to distinguish the mis-matches, a probability density estimate of the link length is constructed by convoluting their histogram with an appropriate kernel function, the width of which was increased until the density function had just two modes (see Silverman [104]). The first, sharp mode comes from the short links, the second, more diffuse mode comes from the mis-matches. **Figure 7.3** shows the density estimation where the vertical scale is arbitrary. The length corresponding to the minimum between the two modes determines the limit to distinguish the mis-matches. The links longer than this length are broken and re-matched by using the Euclidean distance between the points rather than the radial distance as a matching criterion. Extra noise points (if there are any) are available for inclusion in this second match as well. After this, **P** can be re-computed more accurately than before and applied again to make a fine adjustment

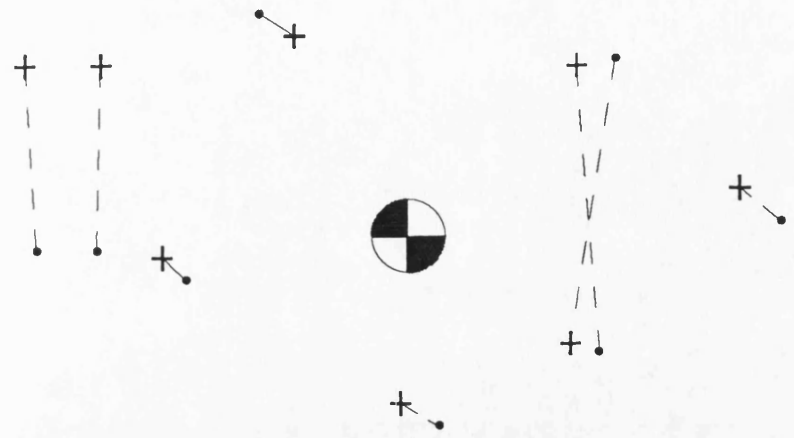


Figure 7.2 Matched and mismatched pairs after rotation
(from Cakir et al. [29])

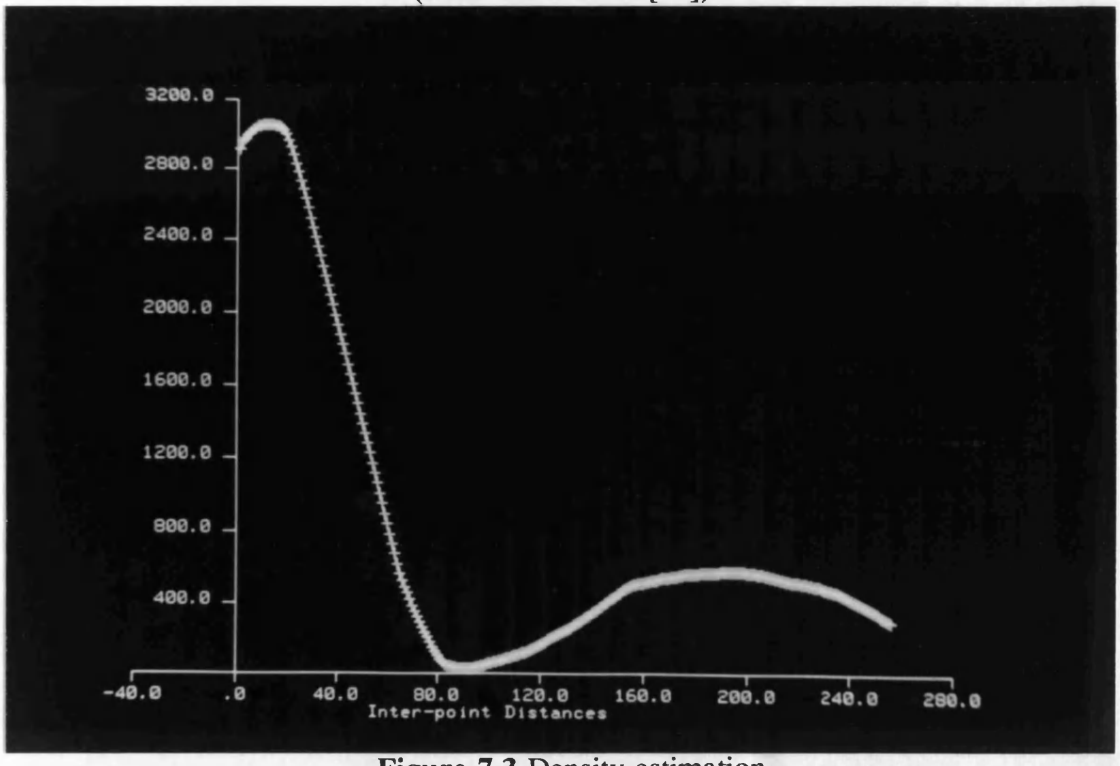


Figure 7.3 Density estimation

in the relative orientation between the two configurations.

Iterating the second re-matching process might be thought a good idea in an attempt to obtain better results, but in practice this is hardly necessary. After a couple of iterations no further improvement in the Procrustes statistic that measures the dissimilarity between the configurations is obtained.

The algorithm outlined in this section is efficient and works accurately on simulated data. It is a robust algorithm in the presence of errors in the configurations. The application of this algorithm to the problem of matching the measured component to the solid model primitives will be explained in the next section.

7.5 Matching by Using the Procrustean Algorithm

As described in the previous section the Procrustean algorithm matches the two point patterns under translation and rotation. However, the aim in this research is to match the *faces* of the measured component to solid model primitives generated by a CAD system; therefore half-spaces need to be matched, not points. Furthermore there is no one-to-one correspondence between the two collections; and there may (because of manufacturing errors) even be different numbers of planes in the model and the measured object.

In order to perform the matching the plane equations are mapped onto *Extended Gaussian Spheres* (EGSs) [62, 75]. Thus, the surface normals of planes can be represented as points on a Gaussian sphere. In the next section Gaussian

spheres and extended Gaussian spheres will be described briefly.

7.5.1 Extended Gaussian Spheres

The *Gaussian sphere* of a collection of planes is the points formed by their normals on the unit ball. Imagine moving the unit surface normal of each plane so that its tail is at the centre of a unit sphere. The head of the unit normal then lies on the surface of the unit sphere. This sphere is called the Gaussian sphere and each point on it corresponds to a particular surface orientation.

When the surface normals are mapped to the Gaussian sphere planar regions become very small clusters, cylinders become unit radius circles, and cones become smaller radius circles (this information is quite useful in extending the work described in this thesis to allow work with cylinders and cones as well. This will be explained in the next chapter).

The extended Gaussian sphere is the pattern of points in space which is obtained by scaling each of these normals by a factor obtained from the planes: for example each plane normal might be scaled by the area of a facet lying in it. The EGS has some nice properties and these properties can be listed as [75]:

1. The extended Gaussian image is not affected by the translation of the object. It rotates in the same fashion as the object in space. This means rotation of the object causes an equal rotation of the extended Gaussian image, since the unit surface normals rotate with the object.

2. The extended Gaussian image is a unique description of a convex object, no two convex objects have the same extended Gaussian image.

Figure 7.4 shows an extended Gaussian image and its corresponding object.

In this research the perpendicular distance from the planes to the origin and the surface areas of the faces are used to scale the normals. This will be described in section 7.5.3. A nice feature of this method is that *any* measurable characteristic of a face (even things such as colour) can be employed to scale the EGS without affecting the rest of the process.

7.5.2 Matching the Faces

The two configurations which contain the scaled normals of the faces of the measured component and the solid model are first matched under translation and this is done by matching their centroids (will be referred to *plane centres* from here on). The plane centre of the configurations (which is minimum distance away from the faces) is calculated as follows:

The squared distance from the faces to the plane centre is

$$f(x,y,z) = \hat{d}^2(x,y,z) = \sum_{i=1}^{n} (a_i\, x + b_i\, y + c_i\, z + d_i)^2$$

where a,b,c are the coefficients of planes and i is the number of faces. In order to find the minimum distance from the faces, x,y and z values which makes f(x,y,z) minimum needs to be found. If the partial derivatives with respect to x,y and z are calculated, and are assigned to zero, three plane equations are obtained:

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{n} 2\cdot(a_i\, x + b_i\, y + c_i\, z + d_i)\cdot(a_i) = 0$$

$$\frac{\partial f}{\partial y} = \sum_{i=1}^{n} 2\cdot(a_i\, x + b_i\, y + c_i\, z + d_i)\cdot(b_i) = 0$$

$$\frac{\partial f}{\partial z} = \sum_{i=1}^{n} 2\cdot(a_i\, x + b_i\, y + c_i\, z + d_i)\cdot(c_i) = 0$$

which can be written as

$$\alpha_1 x + \beta_1 y + \gamma_1 z + \eta_1 = 0$$

$$\alpha_2 x + \beta_2 y + \gamma_2 z + \eta_2 = 0$$

$$\alpha_3 x + \beta_3 y + \gamma_3 z + \eta_3 = 0$$

where α_1 is $\sum_{i=1}^{n} a_i a_i$, β_1 is $\sum_{i=1}^{n} b_i a_i$, γ_1 is $\sum_{i=1}^{n} c_i a_i$, and η_1 is $\sum_{i=1}^{n} d_i a_i$ and so on.

If the intersection point of these three planes is calculated (see [28]), the coordinates of the point which is the minimum distance away from the faces is calculated. This point is the plane centre of the structure.

The next step to be done after the calculating the plane centres of both configurations is to translate the configurations so that the plane centres are at the origin. The new perpendicular distances of the faces from the plane centres are calculated as [87] (see **Figure 7.5**):

$$d_i' = d_i - s$$

where i is the number of faces and s is the displacement between the plane centre and origin and is calculated (from the projection of the displacement vector (dx, dy, dz) onto the surface normal) as

$$s = a\cdot dx + b\cdot dy + c\cdot dz$$

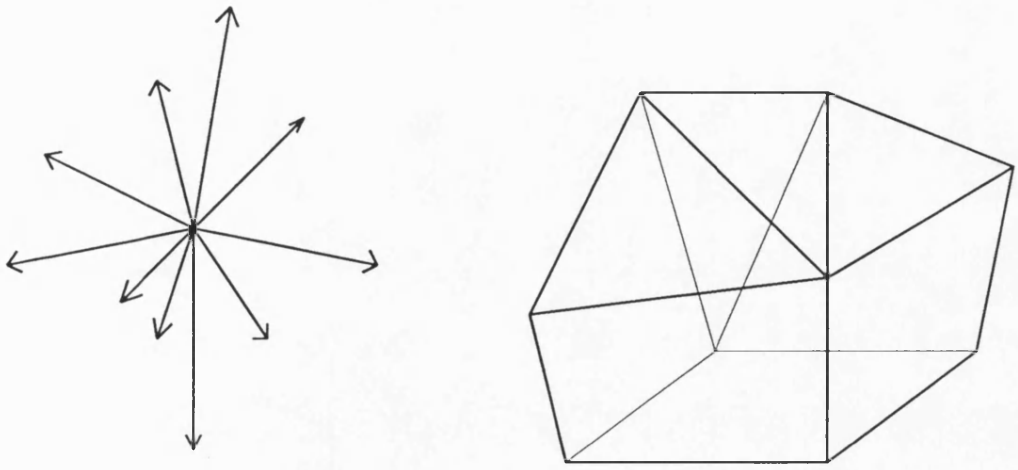


Figure 7.4 An extended Gaussian sphere and its corresponding object

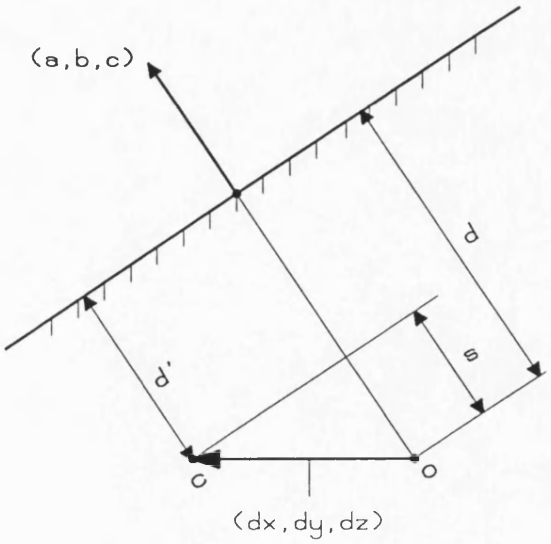


Figure 7.5 The calculation of the displacement

The surface normals (both the measured component's and the solid model's) are then scaled by these new perpendicular distances from the origin and the surface areas of the faces and the point patterns on extended Gaussian spheres are generated. The calculation of surface areas will be explained in the next section.

As the next step the optimal matching under translation is attained by making the plane centres of the point patterns **X** and **Y** coincide. For our purpose the configuration **X** is the points in the EGS of the solid model and the configuration **Y** is the points in the EGS of the measured component. Once the plane centres coincide (**Y** is translated so that its plane centre coincides with that of **X** at the origin) the radial distances of points in **Y** will be invariant under rotation. As explained in section 7.4 the two point patterns are first matched according to their radial distances. Matched patterns after the first matching would allow the rotation matrix **P** to be estimated and **Y** is rotated to roughly the correct orientation.

In section 7.4 a technique of distinguishing the mis-matches was described. But later on it was discovered that forming that kind of histogram to estimate the link length did not improve distinguishing the mis-matches over the simpler technique of breaking all the links and re-matching them according to their actual inter-point distances. After this, **Y** is rotated again to make a fine adjustment in the relative orientation between the measured component and the solid model. The two configurations are matched in such a way that the residual sum of squares between the two extended Gaussian spheres is minimal. The flowchart of the algorithm is given in **Figure 7.6.**

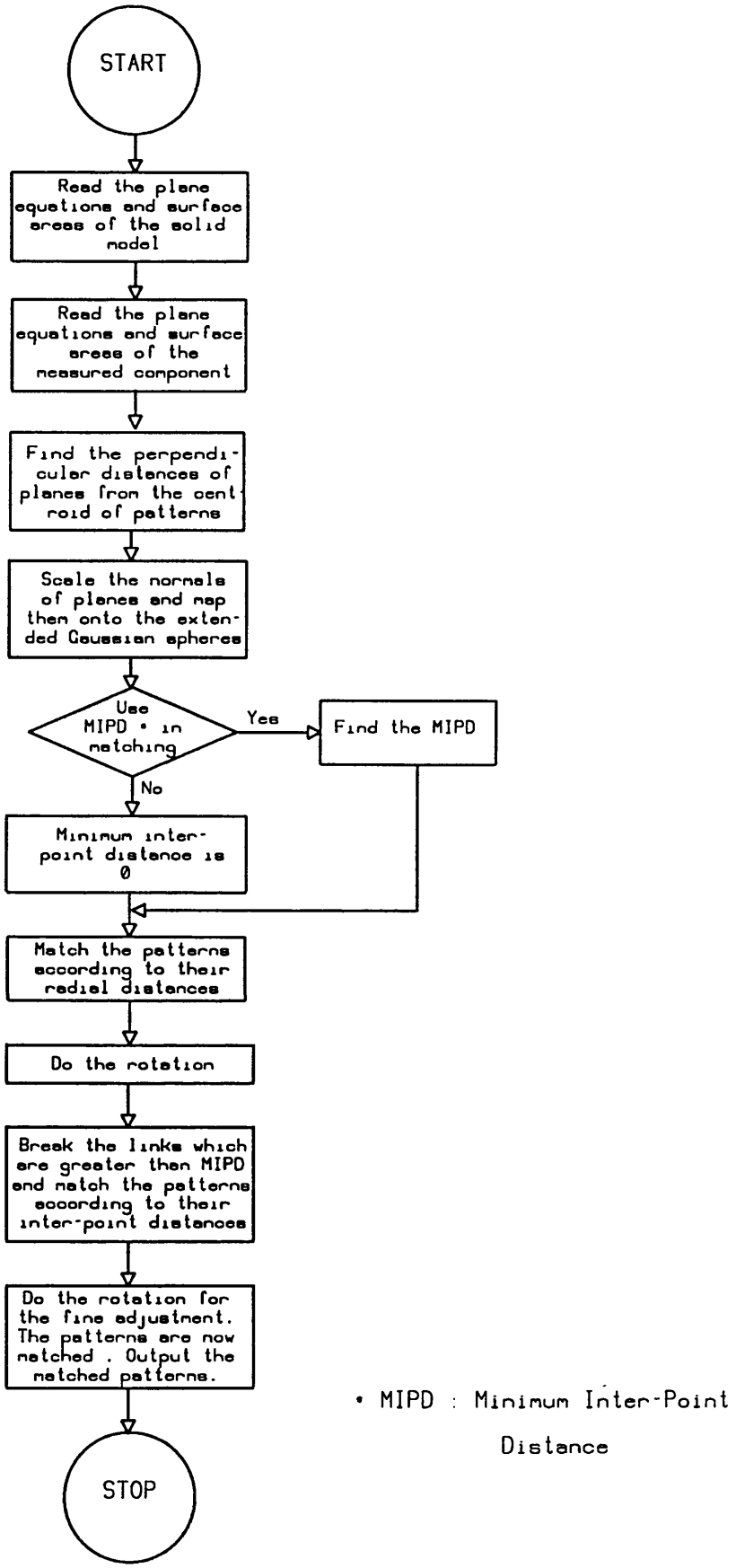


Figure 7.6 The flowchart of the matching algorithm

7.5.3 Scaling the Surface Normals

In order to match the measured component to the solid model, the surface normals in both configurations need to be mapped onto two extended Gaussian spheres. This mapping is done by scaling the normals by two factors: the perpendicular distances of surfaces from the origin and their surface areas. These were multiplied together to give a length for the vectors. Scaling the surface normals by both the perpendicular distances and the surface areas allows the method to recover from any small miscalculation in either of them.

As mentioned in Chapter 6, once the real faces of the measured component are determined their surface areas are calculated from the sum of the areas of surface triangles that form the faces. Since the solid model was already generated the areas of the faces in solid model could also be calculated. However, because of the false chamfers, the surface areas of measured component will be smaller than the surface areas of the solid model. In order to consider the effects of false chamfers, the surface areas of the measured component are scaled:

$$A_i = A_i \cdot x \qquad x = \frac{\sum_{i=1}^{n} A_i + \sum A_F}{\sum_{i=1}^{n} A_i}$$

where A_i is the surface area of a face, A_F is the surface area of a false chamfer, and i is the number of the faces.

As a second alternative, the ratio of the the sum of the surface areas of the solid model to the sum of the surface areas of the measured component can be used as a scale factor. The aim of scaling the surface areas of the measured

component is to make them as close as possible to the surface areas of the solid model. Since the residual sum of squares between the two extended Gaussian spheres needs to be minimal to match the two descriptions, and the surface areas are used to scale the surface normals onto the EGSs, the surface areas of the two descriptions need to be the same (or nearly the same).

As mentioned earlier, there is no one-to-one correspondence between the faces or the surface areas of two configurations. The input of the surface coefficients and the surface areas into the matching algorithm and results will be discussed in the next section.

7.6 Results of the Matching Algorithm

In order to discuss the results of the matching algorithm, the input of the data needs to be explained first. The matching algorithm needs two types of information to achieve the matching: the surface normals of the planes which form the faces of both the measured component and the solid model, and their surface areas. **Figure 7.7** shows a measured component (which is a simple staircase model) and its solid model. Each different colour in figure 7.7 (a) represents a facet and green represents the false chamfers.

The order of half-spaces (faces) of the solid model in a data file prepared by the solid modeller is different from that of the half-spaces of the measured component (which depends on the measuring process), therefore no one-to-one correspondence is available. As mentioned earlier, in some cases, there might be even different numbers of half-spaces in both configurations. Once the surface

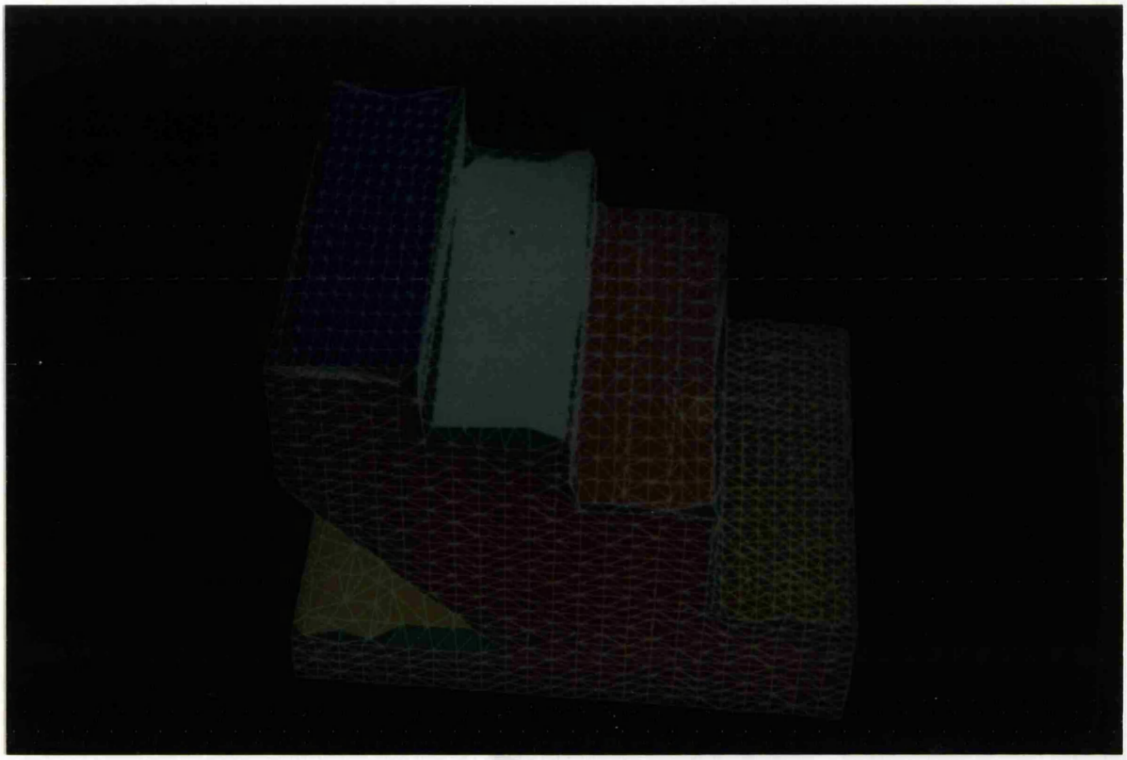


Figure 7.7 (a) A measured component (re-produced from figure 1.7 (b))

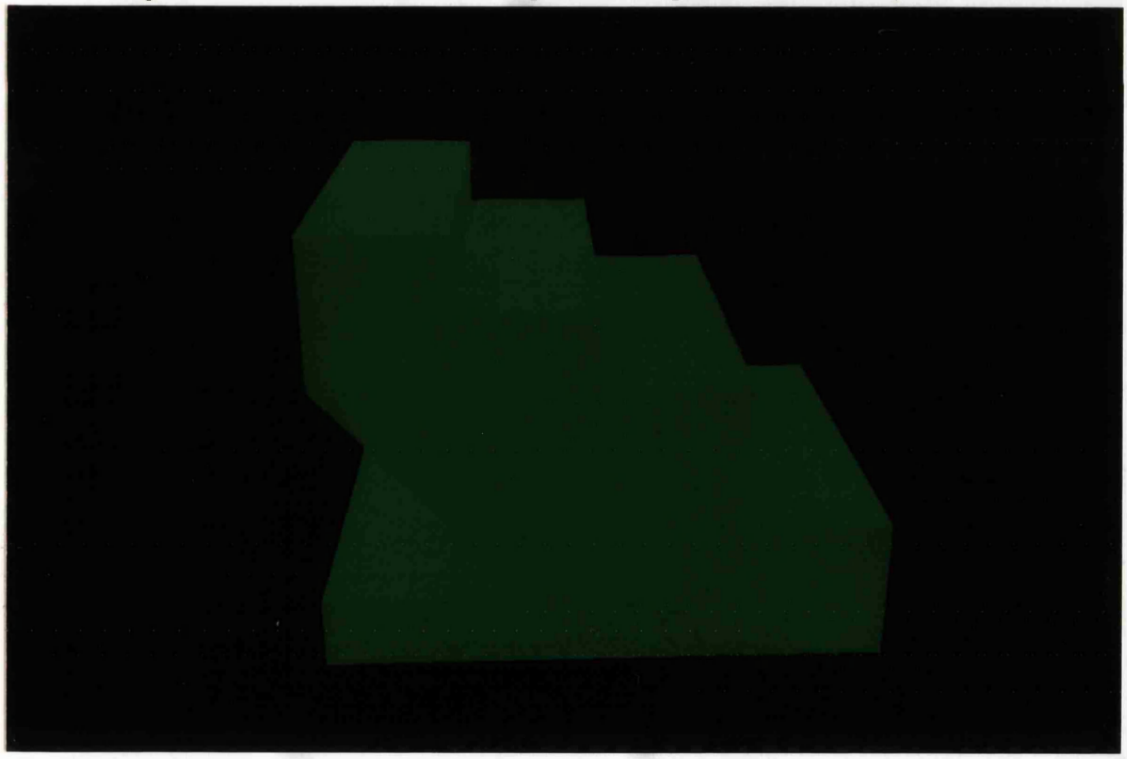


Figure 7.7 (b) Its solid model

normals of the faces are read in by the matching algorithm, their surface areas are read in the same order as the faces. The order of half-spaces and surface areas of a measured component (which is the same staircase model shown in figure 7.7) and of its solid model's are shown in **Figure 7.8.** The perpendicular distances from the plane centre (which is the origin after the translation) are calculated and the surface normals are mapped onto the extended Gaussian spheres. The two configurations are now ready to be matched.

Figure 7.9 and **Figure 7.10** show the result of the matching algorithm. Two data files are read in (first file is for the faces of the solid model, second one is for the measured data). The recovered offset is the difference between the plane centres of two configurations after the matching under translation. Four values of each face are the coefficients of the planar faces, *a, b, c,* and *d* where the equation of each plane is $ax + by + cz + d = 0$ and *d* is the perpendicular distance from the plane centre. As seen from this first output *Face 12* is mis-matched to *Face 4* and *Face 7* is mis-matched to *Face 13* because of near-coincidences in radii (see section 7.4). If the limit distance to distinguish the mis-matches (see section 7.4) is not calculated and is given as 0 (which means all the links will be broken), the second output in figure 7.10, which is the output of matching according to the inter-point distances, is obtained. Since there is no translation or rotation between the measured object and its solid model the recovered offset and the recovered rotation angle is 0 (or very close to 0). As in the second output the faces of the measured component and the solid model are perfectly matched to each other.

Measured Component

Plane Coefficients of STEPSO Ax + By + Cz + D = 0

	A	B	C	D	Areas
Face 1	0.4042278E-04	0.1000000E+01	-0.6262813E-04	-0.8081163E-04	30.6656
Face 2	0.1000000E+01	0.1080815E-03	0.4216455E-04	-0.5157678E-03	19.0814
Face 3	0.8607926E-04	-0.4434678E-04	0.1000000E+01	-0.9997175E+00	18.2636
Face 4	0.1000000E+01	0.2375337E-04	-0.2718664E-04	-0.1999996E+01	5.3573
Face 5	0.7084410E+00	-0.7632673E-05	0.7057700E+00	-0.2825284E+01	26.5962
Face 6	0.1865493E-04	-0.1000000E+01	-0.3522631E-04	0.6000107E+01	30.5200
Face 7	0.7160306E-04	0.6523985E-02	0.9999787E+00	-0.2406028E-01	46.8279
Face 8	0.6083963E-04	0.6569058E-05	0.1000000E+01	-0.3999674E+01	12.5806
Face 9	0.1000000E+01	-0.1935906E-05	-0.1797634E-04	-0.5999934E+01	12.1398
Face 10	0.9999996E+00	0.8907004E-05	0.9424178E-03	-0.4004170E+01	4.9696
Face 11	0.1479737E-02	0.4217801E-04	0.9999989E+00	-0.5998270E+01	12.5375
Face 12	0.1000000E+01	-0.6995658E-04	0.8409857E-04	-0.7999885E+01	12.4043
Face 13	0.1421721E-04	0.1074002E-04	0.1000000E+01	-0.1999983E+01	11.8067
Face 14	0.1984957E-03	0.3631191E-06	0.1000000E+01	-0.5000649E+01	11.5712

Solid Model

Plane Coefficients of STEPS Ax + By + Cz + D = 0

	A	B	C	D	Areas
Face 1	-0.1000000E+01	0.0000000E+00	0.0000000E+00	0.0000000E+00	19.2000
Face 2	0.0000000E+00	-0.1000000E+01	0.0000000E+00	0.0000000E+00	30.4000
Face 3	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.0000000E+00	48.0000
Face 4	0.1000000E+01	0.0000000E+00	0.0000000E+00	-0.8000000E+01	12.0000
Face 5	0.0000000E+00	0.1000000E+01	0.0000000E+00	-0.6000000E+01	30.4000
Face 6	0.0000000E+00	0.0000000E+00	0.1000000E+01	-0.6000000E+01	12.0000
Face 7	-0.1000000E+01	0.0000000E+00	0.0000000E+00	0.2000000E+01	6.0000
Face 8	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.5000000E+01	12.0000
Face 9	-0.1000000E+01	0.0000000E+00	0.0000000E+00	0.4000000E+01	6.0000
Face 10	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.4000000E+01	12.0000
Face 11	-0.1000000E+01	0.0000000E+00	0.0000000E+00	0.6000000E+01	12.0000
Face 12	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.2000000E+01	12.0000
Face 13	0.0000000E+00	0.0000000E+00	0.1000000E+01	-0.1000000E+01	18.0000
Face 14	-0.7071068E+00	0.0000000E+00	-0.7071068E+00	0.2828427E+01	25.8000

Figure 7.8 Plane coefficients of half-spaces and surface areas

```
First .HSP file : steps.hsp
Second .HSP file : stepso.hsp

Recovered offset:   -0.0015  -0.0093   0.0015

Do you want to see the output ?: Y

Face :     9    (   -1.0000000    0.0000000    0.0000000    0.2535212  )
Face :    10    (    0.9999992   -0.0000066    0.0012432   -0.2571232  )

Face :    12    (    0.0000000    0.0000000   -1.0000000   -0.7887323  )
Face :     4    (    1.0000000    0.0000238   -0.0000272    1.7449419  )

Face :     7    (   -1.0000000    0.0000000    0.0000000   -1.7464788  )
Face :    13    (   -0.0000142    0.0000107    1.0000000    0.7902175  )

Face :    10    (    0.0000000    0.0000000   -1.0000000    1.2112677  )
Face :     8    (   -0.0000575    0.0000061    1.0000000   -1.2096615  )

Face :     8    (    0.0000000    0.0000000   -1.0000000    2.2112677  )
Face :    14    (    0.0001985    0.0000004    1.0000000   -2.2096830  )

Face :    11    (   -1.0000000    0.0000000    0.0000000    2.2535212  )
Face :     9    (    1.0000000   -0.0000019   -0.0000180   -2.2550474  )

Face :    13    (    0.0000000    0.0000000    1.0000000    1.7887323  )
Face :     3    (   -0.0000861   -0.0000443    1.0000000    1.7900493  )

Face :     6    (    0.0000000    0.0000000    1.0000000   -3.2112677  )
Face :    11    (   -0.0014797    0.0000422    0.9999989   -3.2134668  )

Face :    14    (   -0.7071068    0.0000000   -0.7071068   -1.7926653  )
Face :     5    (    0.7084410   -0.0000076    0.7057700    1.7970190  )

Face :     4    (    1.0000000    0.0000000    0.0000000   -4.2535212  )
Face :    12    (   -1.0000000    0.0000485   -0.0001330    4.2548103  )

Face :     1    (   -1.0000000    0.0000000    0.0000000   -3.7464788  )
Face :     2    (    1.0000000    0.0001081    0.0000422    3.7448677  )

Face :     5    (    0.0000000    1.0000000    0.0000000   -3.0000000  )
Face :     6    (   -0.0000187   -1.0000000   -0.0000352    3.0092186  )

Face :     2    (    0.0000000   -1.0000000    0.0000000   -3.0000000  )
Face :     1    (    0.0000404    1.0000000   -0.0000626    2.9906161  )

Face :     3    (    0.0000000    0.0000000   -1.0000000   -2.7887323  )
Face :     7    (   -0.0000716    0.0065240    0.9999787    2.7853453  )

Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:       0.0000
Recovered angle around X axis:    0.0073
```

Figure 7.9 Matching according to radial distances

```
First .HSP file : steps.hsp
Second .HSP file : stepso.hsp

Recovered offset:   -0.0016  -0.0093   0.0015

Do you want to see the output ?: N

Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:       0.0000
Recovered angle around X axis:    0.0073
Recovered angle around Y axis:   -0.0004
Recovered angle around Z axis:    0.1684

Face :     9    (   -1.0000000    0.0000000    0.0000000    0.2535212  )
Face :    10    (    0.9999995    0.0001336    0.0009495   -0.2566510  )

Face :    12    (    0.0000000    0.0000000   -1.0000000   -0.7887323  )
Face :    13    (   -0.0000209   -0.0029275    0.9999957    0.7901793  )

Face :     7    (   -1.0000000    0.0000000    0.0000000   -1.7464788  )
Face :     4    (    1.0000000    0.0001513   -0.0000201    1.7448639  )

Face :    10    (    0.0000000    0.0000000   -1.0000000    1.2112677  )
Face :     8    (   -0.0000675   -0.0029317    0.9999957   -1.2096986  )

Face :     8    (    0.0000000    0.0000000   -1.0000000    2.2112677  )
Face :    14    (    0.0001918   -0.0029378    0.9999957   -2.2097211  )

Face :    11    (   -1.0000000    0.0000000    0.0000000    2.2535212  )
Face :     9    (    1.0000000    0.0001256   -0.0000109   -2.2551253  )

Face :    13    (    0.0000000    0.0000000    1.0000000    1.7887323  )
Face :     3    (   -0.0000927   -0.0029826    0.9999956    1.7900111  )

Face :     6    (    0.0000000    0.0000000    1.0000000   -3.2112677  )
Face :    11    (   -0.0014864   -0.0028962    0.9999947   -3.2135048  )

Face :    14    (   -0.7071068    0.0000000   -0.7071068   -1.7926653  )
Face :     5    (    0.7084363   -0.0019911    0.7057719    1.7969365  )

Face :     4    (    1.0000000    0.0000000    0.0000000   -4.2535212  )
Face :    12    (    1.0000000    0.0000573    0.0000909   -4.2549948  )

Face :     1    (   -1.0000000    0.0000000    0.0000000   -3.7464788  )
Face :     2    (    1.0000000    0.0002354    0.0000495    3.7447898  )

Face :     5    (    0.0000000    1.0000000    0.0000000   -3.0000000  )
Face :     6    (    0.0001088   -0.9999956   -0.0029735    3.0092618  )

Face :     2    (    0.0000000   -1.0000000    0.0000000   -3.0000000  )
Face :     1    (   -0.0000871    0.9999959    0.0028756    2.9905729  )

Face :     3    (    0.0000000    0.0000000   -1.0000000   -2.7887323  )
Face :     7    (   -0.0000791    0.0035858    0.9999936    2.7853069  )

Do you want to plot the results later on ?: N
```

Figure 7.10 Matching according to inter-point distances

The results of the matches and mis-matches can be plotted if desired. **Figure 7.11** shows the matches of the staircase model. 14 faces of the measured component matched with the faces of the solid model (each point on an EGS corresponds to a face) in the figure, face numbers are shown in figure 7.11 (b). The red in the figure represents the faces of the solid model and the green represents the measured component's.

The matches and mis-matches of another example on an extended Gaussian sphere is shown **Figure 7.12.** Configurations are deliberately mis-matched in figure 7.12 (b) where the blue colour represents the first configuration and the green colour represents the second one. Since the configurations matched perfectly in figure 7.12 (a), it is difficult to distinguish the colours and configurations. The circle in both figure 7.12 (a) and figure 7.12 (b) corresponds to a cylindrical part of the sample.

If the number of faces are different (which means one of the faces of the measured component is missing or an extra face has been found - noise points, see section 7.4 -) then this mis-matched extra (or missing) face is found by checking how close the matches are and finding any matches which are not as close as the others. **Figure 7.13** shows this. In this example extra face *Face 68* is mis-matched to *Face 54* and this mis-match can easily be spotted by checking the plane coefficients. This example was prepared to test the behaviour of the matching algorithm for a different number of faces in both configurations. The order of the faces in both configurations is the same and this allows the mis-match of the extra face to be shown more clearly. For the simplicity only some matches were shown

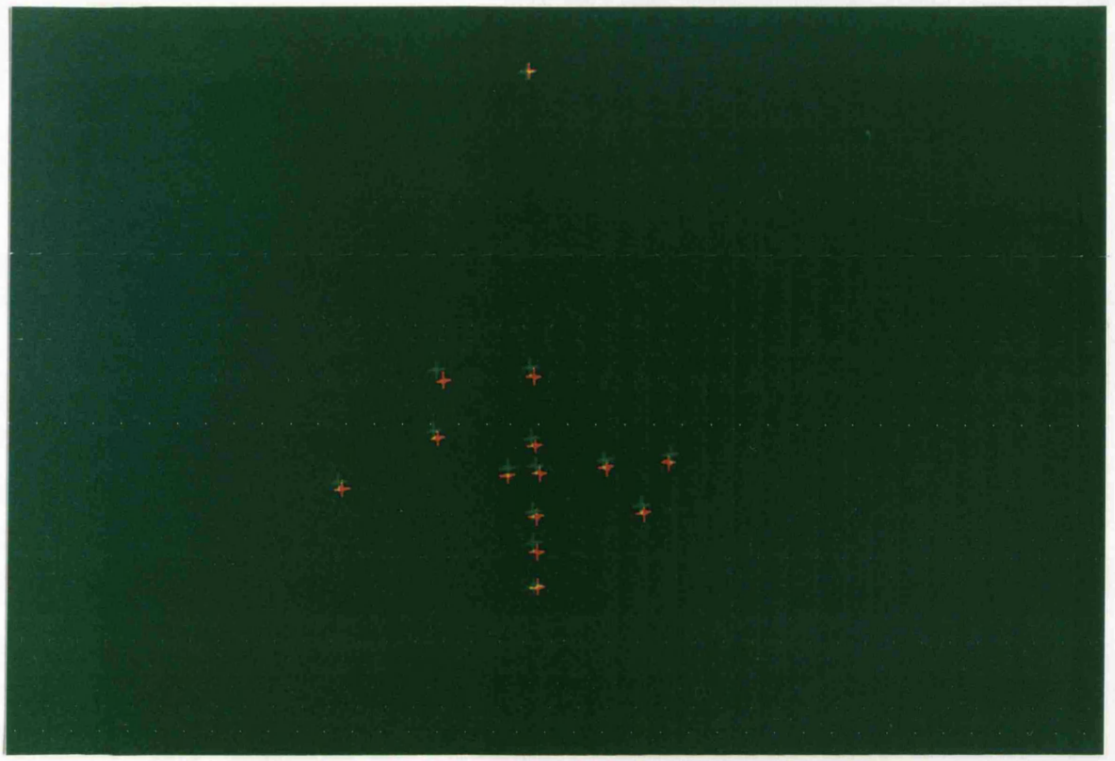


Figure 7.11 (a) Matches on an extented Gaussian sphere

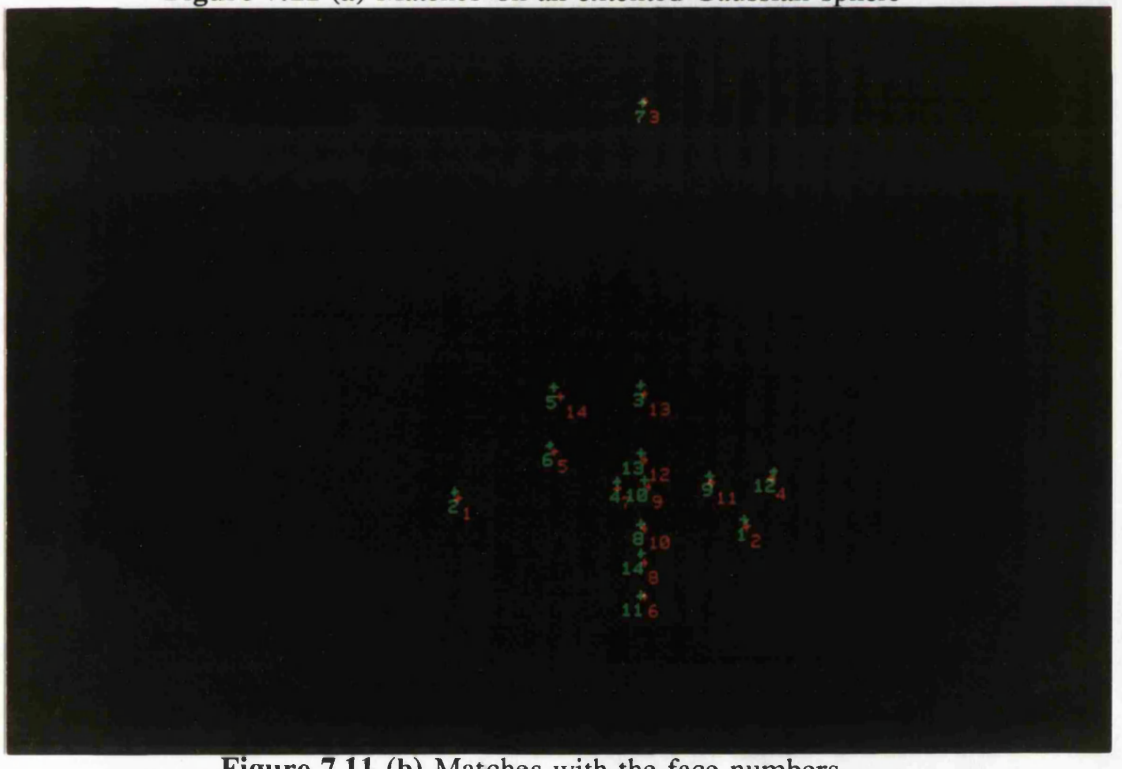


Figure 7.11 (b) Matches with the face numbers

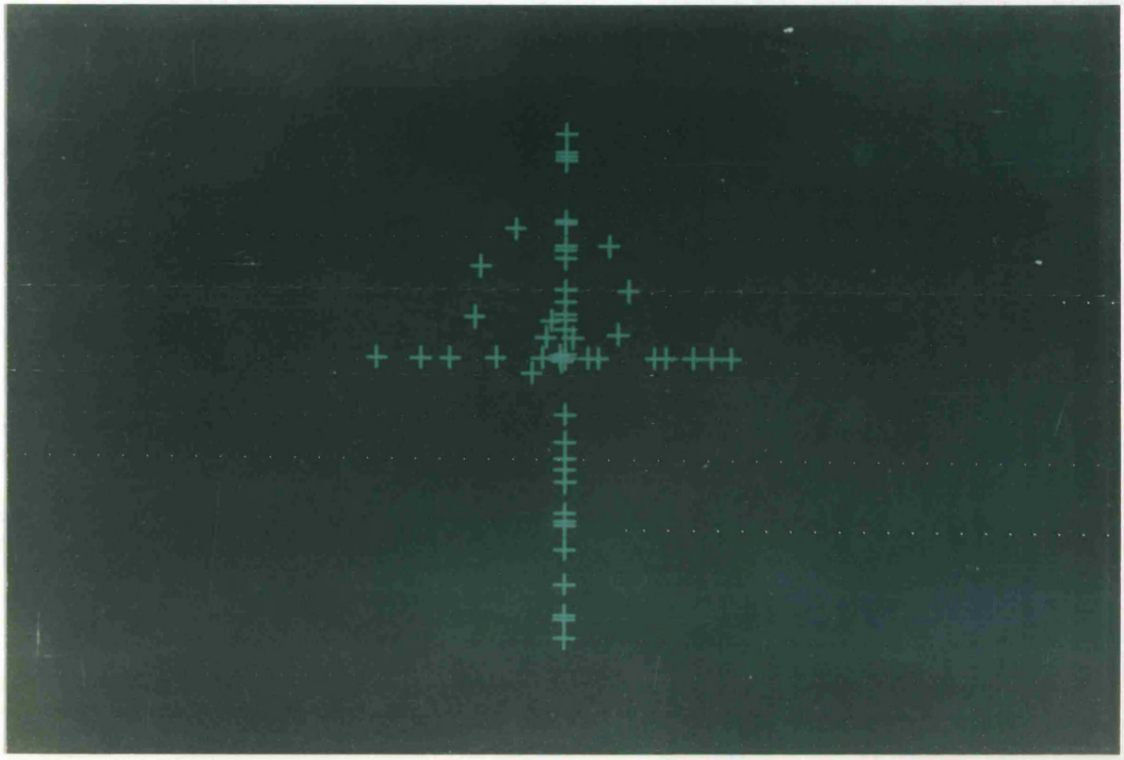


Figure 7.12 (a) Matches on an extented Gaussian sphere

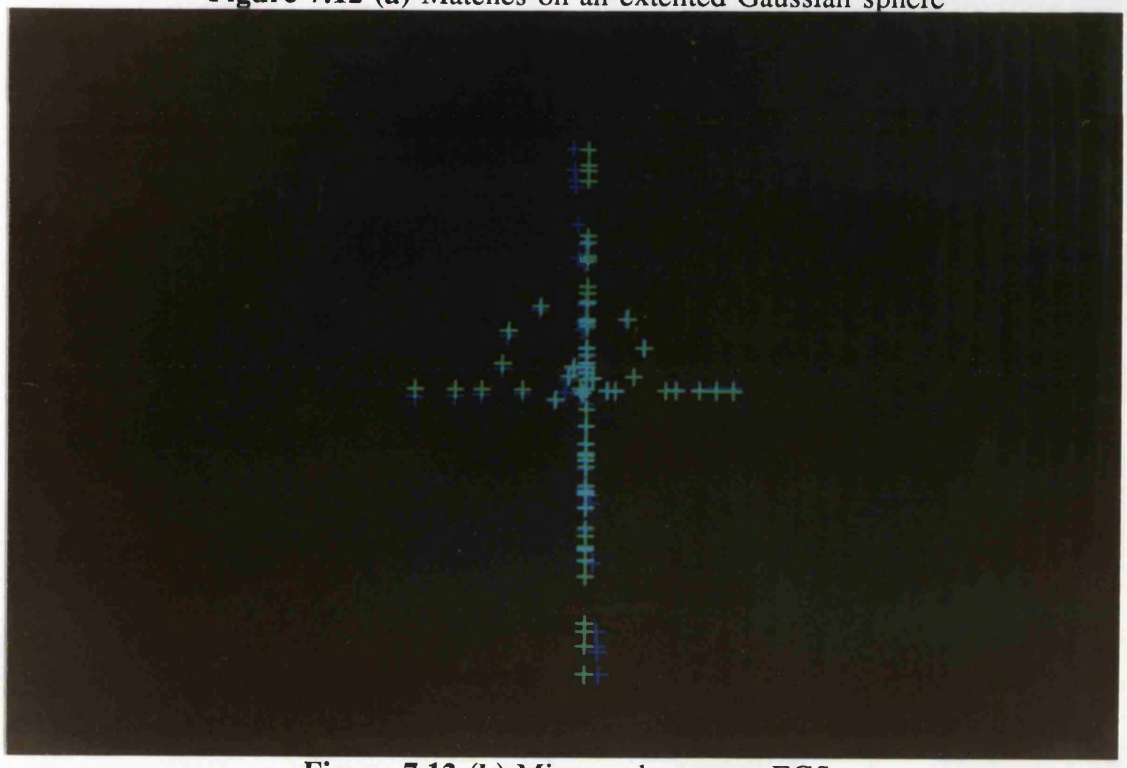


Figure 7.12 (b) Mis-matches on an EGS

```
First .HSP file : cast1p.hsp
Second .HSP file : cast3p.hsp
Recovered offset:   22.9354  -0.3595 -94.7246

Do you want to see the output ?: N
Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:       0.0000
Recovered angle around X axis:    0.0375
Recovered angle around Y axis:   45.0000
Recovered angle around Z axis:    0.0155

Face :    48    (   -0.7071066    0.0000000   -0.7071069   -0.9411774  )
Face :    48    (   -0.7071066    0.0000000   -0.7071069   -0.9411583  )

Face :    13    (   -0.7071069    0.0000000    0.7071066    4.6250000  )
Face :    13    (   -0.7071069    0.0000000    0.7071066    4.6250305  )

Face :    66    (    0.6532813    0.3826834    0.6532816    5.4323745  )
Face :    66    (    0.6532814    0.3826834    0.6532816    5.2948089  )

Face :    68    (    0.0000000   -1.0000000    0.0000000    6.1111069  )
Face :    54    (    0.2705981   -0.9238795    0.2705982    1.8745658  )

Face :    55    (    0.0000001   -1.0000000    0.0000001   -6.3888917  )
Face :    55    (    0.0000001   -1.0000000    0.0000001   -6.0294175  )

Face :    20    (   -0.7071069    0.0000000    0.7071066   -7.3750000  )
Face :    20    (   -0.7071069    0.0000000    0.7071066   -7.3749695  )
```

Figure 7.13 (a) The mis-matched extra face

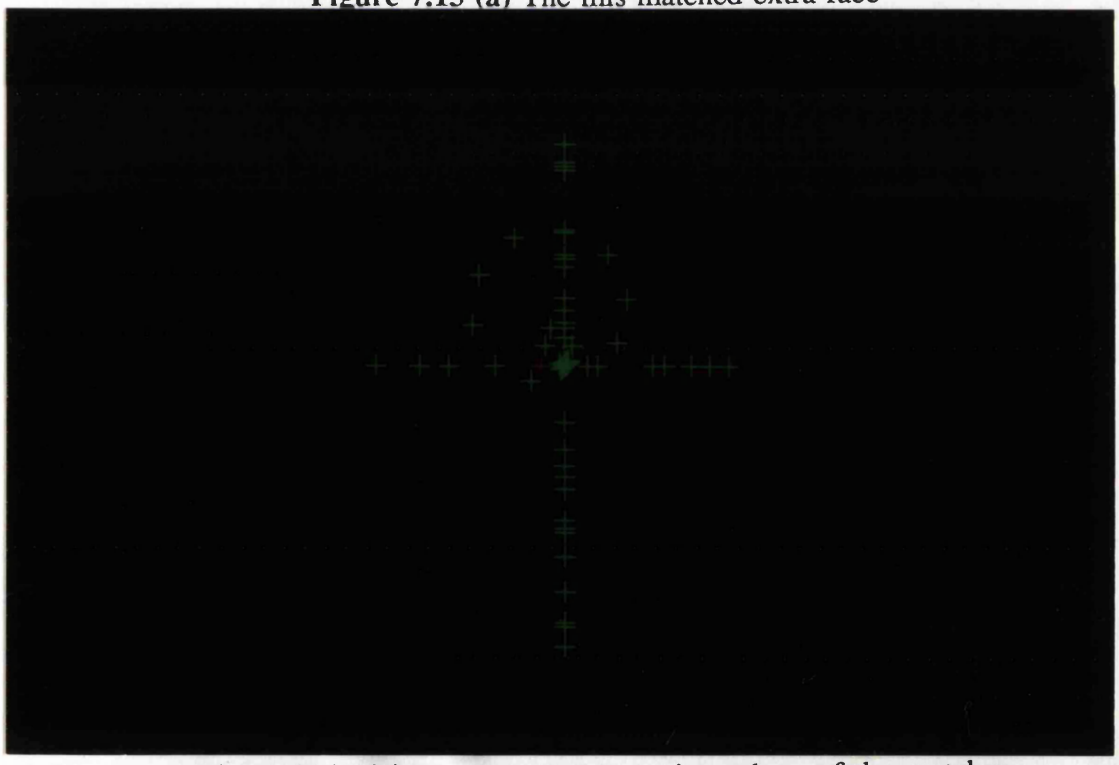


Figure 7.13 (b) The extended Gaussian sphere of the matches

in the figure. The rotation angle between the model and the measured component was 45 degrees. The matches on the EGS are shown in figure 7.13 (b). The single colour in the figure is the mis-matched extra face.

7.7 Problems in Matching

If the measured component to be matched is too symmetrical (which means that not only the perpendicular distances from the origin, but also the surface areas of some faces, are the same), the matching algorithm fails to match these faces and obviously can recover the wrong rotation angle as a result of this. This is not really a problem, or rather it is a problem from which any method must suffer as there is insufficient information for a decision to be made. In addition to this problem, if there are some missing or extra faces in one of the configurations these missing or extra faces change the position of the plane centre and once the plane centre of the configuration is changed, the radial distances of the points are also changed and the configurations do not match perfectly with each other. Scaling surface normals not only by the perpendicular distances from the origin but by the surface areas as well partly solves this second problem but for big changes in the position of the plane centre the problem does still exist. Some examples related to this case and some further results are given in Appendix C.

7.8 Concluding Remarks

In this chapter, as the final step in the work described in this thesis, surface descriptions of the measured component are matched to the surface descriptions of the solid model under translation and rotation. Once the two descriptions are matched, the faces of the measured component may be compared with the corresponding faces of the model and any out-of-tolerance differences reported. Also, if all faces within tolerance are discarded and the whole process is carried out again on bits that don't match, the parts of the component which are of the right shape, but which are in the wrong place can be easily identified and their position and orientation can be computed.

In the next chapter some suggestions will be given for future work. An alternative method which might handle cylinders and cones as well as planar surfaces will also be mentioned.

CHAPTER 8

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

8.1 Introduction

This chapter includes the conclusion of the work described in this thesis and some suggestions for future work.

As mentioned in the earlier chapters the major limitation that the algorithms used in this research have is that most of them can only handle planar surfaces. In this final chapter some suggestions will be given that would extend the research to handle cylindrical and conical components as well as planar surfaces.

8.2 CONCLUSIONS

The motivation of the work described in this thesis was to design an automatic inspection method which compares a set of measurements (taken from the surface of an engineering component by a measuring machine) with a solid model of the same component created by a CAD system. Since the aim of inspection is to find any manufacturing errors, the two descriptions are then ready to be matched to find defects (if there are any).

A group of algorithms were used or developed for this purpose. Some algorithms which were used in this work already existed in the literature on Stochastic Computational Geometry, but had not before been applied to this problem. The new algorithms which were developed were to connect the extant algorithms so as to serve the best solution to the problem.

The initial data were the coordinates of points lying on the surface of the component gathered by a measuring machine (particularly a non-contact type laser measuring machine such as that developed at Bath University [61]). Later on a set-theoretic solid modeller (which was again written at Bath University by John Woodwark [115]) was used to simulate the function of the laser measuring machine. In order to simulate measurement errors the surface points were perturbed slightly by random numbers. Since the implementation was to use the laser measuring machine the algorithms developed used the principles of a laser measuring machine (such as tracing the laser beam backwards in the classification algorithm, Chapter 5) but they could be used with any other sort of measuring machine.

Since the initial data were not suitable to be compared with the solid model primitives (they were in a different forms), the three-dimensional Delaunay triangulation algorithm devised by Adrian Bowyer [25] was used to form a volumetric representation whose surfaces were in the same form as the solid model primitives. The problem was then to find the surfaces.

This problem was solved by finding the triangular faces of the tetrahedral packing (which formed the volumetric representation) that lay on the boundary of the measured component. In order to do so this tetrahedra were classified as solid or air.

Once the surface of the measured component had been found the next stage was to find the real faces of the measured component to be matched with the faces of the solid model. This was done by clustering the surface triangles lying on the same surface. The algorithm dealing with this problem was quite an efficient one but, since it was using scalar products to cluster the triangles, it was unable to handle curved surfaces, such as cylinders or cones. Suggestions to improve the algorithm so as to handle the cylinders and cones will be made in the next section. The clustering algorithm was used not only for clustering the surface triangles, but for deducing the topology between the faces and for finding the false chamfers (which were the products of the measuring machine and needed to be got rid of) as well. It fitted a plane to each cluster by using principle component analysis and output the equations of the planes which represented the real faces of the component. The next thing to do was to match these faces with the faces of the solid model.

At this stage the two descriptions were ready to be matched (they were in the same form: both were planar half-spaces). The only problem which was encountered here was that the descriptions were referred to different coordinate frames. In order to solve this problem the Procrustean algorithm was used to match the faces of the measured component with faces of the solid model. In order to do this the surface normals of the faces were scaled and mapped onto extended Gaussian spheres.

By matching the point patterns on their EGSs under translation and rotation the measured component was compared with its solid model and any out-of-tolerance differences were reported. This comparison allowed the manufacturing errors of the component to be found.

8.3 Suggestions for Future Work

8.3.1 Dealing with Cones and Cylinders

Since the clustering technique described in this research uses the result of the scalar product of the surface normals to cluster the surface triangles, it is unable to handle curved surfaces, such as cylinders or cones. For this reason, only planar faces of the measured component were represented on extended Gaussian spheres and matched with the planar faces of the solid model.

In order to extend the work so as to distinguish cylindrical and conical parts of a measured component and to match them with the corresponding curved faces of the solid model the following method could be used:

Once the surface triangles and their surface normals are found, the surface normals might be mapped onto a Gaussian sphere (or an extended Gaussian sphere, EGS) before clustering the surface triangles. In this technique, as in the technique described in Chapter 7, it might also be a good idea to scale the surface normals by the perpendicular distances of the triangular facets from the *origin* (this would allow the parallel planes to be clustered). After the mapping, the surface triangles lying on the same surface will form small clusters on the ordinary Gaussian sphere (as opposed to the extended one) whereas (as mentioned in section 7.5.1) triangles lying on cylindrical faces will form unit radius circles and the triangles on conical faces will form small radius circles. These different types of clusters would allow the different shapes (such as cylinders and cones as well as the planar surfaces) to be distinguished and different shapes could be matched by just matching the two Gaussian spheres (or EGSs), one for the measured component and one for the solid model.

Matching the planar faces by matching the two EGSs was explained in the previous chapter. In this section some suggestions will be given for matching conical parts and cylindrical parts. Firstly, matching the conical parts:

It is easier to distinguish a cone from its mapping on the Gaussian sphere (see **Figure 8.1**). The included angle of a cone determines the diameter of the small radius circle on the Gaussian sphere. By matching the two same-radius circles the conical parts of the measured component could easily be matched with the parts of the solid model (see **Figure 8.2**).

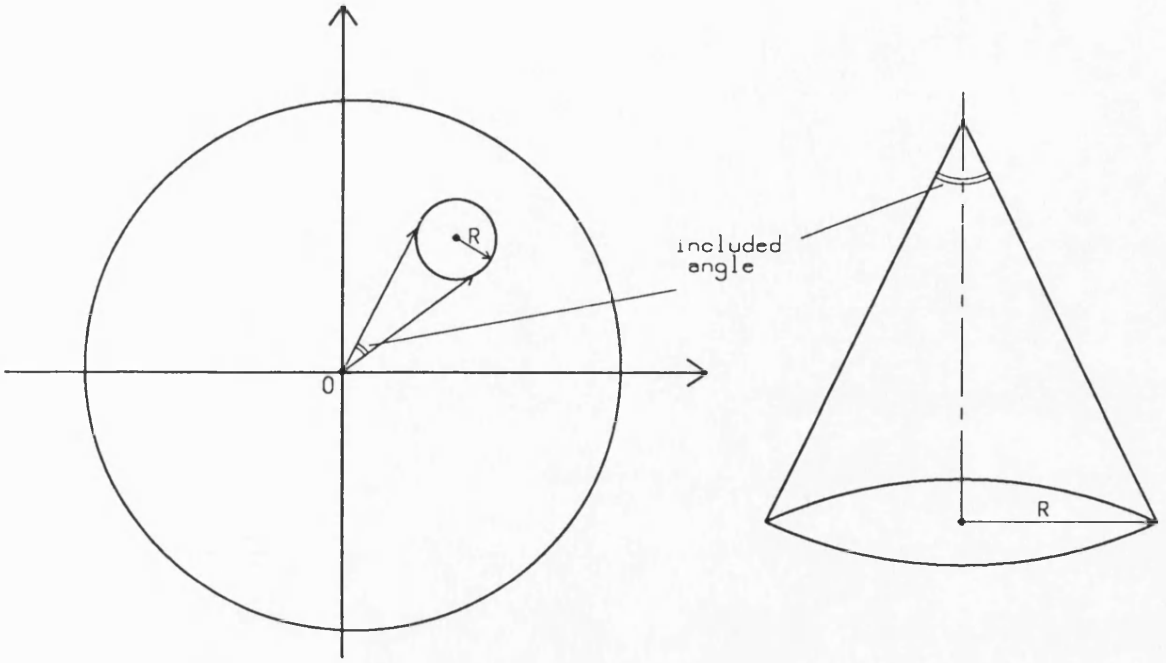


Figure 8.1 Representation of cones on a Gaussian sphere

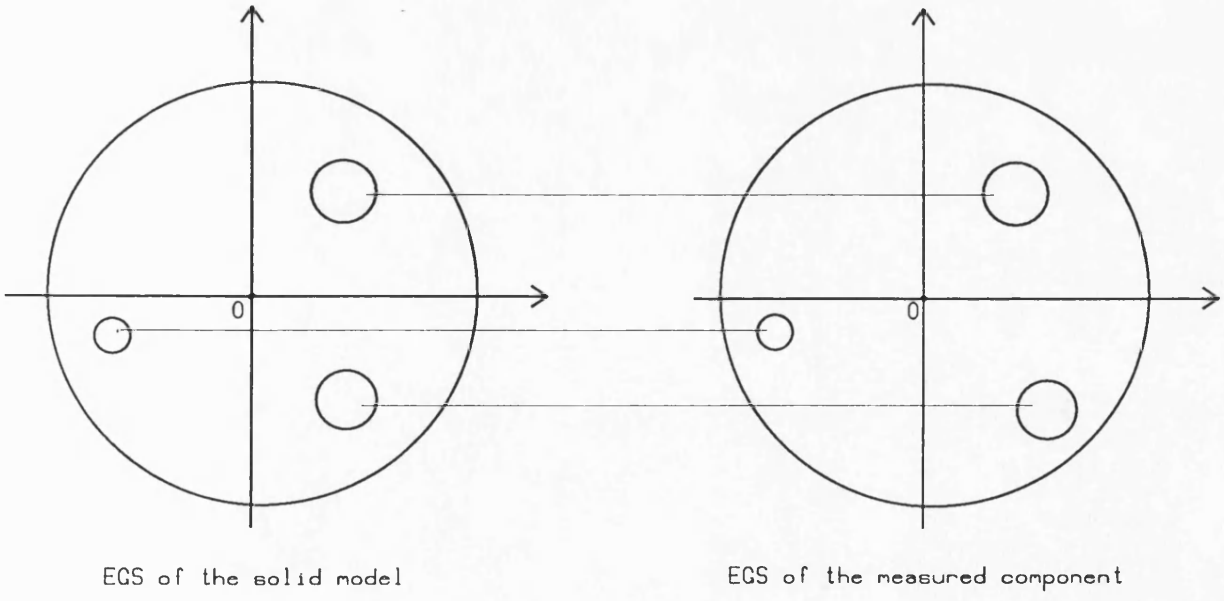


Figure 8.2 Matching the conical parts

On the other hand, matching cylinders is rather difficult. The reason for this is, that whatever their diameters, cylinders are represented as unit radius circles on Gaussian spheres. However, if the links between the surface normals and their corresponding surface triangles are kept, the diameters of the cylinder could be calculated.

If there is more than one cylindrical part in a model, these cylindrical parts might be matched by matching their axes. In order to match axes, first the center point which is minimum distance away from the axes is found and the lines are matched according to their distance from this centre point by using Procrustean matching under translation and rotation. The algebra [26] which calculates the centre point is given in Appendix D. **Figure 8.3** shows the axes of cylinders and their matching.

If there is more than one cone which has the same included angle, the same technique could be used to find these cones, and the cones could again be matched by matching their axes.

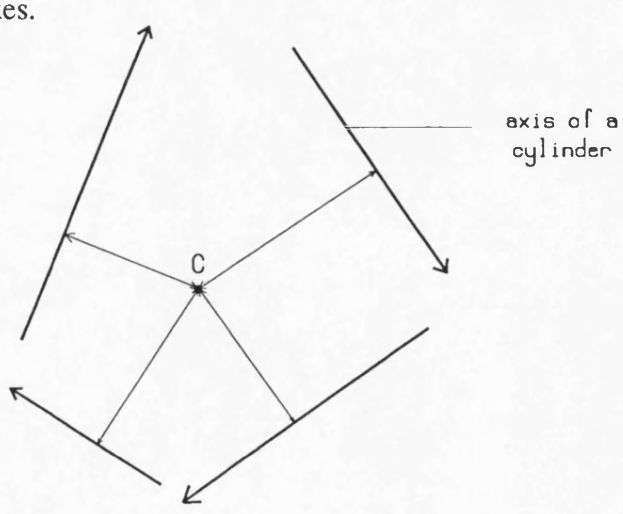


Figure 8.3 Axes of the cylinders and their mutually nearest point

8.3.2 Reducing Processing Time

One of the aims of modern production is to reduce inspection time as much as possible. Since the techniques described in this thesis deal with a large number of surface points and processing these surface points is time-consuming, it would be worthwhile investigating new methods which might reduce that processing time; thus increasing efficiency and reducing inspection time.

In order to reduce processing time, parallel processing might be a useful tool. Parallel processing is a technique for increasing the computation speed for a task, by dividing the algorithm into several sub-tasks and allocating multiple processors to execute multiple sub-tasks simultaneously. Current developments in parallel processing are of increasing interest to those concerned with the creation, display and analysis of pictures. It might also be quite useful in order to increase the accuracy and to reduce the processing time when reconstructing the measured engineering components from their surface points.

8.3.3 Dealing with Symmetry

Most real engineering components have symmetry about the axes (that is actually how they are designed). The problem of symmetry was already mentioned in Chapter 7. In some cases it is worthwhile to try the six possible 180 degree rotations about the axes to find the best match. Possible three 180 degree rotations around x, y and z axis were already tried in order to help to sort out the symmetry problem but did not help much for the case given in Appendix C. The best match

here was defined as the match which minimises the sum of squared distances between the faces of the measured component and the solid model's after each rotation.

As also mentioned in Chapter 7, the matching depends on the position of the plane centres of both the solid model and the measured component. Since the plane centre was the minimum distance away from the faces, extra or missing faces changed the centre of the planes of the measured component and the faces sometimes failed to match. In order to solve this problem surface areas were also considered to scale the surface normals, but for big changes in the position of the plane centre this did not help much either. For this reasons it is recommended that some other measures be found to scale the surface normals which would allow the technique to handle any amount of change in the position of the plane centre because of the missing or the extra faces of measured component. These measures could be the surface colour (if a colour television camera is assisting the measuring machine), the grey level for a monochrome camera (as long as the angle of illumination is taken into account), the surface texture, the surface roughness, some bar codes or markings on the surface and so on.

Alternatively, in order to handle the cases where symmetry or missing or extra faces cause the matching algorithm to fail, the picture of the measured component and the solid model (the picture of the measured component was already produced by using the painter's algorithm [55, 95, 110] and the faces were painted in different colours - see Chapter 6) could be used. The faces that were mis-matched could be corrected interactively by pointing at the faces which should be matched

on both the measured component and the solid model and asking the algorithm to do the matching according to this information.

8.3.4 Dealing with Surface Roughness or Surface Alignments

Algorithms described in this thesis might also be used in checking the surface roughnesses or the surface alignments of the measured engineering components. Since the clustering algorithm produces the neighbourhood relationship information between the faces of the measured component, the angles between the neighbouring faces can be calculated to be compared with the solid model's and any misalignment can be found.

Besides, the roughness of the surfaces might be introduced by just perturbing the measured points not by random numbers but by different sorts of distributions (i.e. Gaussian distribution) and checked again by comparing with the solid model.

REFERENCES

1. **Ahuja, N.,** *Dot pattern processing using Voronoi neighbourhoods,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-4, Number 3, May 1982, page 336-343.

2. **Ahuja, N., Bridwell, N., Nash, C., Huang, T.S.,** *Three-dimensional robot vision,* IEEE Workshop on Industrial Applications of Machine Vision, Research Triangle Park, May 1982, page 206-213.

3. **Anderberg, M.R.,** *Cluster analysis for applications,* Academic Press, New York, 1973.

4. **Baer, A., Eastman, C., Henrion, M.,** *Geometric modelling - a survey,* Computer-Aided Design, Volume 11, Number 5, September 1979, page 253-272.

5. **Bajcsy, R.,** *Three-dimensional scene analysis,* Proc. of the 5th International Conference on Pattern Recognition (Miami Beach, Florida, December 1-4), IEEE, New York, page 1064-1073.

6. **Baker, H.,** *Three-dimensional modelling,* Proc. of the 5th International Joint Conference on Artificial Intelligence (Cambridge, Mass., Aug. 22-25), IJCAI, page 649-655.

7. **Ball, G.H.,** *Classification analysis,* Stanford Research Institute, SRI Project 5533, 1971.

8. **Ballard, D.H., Sabbah, D.,** *Viewer independent shape recognition,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-5, Number 2, March 1983, page 653-659.

9. **Bentley, J.,** *Multidimensional-divide-and-conquer,* Commun. Ass. Comput. Mach., Volume 23, April 1980, page 214-229.

10. **Bentley, J., Shamos, M.,** *Divide-and-conquer in multidimensional space,* Proc. ACM Symp. Theory of Computing, May 1976, page 220-230.

11. **Besant, C.B., Lui, C.W.K.,** *Computer-aided design and manufacture,* Ellis Horwood Ltd., 1986.

12. **Besl, P.J., Jain, R.C.,** *Survey: Three-dimensional object recognition,* Computer Surveys, Vol 17, No 1, March 1985, page 75-145.

13. **Bhanu, B.,** *Representation and shape matching of 3-D objects,* IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-6, May 1984, page 340-350.

14. **Bhanu, B., Faugeras, O.D.,** *Shape matching of two-dimensional objects,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume

PAMI-6, Number 2, March 1984, page 137-156.

15. **Black, S.P.,** *Utilisation of 3-coordinate measuring machines,* International Metrology Conference Nelex 80, paper 2.3, 7-9 October 1980.

16. **Bocquet, J.C., Tichkiewitch, S.,** *An expert system for reconstruction of mechanical objects from projections,* Proceedings of the 6th International Conference on Pattern Recognition (Munich, W.Germany, Oct 19-22), IAPR and IEEE, New York, page 491-496.

17. **Boissonnat, J.D.,** *Shape reconstruction from planar cross-sections,* Proceedings of the IEEE conf. on Computer Vision and Pattern Recognition, San Fransisco, June 1985, page 393-397.

18. **Boissonnat, J.D.,** *Representing 2D and 3D shapes with Delaunay triangulation,* Proc. of the 7th International Conference on Pattern Recognition (Montreal, Canada, July 30-Aug 2), IEEE, New York, page 745-747.

19. **Boissonnat, J.D.,** *Representation of object triangulating points in 3-D space,* Proc. of the 6th International Conference on Pattern Recognition (Munich, West Germany, Oct 19-22), IEEE, New York, page 830-832.

20. **Boissonnat, J.D.,** *Geometric structures for three-dimensional shape representation,* ACM Trans. on Graphics, page 266-286, October 1984.

21. **Boissonnat, J.D., Faugeras, O.D.,** *Triangulation of 3-D objects,* Proc. of the 7th International Joint Conference on Artificial Intelligence (Vancouver, B.C., Canada, Aug 24-28), IJCAI, page 658-660.

22. **Boubez, T.I., Funnell, W.R.J., Lowther, D.A., Pinchuk, A.R., Silvester, P.P.,** *Mesh generation for computational analysis, part I- Electromagnetic and technical considerations of mesh generations,* Computer-Aided Engineering Journal, Volume 3, Number 5, October 1986, page 190-195.

23. **Boubez, T.I., Funnell, W.R.J., Lowther, D.A., Pinchuk, A.R., Silvester, P.P.,** *Mesh generation for computational analysis, part II- Geometric and topological considerations for three-dimensional mesh generation* Computer-Aided Engineering Journal, Volume 3, Number 5, October 1986, page 196-201.

24. **Bowyer, A.,** *SID- Set-theoretic Input to Dora, a language for describing solid objects,* Bath University, School of Engineering, Internal Report, 1986.

25. **Bowyer, A.,** *Computing Dirichlet tesselations,* Computer Journal, Volume 24, Number 2, 1981, page 162-166.

26. **Bowyer, A.,** *Personal communications,* University of Bath.

27. **Bowyer, A., Graham, D., Henry, G.K.,** *The measurement of 3-D features using laser triangulation,* Proc. of the 7th International Conference on Automated Inspection and Product Control, Birmingham 1985, IFS Publications, page 313-322. 1980.

28. **Bowyer, A., Woodwark, J.R.,** *Programmer's geometry,* Butterworths, 1983.

29. **Cakir, M.C., Bowyer, A.,** *Matching measured components into solid models,* Proc. of the International Conference on Theory and Practice of Geometric Modelling, (Blaubeuren, W.Germany, October 3-7), 1988.

30. **Cavendish, J.C., Field, D.A., Frey, W.H.,** *An approach to automatic three-dimensional finite element mesh generation,* International Journal for Numerical Methods in Engineering, Volume 21, page 329-347, 1985.

31. **Chiyokura, H.,** *Solid modelling with DESIGNBASE,* Addison-Wesley Publishing Company, 1988.

32. **Choi, B.K., Shin, H.Y., Yoon, Y.I., Lee, J.W.,** *Triangulation of scattered data in 3D space,* Computer-Aided Design, Volume 20, Number 5, June 1988, page 239-248.

33. **Choong, Y.C.,** *Intelligent robot vision in automated surface finishing,* Ph. D. dissertation, University of Bath, 1982.

34. **Cohen, H.D.,** *A method for the automatic generation of triangular elements on a surface,* International Journal for Numerical Methods in Engineering, Volume 15, Number 3, page 470-477, 1980.

35. **Coyne, B.,** *Three-dimensional coordinate measuring machine survey,* Quality Today, January 1989, page 17-31.

36. **Csendes, Z.J., Shenton, D., Shahnasser, H.,** *Adaptive finite element mesh generation using Delaunay algorithm,* IEEE Transactions on Magnetics, Volume MAG-19, Number 6, 1983, page 2551-2554.

37. **Dane, C., Bajcsy, R.,** *An object-centered three-dimensional model builder,* Proc. of the 6th International Conference on Pattern Recognition, IEEE, New York, page 348-350.

38. **Davis, S.L.,** *Shape matching using relaxation techniques,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-1, Number 1, January 1979, page 60-72.

39. **Diday, E., Govaert, G., Lechevallier, Y., Sidi, J.,** *Clustering in pattern recognition,* Digital Image Processing, J.C. Simon and R.M. Haralick, Eds. Dordrecht, Holland, Reidel, Oct. 1981, page 331-370.

40. **Dudani, S., Breeding, K.J., McGhee, R.D.,** *Aircraft identification by moment invariants,* Trans. Computing, Volume C-26, page 39-45, 1977.

41. **Duncan, J.P., Mair, S.G.,** *Sculptured surfaces in engineering and medicine,* Cambridge University Press, 1983.

42. **Eastman, C.M., Preiss, K.,** *A review of solid shape modelling based on integrity verification,* Computer-Aided Design, Volume 16, Number 2, March 1984, page 66-80.

43. **Everitt, B.,** *Cluster analysis,* Halsted Press, New York, 1974.

44. **Faugeras, O.D.,** *New steps toward a flexible three-dimensional vision system for robotics,* Proc. of the 7th International Conference on Pattern Recognition (Montreal, Canada, July 30-Aug. 2), IEEE, New York, page 796-805.

45. **Faugeras, O.D., Ponce, J.,** *Prism trees: a hierarchical representations of 3-D objects,* Proc. of the 8th International Joint Conference on Artificial Intelligence (Karlsruhe, West Germany, Aug. 8-12), IJCAI, page 982-988.

46. **Faugeras, O.D., Hebert, M.,** *A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces,* Proc. of the 8th International Joint Conference on Artificial Intelligence (Karlsruhe, West Germany, Aug. 8-12), IJCAI, page 998-1001.

47. **Faugeras, O.D., Hebert, M., Mussi, P., Boissonnat, J.D.,** *Polyhedral approximation of objects without holes,* Proc. of the Pattern recognition

and Image Processing Conference (Las Vegas, Nevada, June 14-17), IEEE, New York, page 593-598.

48. **Farin, G.,** *Smooth interpolation to scattered 3D data,* in Barnhill, R.E., and Boehm, W. (eds) Surface in Computer-Aided Geometric Design, North-Holland, Amsterdam, Netherlands, 1983.

49. **Faux, I.D., Pratt, M.J.,** *Computational geometry for design and manufacture,* Ellis Horwood, Chichester, U.K., 1979.

50. **Fuchs, H., Kedem, Z., Uselton, S.P,** *Optimal surface reconstruction from planar contours,* Communications of ACM, Volume 20, Number 10, October 1977, page 69-75.

51. **Ganapathy, S., Dennehy, T.G.,** *A new general triangulation method for planar contours,* ACM Computer Graphics, July 1982, Volume 16, Number 3, page 69-75.

52. **Gilheany, R., Treywin, E.T.,** *Developments in three co-ordinate measuring machines and associated software,* International Metrology Conference Nelex 80, paper 2.2, 7-9 October 1980.

53. **Green, P.J., Sibson, R.,** *Computing Dirichlet tessellation in the plane,* The Computer Journal, Volume 21, Number 2, page 168-173.

54. **Groover, M.P., Zimmers, E.W.,** *CAD/CAM Computer-aided design and manufacturing,* Prentice/Hall International Editions, 1984.

55. **Harrington, S.,** *Computer graphics, a programming approach,* Mc Graw-Hill Inc., 1983.

56. **Hartigan, J.A.,** *Clustering algorithms,* Wiley, New York, 1975.

57. **Hawkes, B.,** *The CADCAM process,* Pitman Publishing, 1988.

58. **Henderson, T.,** *Efficient 3-D object representations for industrial vision systems,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-5, No 6, Nov 1983, page 609-617.

59. **Henderson, T.C.,** *An efficient segmentation method for range data,* SPIE Conf. Robot Vision, Arlington, VA, May 1982, page 46-47.

60. **Henderson, T.C, Bhanu, B.,** *Three-point seed method for the extraction of planar faces from range data,* IEEE Workshop on Industrial Applications of Machine Vision, Research Triangle Park, May 1982, page 181-186.

61. **Henry, G.K.,** *Three-dimensional vision by laser triangulation,* Ph. D. dissertation, University of Bath, 1988.

62. **Horn, B.K.P.,** *Extended Gaussian images,* Proc. of IEEE, December 1972, page 1656-1678.

63. **Horn, B.K.P., Ikeuchi, K.,** *The mechanical manipulation of randomly oriented parts,* Sci. Amer. 251, August 1984, page 100-111.

64. **Ikeuchi, K.,** *Reconstruction of three-dimensional objects using the extended Gaussian image,* Proc. of the 7th International Joint Conference on Artificial Intelligence (Vancouver, B.C., Canada, Aug. 24-28), IJCAI, page 595-600.

65. **Jambu, M., Lebeaux, M.O.,** *Cluster analysis and data analysis,* North-Holland Publishing Company, Netherlands, 1983.

66. **Jarvis, R.A.,** *A perspective on range-finding techniques for computer vision,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-5, Number 2, March 1983, page 122-139.

67. **Kargas, A., Cooley, P., Richards, T.H.E.,** *Interpretation of engineering drawings as solid models,* Computer-Aided Engineering Journal, Volume 5, Number 2, April 1988, page 67-78.

68. **Keppel, E.,** *Approximating complex surfaces by triangulation of contour lines,* IBM J. Res. Develop., Number 19, Jan. 1975, page 2-11.

69. **Krouse, J.K,** *What every engineer should know about CAD/CAM,* Marcel Dekker Inc., New York and Basel, 1982.

70. **Lawson, C.L.,** *Generation of triangular grid with application to contour plotting,* California Institute of Technology, Jet Propulsion Laboratory, Number 299, 1972.

71. **Lee, D.T.,** *Medial axis transformation of a planar shape,* IEEE Trans. on Pattern Analysis and Machine Intelligence, Volume PAMI-4, Number 4, July 1982, page 363-369.

72. **Lee, D.T., Schacter, B.J.,** *Two algorithms for constructing a Delaunay triangulation,* Int. Journal of Computer and Information Sciences, Volume 9, Number 3, 1980.

73. **Lewis, B.A., Robinson, J.S.,** *Triangulation of planar regions with applications,* The Computer Journal, Volume 21, page 324-332, 1978.

74. **Little, J.J.,** *An iterative method for reconstructing convex polyhedra from extended Gaussian images,* Proc. of National Conference on Artificial Intelligence (Washington D.C., Aug 22-26), AAAI, page 247-250.

75. **Little, J.J.,** *Extended Gaussian Images, mixed volumes, and shape reconstruction,* Proc. First ACM Symposium on Computational Geometry, (Baltimore, 5-7 June 1985), page 15-23.

76. **Lloyd, E.L.,** *On triangulation of a set of points in the plane,* Technical Report MIT/LCS/TM-88, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1977.

77. **Lorensen, W.E., Cline, H.E.,** *Marching cubes: A high resolution 3D surface construction algorithm,* ACM Computer Graphics, July 1987, Volume 21, Number 4, page 163-168.

78. **Lowther, D.A., Silvester P.P,** *Computer-aided design in magnetics,* Springer-Verlag, 1985.

79. **McCollum, A.J., Bachelor B.G., Cotter, S.M.,** *Three-dimensional optical sensing,* Proc. 7th International Conference on Automated Inspection and Product Control, Birmingham 1985, IFS Publications, page 161-176.

80. **Meguid, S.A.,** *Integrated computer-aided design of mechanical systems,* Elsevier Applied Science London and New York, 1987.

81. **Mullineux, G.,** *CAD: Computational concepts and methods,* Kogan Page Ltd., 1986.

82. **Newman, W.M., Sproul, R.F.,** *Principles of Interactive Computer Graphics,* Mc Graw Hill, 1979.

83. **O'Rourke, J.,** *Polyhedra of minimal area as 3D object models,* Proc. of the 7th International Joint Conference on Artificial Intelligence, IJCAI, page 664-666.

84. **Oxley, A.,** *Surface fitting by triangulation,* The Computer Journal, Volume 28, Number 3, page 335-339, 1985.

85. **Parthasarathy, S., Birk, J., Dessimoz, J.,** *Laser range-finder for robot control and inspection,* Proc. of the Society for Photo-Optical Instrumentation Engineers Conference on Robot Vision, Volume 336, (Arlington, Va., May 6-7), SPIE, Bellingham, Washington, page 2-11.

86. **Pavlidis, T.,** *A review of algorithms for shape analysis,* Computer Graphics Image Processing, Volume 7, page 243-258, 1978.

87. **Phelan, N.R.,** *Personal communications,* University of Bath.

88. **Piper, B.R.,** *Visually smooth interpolation with triangular Bezier patches,* in Farin, G. (ed) Geometric Modelling: algorithms and trends SIAM, Philadelphia, USA, 1987.

89. **Plummer, J.C.S.,** *Making full use of a solid model database,* CME July/August 1985, page 20-24.

90. **Potmesil, M.,** *Generating models of solid objects by matching 3-D surface matches,* Proc. of the 8th International Joint Conference on Artificial Intelligence, IJCAI, page 1089-1093.

91. **Poulter, K.F.,** *Computer-aided dimensional engineering metrology, advantages and disadvantages,* Proc. of the 7th International Conference on Automated Inspection and Production Control, Birmingham 1985, IFS Publications, page 37-44.

92. **Preparata, F., Hong, S.,** *Convex hulls of finite sets of points in two and three dimensions,* Commun. Ass. Comput. Mach., Volume 20, page 87-93, 1977.

93. **Preperata, F.P., Shamos, M.I.,** *Computational geometry - an introduction,* Springer-Verlag, New York, 1985.

94. **Rhynsburger, D.,** *Analytic delineation of Thiessen polygons,* Geographical analysis, Volume 5, April 1973, page 133-144.

95. **Rogers, D.F.,** *Procedural elements for computer graphics,* McGraw-Hill Inc., 1985.

96. **Rooney, J., Steadman, P.,** *Principles of computer-aided design,* Pitman Publishing, 1987.

97. **Sadek, E.A.,** *A scheme for the automatic generation of triangular finite elements,* International Journal for Numerical Methods in Engineering, Volume 15, page 1813-1822, 1980.

98. **Schmitt, F.J.M, Barsky, B.A, Du, W.H,** *An adaptive subdivision method for surface-fitting from sampled data,* ACM Computer Graphics, Volume 20, Number 4, August 1986, page 179-188.

99. **Shamos, M.I., Hoey, D.,** *Closest-point problems,* Proceedings of the 16th Annual Symposium on the Foundations of Computer Science, page 151-162, October 1975.

100. **Shapira, R., Freeman, H.,** *Reconstruction of curved surface bodies from a set of imperfect projections,* Proc. of the 5th International Joint Conference on Artificial Intelligence (Cambridge, Mass., Aug. 22-25), IJCAI, page 628-634.

101. **Sibson, R.,** *Locally equiangular triangulations,* The Computer Journal, Volume 21, Number 3, page 243-245, 1978.

102. **Sibson, R.,** *SLINK: An optimally efficient algorithm for the single-link cluster method,* The Computer Journal, Volume 16, Number 1, page 30-34, 1972.

103. **Sibson, R.,** *Studies in the robustness of multi-dimensional scaling: Procrustean scaling,* Journal of the Royal Statistical Society, Series B, Volume 40, page 234-238, 1978.

104. **Silverman, B.W.,** *Using Kernel density estimates to investigate multimodality,* Journal of the Royal Statistical Society, Series B, Volume 43, page 97-99, 1981.

105. **Staugaard, A.C.Jr.,** *Robotics and AI: an introduction to applied machine intelligence,* Prentice Hall, Inc., 1987.

106. **Toussaint, G.T.,** *Pattern recognition and geometric complexity,* Proc. of the 5th International Conference on Pattern Recognition, IEEE, New York, page 1324-1347.

107. **Vemuri, B.C., Aggarwal, J.K.,** *Three-dimensional reconstruction of objects from range data,* Proc. of the 7th International Conference on Pattern Recognition, IEEE, New York, page 752-754.

108. **Walker, I.,** *Personal communications,* University of Bath.

109. **Wallis, A.F., Woodwark, J.R.,** *Interrogating solid models,* Proceedings of CAD-84, Butterworths, 1984.

110. **Wallis, A.F.,** *Personal communications* University of Bath.

111. **Watson, D.F.,** *Computing the n-dimensional Delaunay tessellation with applications to Voronoi polytopes,* The Computer Journal, Volume 24, Number 2, 1981, page 167-172.

112. **Woodwark, J.R.,** *Computing shape,* Butterworths, 1986.

113. **Woodwark, J.R.,** *Solid modelling - the set-theoretic approach,* BCS Displays Group, Fundamentals of Geometric Modelling - Review and Potential, Cafe Royal, London, February 1986.

114. **Woodwark, J.R.,** *Shape models in computer integrated manufacture-a review,* Computer-Aided Engineering Journal, Volume 5, Number 3, June 1983, page 103-112.

115. **Woodwark, J.R., Bowyer, A.,** *Better and faster pictures from solid models,* IEEE Computer Aided Enginering Journal, Volume 3, Number 2, 1986.

116. **Wordenweber, B.,** *Automatic mesh generation of two and three-dimensional curvilinear manifolds,* PhD. dissertation, Computer Laboratory, Cambridge University, 1981.

117. **Wordenweber, B.,** *Surface triangulation for picture production,* IEEE Computer Graphics and Applications, Volume 3, Number 8, November 1983, page 45-51.

118. **Zahn, C., Roskies, R,** *Fourier descriptors for plane closed curves,* IEEE Trans. Computing, Volume C-21, page 269-280, 1972.

LIST OF PUBLICATIONS

Cakir, M.C., Bowyer, A.

Matching Measured Components into Solid Models

Proc. of the International Conference on Theory and Practice of Geometric Modelling, Blaubeuren, W.Germany, October 1988.

APPENDIX A

USER INTERFACE

A.1 Introduction

This appendix gives some idea about how to use the software. In the first part the inputs and outputs of the triangulation and finding-the-surface-triangles steps will be explained. In the later sections the inputs and outputs of clustering will be described. The matching algorithm will be explained in section 7.7.

A.2 Triangulation of the Surface Points

The input to the triangulation algorithm is the only information available at the start: the coordinates of the surface points of the component. The algorithm first asks the user if plotting the surface points is desired and if the answer is *Y*, it plots them onto a raster scan graphics display. Different views of surface points can be obtained by simply asking the algorithm to plot the surface points again and again and defining a new view point for each plot. The next question is whether the user wishes to continue to triangulate the surface points or not (see **Figure A1.1**). If the answer is *Y* the algorithm continues, otherwise it stops. In order to see what the ray parameters are at the intersection points with each tetrahedron (to classify the tetrahedra as solid or air), the fourth question in figure A1.1 should also be answered as *Y*.

As seen from the HELP screen there are a couple of steps to triangulate the surface points and to find the surface triangles. The first command, *EXT*, finds the maximum and minimum of the coordinates of the surface points. This information is needed to form a structure around the surface points which guarantees all the surface points are inside. The next command *S* starts the Delaunay triangulation. The four points shown in figure A1.1 (point 1, point 2, point 3, and point 4) are the vertices of the tetrahedron that encloses the surface points. For the given example 4077 surface points are processed to generate the volumetric representation of the measured component. The Delaunay tetrahedra (that form the volumetric three-dimensional structure) and the Dirichlet tessellation can be plotted if *P* is typed as the next command. A windowing facility is also available for the close

```
Do you want to plot the data ? : Y
Measurements file: STEPS.MES

Data range is (   -0.002   -0.002   -0.003) to (    8.002    6.002    6.002)

View position: 4,-15,3
Centre of view: 4,3,3
Top of view: 4.000  -14.000   3.845
Graphics device: B
PLTON: Erase the screen? Y
Do you want to plot the again ?: N

Do you want to continue ? : Y

Do you want to write the data ?: N

If you want any help press  h :H

  Print  ext  to find the maximum and minimum of the given data and
           to form a convex hull
  Press  s  to start the DELAUNAY TRIANGULATION
  Press  p  to plot the triangulation
  Press  l  for the list of neighbouring vertices
  Press  t  to find the tetrahedra which are AIR
  Print DORA to run Dora for the second time to eliminate the extra triangles
  Print SURTRI to make the list of triangles which are on the surface
  Print STOP to stop the program

  But except listing and plotting the triangulation,
  please do these steps in order !


Command ? :EXT
Measurements file: STEPS.MES

  Minimum point : (   -0.002   -0.002   -0.003 )
  Maximum point : (    8.002    6.002    6.002 )


Command ? :S

  Point 1: (    -1.002    -1.002    -1.003 )
  Point 2: (    25.012    -1.002    -1.003 )
  Point 3: (    -1.002    25.012    -1.003 )
  Point 4: (    -1.002    -1.002    25.011 )

 Number of points :       4077


Command ? : P
Measurements file: MODEL.MES
Do you want to open a window ? : Y
Bottom-left corner : 1,1,1
Top-right corner   : 3,3,3
Graphics device: A
```

Cont'd...

```
Command ? : T

  For vertex V    2362 :
  There is NO intersection with this tetrahedron, try the others


  For vertex V     5628 :
  Intersection in first tetrahedron
  Ray parameter T( 2 ): .5333372E+00


  For vertex V     1186 :
  There are 2 intersections with this tetrahedron and this tetrahedron is AIR
  Ray parameter T( 1 ): .5333372E+00
  Ray parameter T( 2 ): .6956486E+00


 For vertex V      2129 :
 There are 2 intersections with this tetrahedron and this tetrahedron is AIR
 Ray parameter T( 3 ): .6956486E+00
 Ray parameter T( 1 ): .1000000E+01


 Now we are OUTSIDE,try the other starting point


Command ? : DORA
Command (.DOR) file: STEPS
Is this your first running of DORA ?:N
How far away do you want to go (in cm.)? :10.0
Your epsilon : 0.001


Command ? : SURTRI
Measurements file :STEPS


Command? : STOP

FORTRAN STOP
```

Figure A1.1 Triangulation and finding surface triangles

observations. The command *L* prints the list of neighbouring vertices for each Delaunay vertex onto the screen.

Once the three-dimensional structure is formed from the aggregation of a set of Delaunay tetrahedra (with the measured surface points as the vertices), the tetrahedra which are solid are determined by typing *T* as the next command. Only the ray parameters of the first ray (at the intersection points with the tetrahedra on its way) are shown in figure A1.1 to give some idea about the outputs of the algorithm.

If *DORA* is typed next (which it normally should be), the process of eliminating the redundant tetrahedra is started. The algorithm reads the half-space information of the solid model of the component and determines where the faces of the component should be. The algorithm then asks if the user is running the solid modeller for the first time and the answer should be *N* to this question. As mentioned in Chapter 5, in order to eliminate the redundant tetrahedra DORA needs to be run for the second time (first run was to simulate the data gathering process and this second run is to simulate the measuring machine's being on-line). For the example given in figure A1.1 the distance which is used to find any redundant tetrahedron is given as 10 cm. The algorithm moves away from the centroid of each tetrahedron at this given distance in positive and negative *x,y,* and *z* directions consecutively, and sends a ray onto the centroid of the tetrahedron to determine whether this tetrahedron is redundant or not (see figure 5.3).

After eliminating the redundant tetrahedra, the surface triangles are found by typing *SURTRI* as the next command. The output of surface triangles is written

into a given file.

This is how the algorithm triangulates the surface points and finds the surface triangles. As written in the HELP menu, apart from listing the neighbouring vertices and plotting the triangulation and tessellation all these steps explained should be done in the given order. The program can be stopped at any stage by typing *STOP* as the next command.

A.3 Clustering the Surface Triangles

The clustering algorithm first reads in the surface triangles from a given file (see **Figure A2.1**). It then asks the user to define the constant which will be used in clustering (it checks the scalar product of two surface normals, if the result is greater than or equal to this given constant, it clusters these surface normals together).

The mid-distance factor which is asked to be defined as the next factor is the factor which determines the place of the mid-plane. Once the clusters were formed the position of mid-plane of each cluster is determined by finding the difference between the maximum and minimum distance values of each cluster and multiplying this difference with the given mid-distance factor.

Next, the weighting factor needs to be determined. The clustering algorithm uses this factor to decide to which cluster the surface triangle belongs when two of its neighbours are in the same cluster and the third one is in different cluster. As this factor gets bigger the chance of the surface triangle being clustered with its

```
.SUR file :STEPTE.SUR

Constant :0.98

Mid Distance Factor:0.52

Weighting Factor : 4.0

 Number of single clusters  :    36
 Number of changed colours  :   322

 Number of single clusters  :     0
 Number of changed colours  :     0

Weighting Factor : 4.0

 Number of single clusters  :    10
 Number of changed colours  :   214

 Number of single clusters  :     0
 Number of changed colours  :     0

Number of faces : 9
 Area of face   1 : 19.2
 Area of face   2 : 32.9
 Area of face   3 : 48.0
 Area of face   4 : 13.8
 Area of face   5 : 32.9
 Area of face   6 : 12.0
 Area of face   7 : 42.0
 Area of face   8 : 18.0
 Area of face   9 : 25.8

 Cluster   5 is doubtfull. NO merging.
 Might be a false chamfer

 Cluster   8 is doubtfull. NO merging.
 Might be a false chamfer

 Doubtful cluster   9 is merged with cluster   4

 Doubtful cluster  11 is merged with cluster  13

 Cluster  14 is doubtfull. NO merging.
 Might be a false chamfer

Weighting Factor : 4.0

 Number of single clusters  :     0
 Number of changed colours  :     1

 Number of single clusters  :     0
 Number of changed colours  :     0


Number of clusters :  9
```

Figure A2.1 Clustering the surface triangles

two neighbours gets higher. The algorithm also reports the number of surface triangles whose three neighbours are in three clusters which are different from the the surface triangle's cluster (single cluster) and the number of surface triangles whose clusters were changed. The algorithm asks for the weighting factor for the second time when it re-organises the clusters after clustering the surface triangles which are lying in parallel planes.

The algorithm then asks the number of faces of the solid model and their surface areas. By using this information and calculating the surface areas of the measured component, it finds which of the clusters are doubtful (they are the clusters whose surface areas are much smaller than the minimum surface area of the solid model). If the number of clusters which are found by the algorithm are more than the number of the faces of the solid model, it either tries to merge the doubtful cluster with any other cluster (it does this by fitting a plane to each cluster and calculating the scalar product between the doubtful cluster and any other clusters. If the result is greater than or equal to the given constant, the doubtful cluster is the part of this cluster) or if no merging is possible it classifies this doubtful cluster as a false chamfer. It organises the clusters again, tests single clusters, changes their clusters and outputs the number of clusters (which is obviously equal to the number of the faces of the solid model).

APPENDIX B

ROTATIONS

B.1 Introduction

This appendix gives general information about the three-dimensional rotations about x,y,z and arbitrary axes.

B.2 Rotations

In three-dimensions the axis about which the rotation will take place needs to be determined. Rotation of a point about the z axis through an angle θ is given by:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where the rotation angle θ is measured anti-clockwise about the origin when looking at the origin from a point on the $+z$ axis.

Thus, rotation about the x axis is:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

and rotation about y axis is given by:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

In order to rotate in clockwise direction a negative angle is used.

When the objects are rotated sequentially about x, y and z axes, the rotation matrix P can be calculated as:

$$P = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$P = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

For the given rotation matrix P:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

the rotation angles about the axes are calculated as:

$$\alpha = Atan\left(\frac{p_{21}}{p_{11}}\right)$$

$$\beta = Atan\left(\frac{-p_{31}}{\sqrt{p^2_{11} + p^2_{21}}}\right)$$

$$\gamma = Atan\left(\frac{p_{32}}{p_{33}}\right)$$

where γ is the rotation angle about the x axis, β about the y axis and α about the z axis.

If the axis of rotation is a general axis, the equivalent rotation matrix is given by:

$$P = \begin{bmatrix} k_x k_x v\Theta + c\Theta & k_x k_y v\Theta - k_z s\Theta & k_x k_z v\Theta + k_y s\Theta \\ k_x k_y v\Theta + k_z s\Theta & k_y k_y v\Theta + c\Theta & k_y k_z v\Theta - k_x s\Theta \\ k_x k_z v\Theta - k_y s\Theta & k_y k_z v\Theta + k_x s\Theta & k_z k_z v\Theta + c\Theta \end{bmatrix}$$

where $c\Theta = \cos\Theta$, $s\Theta = \sin\Theta$, $v\Theta = 1 - cos\Theta$; k_x, k_y, k_z are the coefficients of the general axis $\vec{K}$ and Θ is the rotation angle about the axis.

If the rotation matrix P is given:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

then the rotation angle and the general axis can be calculated as:

$$\Theta = Acos\left(\frac{p_{11} + p_{22} + p_{33} - 1}{2}\right)$$

$$\vec{K} = \frac{1}{2\sin\Theta} \cdot \begin{bmatrix} p_{32} - p_{23} \\ p_{13} - p_{31} \\ p_{21} - p_{12} \end{bmatrix}$$

APPENDIX C

FURTHER RESULTS

C.1 Introduction

In this appendix some more results will be given on matching the measured components with their solid model. These results include the cases where symmetry or missing or extra faces of the measured component yields some mis-matches and the cases where the measured component matches perfectly with its solid model.

C.2 Other Results

The result of the reconstruction and the matching of another component is shown in **Figure C1.1**. This measured component was also perfectly matched with its solid model. No missing or extra face of the measured component and no rotation or translation between the two was reported.

In the other result shown in **Figure C1.2** the translation of 10 cm. and the rotations of 60 degrees about x axis, 30 degrees about the y axis and 45 degrees about z axis of the measured component were recovered, and the faces of the solid model were matched perfectly with the faces of the measured component. Minus sign in rotation angles indicates the clockwise direction of the rotations.

C.3 Symmetry Problem

In **Figure C2.1** the result of matching a symmetric U-shaped measured component with its solid model is shown. As seen from **Figure C2.2**, not only the perpendicular distances from the plane centre, but also the surface areas of some of the faces, are the same. Since the matching algorithm uses these measures to scale the surface normals and the information to make the decision is insufficient, it fails to find the best match when more than one face corresponds to a face of the solid model. Thus it recovers the wrong rotation angle (although it recovers the right translation). There is no rotation or translation between the measured component and the solid model in figure C2.1. As discussed in Chapter 8, possible 180 degree rotations about x, y and z axes were tried to improve the results but no improvement was observed. The default value for the rotation angles in figure C2.1 was

```
First .HSP file : stepte.hsp
Second .HSP file : stepteo.hsp

Recovered offset:    0.0037  -0.0089  -0.0021

Do you want to see the output ?: N

Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:      0.0000
Recovered angle around X axis:   0.0339
Recovered angle around Y axis:   0.3514
Recovered angle around Z axis:   0.1143

Face :     8    (    0.0000000    0.0000000    1.0000000    1.5224915  )
Face :     3    (   -0.0000132   -0.0022024    0.9999976    1.5208458  )

Face :     6    (    0.0000000    0.0000000    1.0000000   -3.4775085  )
Face :     6    (    0.0022369   -0.0026080    0.9999941   -3.4697677  )

Face :     9    (   -0.7071068    0.0000000   -0.7071068   -1.7179902  )
Face :     4    (    0.7070202   -0.0013239    0.7071920    1.7191496  )

Face :     4    (    1.0000000    0.0000000    0.0000000   -4.0928869  )
Face :     9    (    0.9999999    0.0003098    0.0001418   -4.0892870  )

Face :     1    (   -1.0000000    0.0000000    0.0000000   -3.9071131  )
Face :     2    (    0.9999999    0.0003935    0.0001954    3.9107765  )

Face :     7    (    0.5000109    0.0000000    0.8660190   -2.0580111  )
Face :     7    (    0.4999631   -0.0016860    0.8660451   -2.0579853  )

Face :     5    (    0.0000000    1.0000000    0.0000000   -2.9999998  )
Face :     8    (    0.0003025   -0.9999976   -0.0021898    3.0089600  )

Face :     2    (    0.0000000   -1.0000000    0.0000000   -3.0000002  )
Face :     1    (   -0.0002823    0.9999976    0.0021837    2.9911051  )

Face :     3    (    0.0000000    0.0000000   -1.0000000   -2.5224915  )
Face :     5    (   -0.0001303    0.0042500    0.9999909    2.5153420  )

Satisfied ? Or do you want to try rotations around the axes [180.0] : N

Do you want to plot the results later on ?:N
```

Figure C1.1 Matching of a measured component with its solid model

```
First .HSP file : stepso.hsp
Second .HSP file : stepsr3t.hsp
Recovered offset:   -0.0051  -0.0162  10.0016

Do you want to see the output ?: N

Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:      0.0000
Recovered angle around X axis: -59.8692
Recovered angle around Y axis: -29.8770
Recovered angle around Z axis: -45.0582

Face :    10    (    0.9999995    0.0000089    0.0009424    0.2570003  )
Face :     9    (    0.9999999    0.0001249   -0.0001175    0.2535236  )

Face :     4    (    1.0000000    0.0000238   -0.0000272    1.7445094  )
Face :     7    (    0.9999999    0.0001249   -0.0001175    1.7464784  )

Face :    13    (   -0.0000142    0.0000107    1.0000000    0.7959593  )
Face :    12    (    0.0001179   -0.0029180    0.9999957    0.7887317  )

Face :     8    (   -0.0000575    0.0000061    1.0000000    1.2039199  )
Face :    10    (    0.0001179   -0.0029180    0.9999957    1.2112693  )

Face :    14    (    0.0001985    0.0000004    1.0000000    2.2039414  )
Face :     8    (    0.0001179   -0.0029180    0.9999957    2.2112693  )

Face :     9    (    1.0000000   -0.0000019   -0.0000180    2.2554801  )
Face :    11    (    0.9999999    0.0001249   -0.0001175    2.2535236  )

Face :     3    (   -0.0000861   -0.0000443    1.0000000    1.7957899  )
Face :    13    (    0.0001179   -0.0029180    0.9999957    1.7887317  )

Face :    11    (   -0.0014797    0.0000422    0.9999989    3.2077238  )
Face :     6    (    0.0001179   -0.0029180    0.9999957    3.2112683  )

Face :     5    (   -0.7084410   -0.0000076   -0.7057700    1.8008103  )
Face :    14    (   -0.7071902    0.0019750   -0.7070206    1.7926649  )

Face :    12    (    1.0000000    0.0000485   -0.0001330    4.2556951  )
Face :     4    (    0.9999999    0.0001249   -0.0001175    4.2535236  )

Face :     2    (   -1.0000000    0.0001081    0.0000422    3.7435509  )
Face :     1    (   -0.9999999   -0.0001249    0.0001175    3.7464784  )

Face :     6    (   -0.0000187    1.0000000   -0.0000352    2.9907467  )
Face :     5    (   -0.0001245    0.9999956    0.0029180    2.9999992  )

Face :     1    (    0.0000404   -1.0000000   -0.0000626    3.0094717  )
Face :     2    (    0.0001245   -0.9999956   -0.0029180    2.9999958  )

Face :     7    (   -0.0000716    0.0065240   -0.9999787    2.7524774  )
Face :     3    (   -0.0001179    0.0029180   -0.9999957    2.7887317  )

Satisfied ? Or do you want to try rotations around the axes [180.0] : N

Do you want to plot the results later on ?: N
```

Figure C1.2 Matching of a measured component with its solid model

180 degrees.

C.4 Missing or Extra Faces of the Measured Component

As discussed in Chapter 7, the algorithm fails to match some faces of the measured component with its solid model's if the measured component has a different number of faces from its solid model. Since the different number of faces change the position of the centre point (which is minimum distance away from the faces), the radial distances from the centre point were also changed and some mis-matches occur. **Figure C3.1** shows this sort of matching. The measured component in figure C3.1 has an extra face, *Face 15* and because of this extra face, 3 faces were mis-matched (including the extra face) and the rotation angle is recovered by 2 degrees difference.

```
First .HSP file : samp.hsp
Second .HSP file : sampo.hsp

Recovered offset:   -0.0003   0.0008  -0.0140

Do you want to see the output ?: N

Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:      0.0000
Recovered angle around X axis:  -0.0008
Recovered angle around Y axis:  -0.0003
Recovered angle around Z axis: -63.1604

Face :     9    (    0.0000000    0.0000000   -1.0000000    0.0000000  )
Face :     2    (    0.0000043    0.9998813   -0.0154073   -0.0139407  )

Face :     7    (    0.0000000   -1.0000000    0.0000000   -1.0000000  )
Face :     7    (    0.0006424   -0.0413373   -0.9991450   -0.9879967  )

Face :     8    (    0.0000000    1.0000000    0.0000000   -1.0000000  )
Face :     5    (    0.0000567   -0.0009306    0.9999995   -0.9860520  )

Face :     6    (    0.0000000    0.0000000    1.0000000   -2.0000000  )
Face :     1    (    0.0000038    0.0153857    0.9998816   -3.0008683  )

Face :     1    (   -1.0000000    0.0000000    0.0000000   -5.0000000  )
Face :     4    (   -1.0000000   -0.0000121    0.0000049   -5.0003550  )

Face :     4    (    1.0000000    0.0000000    0.0000000   -5.0000000  )
Face :     3    (   -1.0000000    0.0000062   -0.0000080    4.9996801  )

Face :     2    (    0.0000000   -1.0000000    0.0000000   -3.0000000  )
Face :     6    (    0.0000172    0.9998818   -0.0153744    1.9860445  )

Face :     5    (    0.0000000    1.0000000    0.0000000   -3.0000000  )
Face :     8    (    0.0000020    0.9998816   -0.0153896   -2.0140293  )

Face :     3    (    0.0000000    0.0000000   -1.0000000   -2.0000000  )
Face :     9    (   -0.0000006    0.0154181    0.9998811    2.9991558  )

Satisfied ? Or do you want to try rotations around the axes [180.0] : Y

Rotation angle around X axis [180.0] :
Rotation angle around Y axis [180.0] :
Rotation angle around Z axis [180.0] :

Sorry, rotations around axes has NO use !

Do you want to plot the results later on ?:N
```

Figure C2.1 The effect of the symmetry problem on matching

Solid Model

Plane Coefficients of SAMP Ax + By + Cz + D = 0

	A	B	C	D	Areas
Face 1	-0.1000000E+01	0.0000000E+00	0.0000000E+00	0.0000000E+00	20.0000
Face 2	0.0000000E+00	-0.1000000E+01	0.0000000E+00	0.0000000E+00	40.0000
Face 3	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.0000000E+00	60.0000
Face 4	0.1000000E+01	0.0000000E+00	0.0000000E+00	-0.1000000E+02	20.0000
Face 5	0.0000000E+00	0.1000000E+01	0.0000000E+00	-0.6000000E+01	40.0000
Face 6	0.0000000E+00	0.0000000E+00	0.1000000E+01	-0.4000000E+01	40.0000
Face 7	0.0000000E+00	-0.1000000E+01	0.0000000E+00	0.2000000E+01	20.0000
Face 8	0.0000000E+00	0.1000000E+01	0.0000000E+00	-0.4000000E+01	20.0000
Face 9	0.0000000E+00	0.0000000E+00	-0.1000000E+01	0.2000000E+01	20.0000

Measured Component

Plane Coefficients of SAMPO Ax + By + Cz + D = 0

	A	B	C	D	Areas
Face 1	0.2023394E-05	-0.1000000E+01	-0.6642086E-05	0.1213150E-05	40.0171
Face 2	0.9232730E-05	0.1494617E-04	0.1000000E+01	-0.2000031E+01	21.9993
Face 3	0.1000000E+01	0.9649947E-05	-0.7248492E-05	-0.2849862E-05	18.9023
Face 4	0.1000000E+01	-0.2975559E-05	-0.2568686E-04	-0.9999963E+01	18.9055
Face 5	0.5517530E-04	-0.9998668E+00	-0.1632277E-01	0.2047087E+01	19.2166
Face 6	0.3645714E-05	-0.1795530E-04	0.1000000E+01	0.1172760E-03	61.0633
Face 7	0.6444840E-03	0.9996629E+00	-0.2595323E-01	-0.3933066E+01	18.4972
Face 8	0.1154774E-04	-0.2776834E-05	0.1000000E+01	-0.4000078E+01	40.4796
Face 9	0.2339465E-05	-0.1000000E+01	0.2576568E-04	0.5999939E+01	40.0168

Figure C2.2 The plane coefficients of half-spaces and surface normals

```
First .HSP file : steps.hsp
Second .HSP file : stpplo.hsp
Recovered offset:    0.5043  -0.0077  -0.4287

Do you want to see the output ?: N
Do you want to calculate the minimum distance : N
D Term : 0.0

Squared dist between modes:      0.0000
Recovered angle around X axis:  -0.0184
Recovered angle around Y axis:  -2.4140
Recovered angle around Z axis:   0.1557

Face :     9    (   -1.0000000    0.0000000    0.0000000    0.2535212  )
Face :    11    (    0.9990647   -0.0001048    0.0432400    0.2484257  )

Face :    12    (    0.0000000    0.0000000   -1.0000000   -0.7887323  )
Face :    14    (   -0.0422126   -0.0024783    0.9991056    0.3602992  )

Face :     7    (   -1.0000000    0.0000000    0.0000000   -1.7464788  )
Face :     4    (    0.9991230   -0.0000626    0.0418715    2.2515779  )

Face :    10    (    0.0000000    0.0000000   -1.0000000    1.2112677  )
Face :     9    (   -0.0421901   -0.0024793    0.9991065   -1.6398436  )

Face :    12    (    0.0000000    0.0000000   -1.0000000   -0.7887323  )
Face :    10    (    0.9991157   -0.0000630    0.0420448   -1.7491485  )

Face :    11    (   -1.0000000    0.0000000    0.0000000    2.2535212  )
Face :    13    (    0.9991049   -0.0002018    0.0423009   -3.7490228  )

Face :    13    (    0.0000000    0.0000000    1.0000000    1.7887323  )
Face :     3    (   -0.0423452   -0.0024617    0.9991000    1.3594423  )

Face :     8    (    0.0000000    0.0000000   -1.0000000    2.2112677  )
Face :    15    (   -0.0419148   -0.0024830    0.9991181   -2.6398782  )

Face :     4    (    1.0000000    0.0000000    0.0000000   -4.2535212  )
Face :     8    (   -0.7390954   -0.0017076    0.6735985    3.6149382  )

Face :     6    (    0.0000000    0.0000000    1.0000000   -3.2112677  )
Face :    12    (   -0.0436387   -0.0024375    0.9990443   -3.6444119  )

Face :    14    (   -0.7071068    0.0000000   -0.7071068   -1.7926653  )
Face :     5    (    0.6781108   -0.0018001    0.7349574    1.8519076  )

Face :     1    (   -1.0000000    0.0000000    0.0000000   -3.7464788  )
Face :     2    (    0.9991100   -0.0000112    0.0421818    4.2506389  )

Face :     2    (    0.0000000   -1.0000000    0.0000000   -3.0000000  )
Face :     1    (   -0.0000208    0.9999972    0.0023667    2.9922713  )

Face :     5    (    0.0000000    1.0000000    0.0000000   -3.0000000  )
Face :     6    (    0.0000116   -0.9999968   -0.0025197    3.0076690  )

Face :     3    (    0.0000000    0.0000000   -1.0000000   -2.7887323  )
Face :     7    (   -0.0431207    0.0038195    0.9990625    2.3530882  )

Do you want to plot the results later on ?: N
```

Figure C3.1 The effect of an extra face of the measured component on matching

APPENDIX D

MATCHING THE AXES OF CYLINDERS

D.1 Introduction

In order to match the axes of cylinders under translation and rotation the same technique which was used to match the planar faces could be used. One thing that remains invariant under rotation is the distances of axes from the point which is nearest to the axes. In this appendix the algebra which could be used to calculate this nearest point is given.

D.2 Finding the Nearest Point to the Axes

The nearest point to the axes is the point which has the smallest sum of squared distances from the axes (lines). This point is calculated as follows [26] :

1. Find the sum of squared distances from an arbitrary point to all the lines (see [28]).
2. Partially differentiate that sum with respect to x,y and z coordinates of the point and set the partial derivatives to zero to find the minimum (the squared distance sum must be quadratic; therefore it can only have one minimum and one maximum; the maximum value it can take must be infinite; therefore the singular point must be minimum).
3. Solve the resulting linear system to obtain the position of the point. Accumulate the sums needed for the linear equations.

   The linear system will be:

$$\alpha_1 x + \beta_1 y + \gamma_1 z + \eta_1 = 0$$

$$\alpha_2 x + \beta_2 y + \gamma_2 z + \eta_2 = 0$$

$$\alpha_3 x + \beta_3 y + \gamma_3 z + \eta_3 = 0$$

where

$$\kappa_1 = f^4 + (g^2 + h^2 - 2) \cdot f^2 + 1 \qquad \text{and} \qquad \alpha_1 = \sum_{i=1}^{n} \kappa_{1i}$$

$$\psi_1 = g \cdot f^3 + (g^3 + (h^2 - 2) \cdot g) \cdot f \qquad \text{and} \qquad \beta_1 = \sum_{i=1}^{n} \psi_{1i}$$

$$\omega_1 = h \cdot f^3 + (h^3 + (g^2 - 2) \cdot h) \cdot f \qquad \text{and} \qquad \gamma_1 = \sum_{i=1}^{n} \omega_{1i}$$

$$\eta_1 = -\sum_{i=1}^{n} (\kappa_{1i} \cdot x_{0i} + \psi_{1i} \cdot y_{0i} + \omega_{1i} \cdot z_{0i})$$

$$\kappa_2 = f \cdot g^3 + (f^3 + (h^2 - 2) \cdot f \;) \cdot g \qquad \text{and} \qquad \alpha_2 = \sum_{i=1}^{n} \kappa_{2i}$$

$$\psi_2 = g^4 + (f^2 + h^2 - 2) \cdot g^2 + 1 \qquad \text{and} \qquad \beta_2 = \sum_{i=1}^{n} \psi_{2i}$$

$$\omega_2 = h \cdot g^3 + (h^3 + (f^2 - 2) \cdot h \;) \cdot g \qquad \text{and} \qquad \gamma_2 = \sum_{i=1}^{n} \omega_{2i}$$

$$\eta_2 = -\sum_{i=1}^{n} (\kappa_{2i} \cdot x_{0i} + \psi_{2i} \cdot y_{0i} + \omega_{2i} \cdot z_{0i})$$

$$\kappa_3 = f \cdot h^3 + (f^3 + (g^2 - 2) \cdot f \;) \cdot h \qquad \text{and} \qquad \alpha_3 = \sum_{i=1}^{n} \kappa_{3i}$$

$$\psi_3 = g_i \cdot h^3 + (g^3 + (f^2 - 2) \cdot g \;) \cdot h \qquad \text{and} \qquad \beta_3 = \sum_{i=1}^{n} \psi_{3i}$$

$$\omega_3 = h^4 + (g^2 + f^2 - 2) \cdot h^2 + 1 \qquad \text{and} \qquad \gamma_3 = \sum_{i=1}^{n} \omega_{3i}$$

$$\eta_3 = -\sum_{i=1}^{n} (\kappa_{3i} \cdot x_{0i} + \psi_{3i} \cdot y_{0i} + \omega_{3i} \cdot z_{0i})$$

where f, g, h are the line coefficients, x_0, y_0, and z_0 is the origin of each line and i is the number of the lines.

If the intersection point of these three planes is calculated (see [28]), the coordinates of the nearest point which is the minimum distance away from the axes is determined. Once the nearest points of both the axes of the measured component and the solid model's are calculated, configurations are translated so that their nearest points are both at the origin and matched by using the Procrustean algorithm under translation and rotation.

APPENDIX E

PUBLISHED PAPER

This paper - which was given as reference [29] - will be published in the Proceedings of the International Conference on Theory and Practice of Geometric Modelling. The proceedings were in press during the preparation of this thesis.

MATCHING MEASURED COMPONENTS TO SOLID MODELS

M. Cemal Cakir & Adrian Bowyer

School of Mechanical Engineering,
University of Bath,
Bath, England BA2 7AY

Tel: (+44) 225 826826
Email: cc1@uk.ac.bath.maths and ab@uk.ac.bath.maths

Abstract

When components have been made they need to be inspected. In this paper some newly devised automatic methods are described that compare a set of measurements of an engineering component taken by a coordinate measuring machine with a master solid model of the measured component obtained from a CAD system. Once matched, the two may then be compared to find any differences resulting from manufacturing errors and those manufacturing errors can automatically be reported.

1. Introduction

When an engineering component has been manufactured it must often be checked for defects. Ideally such checking should be done against the original design of the component and any out-of-tolerance differences between the two should be reported.

This paper will describe a group of algorithms which allow a collection of points on the surface of a manufactured component (such as might be gathered using a coordinate measuring machine) to be matched automatically with a solid model of the component. **Figure 1** shows the gathered surface points and the solid model of a simple staircase model. Some of these algorithms are extant in the literature on Stochastic Computational Geometry, but have not before been applied to this problem; some are novel and due to the authors. The way in which the matching is achieved is robust in the presence of errors in the component which cause it to differ slightly in shape from the solid model. As the the whole purpose of measurement is to check for such errors, this is particularly important.

The authors' algorithms have been developed especially to handle the large numbers of surface points that may be gathered from a component using a laser non-contact measuring machine developed by one of them and others [3]. However, they would also be quite suitable for use with a conventional coordinate measuring machine.

The solid modeller used by the authors for this work is called DORA. This is a set-theoretic solid modeller developed at Bath by John Woodwark [8]. The algorithms would work just as well with any other set-theoretic or B-rep modeller.

The authors' present system is implemented to deal with faceted components and solid models only. They are currently engaged in extending it to work with curved components.

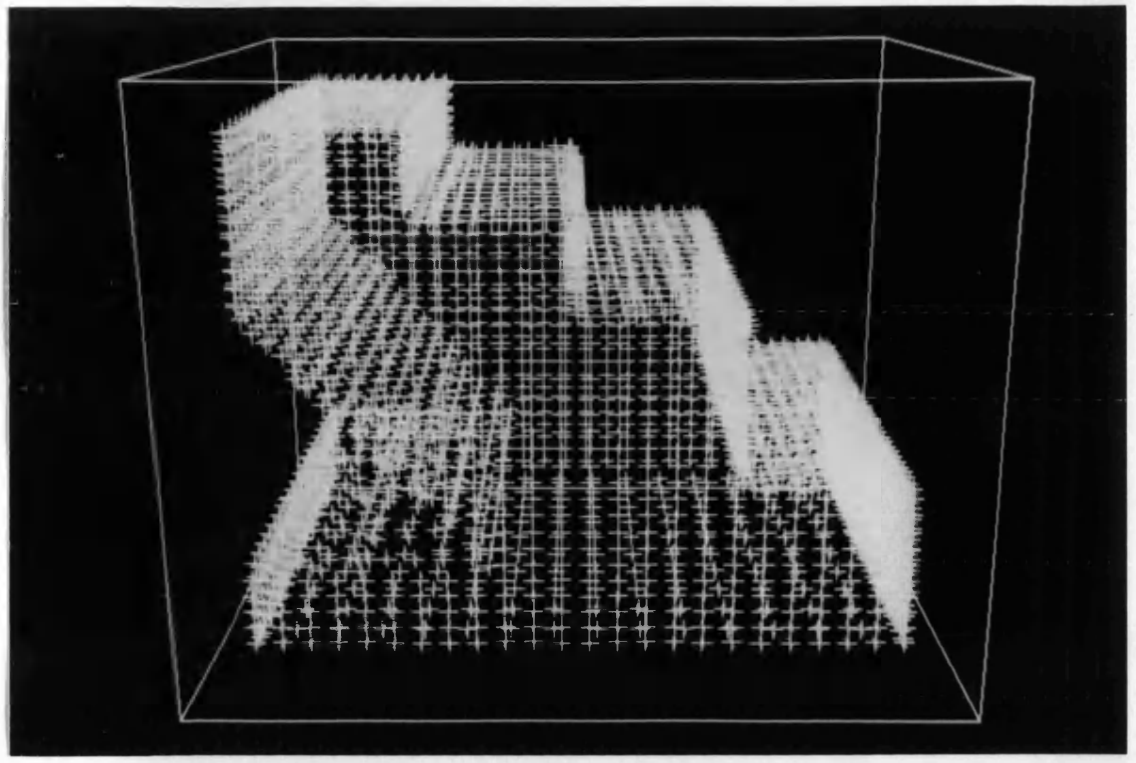


Figure 1a. Gathered surface points

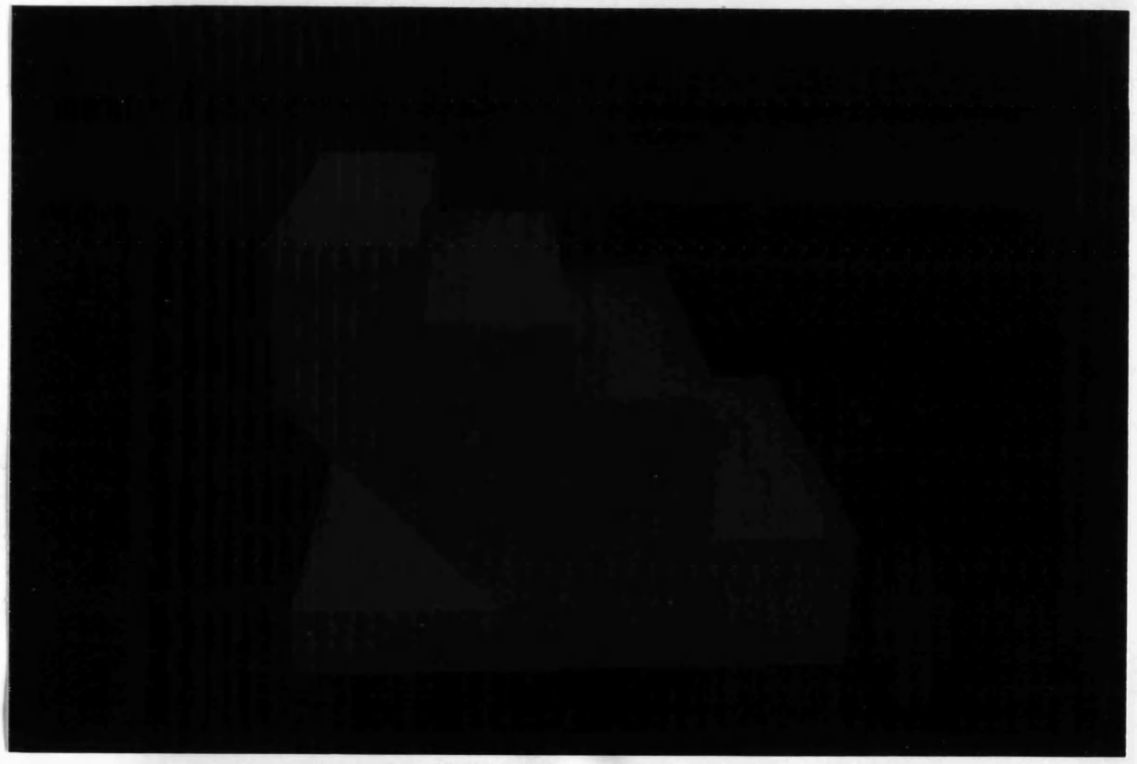


Figure 1b. Solid model of the component

2. The Algorithms

2.1 Triangulation

The data to be matched consist of points in space. The only information about these data (which have to be matched to a collection of solid model primitives) is the positions of the points which lie on the surface of the object, no topological information is available. To obtain topological information about the measured object the *Voroni diagram* or *Dirichlet tessellation* of the measured points is constructed. This technique is substantially similar to one devised by Boissonnat [1], who used an algorithm due to one of the authors (AB). The geometrical dual of the Voroni diagram, obtained by linking the points whose Voroni polyhedra are adjacent across a common face, is called the *Delaunay triangulation*. **Figure 2** shows the Voroni diagram and Delaunay triangulation for a small set of points (15 of them).

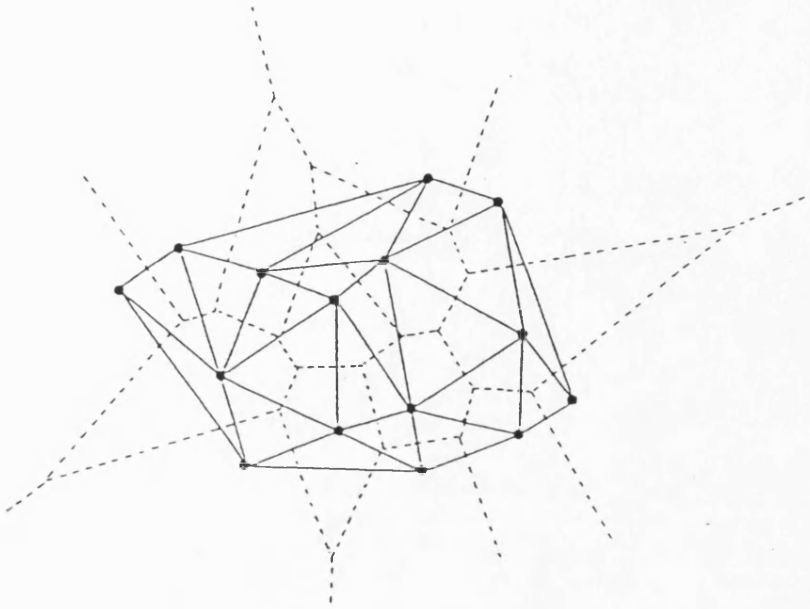


Figure 2. Voroni diagram (dotted lines) and Delaunay triangulation (bold lines).

The authors use an efficient multi-dimensional algorithm devised by one of them to construct the tessellation and triangulation [2]. In three dimensions the Delaunay triangles become a set of packed tetrahedra with the measured points as vertices. They fill the convex hull of the measured points. Some of the triangles that form the surfaces of these tetrahedra will form a complete triangulation of the measured object's surface. The problem to be solved is to find which ones.

2.2 Classification of tetrahedra

To answer this we note that some Delaunay tetrahedra will lie within the measured object and will thus be solid, whereas some will lie outside it and will be air. If the tetrahedra may be so classified, then any of their triangular faces that form a boundary between a solid tetrahedron and an air tetrahedron will be part of the component's surface. How may the tetrahedra be classified in such a way?

In order to measure each point a ray of light must have been directed at it or a measuring probe must have touched it. If the path taken by this is recorded then any Delaunay tetrahedra which it passes through must be air. The algorithm which deals with classification, traces a path (which was used to detect the surface point) backwards from the point, and the tetrahedra which the path passes through are classified as air. Since the point is on the surface and the path is traced backwards all the tetrahedra which the path intersects on its way should be air.

The authors were particularly concerned to deal with data gathered by the laser coordinate measuring machine mentioned in the introduction. Henceforth the path will be considered to be a ray of light, but all the algorithms would work just as well with a mechanical probe path.

The algorithm takes the first surface point and start tracing its ray of light backwards. When it finds the intersection between a triangular face of the first tetrahedron and the ray (since the starting point is one of the forming points, there is always one intersection with the first tetrahedron) it classifies this tetrahedron as air and continues tracing the ray into the tetrahedron which shares the intersected face with the first tetrahedron. After this there are always two intersections with the ray and each tetrahedron (unless the ray intersects an edge or a corner). **Figure 3** shows the ray tracing algorithm in two dimensions.

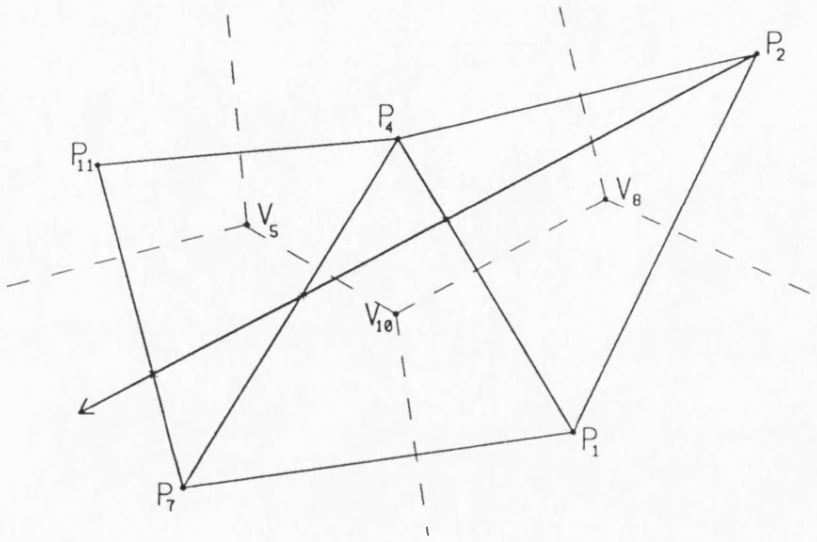


Figure 3. Ray tracing in 2D. P_2 is the starting surface point and dotted lines represents the Voronoi polyhedra.

Since the ray intersects the line P_1P_4 (which corresponds a face in three dimension) the next tetrahedra corresponding to the vertex to be visited is V_{10} which is shares the line P_1P_4 with V_8. After the second intersection with the tetrahedra corresponding to the vertex V_{10}, the next vertex is V_5 which has the common edge P_4P_7 with V_{10} and so on. All the tetrahedra that the ray intersects are classified as air until the ray is beyond the convex hull or when the next vertex visited is already air.

This process is repeated for every surface point and the majority of tetrahedra are thereby classified. Any remaining ambiguities may be resolved by having the measuring machine (which is most useful if on line) take extra measurements which pass through the tetrahedra about which there is still doubt. The algorithm which deals with this problem finds the centroid of each solid tetrahedron, moves a given distance away from the centroid and sends a ray of light towards the centroid. If the surface of the object is beyond the centroid, the tetrahedron is classified as air, otherwise it is solid.

2.3 Clustering

The classification of the tetrahedra as solid or air allows the surface of the object to be found by finding the triangular faces of solid tetrahedra which form a boundary with air tetrahedra. In other words it facilitates the triangulation of the measured component's surface. The next problem to be solved is that of gathering the triangles together in collections, each collection representing a facet of the component.

Each triangle forms a little plane in space. Even on one facet all the triangles will not be exactly co-planar because of measurement errors. The triangles are subjected to cluster analysis to gather them together in collections representing facets. We have used the SLINK algorithm [6] which is based on single-link or nearest neighbour cluster analysis, for a small number of surface points but since the large number of surface points causes us to exceed the memory capacity of the computer with the SLINK algorithm, we have developed a different clustering algorithm. In this algorithm, surface points (which are the vertices of surface triangles) are clustered according to the normals of the planes in which the triangles lie, in other words surface normals.

The scalar product of surface normals of two triangles is calculated and if the result is greater than a number which is close to 1, the two triangles are put into the same cluster. Since the surface normals are normalised the scalar product equals the cosine of the angle between the planes in which the triangles (or surface points) lie. If the result is close to 1 this means that the angle is close to 0 and the triangles lie on the same plane or on parallel ones. The algorithm distinguishes the triangles lying on parallel planes by checking the perpendicular distance from their plane to the origin (the D term in their implicit plane equation, $Ax + By + Cz + D = 0$)

The algorithm defines a mid-plane between the two parallel planes and clusters the triangles which are more distant from origin than the mid-plane in one cluster and the ones which are less distant in another. In the case of more than two parallel planes, first the parallel planes are split into two clusters, each cluster is checked to see whether they contain more than one plane, if they do they are split again and checked again and same process is repeated recursively until the triangles lying in each parallel plane are clustered in separate clusters.

Since the algorithm provides the neighbourhood relationship between the surface triangles (that is to say, for any given triangle, we know its three neighbouring triangles), any mis-clustered triangle is corrected by checking its neighbouring triangles. If all three neighbours are in the same cluster but the triangle is not, it is put into the same cluster with its neighbours. If two neighbours are the same then the surface normals of all three neighbours are inspected and weighting is used to decide if the central triangle ought to be clustered with the pair or not.

Next, the algorithm handles false *chamfers*. False chamfers are artificial features which occur because of the lack of the ability of measuring machine to generate the points exactly on the measured object's edges (**Figure 4**).

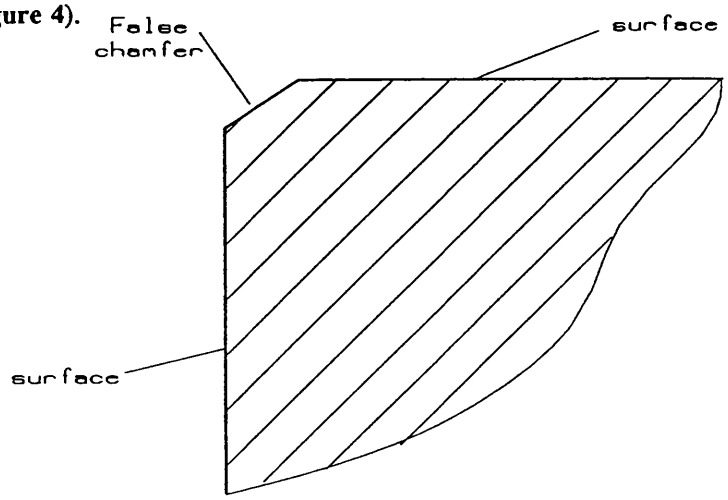


Figure 4. False chamfer

They are products of the measuring system which need to be found to be get rid of. Since the false chamfers should be one triangle wide, the algorithm finds the boundaries of each cluster and thereby finds clusters which are one triangle wide. It classifies these as chamfers and does not consider them to be real faces when matching.

As a result of all the processes the surface triangles (and the surface points obviously) are clustered together in different clusters, each cluster representing a facet. Principle components analysis is then used on the measured points making up the triangle vertices in each collection to obtain a best-fit plane through them all. **Figure 5** shows the result of surface triangulation and clustering on a simple staircase model (the missing lines on some steps are caused by a bug in the plotting program. This will be fixed!). Green colour represents the false chamfers.

Figure 5. Surface triangulation and clustering

2.4 Matching

The measured data and the solid model are now more or less in the same form: two collections of plane equations in space. Unfortunately they will, in general, be referred to different coordinate frames, so, in order to compare them, it is necessary to translate and to rotate them to a best-fit with each other. The authors have extended the technique of *Procrustean matching* [5] to allow this to be done.

The model and the measured planes are first translated so that their centroids are at the origin of coordinates. Either the model or the measured data now have to be rotated. Standard Procrustean rotation requires a known one-to-one correspondence between the two collections of points that are to be matched. In Procrustean rotation, for two roughly similar given configurations, X and Y, each of N points in K dimensional space, Sibson [5] shows that the best match under rotation is obtained by **PY** where orthogonal KxK matrix **P** is given by :

$$P = XY^T (YX^T XY^T)^{\frac{-1}{2}}$$

But in our case we don't have points, we have planes; as yet there is no one-to-one correspondence between the two collections; and there may (because of manufacturing errors) even be different numbers of planes in the model and the measured object.

The rotation is performed upon the points in the two *Extended Gaussian Spheres* (EGSs) [4] of the collections of plane equations. The Gaussian sphere of a collection of planes is the points formed by their normals on the unit ball. The EGS is the pattern of points in space which is obtained by scaling each of these normals by a factor obtained from the planes, for example each plane normal might be scaled by the area of a facet lying in it. In our work we have used the perpendicular distance from the planes to the origin to scale the normals. The EGS thus effectively become the set of points (one on each plane) that are closest to the origin. The authors also intend to try to use facet area for this as well. A nice feature of this method is that *any* measurable characteristic of a face can be employed to scale the EGS without affecting the rest of the process.

One thing that remains invariant under rotation is radius. The points in the two EGSs are matched under radius and then rotated. Once this has been done most points will be correctly matched, but some will have been mis-matched because of near-coincidences in radii (**Figure 6**).

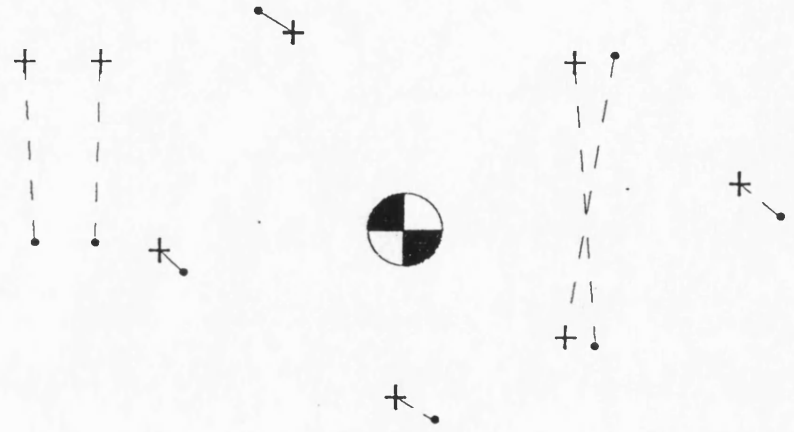


Figure 6. Matched and mis-matched pairs after rotation

Now there are two sorts of links between individual planes in the measured object and the solid model: links that are short in length which represent correct matches and links that are longer representing mis-matches. In the first approach to resolve this, a probability density estimate of the link length was constructed by convoluting their histogram with an appropriate kernel function, the width of which was increased until the density function had just two modes. Silverman [7] uses a Gaussian kernel, but a simple triangle was adequate for this application. The first, sharp mode comes from the short links, the second more diffuse mode comes from mis-matches. **Figure 7** shows the density estimation; the vertical scale is arbitrary.

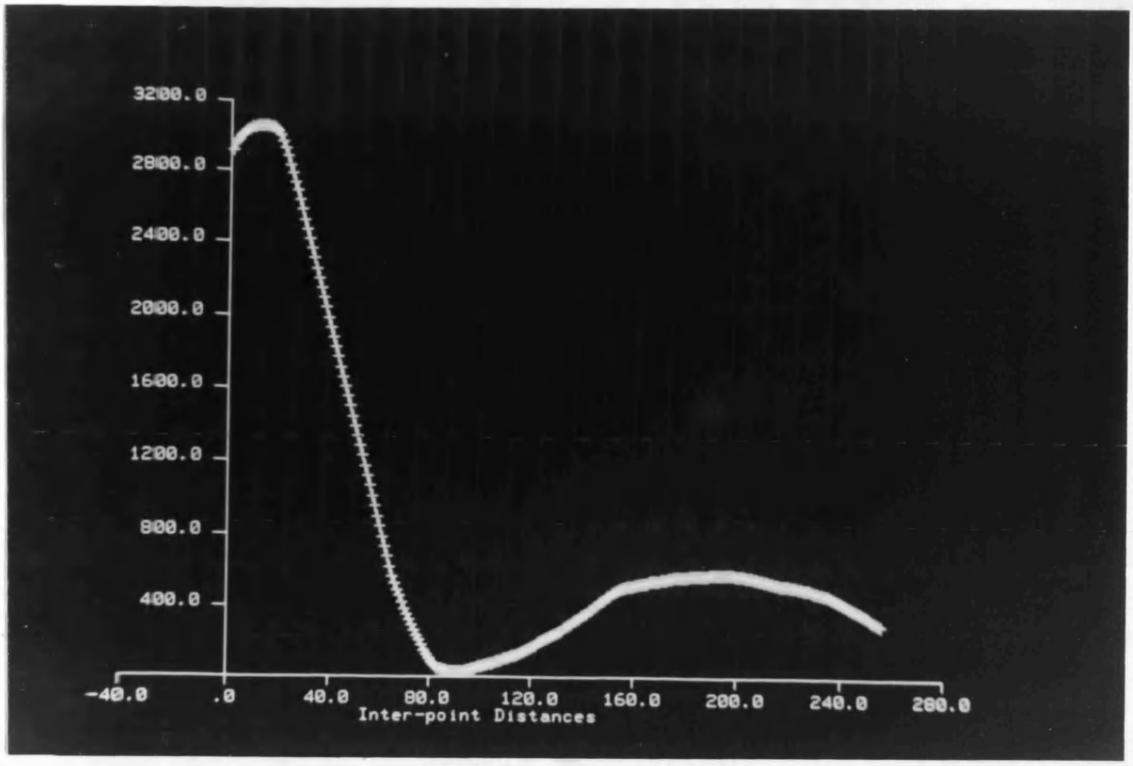


Figure 7. Density estimation

All the links longer than the length corresponding to the minimum between the two modes were broken and re-matched by using the Euclidean distance between them rather than radius as a matching criterion. But later on the authors discovered that forming this sort of histogram to estimate the link length did not improve distinguishing the mis-matches over the simpler technique of breaking all the links and re-matching them using actual inter-point distances. After this, the rotation is done again to make a fine adjustment in the relative orientation between the component and the model. The two are now matched in such a way that the residual sum of squares between the two EGSs is minimal.

Faces of the component may now be compared with the corresponding faces of the model and any out-of-tolerance differences reported. Also, all faces within tolerance can be discarded, and the whole process carried out again on bits that don't match. This allows parts of the component which are of the right shape, but which are in the wrong place, to be identified and *their* position and orientation to be computed.

3. Data Gathering

As was mentioned above the data to be matched consist of points in space. These data may either be gathered by using a laser non-contact measuring machine or by using an algorithm which simulates the process of this measuring machine. The modeller DORA uses a ray-tracer to generate its graphics. The way in which the laser measuring machine works is exactly analogous to this: rays of lights are directed at the object to be measured and the points where they strike the surface are calculated. In order to conduct controlled experiments on the matching system data was created using DORA's ray tracer. This traced a ray of light back from the viewer into a model of the object being measured and found the intersection point of the ray and a surface in the model. It recorded the coordinates of these surface points into a measurements file. To simulate measurement inaccuracies these points were perturbed slightly using a random number generator.

Conclusion

The algorithms explained in this paper have been coded and work efficiently and accurately. They are implemented to deal with the components build up of flat faces and solid models only. The authors are working on the problem of extending the algorithms to cope with curved components.

References

[1] Boissonnat, J.D. *Shape reconstruction from planar cross-sections* Proceedings of the IEEE conf. on Computer Vision and Pattern Recognition, San Fransisco, June 1985, pp 393-397

[2] Bowyer, A. *Computing Dirichlet tessellations* Computer Journal V24, No 2 (1981) pp 162-166

[3] Bowyer, A., Graham, D., and Henry, G. *The measurement of 3-D features using laser triangulation* Proc. 7th International Conference on Automated Inspection and Product Control. Birmingham (1985). IFS Publications.

[4] Little, J.J. *Extended Gaussian Images, Mixed Volumes, and Shape Reconstruction* Proc. First ACM Symposium on Computational Geometry, Baltimore, June 1985

[5] Sibson, R. *Studies in the Robustness of Multidimensional Scaling: Procrustes Statistics* J. R. Statist. Soc. Series B, V40, No 2 (1978) pp 234-238

[6] Sibson, R. *SLINK: An optimally efficient algorithm for the single-link cluster method* Comp. J., V16 No 1 (1972) pp 30-34

[7] Silverman, B.W. (1981).*Using kernel density estimates to investigate multimodality.* Journal of the Royal Statistical Society (Ser. B), Volume 43, pp 97-99.

[8] Woodwark, J.R., and Bowyer, A. *Better and Faster Pictures from Solid Models* IEE Computer Aided Engineering Journal, V3, No 2 (1986)