1990

Improving the analytical recovery of radiostrontium from environmental samples

Luz Selenia Acosta
University of Central Florida

SURFACE MAPPING AND AUTOMATIC TOOL PATH GENERATION

BY

LABICHE FERREIRA
B.E., University of Bombay, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Industrial Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1991

ABSTRACT

With the use of machine vision systems in the manufacturing cycle of a product, the lead time for producing the final product has been substantially reduced. Efforts in the past have aimed at automating the tasks right from the drawing board stage to final production of the product. Such technologies include CAD, CAM and CAPP. However the task of tool path (NC code) generation has not yet been fully automated. In the current techniques, the user plays a crucial role in the NC code generation process.

There is an increasing trend for using machine vision systems in the fabrication of a part. Most machine vision(surface mapping) techniques generate a huge amount of data. Ideally, a CAM system should be capable of accepting data in any format for tool path generation with minimum intervention from the user. This thesis proposes a four step, computer based method for tool path(NC Code) generation from X,Y,Z data, aimed at minimizing if not eliminating the users role. The different techniques of surface mapping and curve fitting are also presented. These four steps extract relevant information needed for the generation of NC code, thereby automating the process traditionally handled by a user a interface.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the advent of machine vision systems the lead time for producing the final product has been substantially reduced. A typical manufacturing cycle consists of three phases : design, production process planning and fabrication. In each of these phases, computers have widely been used. Efforts in the past have been directed at automating the tasks right from the drawing board stage to the final production of the product. Such technologies include computer-aided design (CAD), computer-aided process planning (CAPP), and computer-aided manufacturing (CAM). The current trend in the manufacturing arena is the use of machine vision. Machine vision is now becoming an integral part of advanced manufacturing systems.

Machine vision and computer vision are synonymous. The purpose of machine vision is to make computer capable of understanding environments from visual information. Machine vision involves a variety of intelligent information processing: both pattern processing for extraction of meaningful symbols from visual information and symbol processing for determining what the symbols represent. The term 3D computer vision is used if visual information has to be interpreted as three dimensional scenes.

There is an increasing demand for 3D computer vision. In factories, for example, automated assembly and inspection can be realized with fewer

constraints than conventional one which employ two-dimensional computer vision. 3D vision is being developed for mobile robots capable of passing through corridors and stairs, avoiding obstacles. 3D vision systems can also be used to capture geometric information of an object which could then be used to manufacture the object. There are number of techniques that could be used to capture geometric information depending on the application.

This data could also be used in the creation of geometric CAD databases. For objects with geometric regularity, it is practical to generate them analytically, using one of the several geometric modelling schemes. However, there are many areas in which there is a need to create a database by extracting an object definition from a complex real life object, i.e., one that already exist and which does not have regular geometric properties. Examples of such objects are found in the field of medicine (prosthesis, plastic surgery, replication of ancient artifacts in museums, etc.).In short the acquisition of 3D data of objects opens up the following possibilities:

- producing a copy of the object.
- producing a modified version of the object.
- producing a negative of the object, for example, to produce a mold or die for the object (reverse engineering).
- for inspection of parts.

Application at Kennedy Space Center(KSC)

This thesis topic resulted from a research component in a project between NASA- KSC and the Department of Industrial Engineering and Management Systems at the University of Central Florida(UCF). The project entitled "Productivity Techniques" deals with applying high technology methods and techniques in support of operations at KSC. The focus of this component is to digitize (surface map) the cavity formed by a missing thermal tile on the space shuttle, and link this with a computer controlled milling machine to cut a replacement tile, thereby automating the process of replacement of the tile. The three dimensional information of the tile cavities on the orbiters surface could be obtained by digitizing the cavity with one of the techniques discussed in chapter 2 of this thesis report.

The current method of obtaining the 3D data of the cavity involves:

- "splashing" the cavity with a substance that takes the form of the cavity thereby capturing the geometric representation of the cavity. This is then used to prepare a pattern which is used to manufacture the tile on the gunstock milling machine.

This process of tile replacement has the following drawbacks:

- the process is very laborious.
- the replica or pattern created after splashing the cavity is not an accurate representation of the cavity which may result in a tile that does not meet design specifications.

- The movement of the cutting tool (of the gunstock milling machine) is controlled by the stylus, that follows the patterns surface. This movement depends on the skill of the operator.

A lot of time spent in rework of the newly manufactured tile. This adds to the cost of manufacture per tile and increases the orbiter's stay in the Orbiter Processing Facility (OPF).

By selecting the proper digitizing process (thereby eliminating the splashing process), 3D data of the cavity can be obtained. This data can then be used to define the surfaces of the cavity. After defining the cavity, the information could be loaded to CAM system and finally to a numerically controlled machine to manufacture the new tile.

Thesis Outline

The issue of manufacturing a part from its drawings has not been fully automated. The designer or programmer after generating the drawings of an object needs to write the necessary code (Numerical Control Code) or interact with a CAM package in order to generate the necessary numerical code. Ideally, a CAM package system should be capable of accepting geometric data in any form in order to generate NC code. In cases where the part data is in the form of a CAD drawing, this process would be easy. But in cases where three dimensional data of the object exist (after surface mapping an object by the use of machine vision techniques) , generation of NC code could be laborious. This

is so, since the programmer wouldn't be in a position to tell what kind of surfaces these represent. In such cases the need for a system that deciphers this enormous three dimensional data without making the process cumbersome to the programmer is warranted.

This thesis addresses the issue of tool path generation after surface mapping of an object has been conducted. It is basically an "Reverse Engineering" application. The thesis will be dealing problem of defining surfaces from X-Y-Z data for tool path generation with minimum human intervention. In most CAM packages the user plays a crucial role in the manufacturing process: defining the boundaries of the object, defining surfaces to be machined, the machining sequence, etc.. This thesis attempts to relieve the programmer of these cumbersome tasks. The thesis proposes a four step procedure for the generation of tool path: 1) Data Reduction stage, 2) Geometric Data(shape definition) Extraction Stage, 3) Surface Recognition Stage, and 4) NC Code Generation Stage.

The related issues of surface mapping techniques will be covered in chapter 2. The different methods currently used for obtaining topographical and geometrical information of the objects surface will be dealt in detail. Chapter 3 will cover the principle, operational mechanism and components of laser scanners, which is the target surface mapping technique to be used in the KSC project. The four step procedure proposed for tool path generation and logic for each of the four stages will be explained in chapter 4. Results of these stages

will be presented in chapter 5. The computer programs for the stages can be found in the Appendices.

CHAPTER 2

SURFACE MAPPING TECHNIQUES

Many ways exist for capturing the geometric information of a surface. An obvious way of finding the coordinates of some random points on a surface is to use a Coordinate-Measuring Machine (CMM). The object could be placed on the table and by the use of a probe the coordinates of the surface can be obtained. The limitations of a CMM are :

1) The size of the object that can be measured is restricted by the size of the CMM table.

2) The accuracy of CMM's limits its use where dimensions have to be measured within close tolerances.

3) The measurement of complex profiles may be impossible. This is due to the fact that the probe may not be in a position to reach sharp corners, grooves, peaks etc. Thus the CMM cannot measure a surface to its minutest detail.

Most modern measuring systems are essentially optical. These techniques include:

1) capturing a picture of the object. The image of the objects surface is then analyzed through the use of image processing algorithms to extract pertinent data. The output of such a process would be

geometric information of the objects surface.

2) projecting fringes on the objects surface. A picture of the distorted fringes on the objects surface is captured and stored as a frame on a computer. Analysis of the fringe pattern results in the topological information of the surface.

3) scanning a laser beam across the objects surface. The output is an X-Y-Z representation of the object scanned.

Extraction of tridemsional information from images have proven to be useful in medicine, geology, cartography, and military activities. The sciences involved are photogrammetry and artificial intelligence. The former involves the extraction of 3-D coordinates from images, the latter involves the automatic interpretation of 3-D images.

This chapter gives an overview of methods for collecting surface-point coordinate data with an aim to replicate these surfaces by machining.

Stereo-photogrammetry

The determination of surface geometries by measurements of central perspective photographic projection is called photogrametry. This is one of the earliest methods of recording the topography of both large areas of terrain photograph from an aircraft or buildings and small objects viewed at close range. In this technique two photographs of the terrain or object are taken by cameras at nearby positions. The depth Z of points in the scene at position X,Y

in the plane of the photographs can be deduced from the photographs by measuring the parallax of identifiable points in them. Three methods are described below:

1) Two photographs may be taken from different viewpoints giving projected points $\mathbf{r}_1$, $\mathbf{r}_2$ in a plane which are sufficient to fully determine the coordinates of real spatial points such as $\mathbf{r}$. By forcing the human eyes to receive simultaneously the photographic image appropriate to each eye in real observation, a mental perception of depth(z) dimension of a surface is forced upon the brain. The procedure of observing and recording two projections and their subsequent quantitative analysis is called stereo-photogrammetry.

2) By a device in which shadows of a grating of alternate dark(opaque) and light(transparent) "lines" or bars are optically projected onto the object and subsequently mapped directly and seen as shadow fringes(Moire Technique- will be discussed later) in one photograph. The need for double photography and stereoscopic perception is eliminated.

3) For certain barrel-like objects, the ancient Grecian silhouetting method of Dibutades can be effective, particularly when coupled with modern video techniques. This technique will be described in detail later on in this section.

Photographs are "read" by a skilled, specially trained operators of special analyzing machines which are now obsolete. The output may be Z values at orthogonal rows and columns in X and Y (plan view coordinates) recorded on computer compatible storage media.

This technique of stereo-photogrammetry is highly specialized and requires the special machines to decipher information contained in these photographs. However, the advent of charge-coupled diode (CCD) and other form of light sensing arrays have replaced conventional photographs as recording media. These new techniques of detecting light intensities enable data to be fed directly to computer storage media.

Shadow Moire Contourography

This is a method that deduces depth of surface from one rather than two photographs. Essentially it senses parallax, computes and plots contours of equal elevation as fringes superimposed on the single photograph. When an optical grating of alternatively opaque and transparent bars is illuminated by either a divergent or collimated beam of light, shadows of the opaque bars will be cast onto a surface beyond the grating. If such shadows and the grating form which they are projected are viewed simultaneously from a point not coinciding with the source of illumination, so-called shadow moire fringes are seen. There are many approaches to shadow moire contourography or moire topography. The major applications of the moire method is in the measure deformation in

stress analysis.

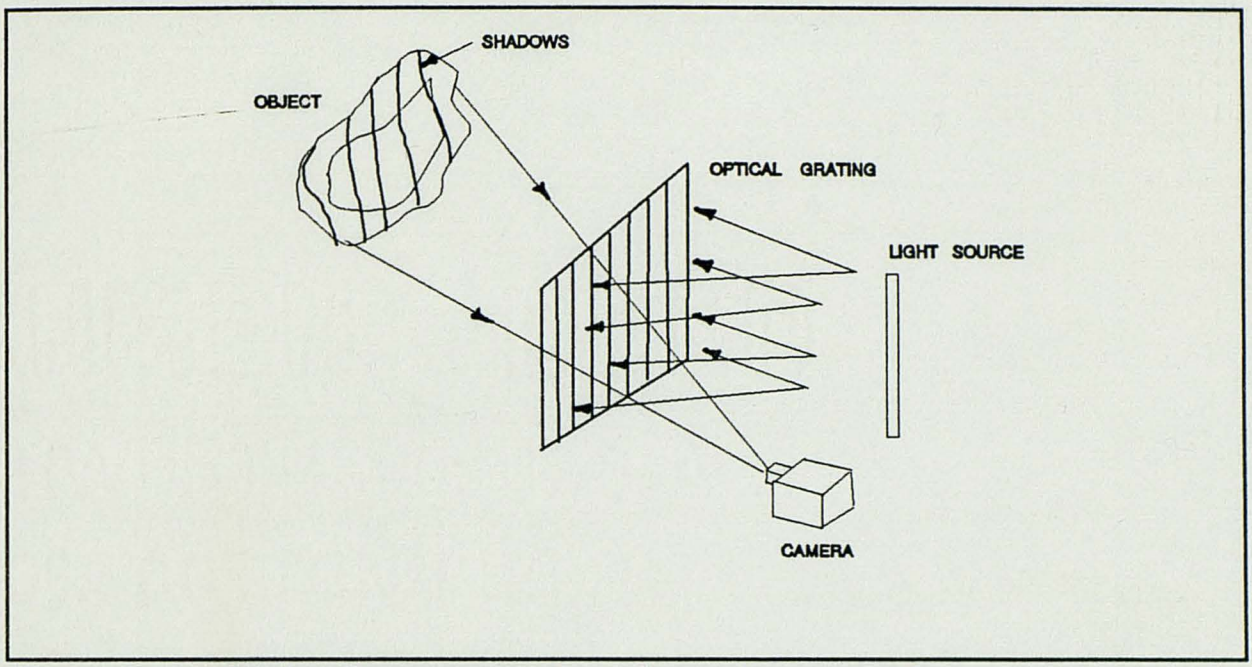

Figure 2.1 Moire Contourography using only one grating.

The Moire method has been successfully used to visualize surface contours of large objects like airplane models or mannequins. In this set up, a light source illuminates an object through a large equispaced plane grating in front of the object and observes its shadow on the object through the same grating (refer figure 2.1). Later, another practical technique was developed that did not use a large grating, instead, a shadow grating produced by projecting a grating onto an object is observed through another grating (refer figure 2.2). The 3D shape of a known object is found by the contour lines in the moire image.

The Takasaki-Terada technique is used for tracing the changes in body shape following surgical operation and for anthropological research. Figure 2.3 shows the apparatus used in this technique. A screen of specially blackened

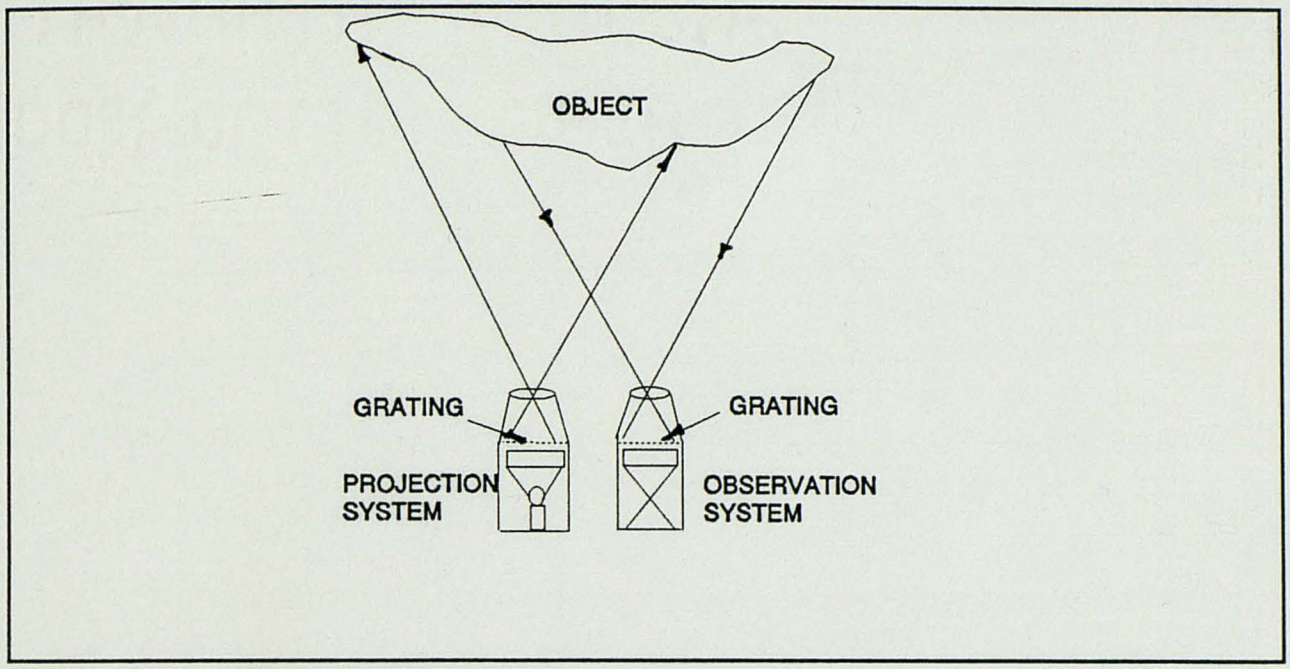

Figure 2.2 Moire Contourography using two gratings.

strings is tensioned within a strong frame. These strings are positioned by a second frame providing a "bridge" for the strings. The strings form a grating of uniformly spaced opaque bars. The frame of strings is mounted vertically in a slightly inclined track in which it travels under gravity in a horizontal direction normal to the vertical direction of the strings during photography. Alternatively the frame may be driven by electric motors from side to side between limits as indicated in figure 2.3. The subject or object is placed behind the screen and illuminated by a high intensity point source lamp sited at an angle to the normal to the plane of the screen. The lamp casts shadows of the strings upon the human subject or inanimate object. Two lamps are used to give shadow free projection on the surfaces. A centrally located 35mm format camera photographs the screen and its shadows on the subject or object while the screen is moving.

To summarize this technique - the X-Y-Z representation of an object is obtained by projecting equally spaced fringes onto the object. A 2D picture is captured by a charge couple device (CCD) and consequently digitized and stored as a frame in a computer. The phase shift of the light fringes on this image leads to the calculation of depth values for every point.

Figure 2.4 shows a schematic representation of the Moire technique.

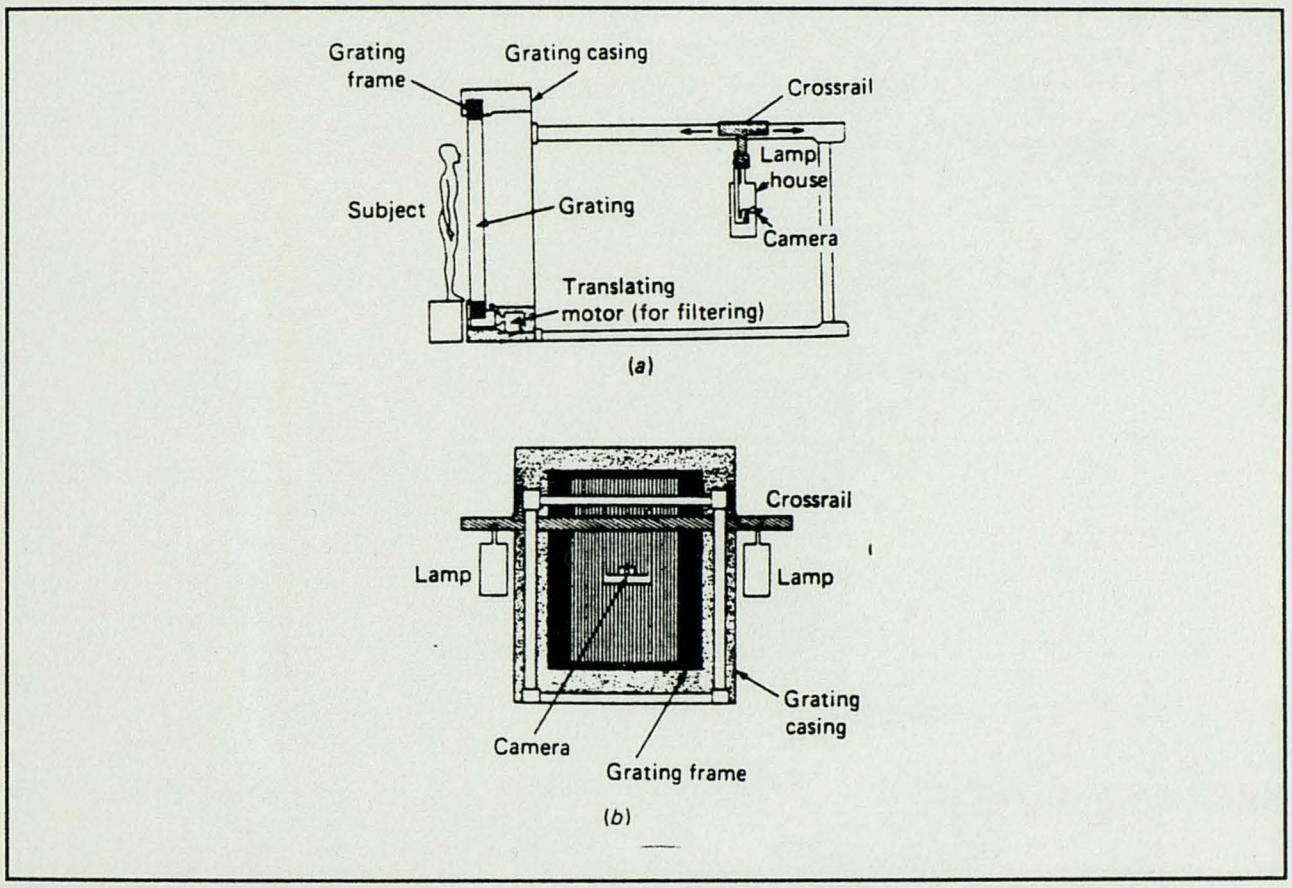

Figure 2.3 Schematic diagram for the Takasaki-Terada technique. (a) Side view of apparatus (b) Front view of apparatus (Duncan and Mair 1983)

Silhouettes

The technique of silhouettes was used by Dibutades to carve the human form from shadows of the model projected from a range of angular directions. Provided the object is free from concave geometry, the rays form a shadow as

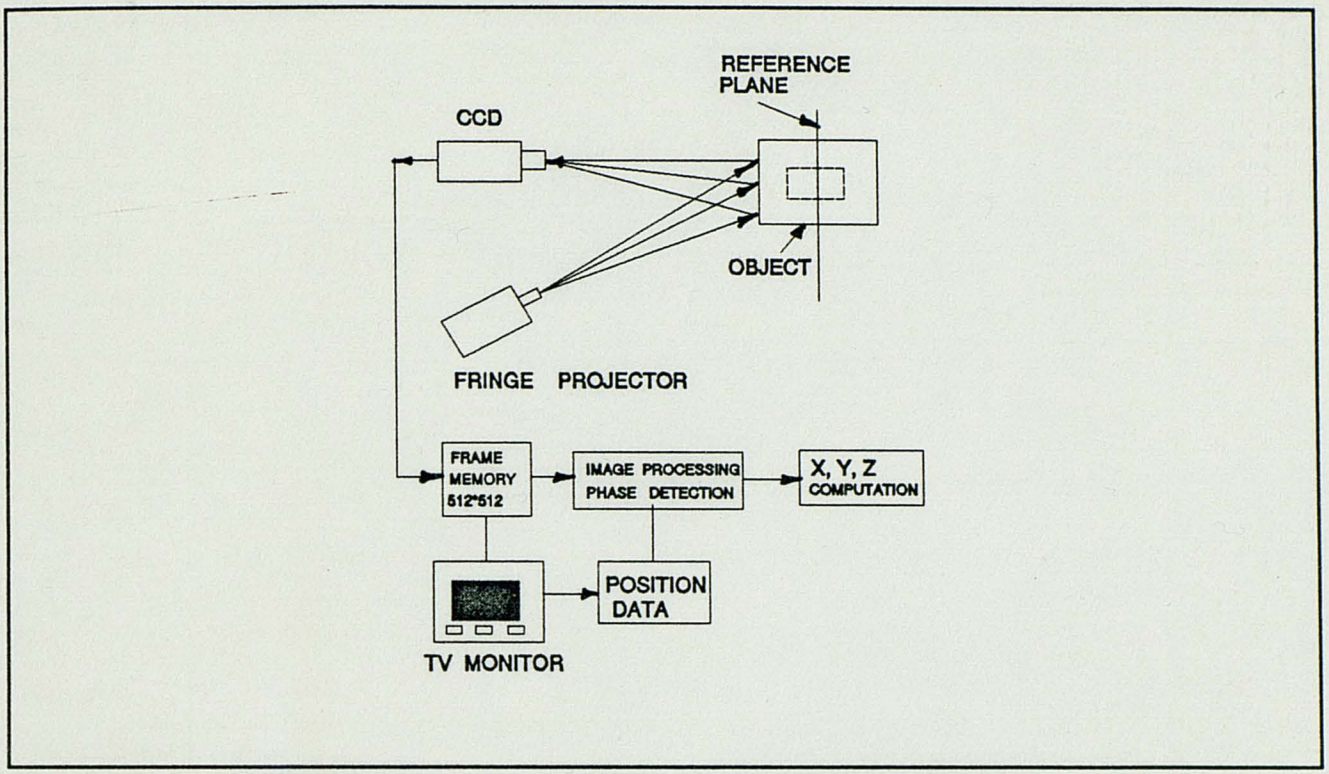

Figure 2.4 Schematic representation of the Moire technique.

an orthographic projection form a cylindrical envelope of the surface for that particular direction. Figure 2.5 shows a typical application for Silhouettes using video recording systems. The shadow for a given angle of projection of light may be digitized (by hand or automatically) and converted to virtual computer stored image. The object can be rotated for a series of angular positions, each of which will yield a silhouette for that position. This may be followed or traced by a cylindrical cutting tool directed by computer-prepared instructions, to produce the desired tubular shape at a specified angle of projection as illustrated in figure 2.6.

A system has been devised for obtaining silhouettes of limbs, limb remnants and the feet of real patients under clinical conditions. This system has been demonstrated and proved using models and simulated arrangements.

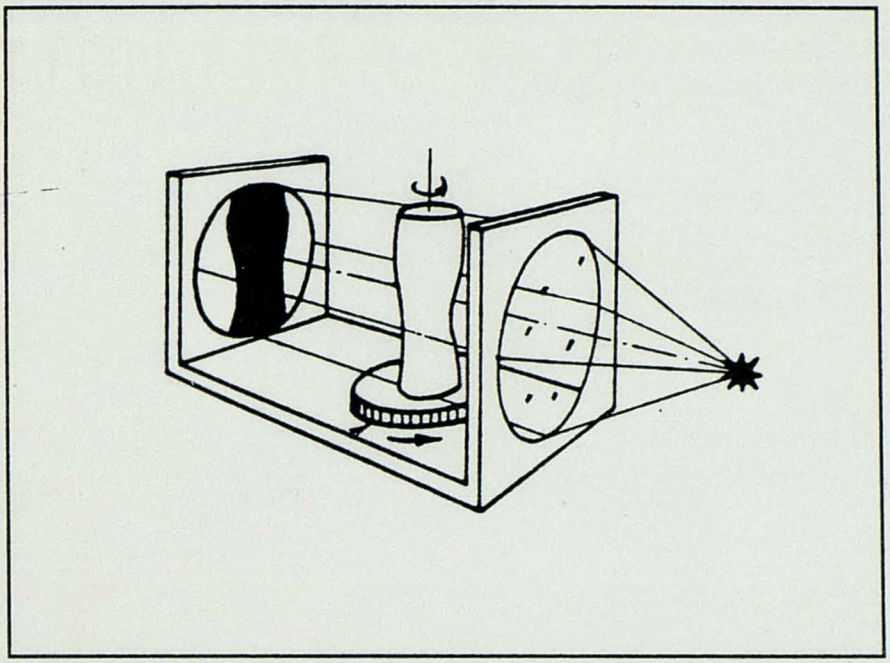

Figure 2.5 Silhouetting by casting a shadow of a surface-bound object in collimated light. (Duncan and Law 1989)

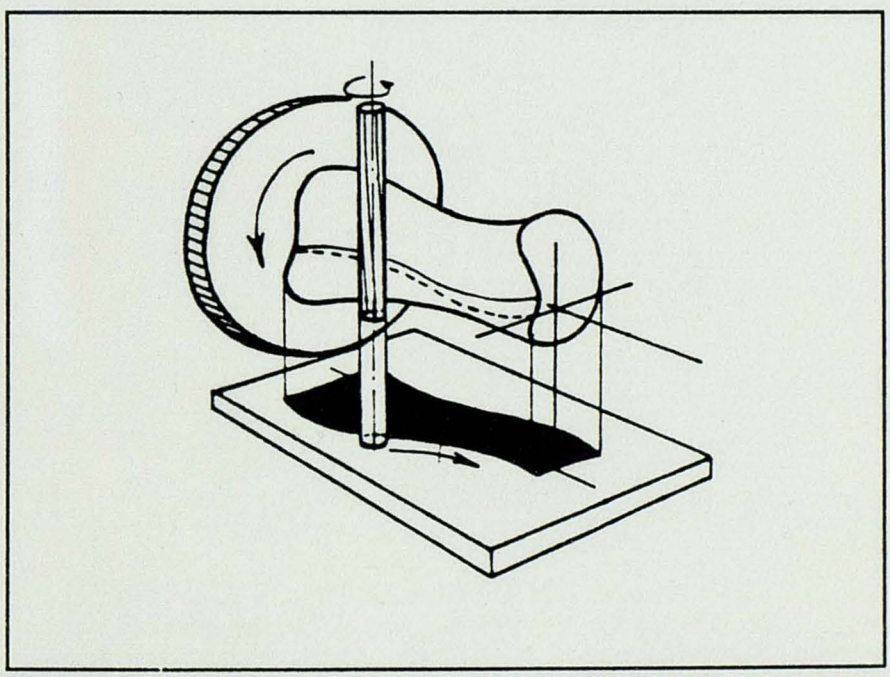

Figure 2.6 The tracing of a silhouette for one angular position by a cylindrical milling cutter. (Duncan and Law 1989)

Beam-Scanning Methods

The latest modern methods of surface mapping employ lasers. The methods that follow use lasers, X-Rays, etc. to capture geometric information of the surface.

Laser Beam Scanning

In this technique the geometric representation of the object is obtained by a scanning laser beam across a surface to be measured. The basic configuration of a 3D laser scanning device consists of a laser light source which produces a narrow light beam which is scanned across the object to be measured through the use of a two dimensional mirror.

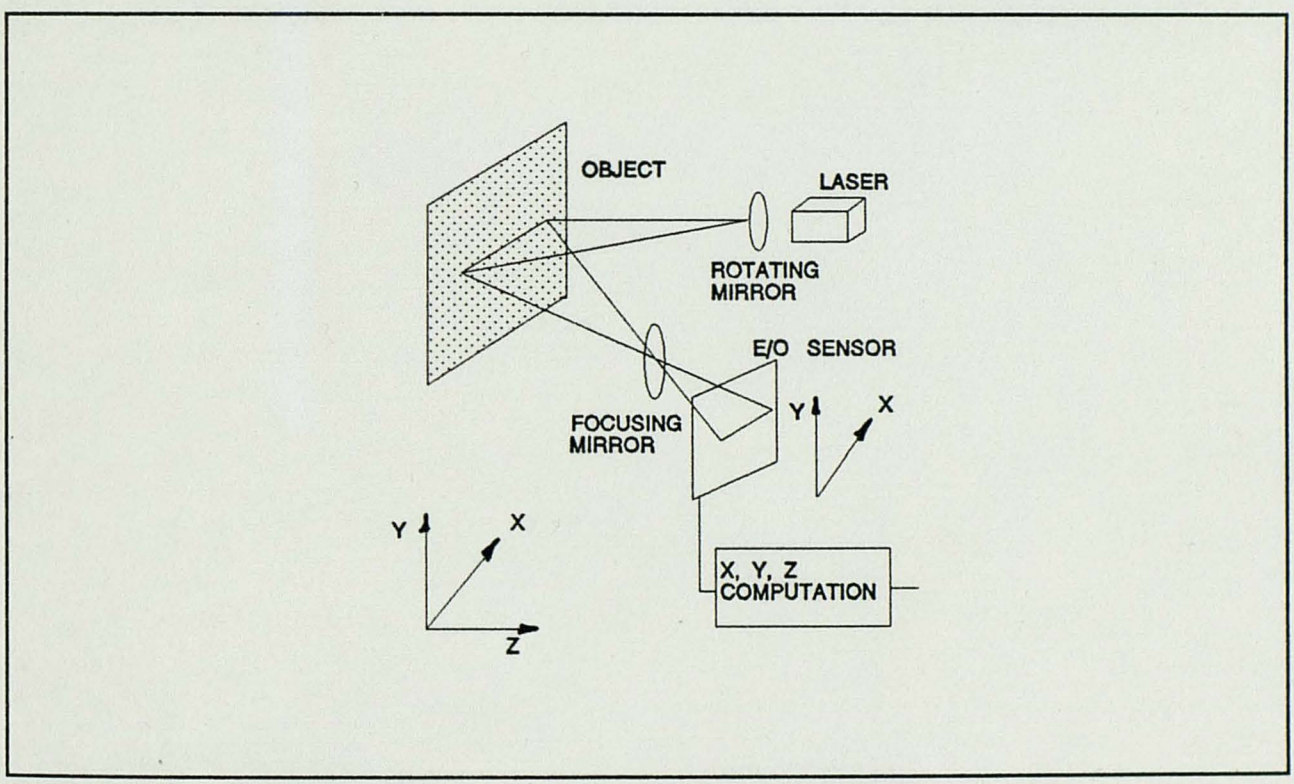

Figure 2.7 Basic configuration of a 3D laser scanner.

A lens collects the reflected beam and displays it on an Electro-Optical (E/O) position sensor. The linear position of the reflected light along with the different

angles of deflection of the scanner are used to calculate the 3-D coordinates of a point on the objects surface. Figure 2.7 depicts the basic configuration of a 3D laser scanner device. Chapter 3 describes in detail the components and operational mechanism of laser scanners.

Computed tomography and CAT scanning

The methods introduced above may be described as "superficial", i.e. measured from observing points above the surface or from outside of any solid enclosed by a surface. Such methods may encounter difficulties such as the shadow effect due to the configuration of the object.

A new principle of surface definition which overcomes some of the difficulties is the Computer Tomography method which was invented by Sir Godfrey Hounsfield, a Nobel Prize winner. This method employs the geometrical idea of sectioning a closed surface by imaginary parallel, closely spaced "cuts" through the surface with the aid of X-ray scanning and large scale matrix inversion in a mainframe or mini-computer. Most large hospitals now have such systems for reconstructing and thus inspecting internal organs and bones of the human bodies of live patients without operation as formerly required. The numerical data used for this pictorial reconstruction can also be used for machining replicas of those organs and bones.

Figure 2.8 shows a typical hospital Computed Axial Tomography (CAT) apparatus. The patient lies on a couch with his or her head inside an annular

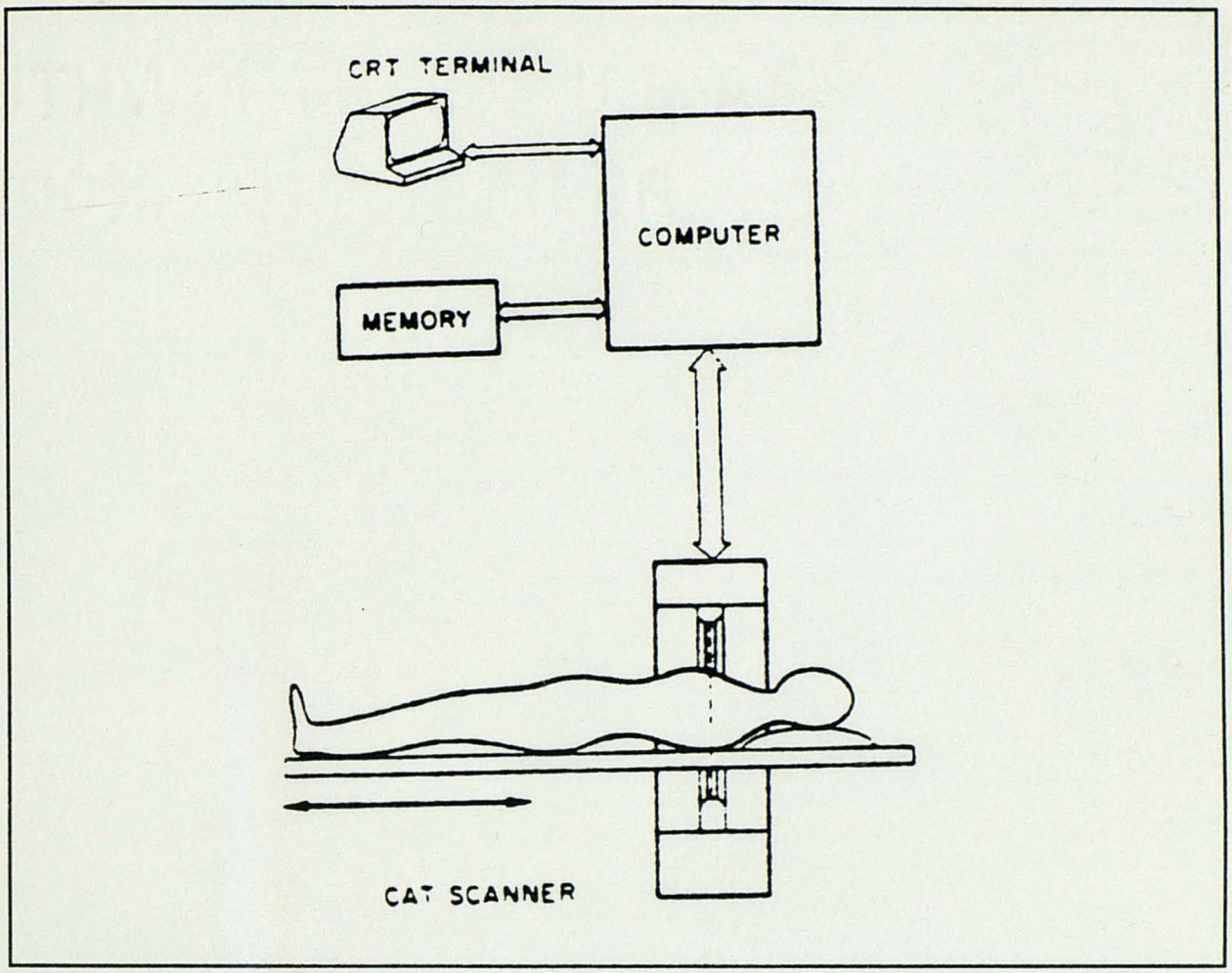

Figure 2.8 A CAT scanner in hospital use. (Duncan and Law 1989)

aperture as illustrated in the diagram. A X-ray gun propagates pencils of X-rays through the skull and through the axis of the aperture to a diametrically opposite receptor which detects the strength of each pencil after some of it has been absorbed by all the various body tissues which it has traversed. The beam is arranged to traverse systematically in many incremental angular directions in a plane normal to the axis. When a full rotation of beam direction has been completed, the rotating beam is moved axially to the next, neighboring plane, usually 1mm or so away. Hounsfield discovered that different tissues absorb different amounts of X-ray energy. The measure of that capacity for absorption

is now the Hounsfield Number.

When Hounsfield Numbers are known the boundaries of bones, for instance, being dense and strong absorbers compared with surrounding tissue, can be deduced in a coordinate frame of reference. This detection of Hounsfield Number differences is so sensitive that boundaries between two soft tissues of only slightly different Hounsfield Number can be detected; and so a kidney, for instance can be "extracted" and imaged for examination. The coordinates involved in image reconstruction can also be used for replication by machining of bones and organs for bone replacement prosthesis.

Ultrasonic time of flight technique.

When a short ultrasonic pulse is transmitted towards the object, some of its energy is reflected back to the transmitter. If the time interval between the transmitted and received pulses is measured, the distance (d) between the source and object is obtained from

$$2d = v_s\, t_f$$

where v_s is the speed of sound under given pressure conditions and t_f is the time of flight of the pulse.

In order to avoid signal attenuation in a practical system, it is not a single pulse, but rather a set of pulses at different frequencies that is transmitted. Commercial systems also provide a correction factor for speed-of-sound fluctuation under varying pressure conditions. An interesting property of

ultrasonic rangefinders is that the baseline separation between emission and reception is null since they are performed by the same unit. This eliminates the shadow effects encountered in all the systems with nonzero baseline separation, for example the laser beam scanning technique.

However, it is difficult to obtain a well focused ultrasonic beam pattern. Another drawback of ultrasonic ranging is that for some incidence angle of the pulse on the objects surface, very little energy is reflected and the return pulse is lost in detector noise. This phenomenon is similar to specular reflection of light on a mirror-like surface.

From the above discussion it is clear that the technology of using machine vision systems for manufacturing does exist. However the choice of technique depends on the application and the environment in which it has to be used. These techniques of surface mapping were studied and evaluated for the cavity digitization study for the NASA-UCF Project. The Fringe Moire and the Laser scanning techniques were the two techniques studied in depth. The Moire technique had limitations such as: bulky equipment, lot of time needed for setup and processing the image and as well as the accuracy with which the object could be measured. The laser beam scanning technique has the characteristics that make it the most promising technique for an industrial environment despite some limitations in its use. This technique will be discussed in detail in Chapter 3 that follows.

CHAPTER 3

LASER BEAM SCANNING

The industrial environment adds constraints and limitations, such as difficult environment, cost, compactness, etc. to the applicability of usual techniques for surface mapping. On the other hand, the proximity of objects allow active methods to be used in order to get 3-D data much more easily than passive techniques, which involves bringing the object to the device or moving the light source by a stationary object.

Active methods where a beam of light, such as a laser beam is superimposed to the naturally lightened scene simplify a lot of signal processing to be done in order to recover distance information. Besides this advantage, the use of a laser beam for surface mapping provides a number of unique advantages. The brightness of the source ensures a good signal-to-noise ratio in most applications. If needed, the ambient light can be filtered out without any significant reduction of the laser light itself. This is because the laser light is emitted over a very narrow bandwidth. The most significant advantage is that most laser sources can be adjusted to emit in a low order Gaussian mode. This property gives maximum light power and minimum divergence. This is the basic advantage of lasers for surface mapping. This means that all the power available at the propagation remains in focus for an

extended length along the range axis. This is not the case for conventional light sources, where for an extended depth of focus along the range of axis, the aperture of the incoherent light projection system must be reduced, which considerably lowers the amount of light available.

The use of the laser scanning technique for surface mapping of a cavity on the orbiters body has been proved to be one of the most promising. This technique has been described briefly in chapter 2, however it is addressed in detail in this chapter. This chapter will cover principle, operational mechanism, types of polygonal mirror scanners, mechanisms of rotating polygonal mirrors, and configuration of polygon/motor assembly.

Active Triangulation

Figure 3.1 shows the basic elements of a system using active triangulation for surface mapping(scanning).It consists of a light source(S), a scanning mechanism(M) to project the light spot onto the object surface and a position sensor(D) with a collecting lens(L) looking off-axis for the light spot. Distance measurement is done by trigonometric algebra applied to the projection direction (scanner angular position) and the detection direction made by the light spot position on the sensor with the principal point of the collecting lens.

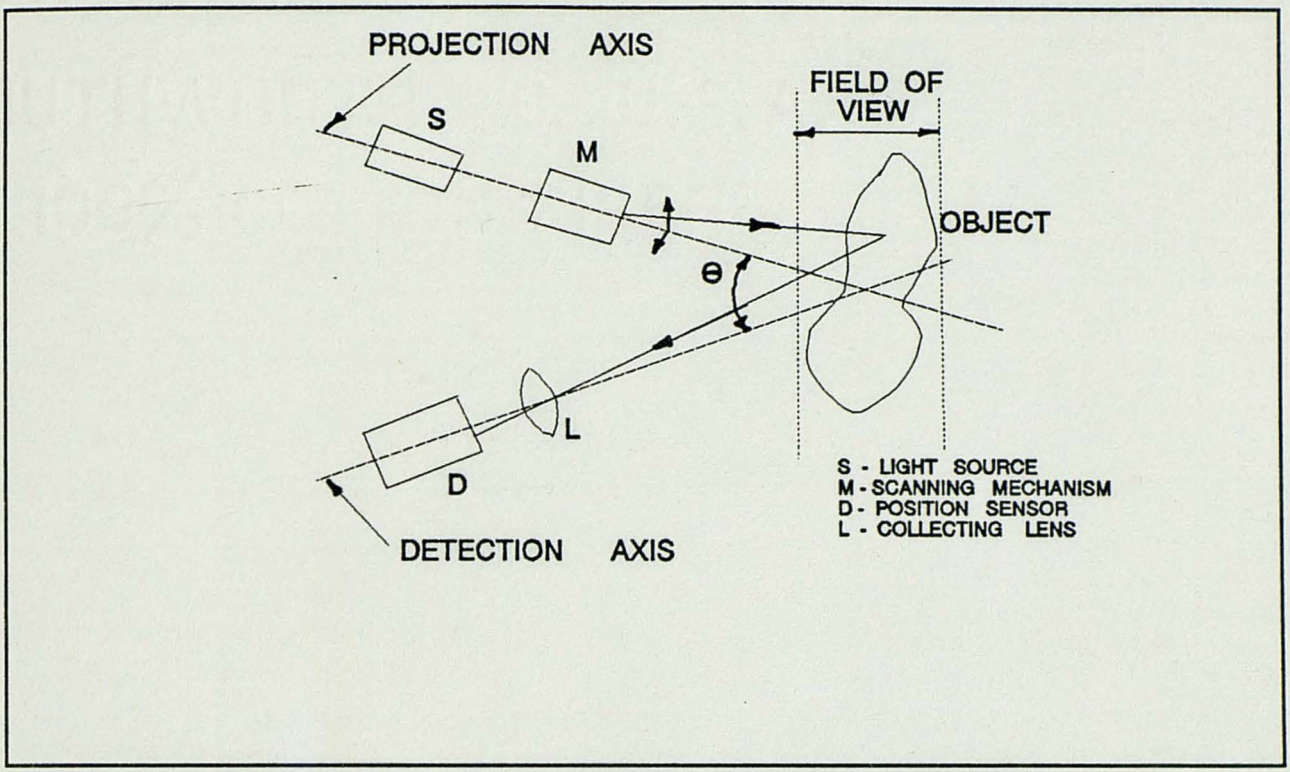

Figure 3.1 Basic laser scanning arrangement by triangulation.

Synchronized Scanning.

One property of the arrangement shown in figure 3.1 is that the measurement resolution can be increased by increasing the off-axis angle(Θ) but this is done at the expense of compactness and field of view. Another drawback is a shadow effect that increases with the angle. This is related to the fact that some part seen by the projection mechanism is not seen by the position sensor. The shadow effect (refer figure 3.2) prevents continuous profile recording and is more serious as the off axis angle increases.

Synchronized scanning eliminates the above drawbacks. The basic idea is to synchronize the projection and detection in a way that the detected light spot on the position sensor keeps it spatial position stable when the projected beam is scanning a flat surface (scanning being parallel to the surface). The

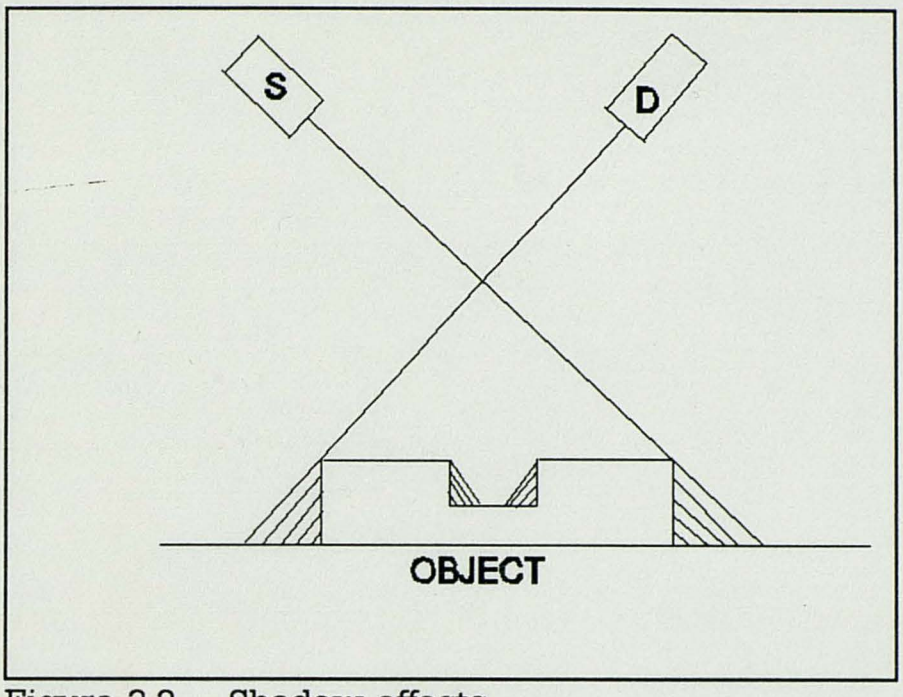

Figure 3.2 Shadow effects.

implementation of a synchronized scanning mechanism can be done electronically or geometrically.

Figure 3.3 shows an arrangement to electrically synchronize two galvanometer driven motors G1, G2. In this case the source signal is a ramp, then the output signal from a position sensor seen on the oscilloscope provides a direct profile reading for which the time axis is proportional to displacement along the X axis. Amplitude of the deflection is proportional to departure of the object surface from the reference plane. One interesting feature of that arrangement is that the position of the reference plane can be set electronically by modifying the phase relationship of the two excitation signals. In order to get a surface profile measurement a third galvanometer driven motor(indicated by G3 on figure 3.3) is used to deflect perpendicularly to the page both the projected and received beams.

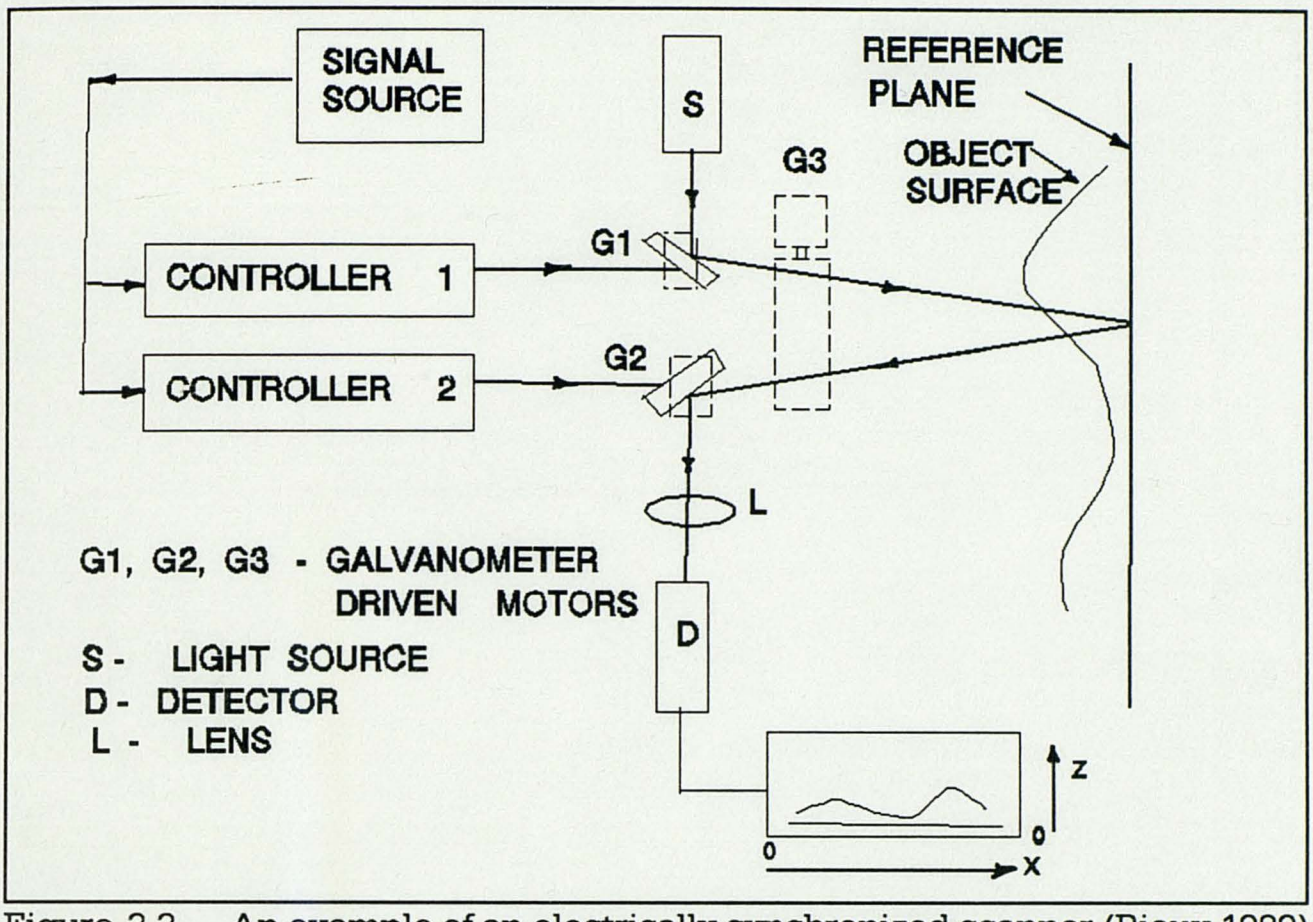

Figure 3.3 An example of an electrically synchronized scanner.(Rioux 1989)

Figure 3.4 shows an arrangement that is geometrically synchronized. Both mirrors F1 and F2 are fixed when the device is an operation. Angular adjustment of those mirrors sets the position of the reference plane in space. Synchronization is realized using a pyramidal rotating mirror(P). By geometrical analysis, synchronization can be achieved by using two opposite facets of the scanner. The arrangement is simpler to implement than electrical synchronization and is also more precise and stable. Another feature is the ability of rotating mirror scanners to rotate at very high speed (approx 10,000 lines/s) compared to galvanometers (approx 100 lines/s).

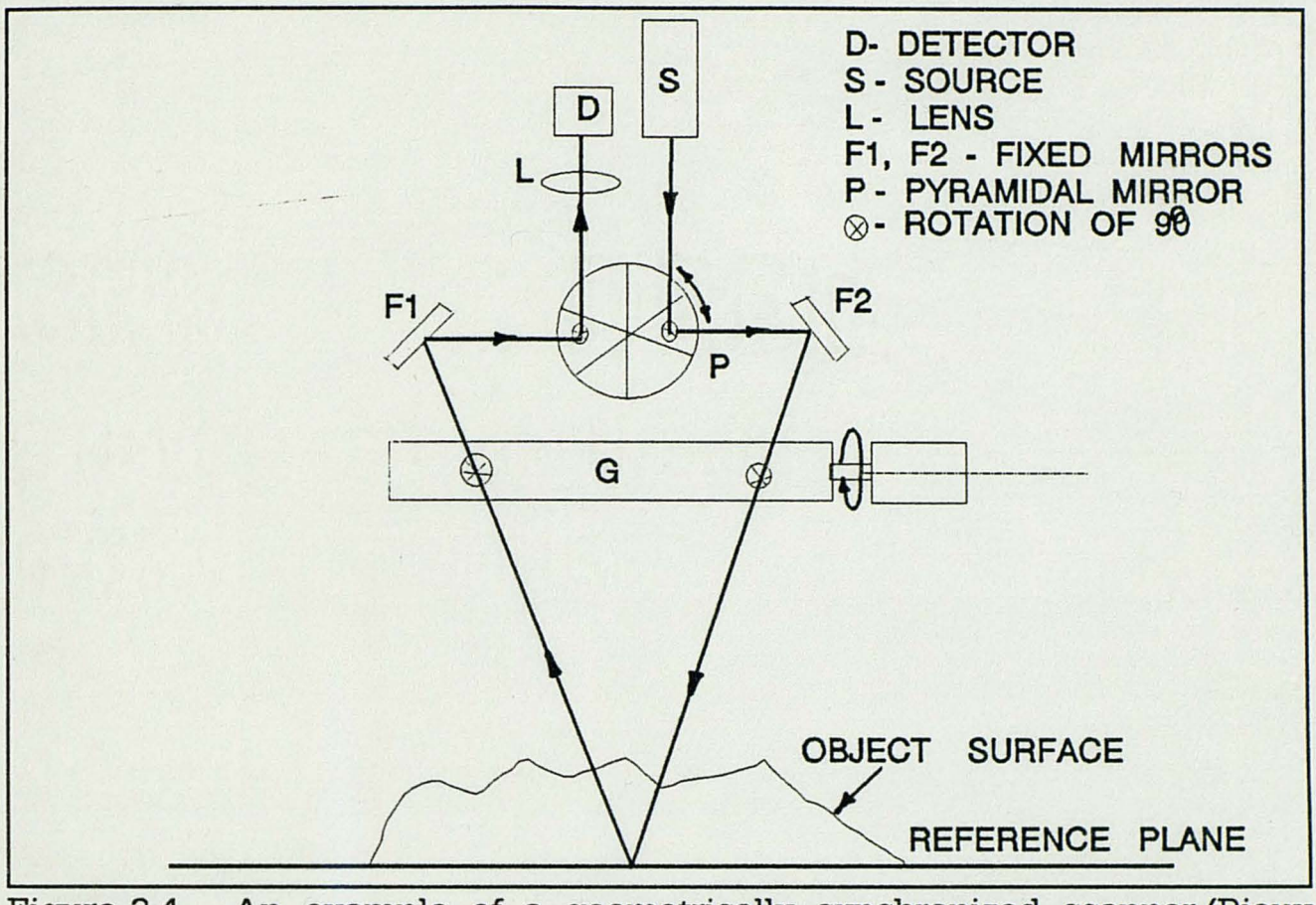

Figure 3.4 An example of a geometrically synchronized scanner.(Rioux 1989)

As seen from figure 3.4, the major components of the scanning arrangement are : polygonal scanner(mirror), motor for rotating this mirror, light source and a position sensor. A three dimensional setup of the arrangement is shown in figure 3.5.

Light Source

Although any light source can be used, a laser beam has many advantages over conventional ones:

- interferometric filter can be used to improve signal to noise ratio at detection.
- brightness is orders of magnitude higher than an incoherent

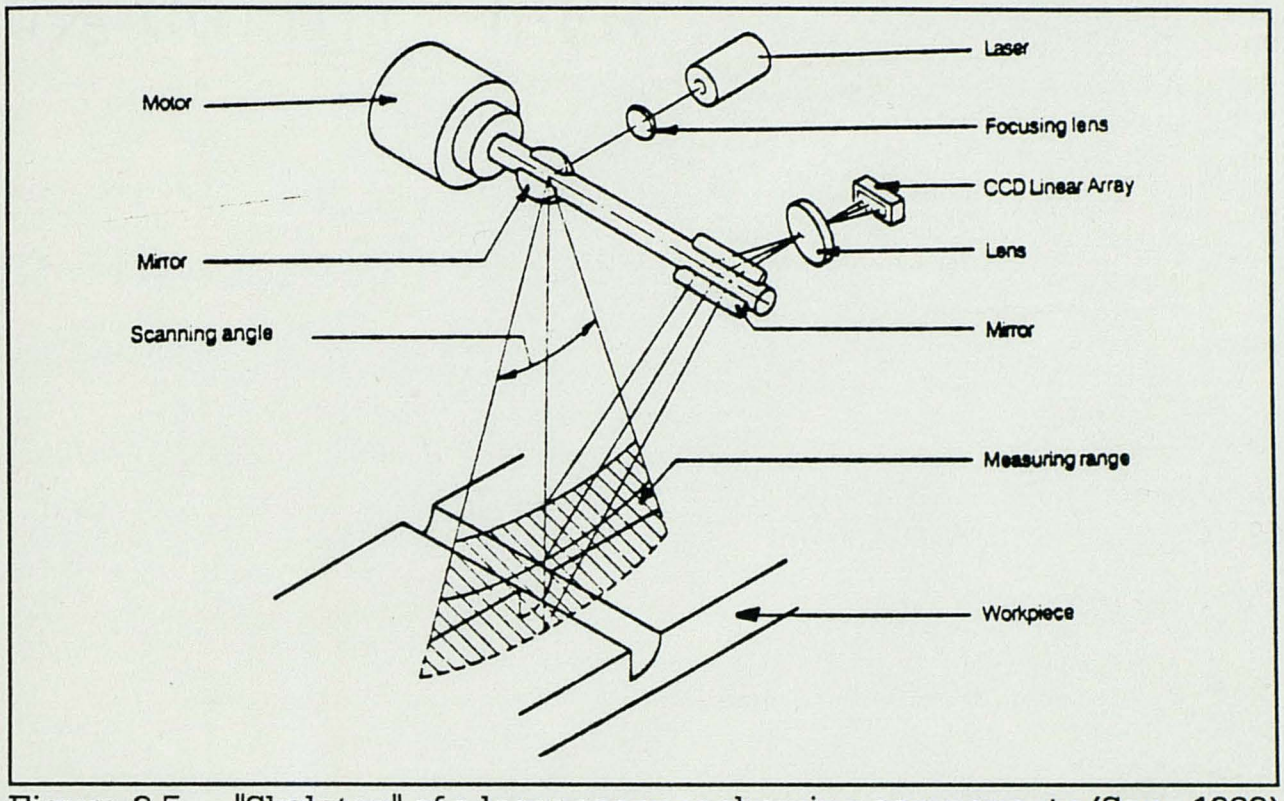

Figure 3.5 "Skeleton" of a laser scanner showing components.(Sanz 1988)

- compact devices (specially diode lasers).
- cheaper.
- very large depth of field due to spatial coherence.

Position Sensors

There are basically two types of position sensors, these are uni-dimensional (or linear devices) and bi-dimensional sensors. A synchronized approach to scanning can allow the use of a linear device for both, line profile and surface profile measurements. There are three main types of such sensors.

The **first** one is an analog device referred to as lateral effect photodiode (LEP). It is made of a photodiode on top of a resistive layer. Interesting features

of the device includes:

- the response time that can be as low as 500ns for small devices.
- low cost sensor.
- insensitivity to large amounts of defocussing.
- the signal output amplitude is proportional to the centroid of the spot.

The **second** type is the linear array of discrete photoreceptors (photodiode arrays, CCD, CID, etc).The major advantages of these devices are:

- high sensitivity.
- very high spatial resolution.
- very good geometrical precision.
- such a device is required for mapping large volume scenes or distant objects. For example, a typical arrangement using a low power laser to map 3-D coordinates of a scene using a LEP sensor will be limited to objects no more distant than about one meter from the camera. In contrast a CCD linear array can map objects as distant as 10 meters.

The disadvantages of these devices are:

- more expensive than lateral effect diodes(LEP).
- a full line scan is needed to get position information.
- much more signal processing than LEP is needed to extract position signals.
- they are limited in dynamic range due to sensitivity to defocussing.

A **third** type is the dual element photodiode. This device is useless with

usual triangulation geometry, but can be used advantageously with synchronized approaches. It consists of two photodetectors located side by side. It has the advantage of having a bandwidth of more than 30 MHz. Signal processing circuitry required to cope with such a speed is the limiting factor for this device. It is also sensitive to geometric distortion of the spot and has a very small displacement range. Nevertheless, its low cost makes it attractive for the development of a 3-D camera.

Scanners

Choice of scanners to be used is application dependent. Parameters to be taken into consideration are: scanning speed, resolution, random access operation, no moving parts requirement and cost. Scanners can be grouped into three types:

- galvanometer driven.
- rotating polygon mirrors.
- acousto-optic devices.

Galvanometer driven and acousto-optic devices can be addresses at random with a typical response time of 1ms and 10 μs. Acousto-optic devices have the advantage of no moving parts. However, they are more expensive and limited in resolution. Another drawback of acousto-optic devices is the difficulty to get large scan angles.

Polygon mirrors are a good choice for very high speed requirements, but they cannot be addressed randomly. Typical speeds are 2 to 10 thousand lines

per second with 2 to 5 thousand resolved spots per line. Scan angles cover typically 20 degrees to 120 degrees. Polygonal scanners will be dealt in the next section.

Polygonal Mirrors

Most laser scanning systems designed to date use a polygonal mirror to scan the laser beam on the objects surface. The polygonal mirror is the heart of the laser scanning system. The primary advantages of polygonal mirrors are speed, the availability of wide scan angles, and velocity stability. These scanners are usually rotated continuously in one direction at a fixed speed to provide repetitive unidirectional scans which are superimposed in the scan field.

A common method of producing rotation is to fasten the polygonal mirror directly to an electric motor shaft. The combined inertia of the polygon and motor rotor assembly contribute to the rotational stability. The relatively high inertia of polygons and drive motors on the other hand render them impractical for application requiring rapid changes in scan velocity or start/stop formats.

Rotational speeds upto 120,000 rpm are practical for alternating-current(AC) motors. For applications where rotational rates exceed the capability of electrical motors, gas turbines provide an alternative. Rotational speeds in the range of 90,000 - 1,000,000 rpm can be obtained using compressed-air.

There are basically four types of polygonal scan mirrors which have been

There are basically four types of polygonal scan mirrors which have been developed and used over the years. These are classified based on their shape: regular polygons, irregular polygons, inverted polygons and pyramidal polygons.

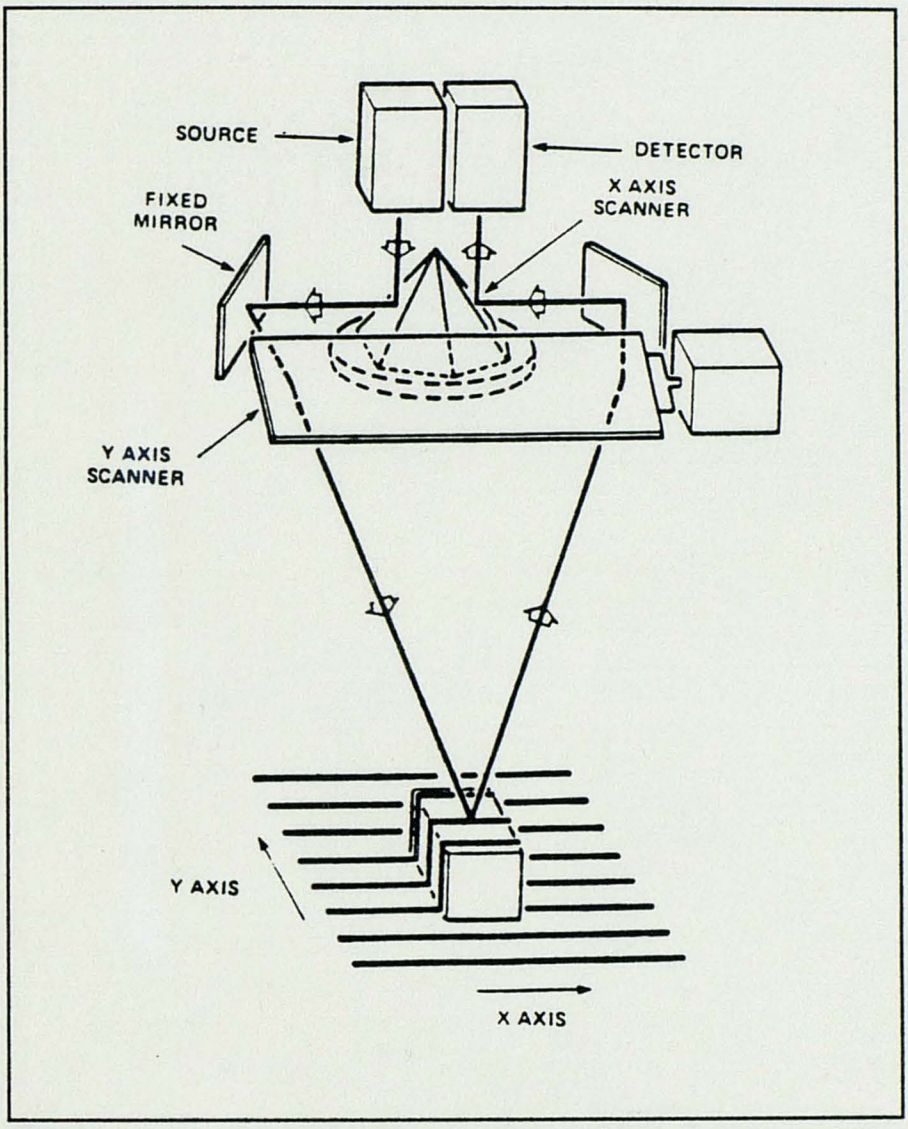

Figure 3.6 Synchronized scanning arrangement using a pyramidal polygonal mirror. (Boulanger 1986)

Mounting of Polygonal Scanners(Mirrors).

A typical mounting method is to fasten the polygon mirror to the shaft of the motor which is used to rotate it. In this case the datum surface of the

quality surfaces so that when the two are firmly held together, distortions do not occur. There are two basic configurations of polygon/motor assembly. These are: (1) Cantilevered and (2) Captured designs. These terms refer to the location of the polygon on the motor shaft relative to the bearings.

The "Cantilevered Design".

The cantilevered approach (refer figure 3.7) is commonly used for low speeds (10,000 rpm or less) and/relaxed tracking tolerances (1 arc minute or greater). The primary advantage of this design is its adaptability to the use of standard commercial motors.

The main disadvantages of this design type are:

1) Limited stiffness. The output shaft must be small enough to go through the bearing.
2) Difficult bearing replacement. The polygon mounting hub is usually interference fitted and must removed to replace the bearing behind it.
3) Limited dynamic balancing capability. The motor end of rotor is inaccessible after the polygon is installed.

The "Captured Design".

The captured approach (refer figure 3.9) is recommended for application requiring high speed and/or high tracking accuracy. In this approach the center-

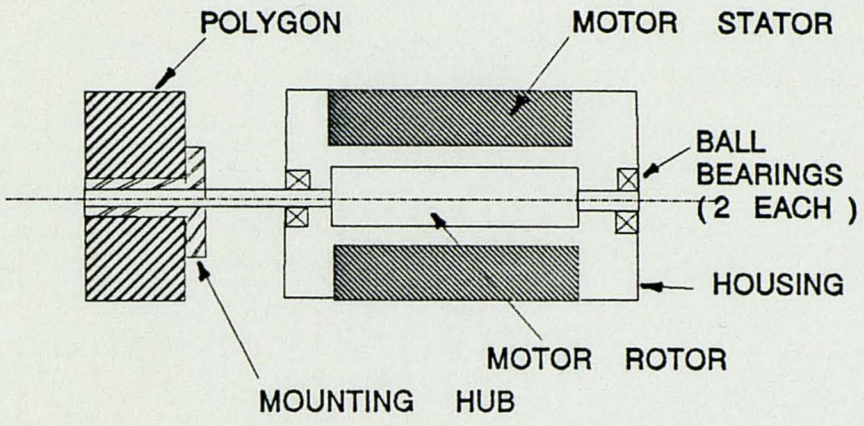

Figure 3.7 Cross-section of "cantilevered" polygon/motor construction.

to-center distance between bearings is increased, thereby reducing the contribution of bearing runout due to shaft angular runout. A major benefit of the captured design is that the entire rotor assembly and bearings may be assembled outside the housing and two-plane dynamically balanced. Additionally, bearings may be replaced without having to remove the polygon mounting hub.

The disadvantages of the captured design are :

1 Lack of adaptability to standard commercial motors.

2 Rotor must be removed to install or remove polygon.

The technology of laser beam scanning has long being used in the welding industry and in the field of robotics. In welding, the laser beam guides

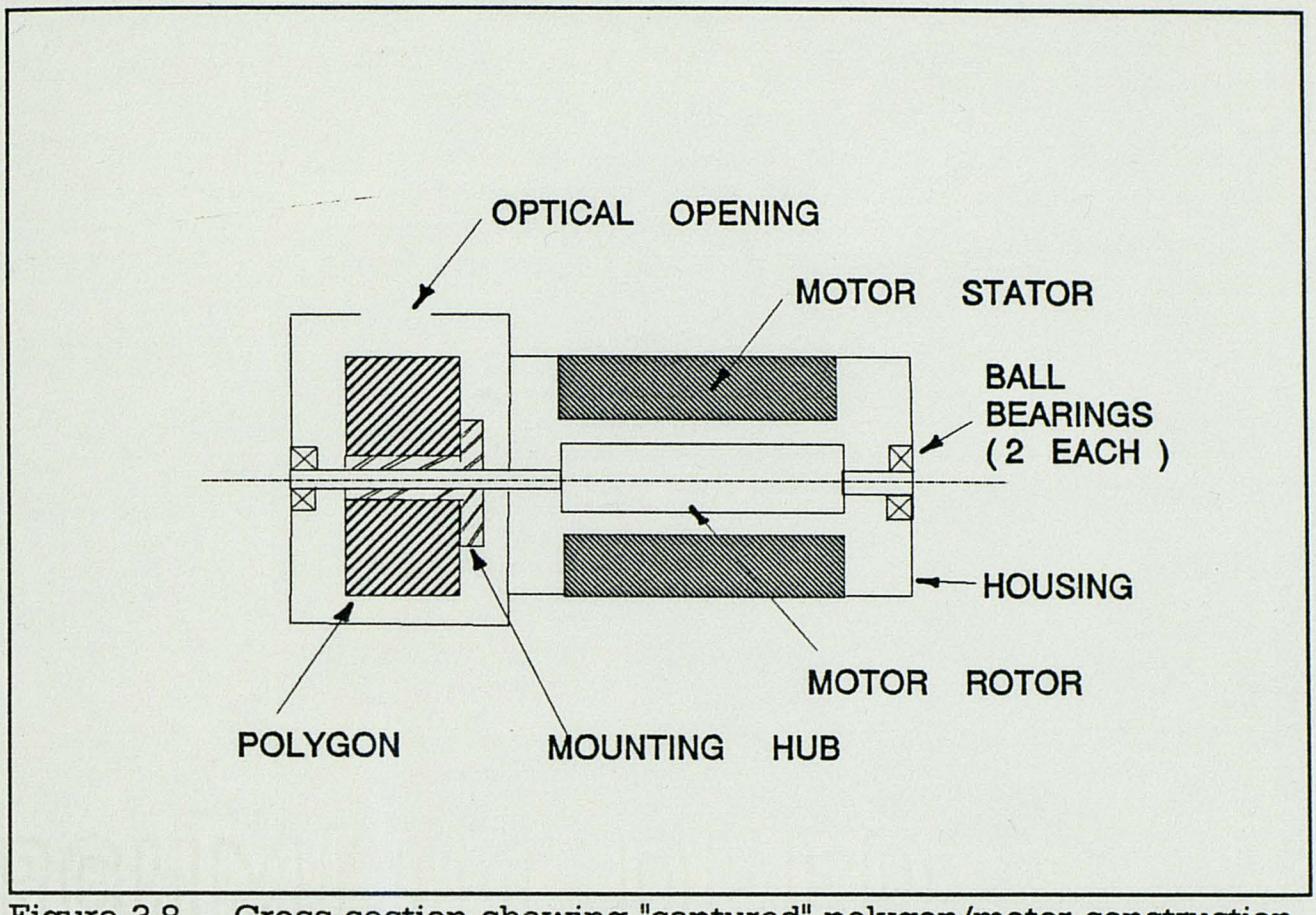

Figure 3.8 Cross-section showing "captured" polygon/motor construction.

the welding torch along the center line of the weld line. In the field of robotics, the laser scanner has been used for guiding the robot to carry out tasks after identifying parts in the scene. Besides these applications, the technology has great potential especially in manufacturing, conservation of artifacts, and inspection. The output after scanning or surface mapping is an X-Y-Z representation of the object. The next section of this thesis deals with the issue of generating the NC code for the manufacture of the object scanned. The proposed four step procedure is discussed in detail in the chapters to follow.

CHAPTER 4

TOOL PATH GENERATION

From the previous chapters, it is evident that most surface mapping techniques generate a huge amount of point data. Typically an application in manufacturing is to fabricate a "replica" of a mapped object. This may be accomplished by generating a tool path for material removal by a tool, for example a milling cutter. For the generation of tool path or NC code, only a few points are needed. For the case of a plane, only four points are needed. We propose here a four step procedure for the generation of tool path. In the case of complex objects where surfaces may take any shape an extension of the method proposed is warranted. More complex routines for curve fitting or surface fitting may be employed. However the flow of the process would be the same. This algorithm described for the proposed stages is only limited to objects of planer surfaces and straight edges, however it could be modified to handle a variety of objects of complex shapes and configurations.

To test this four step procedure a sample part (refer figure 4.1) was considered. A point data file was generated for this part which provided for the necessary scan data and the systems algorithms were used to validate the proposed technique. This section introduces the four steps and gives a description of each step.

The procedure includes four consecutive: 1) Data Reduction, 2) Geometric Data Extraction / Shape Definition, 3) Surface Recognition, and 4) NC Code Generation. The data reduction stage manipulates the data into a manageable number of points. A geometric data extraction module identifies what these points represent. This serves as an input to the surface recognition module that identifies what kind of surface it is, i.e. planer, taper, or edge. A Numerical control (NC) program for the tool path is then generated for either the fabrication of the object or its die for its mold. The algorithm could be modified to handle a wider range of geometric entities.

TOOL PATH GENERATION SYSTEM

A prime purpose for surface mapping of an object is to use the data for manufacturing the object or a die for its fabrication. Scanning may be done using any one of the methods described in chapter 2. The data generated after scanning is an X, Y, Z representation of the surface or object. The next stage could be to use this data for generating a NC program for machining the object or its die. Translating data points resulting from scanning into a NC program is not an easy task. Data has to go through several algorithms/routines before its geometric configuration is identified. The algorithm presented here is an attempt to automate a process traditionally handled through user interface. The human identifies surfaces, planes, etc. and critical dimensions. The algorithm described in this section is for planer surfaces, where a minimum of three or

more points are needed to define a plane. These planes are then used to define boundaries and edges.The algorithm developed and an explanation of the code (C Language) for each of these stages follows. The expanded version to handle a variety of configurations has being studied and is proposed in the chapter 6. Complex cases where surfaces with varying curvatures require previous knowledge about the surface characteristics, so that the proper algorithm will control the tool motion. A major benefit of the surface mapping to tool path algorithm is its elimination of human labor in generating NC code.

The surface mapping to tool path algorithm developed is for a simple rectangular object with planer surfaces and straight line grooves. The object in figure 4.1 is used for demonstration of the algorithm. The following assumptions need to be considered:

1) Surfaces do not have a gradient in the any direction.

2) The variation of surface depth is only in the Z direction.

3) The scanning intervals in the X and Y are predefined.

4) The number of scanning points per line is known.

The data obtained after surface mapping has to be analyzed and presented in a format which is acceptable for a CNC machine. This involves the following stages:

1) Data reduction.

2) Geometric data extraction(shape definition).

3) Surface recognition.

4) NC code generation.

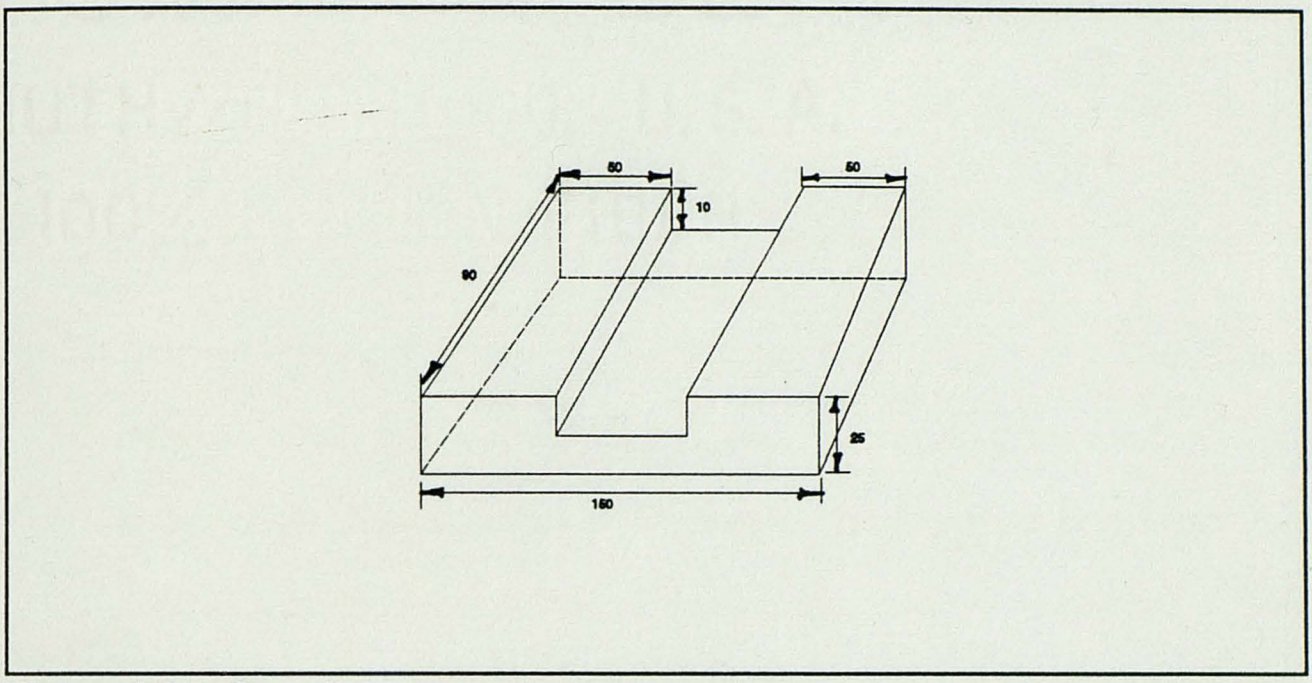

Figure 4.1 Sample part.

Figure 4.2 is a schematic representation of the four stages. The data reduction stage reduces the vast amount of data generated to a manageable amount which will suffice for NC code generation. This forms the input to the geometric data extraction module. The purpose of this stage is to determine what these points represent, that is whether these points represent a line, arc, circle, or edge. The output of this stage serves as an input to the surface recognition module. This module determines the relationship between the geometric entities identified in the previous stage. In the NC code generation module the NC code is generated based on the type of the surface feature that have been identified. These stages are described in detail in the following sections.

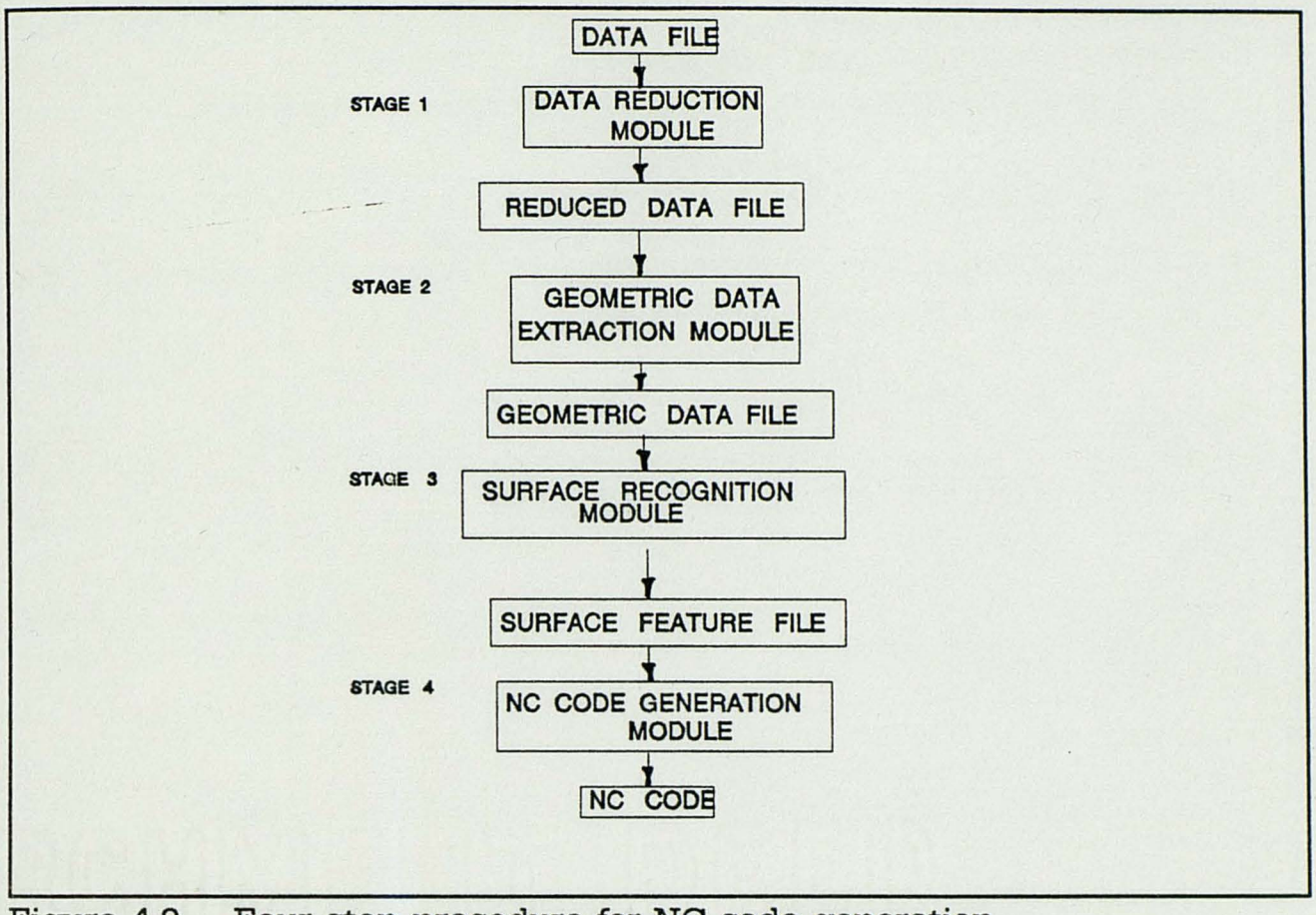

Figure 4.2 Four step procedure for NC code generation.

DATA REDUCTION STAGE

An enormous amount of data is generated after scanning. Analysis of this data (obtained after sending a test panel to one of the promising vendors) revealed that the object was scanned at intervals relatively close to each other. This generated an amount of data in excess of that needed for the tool path generation. The following options are available :

1) Digitize only at critical points. Thereby limiting the points necessary for the generation of tool path statements.

2) Reducing the data to a reasonable amount. This would mean that points in the data file would be analyzed to determine if they were a part of a line, curve, etc.

This would include analysis of the X, Y, Z points contained in the data file. A description of the proposed algorithm for data reduction and stages leading to generation of NC code are described in the following sections. Figure 4.3 shows the flowchart for this stage. The steps involved in the data reduction algorithm include:

1) Create a record for each scanned line,

2) Consider the first point - P[i], from the data file. This point will be used as a reference point.

3) Take the second point - P[i+1].

4) Compare X[i] with X[i+1] **or** Y[i] with Y[i+1].

5) If X[i] = X[i+1]

or

Y[i] = Y[i+1]

Then

compare Z[i] and Z[i+1].

6) If

Z[i] not = Z[i+1]

retain P[i].

If not go to the next point.

7) Consider P[i+1] (if P[i] has been retained), and compare it with the next point. Repeat steps 5, 6, and 7.

A record is created for each scanned line. The result of this data reduction stage is a reduced data file of a manageable size. This file then serves as an input for the geometric data extraction module.

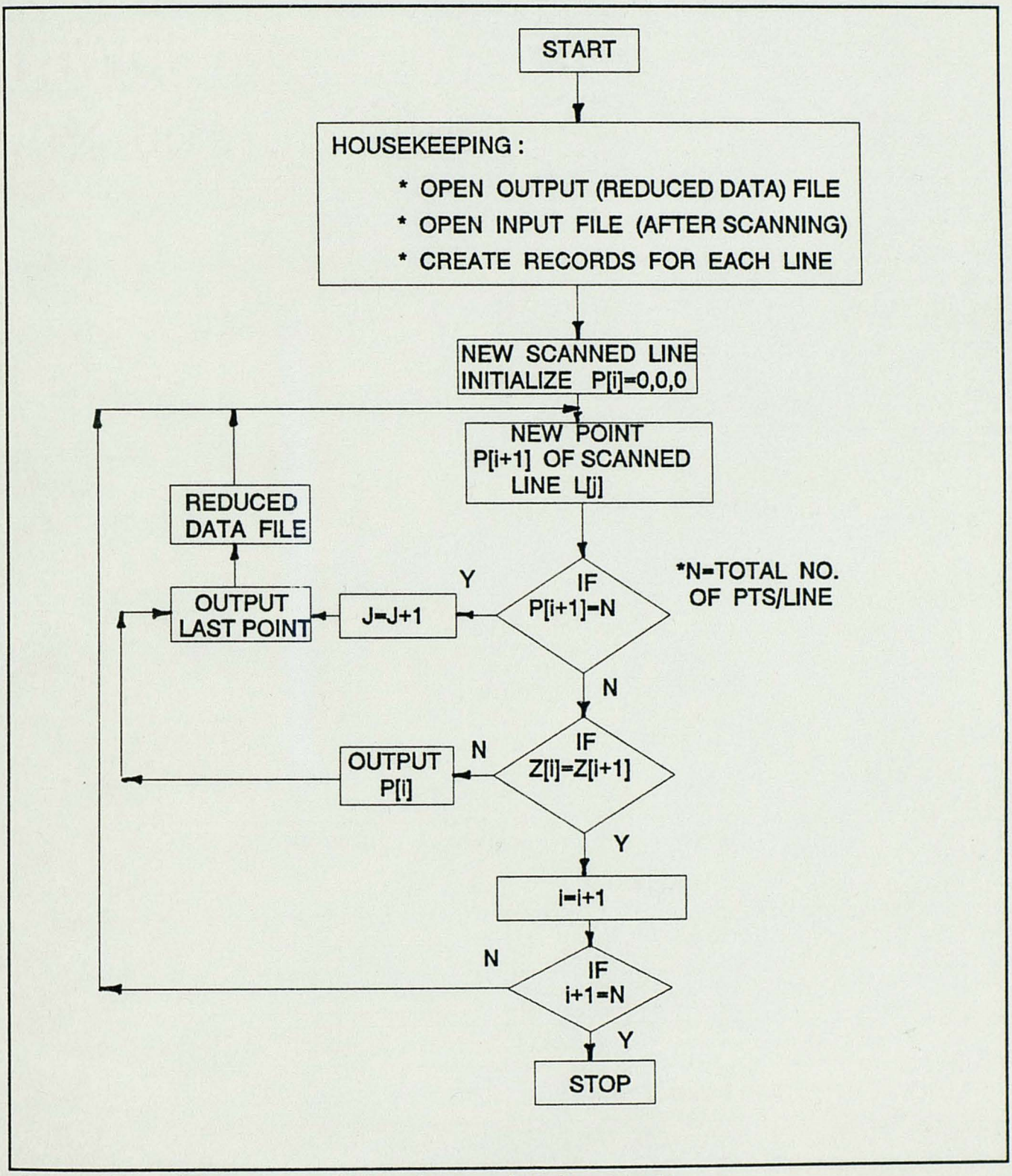

Figure 4.3 Flow chart of the data reduction stage.

GEOMETRIC DATA EXTRACTION / SHAPE DEFINITION STAGE

As seen in figure 4.1, the sample part selected for testing the validity of this four step procedure has a simple configuration. All the surfaces are planer in nature. For this sample part this stage does not have a significant role to play. However, if at all the four step procedure has to be tested for other parts having complex geometry, this stage will play a crucial role in defining a surface from the humongous amount of data generated after surface mapping. For complex shapes, the data reduction stage would not result in a file of noticeable reduction in amount of raw data. For complex shapes, the surfaces would have to be defined through the use of curves and surface fitting techniques. Presented in this section are some of the theories that could be employed for defining surfaces of complex geometry. These theories will form the backbone of algorithms for modifying the four step procedure to handle complex shapes in the future.

Curves and Surface Fitting Techniques.

This section discusses the various methods and theories for defining surfaces with the ultimate goal of defining the object and using this information in any CAM system. Normally it is the part programmer who determines the cutting conditions - cutting speed, feed, coolant, etc. by means of tables and charts, and by making the necessary assumptions on the machinability of the material to be used. The curves and surfaces produced by means of numerical

control may be classified in three categories:

(a) Curves and surfaces made up from the juxtaposition of segments belonging to mathematical curves and surfaces.

(b) Non-mathematical curves and surfaces obtained by fitting them to experimental points, and which must be reproduced within strict tolerance limits.

(c) Non-mathematical curves and surfaces, the production of which requires adherence to very strict tolerances.

The majority of shapes of mechanical parts are found in the first category. Surfaces consists of the plane, the cylinder and cone of revolution, the sphere, the torus, the quadratic surfaces and some ruled surfaces.

The second category comprises of all the surfaces obtained by fitting after experimentation. They are for example the shapes of ship hulls, the fuselages and wings of aircraft, blades of turbines, etc. are a few of the shapes that belong to this category. These surfaces are purely experimental, and cannot be interpreted a priori except by measuring coordinates on a close grid formation.

The last category is made up of surfaces, such as automobile-body shapes which are created in a purely intuitive manner to satisfy aesthetic notions. Their reproduction is not liable to such strict tolerances as the surfaces of the preceding category.

The general problem divides into two: according to whether a curve or

a surface is to be produced. A curve segment is a point-bounded collection of points whose coordinates are given by continuous, one-parameter, single-valued mathematical functions of the form:

$$x = x(u) \qquad y = y(u) \qquad z = z(u).$$

The parametric variable u is constrained to the interval $u \in [0,1]$, and the positive sense on a curve is the sense in which u increases. The curve is point-bounded because it has two definite end points, one at u=0 and the other at u=1.

Bezier identified the fundamental property of parametric curves: their shape depends on only the relative position of the points defining their characteristic vectors and is independent of the position of the total set of points with respect to the coordinate system in use. This is an essential characteristic for many applications, such as CAD/CAM modelling. In general, to transform an axis dependent curve, one must compute the coordinates of every point required in the original system, then transform each into the new system. For axis-independent curves, it is sufficient to transform the points defining the characteristic vectors from one system to another. A few important curve and surface defining methods are presented below.

Spline Curves

The spline curve is perhaps the single most important curve in both the aircraft and shipbuilding industries. A drafting tool called a spline is a strip of

plastic or other material that is flexed to pass through a series of key design points (control points) already established on a drawing. Weights called ducks hold the spline in place while the draftsman uses the spline as a guide to draw a smooth curve formed by it through the design points. A spline curve can be drawn through any set of n points that imply a smooth curve.

A spline behaves structurally exactly like a beam, with bending deflections forming into a smooth curve. As long as the distribution of control points the material and the stiffness of the spline allow the spline to deform elastically, any spline will from the same curve for the same set of control points. This curve is often called an elastic curve, or minimum energy curve.

Bezier Curves

Some curve defining techniques interpolate a given set of points, which means that the curve produced passes exactly through the points. An alternative approach defines a curve that only approximates or approaches the given points. Interpolation techniques have certain disadvantages when incorporated into an interactive CAD program. Specifically, this is so because one does not get a strong intuitive feel for how to change or control the shape of a curve. For example, if we try to change the shape of a spline interpolated curve by moving one or more of the interpolated points, we may produce the unexpected. It is much easier if we can control curve shape in a predictable way by changing only a few parameters. Bezier's curve partially satisfies this

need.

Bezier curves were formulated by P. Bezier of the French automobile company - Renault. The result of Bezier's work was the UNISURF system, used by Renault since 1972 to design the sculptured surfaces of many of their automobile bodies. At the heart of the UNISURF system are the curves and surfaces that bear his name. Bezier formulated the curve with the principle that any point on a curve segment must be given by a parametric function of the following form:

$$\mathbf{p}(u) = \Sigma\ \mathbf{p}_i\ f_i(u) \qquad u \in [0,1] \qquad (1)$$

where the vectors $\mathbf{p}_i$ represent the n+1 vertices of a characteristic polygon (refer figure 4.4). These vertices are also called control points.

Bezier laid down certain properties that the $f_i(u)$ blending functions must have.

1) The functions must interpolate the first and last vertex points, that is, the curve segment must start on $\mathbf{p}_0$ and end on $\mathbf{p}_n$. It is upto the user to control the starting and ending points of a Bezier curve.

2) The tangent at $\mathbf{p}_0$ must be given by $\mathbf{p}_1$-$\mathbf{p}_0$, and the tangent at $\mathbf{p}_n$ by $\mathbf{p}_n$-$\mathbf{p}_{n-1}$. This gives the user direct control of the tangent of the curve at each end.

3) This requirement is generalized for higher derivatives at the curve's end points. Thus, the second derivatives at $\mathbf{p}_0$ must be determined by $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$. In general, the r^{th} derivative at an end

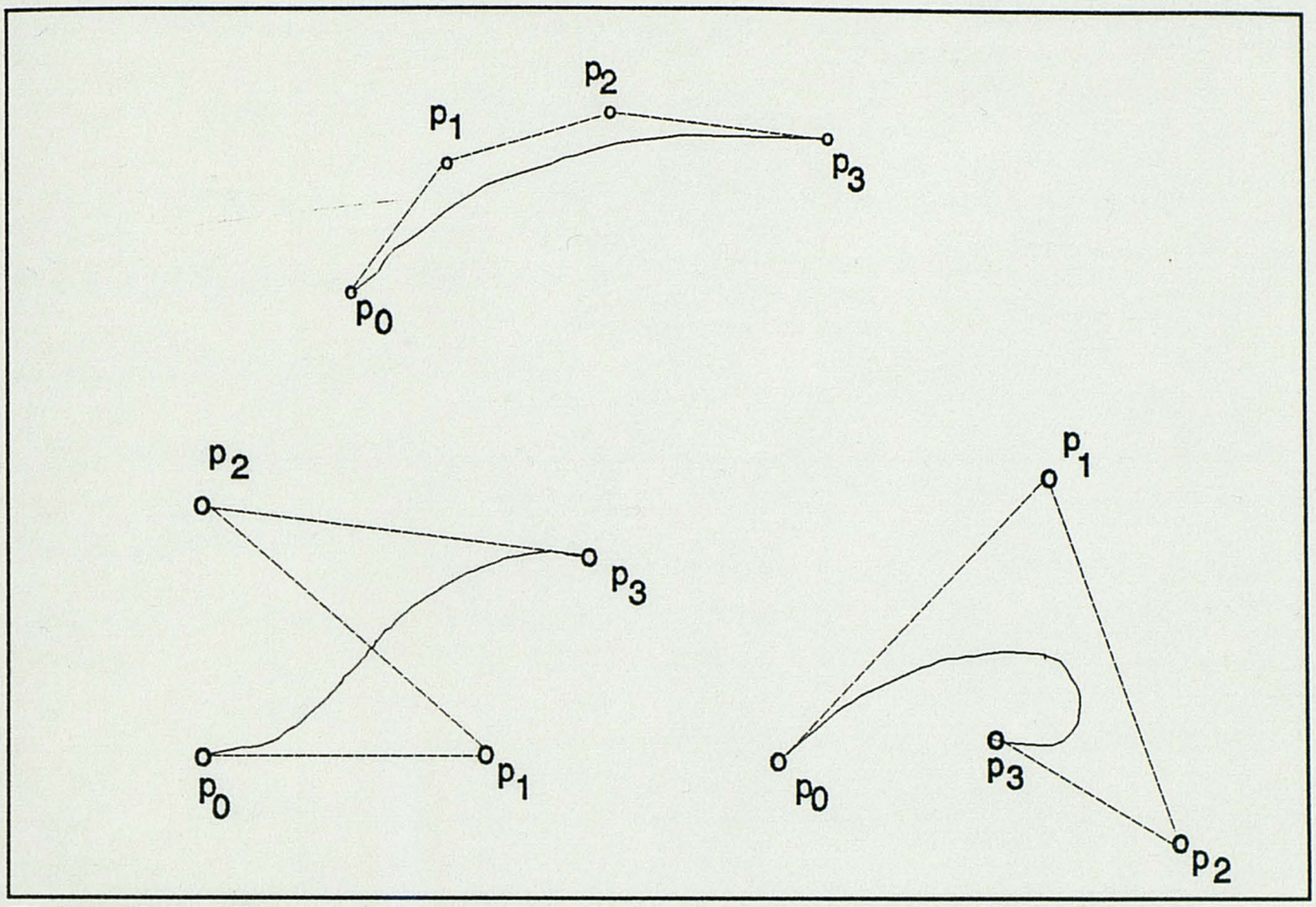

Figure 4.4 Bezier curves.

point must be determined by its r neighboring vertices. This allows the user unlimited control of the continuity at the points between curve segments of a composite Bezier curve.

4) The functions $f_i(u)$ must be symmetrical with respect to u and (1-u). This means that one can reverse the sequence of the vertex points defining the curve without changing the shape of the curve. In effect, this reverses the direction of parametrization.

Bezier chose a family of function called Bernstein polynomial to satisfy these conditions mentioned above. He originally chose a form of vector representation that used the sides of the characteristic polygon. The function Bezier selected depends on the number of vertices used to specify a particular curve.

Equation 1 becomes

$$\mathbf{p}(u) = \Sigma\ \mathbf{p}_i\ B_{in}(u) \qquad u \in [0,1] \qquad (2)$$

where $B_{in}(u) = C(n,i)\ u^i\ (1-u)^{n-i}$

and $C(n,i) = n!\ /\ i!\ (n-i)!$.

The Bezier curve is widely used as a modeling technique in a CAD system. The properties of the Bezier curve that make it an unusually effective interactive design tool are listed as follows:

1) The curve has end points in common with the polygon (the other vertices are usually not on the curve).

2) The slope of the tangent vectors at the end points equals the slope of the first and last segments of the polygon.

3) The curve lies entirely within the convex figure defined by the extreme points of the polygon (often called the convex hull).

4) Bezier curves are variation diminishing. This means that they never oscillate wildly away from their defining control points.

5) When compared to conventional polynomials or splines, all that is needed for a Bezier curve is the data points.

6) The parametric formulation allows a curve to represent multiple-valued shapes.

B-Spline Curves

Most curve defining techniques do not provide for local control of shape. Consequently local changes (for example, a small change in the position of a point on a spline curve or of a vertex of a characteristic polygon of a Bezier polygon) tend to be strongly propagated throughout the entire curve. This is sometimes described as a global propagation of change. The B-spline curve avoids this problem by using a special set of blending functions that has only local influence and depends on only a few neighboring control points.

B-spline curves are similar to Bezier curves in that a set of blending functions combines the effects of n+1 control points $\mathbf{p}_i$ given by

$$p(u) = \sum_{i=0}^{n} p_i \, N_{i,k(u)} \tag{3}$$

The equation of a bezier curve is

$$p(u) = \sum_{i=0}^{n} p_i B_{i,n(u)} \tag{4}$$

By comparing the equation of the Bezier curve and the B-spline curve, the most important difference is the way the blending function $N_{i,k(u)}$ are formulated. For Bezier curves, the number of control points determine the degree of the blending function polynomials. For the B-spline curves, the degree of these polynomials is specially controlled by a parameter k and usually independent of the number of control points. The B-spline blending functions are defined recursively by the following expressions:

$$N_{i,i(u)} = 1 \text{ if } t_i \le u < t_{i+1} \tag{5}$$

and

$$N_{i,K(u)} = (u - t_i)\frac{N_{i,K-1}(u)}{t_{i+K-1} - t_i} + \frac{(t_{i+K} - u)N_{i+1,K-1}(u)}{t_{i+K} + t_{i+1}} \quad (6)$$

where k controls the degree (k-1) of the resulting polynomial in u and thus also controls the continuity of the curve. The t_i are called **knot values**. They relate the parametric variable u to p_i control points. The range of the parametric variable u is

$$0 \leq u \leq n\text{-}k+2$$

B-spline curves and Bezier curves have many advantages in common: Control points influence the curve shape in a predictable, natural way, making them good candidates for use in an interactive environment. Both types of curves are variation diminishing, axis independent, and multivalued and both exhibit the convex hull property. However, it is the local control of curve shape possible with B-splines that gives the techniques an advantage over the Bezier technique, as does the ability to add control points without increasing the degree of the curve.

Surfaces

The simplest mathematical element to model a surface is a patch. A patch is a curve bounded collection of points whose coordinates are given by continuous, two-parameter, single valued mathematical function of the form

$$x = x(u,w) \quad y=y(u,w) \quad z=z(u,w)$$

The parametric variables u and w are constrained to the intervals u,w ϵ [0,1]. Fixing the values of one of the parametric variables results in a curve on the patch in terms of the other variable, which remains free. By continuing this process first for one variable and then the other for any number of arbitrary values in the allowed interval, the result is a parametric net of two one parametric families of curves on the patch such that just one curve of each point **p**(u,w).

Associated with every patch is a set of boundary conditions (refer figure 4.5). There are the four corner points and four curves defining its edges, the tangent and twist vectors. For any ordinary patch, there are always four and only four corner points and edge curves. This is due to possible combination of the two parametric variables. The corner points are found by substituting these four combinations of 0 and 1 into **p**(u,w) to obtain **p**(0,0), **p**(0,1), **p**(1,0), and **p**(1,1). The edge or boundary curves are functions of one of the two parametric variation. These can be obtained by allowing any of the variables to remain free, while fixing the other to its limiting values. This procedure results in four and only four possible combinations yielding the functions of the four parametric boundary curves **p**(u,0), **p**(u,1), **p**(0,w), and **p**(1,w).

A major advantage of the parametric representation of surfaces is the complete control one has over the domain of a surface modeling operation simply by an appropriate choice of the parametrization scheme. By carefully specifying subsets of a particular domain $[u_{min}, u_{max}] \times [w_{min}, w_{max}]$, one can

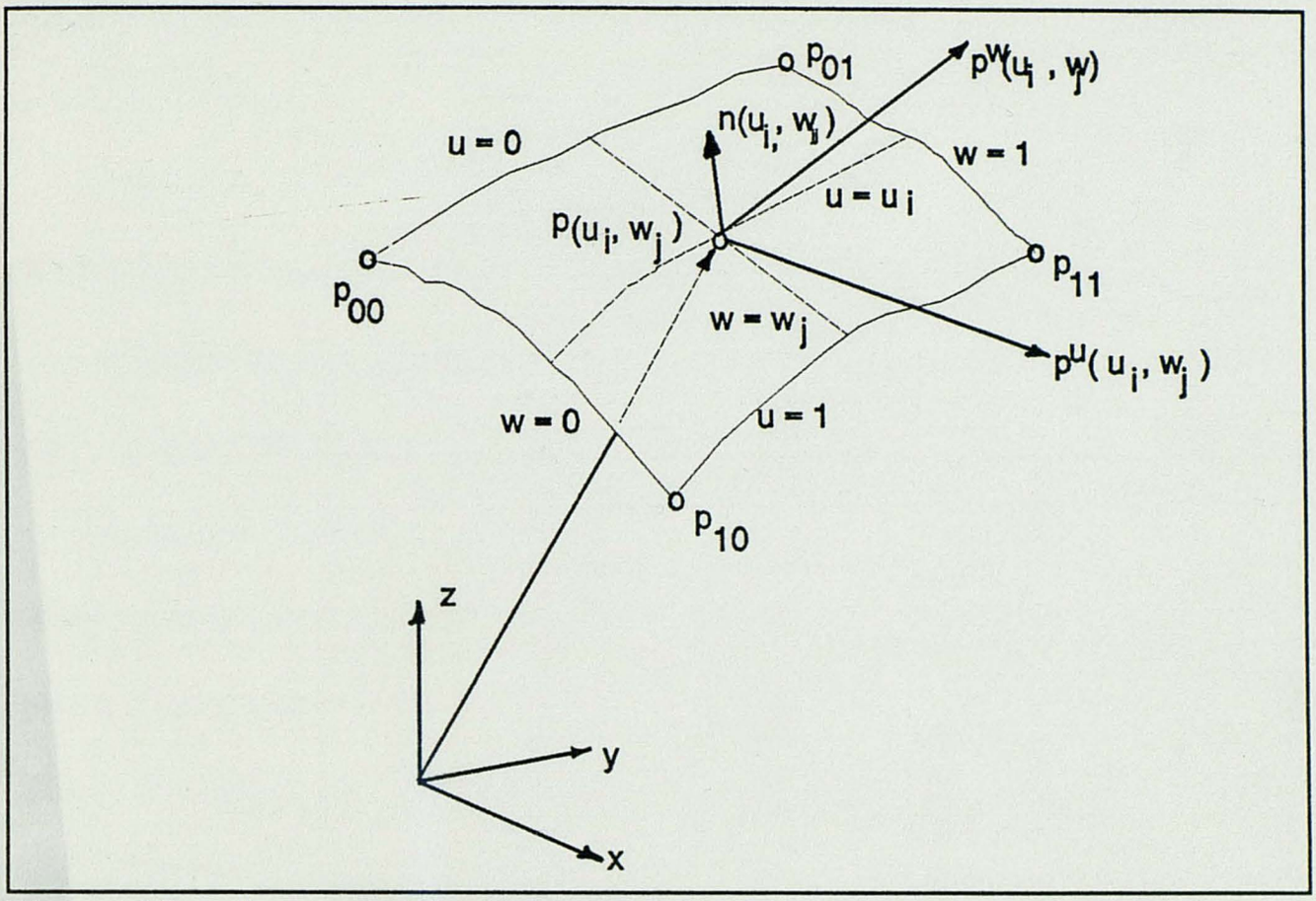

Figure 4.5 A parametric surface patch.

readily define certain sections of a surface. This feature is useful when a surface is composed of several patches. Considerable amount of work in the area of surface description has been done over the past decade. To date there have been over 106 different methods of surface representation methods developed so far. The section to follow will describe in brief some of the systems that have been developed. It may be noted that most of the systems of surface representation to date use some of the above mentioned theories.

Bezier Surfaces

Just as the Bezier curve has a characteristic polygon the Bezier surface has a characteristic polyhedron. Points on the Bezier surface are given by a simple

extension of the general equation for points on a Bezier curve.

$$p(u,w) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{ij} B_{i,m(u)} B_{j,n(w)} \qquad u,w \in [0,1] \tag{7}$$

where $\mathbf{p}_{ij}$ are the vertices of the characteristic polyhedron that form an (m+1) x (n+1) rectangular array of points , and $B_{i,m}(u)$ and $B_{j,n}(w)$ are defined as curves. The matrix **P** contains the position vectors for points that define the characteristic polyhedron and thereby the Bezier surface patch. Figure 4.6 illustrates these points, the polyhedron, and the resulting patch. In the Bezier formulation, only the four corner points $\mathbf{p}_{11}$, $\mathbf{p}_{41}$, $\mathbf{p}_{14}$, and $\mathbf{p}_{44}$ actually lie on the patch. The points $\mathbf{p}_{21}$, $\mathbf{p}_{31}$, $\mathbf{p}_{12}$, $\mathbf{p}_{12}$, $\mathbf{p}_{13}$, $\mathbf{p}_{42}$, $\mathbf{p}_{43}$, $\mathbf{p}_{24}$, and $\mathbf{p}_{34}$ control the slope of the boundary curves. The remaining four points $\mathbf{p}_{22}$, $\mathbf{p}_{32}$, $\mathbf{p}_{23}$, and $\mathbf{p}_{33}$ control the slopes along the boundary curves.

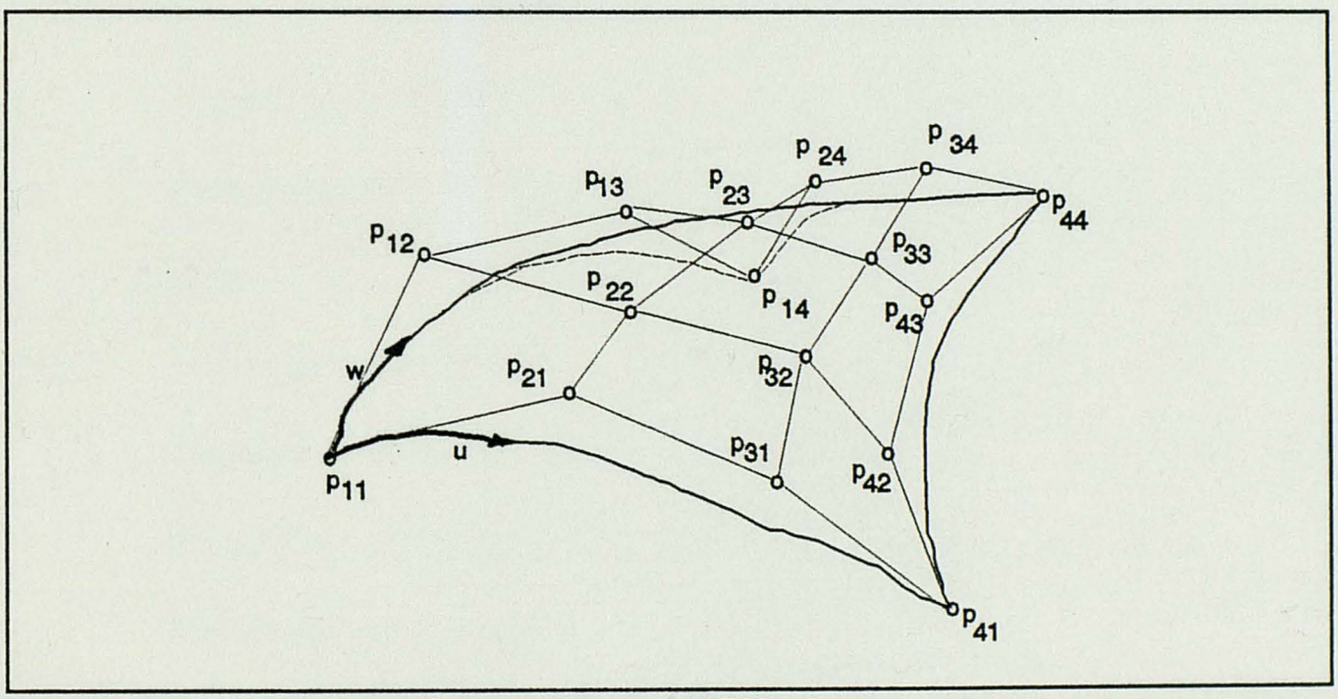

Figure 4.6 A bezier surface. (Bezier 1973)

B-Spline Surface

The formulation of a B-spline surface follows directly from that of B-spline curves. This relationship is analogous to that between Bezier curves and surfaces. Further more, the B-spline surface, like the Bezier surface, is defined in terms of a characteristic polyhedron. The approximation is weaker the higher values of k and l. Thus,

$$p(u,w) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} N_{i,k(u)} N_{j,l(w)} \quad (8)$$

The $\mathbf{p}_{ij}$ are the vertices of the defining the polyhedron, and the $N_{i,k}(u)$ and $N_{j,l}(w)$ are the blending functions of the same from as those for B-spline curves (refer equation 3) The degree of each of the blending function polynomials $N_{i,k}(u)$ and $N_{j,k}(w)$ is controlled by k and l respectively.

SURFACE RECOGNITION STAGE

Three pieces of information are important for NC part programming application:

(1) the location of each surface feature,

(2) the type of each surface feature, and

(3) the relationship between each pair of surface features.

The next step after determining which geometric shape the points represent is to determine the relationship between these shapes. These lines represent an edge or are a part of a surface which might be represented by

three or more points. Figure 4.7 shows the flowchart for this stage.

The structure of the algorithm for determining the relationships between the entities is described below:

1) Consider the first set of points contained in the first record - P, (that represent the first segment of the first scanned line). Compare it with the first set of points of second record - P+1, of the second scanned line.

2) The slopes between these points are then computed. If the slopes are equal, then this could be taken to be a planer surface. If the slopes are not equal, then a change in surface orientation is recorded, i.e. taper, convex, etc.(not yet considered).

3) If the slopes of the two lines are equal then a surface is recorded which can be represented by these four points.

4) Steps 1 - 3 are repeated for each segment within every scanned line.

5) The relationship between surfaces should be stored in the order in which they are recognized. These will be called in the same order for NC code generation.

6) Each stored surface is given a code as an identification, i.e. 1 -for planer, 2 - for taper, etc.. This data forms an input for the NC part program generation algorithm which is described below. This data file is referred to as the surface feature file.

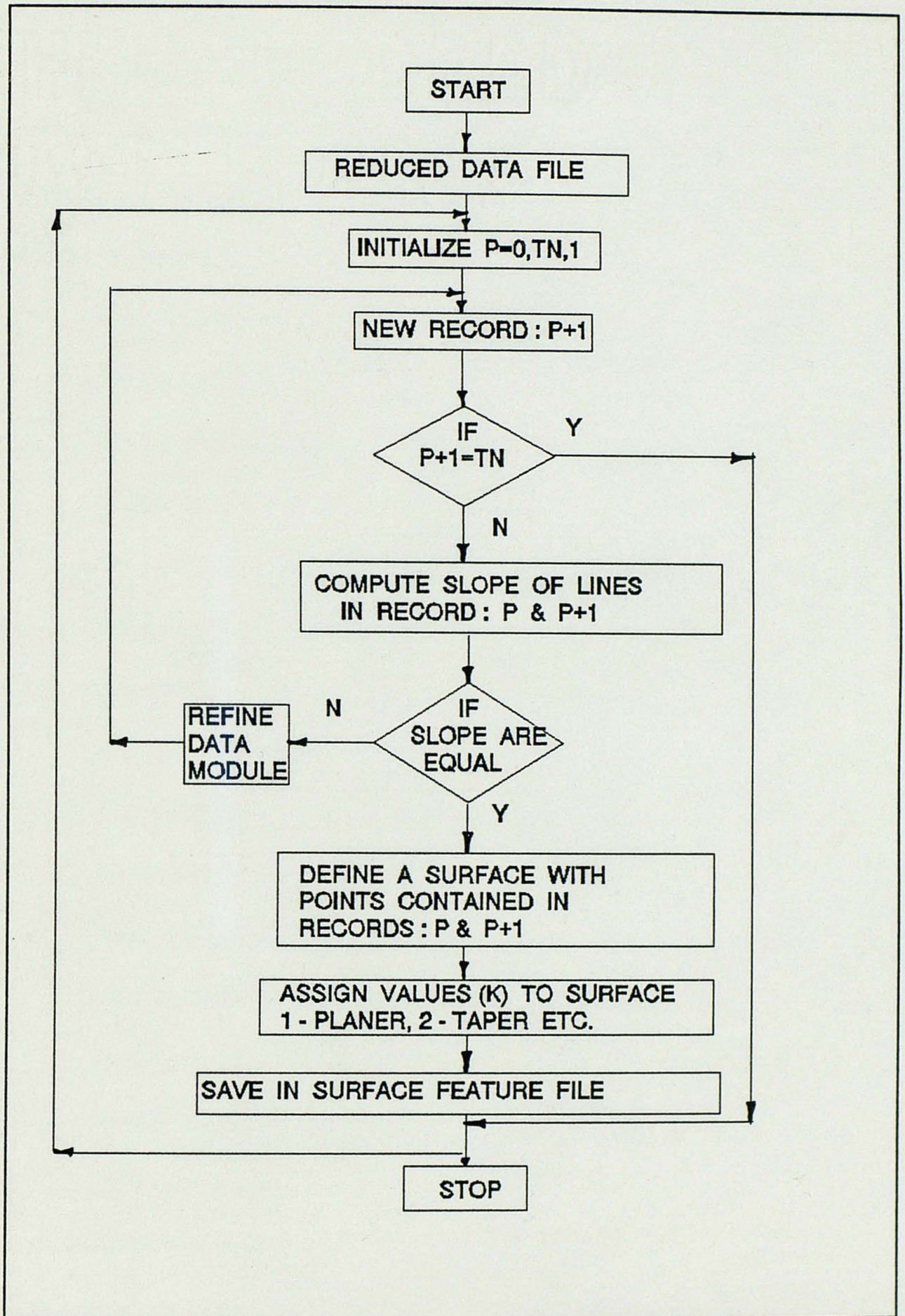

Figure 4.7 Flowchart of the surface recognition stage.

NC CODE GENERATION STAGE

Surface orientation and locations of both the start and end points of each feature are available from the surface feature file.The routine of the NC code generation algorithm is described below and the flowchart of the algorithm is shown in figure 4.8.

1) Read the surface code (k) of each surface (which was identified in the previous stage).

2) Set depth of cut (d) and feed rate (f).

3) Based on the value of K, call the appropriate sub-routine for machining that surface. That is, the surface could be planer, circular, convex or concave. Depending on this value of K, the routine from the main program jumps to the sub-routine for generating NC code for the appropriate surface.

4) Repeat this procedure for all the surface features contained in the surface feature file.

5) After all the surface features have been processed, select an appropriate depth of cut for the finish cut.

6) Move the tool to the appropriate position and finish cut to generate the shape desired.

The output of this stage is a numerical control program that can be used to machine the part.

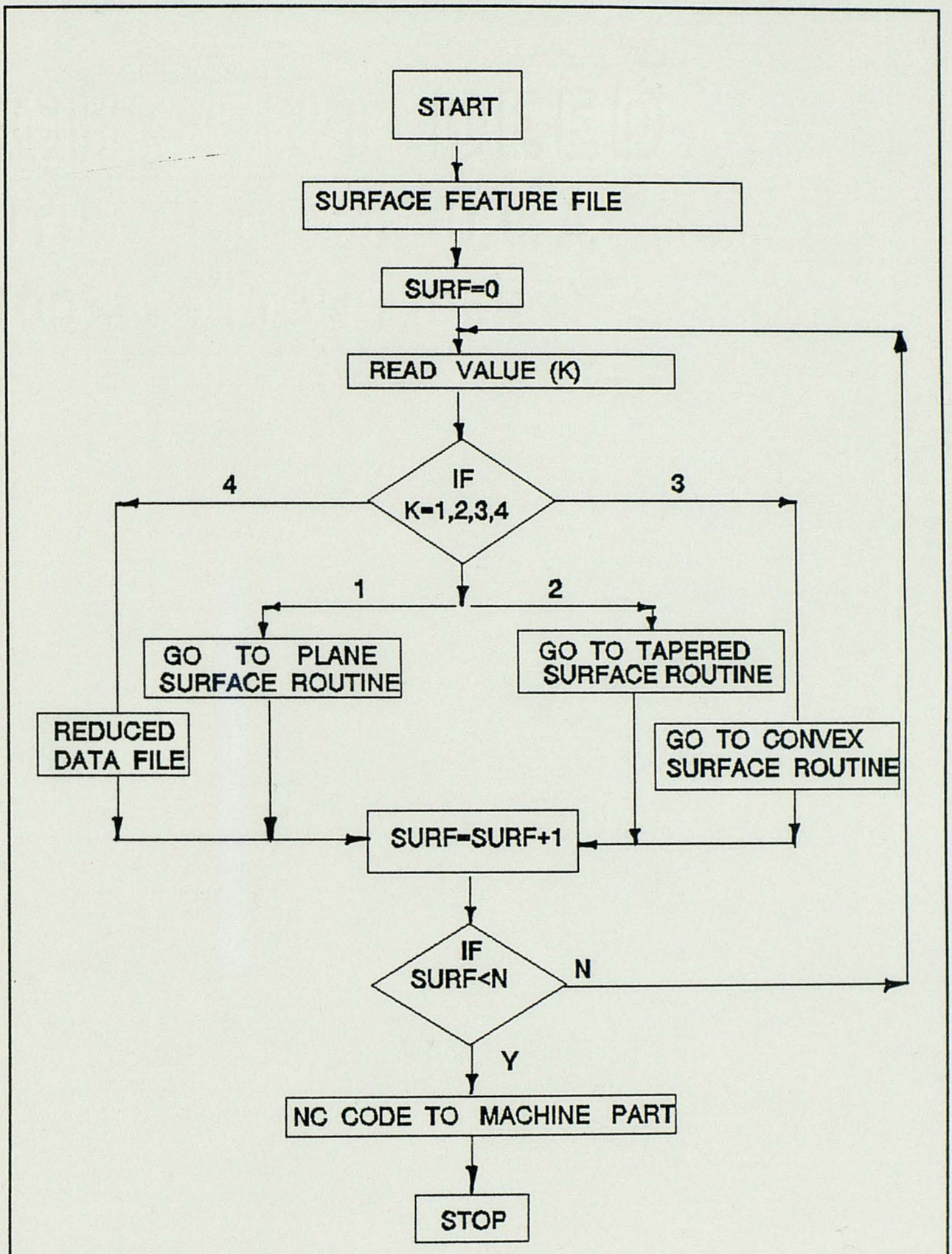

Figure 4.8 Flowchart of the NC code generation stage.

The results of each of the stages describe in this chapter are presented in the next chapter.

CHAPTER 6

ANALYSIS OF THE RESULTS

The results of the proposed algorithm are presented in this section. The code for each of these stages can be found in the Appendix.

Data Reduction Stage.

The data file for the sample part is not enclosed due to its size - 1640 points. The data file for the sample part was created for two directions - parallel to the X axis and parallel to the Y axis. The data file for the X axis had data for 31 lines of scanning and that for the Y axis had 19 lines. The data file for the X axis resulted in the reduced data file REDF, portion of which is listed in this section. The code for this stage can be found in Appendix A and the full listing of the REDF file in Appendix B. The number 6 in the REDF file stands for the number of segments per line detected. The data file for the Y axis resulted in the reduced data file REDL (sample listed in this section. The code for this file can be found in Appendix C and the listing of the REDL file in Appendix D.

REDF - Reduced data file for parallel to X-axis.
6
25.012 0.000 0.065
25.309 0.000 25.053
30.097 0.000 25.053
75.067 0.000 25.053

REDL - Reduced data file for parallel to Y-axis.

```
3
25.077 0.000 0.087
25.131 0.000 25.097
25.109 5.096 25.191
25.138 90.018 25.130
25.195 90.099 20.154
25.182 90.028 0.020
```

Surface Recognition Stage

These two reduced data files form the input to the surface recognition stage. In the surface recognition stage, the algorithm defines a surface with four points. The output for the REDF file (parallel to the X-axis) is the SURF1 file listed in this section. It shows the total number of surfaces and a number for each of the surfaces identified. The code for this stage can be found in Appendix E. Similarly the output for the REDL file (parallel to the Y-axis) is SURF2, which is listed in this section. The code for this stage can be found in Appendix F.

SURF1 - Surface feature file for parallel to X axis.

```
6
1
25.012 0.000 0.065
25.309 0.000 25.053
25.014 90.000 0.019
25.351 90.000 25.021
2
30.097 0.000 25.053
75.067 0.000 25.053
30.008 90.000 25.021
```

```
75.090 90.000 25.021
3
80.082 0.000 15.087
125.133 0.000 15.087
80.106 90.000 15.071
125.142 90.000 15.071
4
125.133 0.000 20.110
125.133 0.000 25.133
125.142 90.000 20.143
125.142 90.000 25.139
5
130.146 0.000 25.133
175.171 0.000 25.133
130.220 90.000 25.139
175.213 90.000 25.139
6
175.171 0.000 20.222
175.171 0.000 0.174
175.213 90.000 20.199
175.213 90.000 0.204
```

SURF2 - Surface feature file for parallel to Y axis.

```
5
1
25.077 0.000 0.087
25.131 0.000 25.097
175.111 0.000 0.085
175.112 0.000 25.081
2
25.109 5.096 25.191
25.138 90.018 25.130
75.113 5.030 25.025
75.127 90.049 25.036
3
80.171 5.092 15.118
80.206 90.068 15.040
120.097 5.037 15.032
120.188 90.072 15.106
4
```

125.183 5.076 25.045
125.156 90.021 25.007
175.098 5.025 25.110
175.224 90.091 25.163
5
25.195 90.099 20.154
25.182 90.028 0.020
175.294 90.108 20.218
175.237 90.180 0.042

NC Code Generation Stage.

The next step is to use this data form the surface feature file for generation of NC code. The data in each of the surface feature file forms the necessary data for the geometry definition needed in the writing of APT statements. The NC code for this stage is listed below. The program (C Language) for this stage can be found in Appendix G.

```
PART NAME - SAMPLE PART
P1 = POINT/25.012,0.000,0.065
P2 = POINT/25.309,0.000,25.053
P3 = POINT/25.014,90.000,0.019
PL1 = PLANE/P1,P2,P3
P4 = POINT/25.077,0.000,0.087
P5 = POINT/25.131,0.000,25.097
P6 = POINT/175.111,0.000,0.085
PL2 = PLANE/P4,P5,P6
P7 = POINT/25.109,0.000,25.053
P8 = POINT/25.138,90.018,25.053
P9 = POINT/75.113,5.030,25.021
PL3 = PLANE/P7,P8,P9
P10 = POINT/80.082,0.000,15.087
P11 = POINT/80.206,90.068,15.087
P12 = POINT/120.097,5.037,15.071
PL4 = PLANE/P10,P11,P12
P13 = POINT/125.133,0.000,20.110
P14 = POINT/125.133,0.000,25.133
P15 = POINT/125.142,90.000,20.143
PL5 = PLANE/P13,P14,P15
```

```
P17 = POINT/125.156,90.021,25.133
P18 = POINT/175.098,5.025,25.139
PL6 = PLANE/P16,P17,P18
P19 = POINT/175.171,0.000,20.222
P20 = POINT/175.171,0.000,0.174
P21 = POINT/175.213,90.000,20.199
PL7 = PLANE/P19,P20,P21
P22 = POINT/25.195,90.099,20.154
P23 = POINT/25.182,90.028,0.020
P24 = POINT/175.294,90.108,20.218
PL8 = PLANE/P22,P23,P24
SP = POINT/0,0,0

$$
FEDRAT/120
SPINDL/1200,CCW
INTOL/0.005
OUTOL/0.005
CUTTER/20
COOLNT/ON

$$
FROM/SP
THICK/0.02
GO/TO,PL1
GOFWD,PL1,PAST,25.014,90.000,0.019
GO/TO,PL2
GOFWD,PL2,PAST,175.111,0.000,0.085
GO/TO,PL3
GOFWD,PL3,PAST,75.113,5.030,25.021
GO/TO,PL4
GOFWD,PL4,PAST,120.097,5.037,15.071
GO/TO,PL5
GOFWD,PL5,PAST,125.142,90.000,20.143
GO/TO,PL6
GOFWD,PL6,PAST,175.098,5.025,25.139
GO/TO,PL7
GOFWD,PL7,PAST,175.213,90.000,20.199
GO/TO,PL8
GOFWD,PL8,PAST,175.294,90.108,20.218
GO/TO,SP
THICK/0.00
GO/TO,PL1
GOFWD,PL1,PAST,25.014,90.000,0.019
```

```
GO/TO,PL2
GOFWD,PL2,PAST,175.111,0.000,0.085
GO/TO,PL3
GOFWD,PL3,PAST,75.113,5.030,25.021
GO/TO,PL4
GOFWD,PL4,PAST,120.097,5.037,15.071
GO/TO,PL5
GOFWD,PL5,PAST,125.142,90.000,20.143
GO/TO,PL6
GOFWD,PL6,PAST,175.098,5.025,25.139
GO/TO,PL7
GOFWD,PL7,PAST,175.213,90.000,20.199
GO/TO,PL8
GOFWD,PL8,PAST,175.294,90.108,20.218
GO/TO,SP
COOLNT/OFF
SPINDL/OFF
FINI
```

CONCLUSION

The proposed procedure of generation of NC code seems feasible. However, the second stage - Geometric Data / Shape definition stage didn't play a role at all due to the configuration of the part. This stage and its theories were studied and discussed in detail in chapter 4. This stage will play an important role in the event of the modification of this four step procedure to handle complex shapes.

CHAPTER 6

CONCLUSIONS AND POTENTIAL RESEARCH TOPICS

Techniques for surface mapping are promising. The technique of laser beam scanning seems to be most suitable for capturing data in real time, while the Moire and image processing based techniques is equally promising for off-line data capturing. Generation of tool path automatically is also feasible, especially for geometrically simple objects. This was proven through the application of the proposed four steps. These four steps for tool path generation from surface mapping are: 1)Data Reduction Stage - a stage in which the size of the data file is reduced, 2) Geometric Data Extraction / Shape Definition Stage - the points resulting from the first are analyzed and geometric entities are defined, 3) Surface Recognition Stage - this stage defines a label for each surface, and 4) NC Code Generation Stage - the NC code is generated for surfaces identified in the surface recognition stage. The code for these stages for a simple object were developed and applied to test the validity of the proposed four step procedure. The results demonstrates the feasibility of automating the process of NC code generation after surface mapping.

The method by no means implies that the methodology used here can be used generically. Other methods, such as polygon or mesh overlap could be much more promising in other applications. However if enough routines can be

developed and integrated in a library guided by some kind of intelligence to select the best routines which may be applied to define the object before the generation of tool path statements. It is equally important to develop a translator, so that the NC code generated (for the tool path) is compatible with any machine. The generation of surfaces or curves from point data has not yet been perfected and it should be considered in future research. The user still has to specify the number of control points and the degree of accuracy required. If this task can be fully automated, then even the complex CAM systems available can be handled by a novice, and we may one day achieve the "concept to a reality" automatically.

A number of potential research topics were identified in this research. These include the following:

1) Selecting optimal angles for scanning to avoid the shadow effect.

2) Resolve the problems associated with the integration of different scans of the same object.

3) The possible use of intelligence in the generation of curves and surfaces.

4) The need for additional algorithms for surface fitting from three dimensional data.

APPENDICES

APPENDIX A

DATA REDUCTION STAGE - PARALLEL TO X-AXIS

DATA REDUCTION STAGE - PARALLEL TO X-AXIS

```
/* DATA REDUCTION STAGE - PARALLEL TO X-AXIS*/
/* Function Prototype *******/


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>
#define X_limit 0.5
#define X_max_limit 2.5
#define Y_limit 0.5
#define Z_limit 0.5  /* TO DETECT CHANGE IN Z*/

/*  Global Variable Declaration  */
int In_Rec_n=1,Rec_n=1,Seg_n=1,n_pts_bet=1,check_z,store=0,pt=0;

float prev_X,prev_Y,prev_Z;
float X,Y,Z;
float temp_X,temp_Y,temp_Z;

FILE *f1ptr, *f2ptr;


struct XYZ
{
  float X;
  float Y;
  float Z;
};

struct Segment
{
        struct XYZ start_pt;
        struct XYZ end_pt;
        int n_pts_bet;
};
```

```
/*structure for each line parallel to X */

struct RecordY
{
 struct  Segment segment[10];
 int SEG_N;
}parallel_X[19];


main()
{
int j,k,p,q;
char name1[35], name2[35];

printf("Enter filename(data file) \n");
scanf("%s", name1);
printf("Enter number of lines Scanned on the object (parallel to  X axis)\n");
scanf("%d",&In_Rec_n);
printf("Enter name of reduced data file \n");
scanf("%s",name2);

if((f1ptr = fopen(name1, "r")) ==NULL)
   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }

if((f2ptr = fopen(name2, "w+")) ==NULL)

   {
   printf("Can't write to file %s \n", name2);
   exit(1);
   }


/* READS IN FIRST POINT IN THE FILE */
  fscanf(f1ptr,"%f %f %f",&prev_X,&prev_Y,&prev_Z);
  parallel_X[Rec_n].segment[Seg_n].start_pt.X=prev_X;
  parallel_X[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
  parallel_X[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;


/* For scanning parallel to X axis */
do{
```

```
fscanf(f1ptr,"%f %f %f", &X,&Y,&Z);  pt++;

if( prev_Y!=Y) {

 parallel_X[Rec_n].segment[Seg_n].end_pt.X=temp_X;
        parallel_X[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
        parallel_X[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
        parallel_X[Rec_n].SEG_N=Seg_n;
        if(pt>1)        parallel_X[Rec_n].segment[Seg_n].n_pts_bet=pt-2;

        Rec_n++;    Seg_n=1;     store=0;      pt=0;

        prev_X=X;
        prev_Y=Y;
        prev_Z=Z;

        parallel_X[Rec_n].segment[Seg_n].start_pt.X=prev_X;
        parallel_X[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
        parallel_X[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;
        fscanf(f1ptr,"%f %f %f", &X,&Y,&Z);pt++;

}

if( fabs(prev_X-X) < X_limit ){
        temp_X=X;
        temp_Y=Y;
        temp_Z=Z;
}

if(  (fabs(prev_X-X)> X_max_limit) && ( fabs( prev_Z-Z) < Z_limit ) ){

/* since z values are not exceeding the limit we will scan till */
/* we find values of x not changing and z changes                */

  do{
        temp_X=X;temp_Y=Y;temp_Z=Z;
        fscanf(f1ptr,"%f %f %f",&X,&Y,&Z);  pt++;

        if ( fabs( temp_Z-Z) > Z_limit )  store=1;
        if (fabs( temp_X-X) < X_max_limit) store=1;
    } while( store==0);

        if (store==1){
                /* storing end-points */
```

```
                parallel_X[Rec_n].segment[Seg_n].end_pt.X=temp_X;
                parallel_X[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
                parallel_X[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
                if(pt>1)
parallel_X[Rec_n].segment[Seg_n].n_pts_bet=pt-2;
                pt=0;
                Seg_n++;
                /* assigning previous points as starting point of next
segment */

                prev_X=X;
                prev_Y=Y;
                prev_Z=Z;

                parallel_X[Rec_n].segment[Seg_n].start_pt.X=prev_X;
                parallel_X[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
                parallel_X[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;

                store=0;
            }

            else {
                temp_X=X;
                temp_Y=Y;
                temp_Z=Z;
            }
        }

        if( (fabs(prev_X-X)> X_max_limit) && ( fabs( prev_Z-Z) > Z_limit )){

            parallel_X[Rec_n].segment[Seg_n].end_pt.X=temp_X;
            parallel_X[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
            parallel_X[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
            if(pt>1) parallel_X[Rec_n].segment[Seg_n].n_pts_bet=pt-2;
            pt=0;
            Seg_n++;
            /* assigning previous points as starting point of next segment
*/          prev_X=X;
            prev_Y=Y;
            prev_Z=Z;

            parallel_X[Rec_n].segment[Seg_n].start_pt.X=prev_X;
            parallel_X[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
            parallel_X[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;
```

```
        }

temp_X=X;
temp_Y=Y;
temp_Z=Z;
printf(" Rec_n %d \n",Rec_n);
}while(Rec_n <= In_Rec_n);

   fclose(f1ptr);



for ( j=1; j<=In_Rec_n; j++)
        for (k=1; k<=parallel_X[j].SEG_N; k++)
        {
                printf(" \nFor Record_%d Segment_%d \nSt_pt X %2.3f Z %2.3f
\nEnd_pt X %2.3f  Z %2.3f \n pts %d\n",j,k,

parallel_X[j].segment[k].start_pt.X,parallel_X[j].segment[k].start_pt.Z,

parallel_X[j].segment[k].end_pt.X,parallel_X[j].segment[k].end_pt.Z,parallel_X[
j].segment[k].n_pts_bet);
        }
for ( p=1; p<=In_Rec_n; p++)
    {
    fprintf(f2ptr,"%d\n",parallel_X[p].SEG_N);
        for (q=1; q<=parallel_X[p].SEG_N; q++)
        {
                fprintf(f2ptr,"%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",

parallel_X[p].segment[q].start_pt.X,parallel_X[p].segment[q].start_pt.Y,parallel
_X[p].segment[q].start_pt.Z,

parallel_X[p].segment[q].end_pt.X,parallel_X[p].segment[q].end_pt.Y,parallel_
X[p].segment[q].end_pt.Z);
        } }
 fclose(f2ptr);
}
```

APPENDIX B

REDF: REDUCED DATA FILE - PARALLEL TO X AXIS

REDF

6
25.012 0.000 0.065
25.309 0.000 25.053
30.097 0.000 25.053
75.067 0.000 25.053
80.082 0.000 15.087
125.133 0.000 15.087
125.133 0.000 20.110
125.133 0.000 25.133
130.146 0.000 25.133
175.171 0.000 25.133
175.171 0.000 20.222
175.171 0.000 0.174
6
25.042 5.000 0.073
25.286 5.000 25.014
30.048 5.000 25.014
75.007 5.000 25.014
80.060 5.000 15.085
125.099 5.000 15.085
125.099 5.000 20.130
125.099 5.004 25.104
130.181 5.000 25.104
175.137 5.000 25.104
175.137 5.000 20.127
175.137 5.000 0.163
6
25.010 10.000 0.017
25.237 10.000 25.093
30.037 10.000 25.093
75.086 10.000 25.093
80.146 10.000 15.166
125.150 10.000 15.166
125.150 10.000 20.247
125.150 10.000 25.172

130.219 10.000 25.172
175.234 10.000 25.172
175.234 10.000 20.219
175.234 10.000 0.180
6
25.070 15.000 0.047
25.203 15.000 25.013
30.060 15.000 25.013
75.100 15.000 25.013
80.187 15.000 15.105
125.145 15.000 15.105
125.145 15.000 20.129
125.145 15.000 25.201
130.244 15.000 25.201
175.202 15.000 25.201
175.202 15.000 20.260
175.202 15.000 0.204
6
25.029 20.000 0.090
25.222 20.000 25.096
30.092 20.000 25.096
75.042 20.000 25.096
80.134 20.000 15.164
125.139 20.000 15.164
125.139 20.000 20.170
125.139 20.000 25.218
130.238 20.000 25.218
175.230 20.000 25.218
175.230 20.000 20.275
175.230 20.000 0.289
6
25.076 25.000 0.078
25.378 25.000 25.056
30.005 25.000 25.056
75.073 25.000 25.056
80.084 25.000 15.070
125.149 25.000 15.070
125.149 25.000 20.125
125.149 25.000 25.140
130.212 25.000 25.140
175.222 25.000 25.140
175.222 25.000 20.211
175.222 25.000 0.216
6

25.023 30.000 0.055
25.262 30.000 25.052
30.087 30.000 25.052
75.048 30.000 25.052
80.142 30.000 15.083
125.106 30.000 15.083
125.106 30.000 20.177
125.106 30.000 25.141
130.126 30.000 25.141
175.175 30.000 25.141
175.175 30.000 20.229
175.175 30.000 0.175
6
25.059 35.000 0.046
25.241 35.000 25.055
30.021 35.000 25.055
75.079 35.000 25.055
80.109 35.000 15.067
125.171 35.000 15.067
125.171 35.000 20.158
125.171 35.000 25.118
130.211 35.000 25.118
175.204 35.000 25.118
175.204 35.000 20.176
175.204 35.000 0.201
6
25.096 40.000 0.050
25.279 40.000 25.031
30.048 40.000 25.031
75.053 40.000 25.031
80.106 40.000 15.042
125.091 40.000 15.042
125.091 40.000 20.080
125.091 40.000 25.096
130.175 40.000 25.096
175.141 40.000 25.096
175.141 40.000 20.112
175.141 40.000 0.184
6
25.014 45.000 0.073
25.295 45.000 25.092
30.089 45.000 25.092
75.008 45.000 25.092
80.064 45.000 15.171

125.036 45.000 15.171
125.036 45.000 20.181
125.036 45.000 25.224
130.079 45.000 25.224
175.037 45.000 25.224
175.037 45.000 20.302
175.037 45.000 0.236
6
25.077 50.000 0.092
25.343 50.000 25.064
30.093 50.000 25.064
75.017 50.000 25.064
80.026 50.000 15.135
125.077 50.000 15.135
125.077 50.000 20.203
125.077 50.000 25.212
130.096 50.000 25.212
175.153 50.000 25.212
175.153 50.000 20.228
175.153 50.000 0.230
6
25.053 55.000 0.006
25.378 55.000 25.026
30.066 55.000 25.026
75.030 55.000 25.026
80.056 55.000 15.093
125.054 55.000 15.093
125.054 55.000 20.121
125.054 55.000 25.103
130.083 55.000 25.103
175.066 55.000 25.103
175.066 55.000 20.127
175.066 55.000 0.114
6
25.055 60.000 0.018
25.231 60.000 25.026
30.033 60.000 25.026
75.089 60.000 25.026
80.136 60.000 15.095
125.172 60.000 15.095
125.172 60.000 20.165
125.172 60.000 25.133
130.233 60.000 25.133
175.202 60.000 25.133

175.202 60.000 20.163
175.202 60.000 0.162
6
25.090 65.000 0.024
25.354 65.000 25.007
30.091 65.000 25.007
75.049 65.000 25.007
80.077 65.000 15.040
125.125 65.000 15.040
125.125 65.000 20.052
125.125 65.000 25.123
130.154 65.000 25.123
175.170 65.000 25.123
175.170 65.000 20.190
175.170 65.000 0.204
6
25.039 70.000 0.070
25.497 70.000 25.024
30.082 70.000 25.024
75.052 70.000 25.024
80.056 70.000 15.026
125.063 70.000 15.026
125.063 70.000 20.070
125.063 70.000 25.058
130.141 70.000 25.058
175.137 70.000 25.058
175.137 70.000 20.151
175.137 70.000 0.071
6
25.080 75.000 0.059
25.337 75.000 25.080
30.022 75.000 25.080
75.054 75.000 25.080
80.065 75.000 15.093
125.101 75.000 15.093
125.101 75.000 20.160
125.101 75.000 25.141
130.133 75.000 25.141
175.160 75.000 25.141
175.160 75.000 20.240
175.160 75.000 0.216
6
25.013 80.000 0.063
25.281 80.000 25.084

30.009 80.000 25.084
75.048 80.000 25.084
80.105 80.000 15.087
125.077 80.000 15.087
125.077 80.000 20.109
125.077 80.000 25.180
130.117 80.000 25.180
175.168 80.000 25.180
175.168 80.000 20.228
175.168 80.000 0.209
6
25.041 85.000 0.089
25.304 85.000 25.094
30.071 85.000 25.094
75.016 85.000 25.094
80.044 85.000 15.187
125.024 85.000 15.187
125.024 85.000 20.281
125.024 85.000 25.263
130.030 85.000 25.263
175.107 85.000 25.263
175.107 85.000 20.265
175.107 85.000 0.352
6
25.014 90.000 0.019
25.351 90.000 25.021
30.008 90.000 25.021
75.090 90.000 25.021
80.106 90.000 15.071
125.142 90.000 15.071
125.142 90.000 20.143
125.142 90.000 25.139
130.220 90.000 25.139
175.213 90.000 25.139
175.213 90.000 20.199
175.213 90.000 0.204

APPENDIX C

DATA REDUCTION STAGE - PARALLEL TO Y-AXIS

DATA REDUCTION STAGE - PARALLEL TO Y-AXIS

```
/* DATA REDUCTION STAGE */
/* PARALLEL TO Y AXIS */
/* Function Prototype *******/


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>
#define Y_limit 0.5
#define Y_max_limit 2.5
#define X_max_limit 2.5
#define Z_limit 1.0  /* TO DETECT CHANGE IN Z*/

/*  Global Variable Declaration  */
int In_Rec_n=1,Rec_n=1,Seg_n=1,n_pts_bet=1,check_z=0,store=0,pt=0;

float prev_X,prev_Y,prev_Z;
float X,Y,Z,int_X, int_Y,int_Z;
float temp_X,temp_Y,temp_Z;

FILE *f1ptr, *f2ptr;


struct XYZ
{
  float X;
  float Y;
  float Z;
};

struct Segment
{
        struct XYZ start_pt;
        struct XYZ end_pt;
        int n_pts_bet;
};
```

```
/*structure for each line parallel to Y */
struct RecordY
{
 struct  Segment segment[10];
 int SEG_N,TSEG_N;
}parallel_Y[31];



main()
{
int j,k,p,q;
char name1[35], name2[35];

printf("Enter filename(data file) \n");
scanf("%s", name1);

printf("Enter name of reduced data file \n");
scanf("%s",name2);

printf("Enter number of Lines Scanned on the object (parallel to Y axis)\n");
scanf("%d",&In_Rec_n);

if((f1ptr = fopen(name1,"r")) ==NULL)

   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }

if((f2ptr = fopen(name2, "w+")) ==NULL)

   {
   printf("Can't write to file %s \n", name2);
   exit(1);
   }


/* READS IN FIRST POINT IN THE FILE */
  fscanf(f1ptr,"%f %f %f",&prev_X,&prev_Y,&prev_Z);

  parallel_Y[Rec_n].segment[Seg_n].start_pt.X=prev_X;
  parallel_Y[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
  parallel_Y[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;
```

```
/* For scanning parallel to Y axis */

do{
        fscanf(f1ptr,"%f %f %f", &X,&Y,&Z);  pt++;

        if( fabs(prev_X - X ) >  X_max_limit ){

                parallel_Y[Rec_n].segment[Seg_n].end_pt.X=temp_X;
                parallel_Y[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
                parallel_Y[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
                parallel_Y[Rec_n].SEG_N=Seg_n;
                if(pt>1)        parallel_Y[Rec_n].segment[Seg_n].n_pts_bet=pt-2;

                Rec_n++;    Seg_n=1;     store=0;      pt=0;
                check_z=0;
                prev_X=X;
                prev_Y=Y;
                prev_Z=Z;
               parallel_Y[Rec_n].segment[Seg_n].start_pt.X=prev_X;
                parallel_Y[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
                parallel_Y[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;

                fscanf(f1ptr,"%f %f %f", &X,&Y,&Z);pt++;

        }

        if( fabs(prev_Y-Y) < Y_limit ){
                temp_X=X;
                temp_Y=Y;
                temp_Z=Z;

        }

        if (fabs (prev_Y - Y) > Y_max_limit)
        {

                parallel_Y[Rec_n].segment[Seg_n].end_pt.X=temp_X;
                parallel_Y[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
                parallel_Y[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
                if(pt>1)
                parallel_Y[Rec_n].segment[Seg_n].n_pts_bet=pt-2;
                pt=0;
                Seg_n++;
```

```
            /*fscanf(f1ptr, "%f %f %f", &prev_X, &prev_Y, &prev_Z); pt++;*/
            prev_X =X;
            prev_Y =Y;
            prev_Z =Z;

            parallel_Y[Rec_n].segment[Seg_n].start_pt.X=prev_X;
            parallel_Y[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
            parallel_Y[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;


        /* since z values are not exceeding the limit we will scan till */
        /* Find values of x not changing and z changes                  */

          do{
              temp_X = X; temp_Y = Y; temp_Z =Z;
               fscanf(f1ptr,"%f %f %f",&X,&Y,&Z);  pt++;

               if ( fabs( prev_Z - Z) > Z_limit )  store=1;
               if ( fabs(prev_Y - Y) < Y_limit) store = 1;

           } while( store==0);

               if (store==1){
                       /* storing end-points */

                       parallel_Y[Rec_n].segment[Seg_n].end_pt.X=temp_X;
                       parallel_Y[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
                       parallel_Y[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
                       if(pt>1)
parallel_Y[Rec_n].segment[Seg_n].n_pts_bet=pt-2;
                       pt=0;
                       Seg_n++;
                       /* assigning previous points as starting point of next
segment */

                       /*fscanf(f1ptr,  "%f  %f  %f",  &prev_X,  &prev_Y,
                       &prev_Z);pt++;*/

                       prev_X=X;
                       prev_Y=Y;
                       prev_Z=Z;
                       parallel_Y[Rec_n].segment[Seg_n].start_pt.X=prev_X;
                       parallel_Y[Rec_n].segment[Seg_n].start_pt.Y=prev_Y;
                       parallel_Y[Rec_n].segment[Seg_n].start_pt.Z=prev_Z;
```

```
                store=0;
        }

        else {
                temp_X=X;
                temp_Y=Y;
                temp_Z=Z;
        }
}


/*if ((fabs(prev_Y-Y) < Y_limit)&&(check_z ==1))*/ {
        /*
        temp_X =X;
        temp_Y =Y;
        temp_Z =Z;
 */
 }
 /*else*/


        /*
        parallel_Y[Rec_n].segment[Seg_n].end_pt.X=temp_X;
        parallel_Y[Rec_n].segment[Seg_n].end_pt.Y=temp_Y;
        parallel_Y[Rec_n].segment[Seg_n].end_pt.Z=temp_Z;
        if(pt>1) parallel_Y[Rec_n].segment[Seg_n].n_pts_bet=pt-2;
        pt=0;
        Seg_n++;
        */
        /* assigning previous points as starting point of next segment*/
        /*
        prev_X=X;
        prev_Y=Y;
        prev_Z=Z;
        */

temp_X=X;
temp_Y=Y;
temp_Z=Z;

printf(" Rec_n %d \n",Rec_n);
}while(Rec_n <= In_Rec_n);

  fclose(f1ptr);
```

```
for ( j=1; j<=In_Rec_n; j++)
        for (k=1; k<=parallel_Y[j].SEG_N; k++)
        {
                printf(" \nFor Record_%d Segment_%d \nSt_pt X %2.3f Y %2.3f Z
%2.3f \nEnd_pt X %2.3f  Y %2.3f Z %2.3f \n pts %d\n",j,k,

paralle    j].segment[k].start_pt.X,parallel_Y[j].segment[k].start_pt.Y,parallel_Y
[j].segment[k].start_pt.Z,

parallel_Y[j].segment[k].end_pt.X,parallel_Y[j].segment[k].end_pt.Y,parallel_Y[j
].segment[k].end_pt.Z,parallel_Y[j].segment[k].n_pts_bet);

      }
for ( p=1; p<=In_Rec_n; p++)
    {
    fprintf(f2ptr,"%d\n",parallel_Y[p].SEG_N);
        for (q=1; q<=parallel_Y[p].SEG_N; q++)
        {
                fprintf(f2ptr,"%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",

parallel_Y[p].segment[q].start_pt.X,parallel_Y[p].segment[q].start_pt.Y,parallel
_Y[p].segment[q].start_pt.Z,

parallel_Y[p].segment[q].end_pt.X,parallel_Y[p].segment[q].end_pt.Y,parallel_
Y[p].segment[q].end_pt.Z);

      }    }
 fclose(f2ptr);

}
```

APPENDIX D

REDL: REDUCED DATA FILE - PARALLEL TO Y-AXIS

REDL

3
25.077 0.000 0.087
25.131 0.000 25.097
25.109 5.096 25.191
25.138 90.018 25.130
25.195 90.099 20.154
25.182 90.028 0.020
3
30.118 0.000 0.091
30.141 0.000 25.069
30.165 5.038 25.092
30.118 90.084 25.116
30.157 90.178 20.125
30.127 90.151 0.012
3
35.072 0.000 0.042
35.161 0.000 25.092
35.184 5.032 25.192
35.188 90.077 25.117
35.199 90.099 20.182
35.201 90.119 0.012
3
40.127 0.000 0.048
40.152 0.000 25.009
40.154 5.031 25.083
40.183 90.044 25.026
40.243 90.077 20.097
40.241 90.092 0.038
3
45.048 0.000 0.094
45.159 0.000 25.094
45.160 5.019 25.158
45.153 90.040 25.124
45.218 90.073 20.209
45.249 90.091 0.021
3
50.126 0.000 0.081

50.146 0.000 25.035
50.169 5.074 25.051
50.218 90.078 25.068
50.228 90.141 20.160
50.282 90.102 0.092
3
55.038 0.000 0.054
55.048 0.000 25.001
55.105 5.063 25.057
55.121 90.012 25.048
55.139 90.065 20.121
55.147 90.077 0.013
3
60.081 0.000 0.076
60.064 0.000 25.069
60.111 5.082 25.106
60.115 90.076 25.151
60.197 90.157 20.196
60.175 90.158 0.015
3
65.072 0.000 0.030
65.079 0.000 25.012
65.125 5.054 25.091
65.219 90.017 25.110
65.318 90.041 20.146
65.261 90.020 0.016
3
70.043 0.000 0.033
70.035 0.000 25.078
70.006 5.056 25.139
70.166 90.081 25.167
70.190 90.112 20.249
70.257 90.148 0.023
3
75.143 0.000 0.017
75.178 0.000 25.018
75.113 5.030 25.025
75.127 90.049 25.036
75.144 90.098 20.094
75.135 90.143 0.047
3
80.081 0.009 0.058
80.164 0.003 15.028
80.171 5.092 15.118

80.206 90.068 15.040
80.208 90.108 10.112
80.294 90.118 0.063
3
85.696 0.030 0.030
85.785 0.041 15.082
85.848 5.093 15.094
85.847 90.075 15.108
85.831 90.115 10.135
85.856 90.141 0.042
3
90.901 0.014 0.023
90.107 0.083 15.036
90.172 5.059 15.006
90.216 90.142 15.077
90.223 90.135 10.133
90.252 90.202 0.016
3
95.604 0.076 0.048
95.611 0.014 15.051
95.628 5.110 15.092
95.642 90.110 15.137
95.702 90.184 10.209
95.655 90.195 0.018
3
100.132 0.020 0.020
100.076 0.041 15.021
100.165 5.153 15.091
100.117 90.171 15.086
100.150 90.260 10.087
100.177 90.203 0.052
3
105.598 0.031 0.030
105.600 0.030 10.065
105.556 5.045 15.003
105.624 90.118 15.018
105.562 90.096 10.039
105.612 90.104 0.050
3
110.658 0.031 0.031
110.097 0.071 15.098
110.075 5.085 15.031
110.093 90.139 15.068
110.128 90.117 10.100

110.146 90.216 0.042
3
115.165 0.032 0.032
115.642 0.030 15.040
115.609 5.015 15.071
115.657 90.026 15.134
115.632 90.019 10.105
115.707 90.108 0.032
3
120.692 0.031 0.031
120.081 0.040 15.003
120.097 5.037 15.032
120.188 90.072 15.106
120.104 90.090 10.113
120.134 90.142 0.193
3
125.104 0.000 0.013
125.200 0.000 25.002
125.183 5.076 25.045
125.156 90.021 25.007
125.253 90.090 20.036
125.251 90.094 0.067
3
130.076 0.000 0.050
130.050 0.000 25.040
130.048 5.031 25.095
130.121 90.018 25.047
130.145 90.078 20.054
130.155 90.018 0.107
3
135.173 0.000 0.017
135.121 0.000 25.019
135.129 5.084 25.042
135.187 90.003 25.087
135.210 90.095 20.121
135.206 90.077 0.155
3
140.078 0.000 0.054
140.125 0.000 25.086
140.118 5.082 25.162
140.134 90.021 25.114
140.182 90.058 20.189
140.188 90.040 0.041
3

145.058 0.000 0.083
145.143 0.000 25.061
145.205 5.094 25.147
145.202 90.085 25.112
145.244 90.143 20.139
145.266 90.115 0.031
3
150.102 0.000 0.003
150.156 0.000 25.024
150.230 5.074 25.042
150.215 90.006 25.111
150.310 90.101 20.194
150.291 90.036 0.031
3
155.062 0.000 0.085
155.049 0.000 25.007
155.100 5.021 25.027
155.176 90.026 25.034
155.207 90.051 20.053
155.232 90.076 0.086
3
160.105 0.000 0.037
160.077 0.000 25.003
160.123 5.043 25.059
160.202 90.091 25.038
160.273 90.142 20.090
160.227 90.177 0.031
3
165.068 0.000 0.045
165.059 0.000 25.015
165.121 5.003 25.056
165.125 90.030 25.061
165.178 90.109 20.151
165.195 90.115 0.032
3
170.130 0.000 0.017
170.157 0.000 25.099
170.119 5.004 25.123
170.101 90.044 25.176
170.198 90.119 20.233
170.193 90.092 0.042
3
175.111 0.000 0.085
175.112 0.000 25.081

175.098 5.025 25.110
175.224 90.091 25.163
175.294 90.108 20.218
175.237 90.180 0.042

APPENDIX E

SURFACE RECOGNITION STAGE - PARALLEL TO X-AXIS

SURFACE RECOGNITION STAGE - PARALLEL TO X-AXIS

```
/* SURFACE RECOGNITION STAGE */

/*PARALLEL TO X AXIS */

/* Function Prototype */


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>
#define Z_limit 0.5  /* TO DETECT CHANGE IN Z*/

/*  Global Variable Declaration  */
int In_Rec_n=1,Rec_n=1,Seg_n=1,n_pts_bet=1,pt=0;
float prev_X,prev_Y,prev_Z;

float X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,X4,Y4,Z4;

float temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2;

f                    l                    o                    a                    t
SLOPE_X1,SLOPE_Y1,SLOPE_Z1,SLOPE_Z2,SLOPE_X2,SLOPE_Y2,SLOPE_1,SL
OPE_2;

FILE *f1ptr, *f2ptr;


struct XYZ
{
  float X;
  float Y;
  float Z;
};
```

```
struct Segment
{
        struct XYZ start_pt;
        struct XYZ end_pt;
        int n_pts_bet;
};

/*structure for each line */

struct RecordX
{
 struct  Segment segment[10];
 int SEG_N;
 int SURF_N;
}parallel_X[21];


struct POINT
{
float X;
float Y;
float Z;
};

/* Structure for each surface */

struct  SURFY_XYZ
{
 float X;
 float Y;
 float Z;
};


struct SURFACE_X
{
 struct SURFY_XYZ point_1;
 struct SURFY_XYZ point_2;
 struct SURFY_XYZ point_3;
 struct SURFY_XYZ point_4;
 int SURF_N;
}SURF_X[15];

main()
```

```
{
int j=1,k,p,q;
char name1[35], name2[35];
int surf_n=1;

printf("Enter filename (reduced data file for parallel to X axis) \n");
scanf("%s", name1);
printf("Enter number of lines Scanned on the object (parallel to  X axis)\n");
scanf("%d",&In_Rec_n);
In_Rec_n++;


printf("Enter name of surface feature file \n");
scanf("%s",name2);


if((f1ptr = fopen(name1, "r")) ==NULL)
   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }
if((f2ptr = fopen(name2, "w+")) ==NULL)
   {
   printf("Can't write to file %s \n", name2);
   exit(1);
   }


/* READS POINTS IN REDUCED DATA FILE TO STRUCTURES */
for(p=1;p<In_Rec_n;p++)
   {
   fscanf(f1ptr,"%d",&parallel_X[p].SEG_N);
        for (q=1; q<=parallel_X[p].SEG_N;q++)
        {
        fscanf(f1ptr,"%f %f %f %f %f %f",
        &prev_X,&prev_Y,&prev_Z,&X,&Y,&Z);



parallel_X[p].segment[q].start_pt.X=prev_X;parallel_X[p].segment[q].start_pt.
Y=prev_Y;

parallel_X[p].segment[q].start_pt.Z=prev_Z;parallel_X[p].segment[q].end_pt.X
=X;
```

```
parallel_X[p].segment[q].end_pt.Y=Y;parallel_X[p].segment[q].end_pt.Z=Z;
      }
   }
   fclose(f1ptr);


/* PRINTS DATA JUST READ IN ON THE SCREEN */

/*

for(p=1;p<In_Rec_n;p++)
   {
        printf("%d\n",parallel_X[p].SEG_N);
        for (q=1; q<=parallel_X[p].SEG_N;q++)
                {
                printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",

parallel_X[p].segment[q].start_pt.X,parallel_X[p].segment[q].start_pt.Y,

parallel_X[p].segment[q].start_pt.Z,parallel_X[p].segment[q].end_pt.X,

parallel_X[p].segment[q].end_pt.Y,parallel_X[p].segment[q].end_pt.Z);
                }

    }
*/
printf("%d\n",parallel_X[p-1].SEG_N);

      q=1;
      {
      p=1;
      X1 = parallel_X[p].segment[q].start_pt.X;
      Y1 = parallel_X[p].segment[q].start_pt.Y;
      Z1 = parallel_X[p].segment[q].start_pt.Z;
      X2 = parallel_X[p].segment[q].end_pt.X;
      Y2 = parallel_X[p].segment[q].end_pt.Y;
      Z2 = parallel_X[p].segment[q].end_pt.Z;


      printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);


      SURF_X[j].point_1.X = X1;
```

```
    SURF_X[j].point_1.Y = Y1;
    SURF_X[j].point_1.Z = Z1;
    SURF_X[j].point_2.X = X2;
    SURF_X[j].point_2.Y = Y2;
    SURF_X[j].point_2.Z = Z2;


   for(p=2;p<In_Rec_n;p++)
     {
     q=1;

     SLOPE_Y1 = fabs(Y2-Y1);
     SLOPE_Z1 = fabs(Z2-Z1);


/* Comparing points of q segment of the p+1 record */

    X3 = parallel_X[p].segment[q].start_pt.X;
    Y3 = parallel_X[p].segment[q].start_pt.Y;
    Z3 = parallel_X[p].segment[q].start_pt.Z;
    X4 = parallel_X[p].segment[q].end_pt.X;
    Y4 = parallel_X[p].segment[q].end_pt.Y;
    Z4 = parallel_X[p].segment[q].end_pt.Z;


    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);


    SLOPE_Y2 = fabs(Y4-Y3);
    SLOPE_Z2 = fabs(Z4-Z3);

   if (SLOPE_Z1==0)
     {
     SLOPE_1=0;
     }
   else
        {
        SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
        }

 if (SLOPE_Z2==0)
     {
     SLOPE_2=0;
     }
```

```
else
      {
      SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
      }


if (SLOPE_Y1==0)
   {
   SLOPE_1=0;
   }
else
      {
      SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
      }

if (SLOPE_Y2==0)
      {
      SLOPE_2=0;
      }
else
    {
    SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
    }


if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
      {
      temp_X1 = X3;
      temp_Y1 = Y3;
      temp_Z1 = Z3;
      temp_X2 = X4;
      temp_Y2 = Y4;
      temp_Z2 = Z4;
      }

   printf("%2.3f %2.3f %2.3f\n%2.3f
%2.3f%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
   printf("p is %d  In rec is %d\n\n",p,In_Rec_n);


 if ( p == (In_Rec_n)-1)

      {
      SURF_X[j].point_3.X = temp_X1;
```

```
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z = temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        }

 if ((fabs(SLOPE_1-SLOPE_2)) > 5)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z =temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        }
        }

}

/*printf(" End of q=1  \n");*/

     q=2;
     {
     p=1;
     X1 = parallel_X[p].segment[q].start_pt.X;
     Y1 = parallel_X[p].segment[q].start_pt.Y;
     Z1 = parallel_X[p].segment[q].start_pt.Z;
     X2 = parallel_X[p].segment[q].end_pt.X;
     Y2 = parallel_X[p].segment[q].end_pt.Y;
     Z2 = parallel_X[p].segment[q].end_pt.Z;


     printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);
```

```
    SURF_X[j].SURF_N=surf_n;
    SURF_X[j].point_1.X = X1;
    SURF_X[j].point_1.Y = Y1;
    SURF_X[j].point_1.Z = Z1;
    SURF_X[j].point_2.X = X2;
    SURF_X[j].point_2.Y = Y2;
    SURF_X[j].point_2.Z = Z2;

  for(p=2;p<In_Rec_n;p++)
    {
    q=2;
    SLOPE_X1 = fabs(X2-X1);
    SLOPE_Y1 = fabs(Y2-Y1);


/* Comparing points of q segment of the p+1 record */

    X3 = parallel_X[p].segment[q].start_pt.X;
    Y3 = parallel_X[p].segment[q].start_pt.Y;
    Z3 = parallel_X[p].segment[q].start_pt.Z;
    X4 = parallel_X[p].segment[q].end_pt.X;
    Y4 = parallel_X[p].segment[q].end_pt.Y;
    Z4 = parallel_X[p].segment[q].end_pt.Z;


    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);


    SLOPE_X2 = fabs(X4-X3);
    SLOPE_Y2 = fabs(Y4-Y3);

  if (SLOPE_Y1==0)
    {
    SLOPE_1=0;
    }
  else
      {
      SLOPE_1 = SLOPE_X1/SLOPE_Y1;
      }
/*
 if ((SLOPE_X1==0)
    {
    SLOPE_1=0;
    }
```

```
   else
        {
        SLOPE_1 = SLOPE_X1/SLOPE_Y1;
        }

*/

 if (SLOPE_Y2==0)
     {
     SLOPE_2=0;
     }
   else
        {
        SLOPE_2 = SLOPE_X2/SLOPE_Y2;
        }

/*

   if (SLOPE_X2==0)
        {
        SLOPE_2=0;
        }
   else
      {
      SLOPE_2 = SLOPE_X2/SLOPE_Y2;
      }
*/

   if (fabs(SLOPE_1-SLOPE_2)<= 0.5)
        {
        temp_X1 = X3;
        temp_Y1 = Y3;
        temp_Z1 = Z3;
        temp_X2 = X4;
        temp_Y2 = Y4;
        temp_Z2 = Z4;
        }

        printf("%2.3f   %2.3f   %2.3f\n%2.3f   %2.3f
%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
    printf("p is %d  In rec is %d\n\n",p,In_Rec_n);
```

```
if ( p == (In_Rec_n)-1)

      {
      SURF_X[j].point_3.X = temp_X1;
      SURF_X[j].point_3.Y = temp_Y1;
      SURF_X[j].point_3.Z = temp_Z1;
      SURF_X[j].point_4.X = temp_X2;
      SURF_X[j].point_4.Y = temp_Y2;
      SURF_X[j].point_4.Z = temp_Z2;
      SURF_X[j].SURF_N=surf_n;
      j++;
      surf_n ++;
      }

if (fabs(SLOPE_1-SLOPE_2)> 5)
     {
      SURF_X[j].point_3.X = temp_X1;
      SURF_X[j].point_3.Y = temp_Y1;
      SURF_X[j].point_3.Z = temp_Z1;
      SURF_X[j].point_4.X = temp_X2;
      SURF_X[j].point_4.Y = temp_Y2;
      SURF_X[j].point_4.Z =temp_Z2;
      SURF_X[j].SURF_N=surf_n;
      j++;
      surf_n ++;
      SURF_X[j].point_1.X = X3;
      SURF_X[j].point_1.Y = Y3;
      SURF_X[j].point_1.Z = Z3;
      SURF_X[j].point_2.X = X4;
      SURF_X[j].point_2.Y = Y4;
      SURF_X[j].point_2.Z = Z4;
      SURF_X[j].SURF_N=surf_n;
      }
    }
    }


    /*printf(" End of q=2  \n");*/
    q=3;
    {
    p=1;
    X1 = parallel_X[p].segment[q].start_pt.X;
    Y1 = parallel_X[p].segment[q].start_pt.Y;
    Z1 = parallel_X[p].segment[q].start_pt.Z;
```

```
X2 = parallel_X[p].segment[q].end_pt.X;
Y2 = parallel_X[p].segment[q].end_pt.Y;
Z2 = parallel_X[p].segment[q].end_pt.Z;


printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);


SURF_X[j].SURF_N=surf_n;
SURF_X[j].point_1.X = X1;
SURF_X[j].point_1.Y = Y1;
SURF_X[j].point_1.Z = Z1;
SURF_X[j].point_2.X = X2;
SURF_X[j].point_2.Y = Y2;
SURF_X[j].point_2.Z = Z2;


for(p=2;p<In_Rec_n;p++)
  {
  q=3;
  SLOPE_X1 = fabs(X2-X1);
  SLOPE_Y1 = fabs(Y2-Y1);


/* Comparing points of q segment of the p+1 record */

  X3 = parallel_X[p].segment[q].start_pt.X;
  Y3 = parallel_X[p].segment[q].start_pt.Y;
  Z3 = parallel_X[p].segment[q].start_pt.Z;
  X4 = parallel_X[p].segment[q].end_pt.X;
  Y4 = parallel_X[p].segment[q].end_pt.Y;
  Z4 = parallel_X[p].segment[q].end_pt.Z;


  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);


  SLOPE_X2 = fabs(X4-X3);
  SLOPE_Y2 = fabs(Y4-Y3);

 if (SLOPE_Y1==0)
   {
   SLOPE_1=0;
   }
```

```
  else
       {
       SLOPE_1 = SLOPE_X1/SLOPE_Y1;
       }

if (SLOPE_Y2==0)
     {
     SLOPE_2=0;
     }
  else
       {
       SLOPE_2 = SLOPE_X2/SLOPE_Y2;
       }

  /*
  if (SLOPE_X1==0)
     {
     SLOPE_1=0;
     }
  else
       {
       SLOPE_1 = SLOPE_X1/SLOPE_Y1;
       }

  */
  /*
  if (SLOPE_X2==0)
       {
       SLOPE_2=0;
       }
  else
      {
      SLOPE_2 = SLOPE_X2/SLOPE_Y2;
      }
   */

  if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
       {
       temp_X1 = X3;
       temp_Y1 = Y3;
       temp_Z1 = Z3;
       temp_X2 = X4;
       temp_Y2 = Y4;
       temp_Z2 = Z4;
```

```
        }

    printf("%2.3f  %2.3f  %2.3f\n%2.3f  %2.3f  %2.3f\n,
temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
    printf("p is %d  In rec is %d\n\n",p,In_Rec_n);



 if ( p == (In_Rec_n)-1)

        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z = temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        }

 if ((fabs(SLOPE_1-SLOPE_2)) > 1)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z =temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        SURF_X[j].point_1.X = X3;
        SURF_X[j].point_1.Y = Y3;
        SURF_X[j].point_1.Z = Z3;
        SURF_X[j].point_2.X = X4;
        SURF_X[j].point_2.Y = Y4;
        SURF_X[j].point_2.Z = Z4;
        SURF_X[j].SURF_N=surf_n;
        }
        }
    }
```

```
 /*printf(" End of q=3  \n");*/

q=4;
  {
  p=1;
  X1 = parallel_X[p].segment[q].start_pt.X;
  Y1 = parallel_X[p].segment[q].start_pt.Y;
  Z1 = parallel_X[p].segment[q].start_pt.Z;
  X2 = parallel_X[p].segment[q].end_pt.X;
  Y2 = parallel_X[p].segment[q].end_pt.Y;
  Z2 = parallel_X[p].segment[q].end_pt.Z;


  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);


  SURF_X[j].point_1.X = X1;
  SURF_X[j].point_1.Y = Y1;
  SURF_X[j].point_1.Z = Z1;
  SURF_X[j].point_2.X = X2;
  SURF_X[j].point_2.Y = Y2;
  SURF_X[j].point_2.Z = Z2;


for(p=2;p<In_Rec_n;p++)
  {
  q=4;
  SLOPE_Y1 = fabs(Y2-Y1);
  SLOPE_Z1 = fabs(Z2-Z1);


/* Comparing points of q segment of the p+1 record */

  X3 = parallel_X[p].segment[q].start_pt.X;
  Y3 = parallel_X[p].segment[q].start_pt.Y;
  Z3 = parallel_X[p].segment[q].start_pt.Z;
  X4 = parallel_X[p].segment[q].end_pt.X;
  Y4 = parallel_X[p].segment[q].end_pt.Y;
  Z4 = parallel_X[p].segment[q].end_pt.Z;


  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
  SLOPE_Y2 = fabs(Y4-Y3);
  SLOPE_Z2 = fabs(Z4-Z3);
```

```
  if (SLOPE_Z1==0)
     {
     SLOPE_1=0;
     }
  else
        {
        SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
        }

if (SLOPE_Z2==0)
     {
     SLOPE_2=0;
     }
  else
        {
        SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
        }

  /*
  if (SLOPE_Y1==0)
     {
     SLOPE_1=0;
     }
  else
        {
        SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
        }



if (SLOPE_Y2==0)
        {
        SLOPE_2=0;
        }
  else
       {
       SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
       }
 */

  if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
        {
        temp_X1 = X3;
        temp_Y1 = Y3;
```

```
        temp_Z1 = Z3;
        temp_X2 = X4;
        temp_Y2 = Y4;
        temp_Z2 = Z4;
        }


        printf("%2.3f   %2.3f   %2.3f\n%2.3f   %2.3f
%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
    printf("p is %d  In rec is %d\n\n",p,In_Rec_n);


 if ( p == (In_Rec_n)-1)

        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z = temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n++;
        }

 if ((fabs(SLOPE_1-SLOPE_2)) > 5)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z =temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        SURF_X[j].point_3.X = X3;
        SURF_X[j].point_3.Y = Y3;
        SURF_X[j].point_3.Z = Z3;
        SURF_X[j].point_4.X = X4;
        SURF_X[j].point_4.Y = Y4;
        SURF_X[j].point_4.Z = Z4;
        SURF_X[j].SURF_N=surf_n;
```

```
    }
    }
  }
  /*printf(" End of q=4 \n");*/
    q=5;
    {
    p=1;
    X1 = parallel_X[p].segment[q].start_pt.X;
    Y1 = parallel_X[p].segment[q].start_pt.Y;
    Z1 = parallel_X[p].segment[q].start_pt.Z;
    X2 = parallel_X[p].segment[q].end_pt.X;
    Y2 = parallel_X[p].segment[q].end_pt.Y;
    Z2 = parallel_X[p].segment[q].end_pt.Z;


    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);


    SURF_X[j].SURF_N=surf_n;
    SURF_X[j].point_1.X = X1;
    SURF_X[j].point_1.Y = Y1;
    SURF_X[j].point_1.Z = Z1;
    SURF_X[j].point_2.X = X2;
    SURF_X[j].point_2.Y = Y2;
    SURF_X[j].point_2.Z = Z2;


  for(p=2;p<In_Rec_n;p++)
    {
    q=5;
    SLOPE_X1 = fabs(X2-X1);
    SLOPE_Y1 = fabs(Y2-Y1);


/* Comparing points of q segment of the p+1 record */

  X3 = parallel_X[p].segment[q].start_pt.X;
  Y3 = parallel_X[p].segment[q].start_pt.Y;
  Z3 = parallel_X[p].segment[q].start_pt.Z;
  X4 = parallel_X[p].segment[q].end_pt.X;
  Y4 = parallel_X[p].segment[q].end_pt.Y;
  Z4 = parallel_X[p].segment[q].end_pt.Z;

  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
```

```
 SLOPE_X2 = fabs(X4-X3);
 SLOPE_Y2 = fabs(Y4-Y3);

if (SLOPE_Y1==0)
   {
   SLOPE_1=0;
   }
 else
     {
     SLOPE_1 = SLOPE_X1/SLOPE_Y1;
     }

if (SLOPE_Y2==0)
   {
   SLOPE_2=0;
   }
 else
     {
     SLOPE_2 = SLOPE_X2/SLOPE_Y2;
     }

 /*
 if (SLOPE_X1==0)
   {
   SLOPE_1=0;
   }
 else
     {
     SLOPE_1 = SLOPE_X1/SLOPE_Y1;
     }


 if (SLOPE_X2==0)
     {
     SLOPE_2=0;
     }
 else
    {
    SLOPE_2 = SLOPE_X2/SLOPE_Y2;
    }

 */

 if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
```

```
        {
        temp_X1 = X3;
        temp_Y1 = Y3;
        temp_Z1 = Z3;
        temp_X2 = X4;
        temp_Y2 = Y4;
        temp_Z2 = Z4;
        }

        printf("%2.3f  %2.3f  %2.3f\n%2.3f  %2.3f
%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
    printf("p is %d  In rec is %d\n\n",p,In_Rec_n);


 if ( p == (In_Rec_n)-1)

        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z = temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        }

 if ((fabs(SLOPE_1-SLOPE_2)) > 0.5)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z =temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        SURF_X[j].point_1.X = X3;
        SURF_X[j].point_1.Y = Y3;
        SURF_X[j].point_1.Z = Z3;
        SURF_X[j].point_2.X = X4;
        SURF_X[j].point_2.Y = Y4;
```

```
    SURF_X[j].point_2.Z = Z4;
    SURF_X[j].SURF_N=surf_n;
    }
    }
  }
  /*printf(" End of q=5 \n");*/

  q=6;
  {
  p=1;
  X1 = parallel_X[p].segment[q].start_pt.X;
  Y1 = parallel_X[p].segment[q].start_pt.Y;
  Z1 = parallel_X[p].segment[q].start_pt.Z;
  X2 = parallel_X[p].segment[q].end_pt.X;
  Y2 = parallel_X[p].segment[q].end_pt.Y;
  Z2 = parallel_X[p].segment[q].end_pt.Z;

  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);

  SURF_X[j].point_1.X = X1;
  SURF_X[j].point_1.Y = Y1;
  SURF_X[j].point_1.Z = Z1;
  SURF_X[j].point_2.X = X2;
  SURF_X[j].point_2.Y = Y2;
  SURF_X[j].point_2.Z = Z2;

 for(p=2;p<In_Rec_n;p++)
  {
  q=6;

  SLOPE_Y1 = fabs(Y2-Y1);
  SLOPE_Z1 = fabs(Z2-Z1);


/* Comparing points of q segment of the p+1 record */

  X3 = parallel_X[p].segment[q].start_pt.X;
  Y3 = parallel_X[p].segment[q].start_pt.Y;
  Z3 = parallel_X[p].segment[q].start_pt.Z;
  X4 = parallel_X[p].segment[q].end_pt.X;
  Y4 = parallel_X[p].segment[q].end_pt.Y;
  Z4 = parallel_X[p].segment[q].end_pt.Z;

  printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
```

```
 SLOPE_Y2 = fabs(Y4-Y3);
 SLOPE_Z2 = fabs(Z4-Z3);

if (SLOPE_Z1==0)
   {
   SLOPE_1=0;
   }
 else
      {
      SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
      }

if (SLOPE_Z2==0)
   {
   SLOPE_2=0;
   }
 else
      {
      SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
      }


 if (SLOPE_Y1==0)
   {
   SLOPE_1=0;
   }
 else
      {
      SLOPE_1 = SLOPE_Y1/SLOPE_Z1;
      }


 if (SLOPE_Y2==0)
      {
      SLOPE_2=0;
      }
 else
    {
    SLOPE_2 = SLOPE_Y2/SLOPE_Z2;
    }

 if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
      {
      temp_X1 = X3;
```

```
        temp_Y1 = Y3;
        temp_Z1 = Z3;
        temp_X2 = X4;
        temp_Y2 = Y4;
        temp_Z2 = Z4;
        }

        printf("%2.3f  %2.3f  %2.3f\n%2.3f  %2.3f
%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
    printf("p is %d  In rec is %d\n\n",p,In_Rec_n);


 if ( p == (In_Rec_n)-1)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z = temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        }

 if ((fabs(SLOPE_1-SLOPE_2)) > 5)
        {
        SURF_X[j].point_3.X = temp_X1;
        SURF_X[j].point_3.Y = temp_Y1;
        SURF_X[j].point_3.Z = temp_Z1;
        SURF_X[j].point_4.X = temp_X2;
        SURF_X[j].point_4.Y = temp_Y2;
        SURF_X[j].point_4.Z =temp_Z2;
        SURF_X[j].SURF_N=surf_n;
        }
        }
     }
  /*printf(" End of q=6 \n");*/
   for ( j=1;j<= surf_n;j++)
   {
   printf("\nSURFACE_%d\n",j);

   printf("P1 X %2.3f Y %2.3f Z %2.3f\n",
          SURF_X[j].point_1.X,SURF_X[j].point_1.Y,SURF_X[j].point_1.Z);

   printf("P2 X %2.3f Y %2.3f Z %2.3f\n",
```

```
        SURF_X[j].point_2.X,SURF_X[j].point_2.Y,SURF_X[j].point_2.Z);

    printf("P3 X %2.3f Y %2.3f Z %2.3f\n",
        SURF_X[j].point_3.X,SURF_X[j].point_3.Y,SURF_X[j].point_3.Z);

    printf("P4 X %2.3f Y %2.3f Z %2.3f\n",
        SURF_X[j].point_4.X,SURF_X[j].point_4.Y,SURF_X[j].point_4.Z);
    }

    fprintf(f2ptr,"%d\n",surf_n);
    for ( j=1;j<= surf_n;j++)
    {
    fprintf(f2ptr,"%d\n",j);
    fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
         SURF_X[j].point_1.X,SURF_X[j].point_1.Y,SURF_X[j].point_1.Z);

    fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_X[j].point_2.X,SURF_X[j].point_2.Y,SURF_X[j].point_2.Z);

    fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_X[j].point_3.X,SURF_X[j].point_3.Y,SURF_X[j].point_3.Z);

    fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_X[j].point_4.X,SURF_X[j].point_4.Y,SURF_X[j].point_4.Z);
    }

   fclose(f2ptr);
    /*
    for ( j=1;j<= surf_n;j++)
    {
    printf ("\nSURFACE_%d POINT_1 X %2.3f Y %2.3f Z %2.3f\n",
    j,SURF_X[j].point_1.X,SURF_X[j].point_1.Y,SURF_X[j].point_1.Z);

    printf ("\nSURFACE_%d POINT_2 X %2.3f Y %2.3f Z %2.3f\n",
    j,SURF_X[j].point_2.X,SURF_X[j].point_2.Y,SURF_X[j].point_2.Z);

    printf ("\nSURFACE_%d POINT_3 X %2.3f Y %2.3f Z %2.3f\n",
    j,SURF_X[j].point_3.X,SURF_X[j].point_3.Y,SURF_X[j].point_3.Z);

    printf ("\nSURFACE_%d POINT_4 X %2.3f Y %2.3f Z %2.3f\n",
    j,SURF_X[j].point_4.X,SURF_X[j].point_4.Y,SURF_X[j].point_4.Z);
    }
    */
}
```

APPENDIX F

SURFACE RECOGNITION STAGE - PARALLEL TO Y-AXIS

SURFACE RECOGNITION STAGE - PARALLEL TO Y-AXIS

```
/* SURFACE RECOGNITION STAGE */

/*PARALLEL TO Y AXIS */

/* Function Prototype */


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>
#define Z_limit 0.5  /* TO DETECT CHANGE IN Z*/

/*  Global Variable Declaration  */
int In_Rec_n=1,Rec_n=1,Seg_n=1,n_pts_bet=1,pt=0;

float prev_X,prev_Y,prev_Z;

float X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,X4,Y4,Z4;

float temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2;

f                    l                    o                    a                    t
SLOPE_X1,SLOPE_Y1,SLOPE_Z1,SLOPE_Z2,SLOPE_X2,SLOPE_Y2,SLOPE_1,SL
OPE_2;

FILE *f1ptr, *f2ptr;


struct XYZ
{
  float X;
  float Y;
  float Z;
};
```

```
struct Segment
{
        struct XYZ start_pt;
        struct XYZ end_pt;
        int n_pts_bet;
};

/*structure for each line */

struct RecordY
{
 struct  Segment segment[10];
 int SEG_N;
 int SURF_N;
}parallel_Y[31];


struct POINT
{
float X;
float Y;
float Z;
};

/* Structure for each surface */

struct  SURFY_XYZ
{
 float X;
 float Y;
 float Z;
};


struct SURFACE_Y
{
 struct SURFY_XYZ point_1;
 struct SURFY_XYZ point_2;
 struct SURFY_XYZ point_3;
 struct SURFY_XYZ point_4;
 int SURF_N;
}SURF_Y[15];
```

```
main()
{
int j=1,k,p,q;
char name1[35], name2[35];
int surf_n=1;

printf("Enter filename (reduced data file) \n");
scanf("%s", name1);
printf("Enter number of lines Scanned on the object (parallel to  Y axis)\n");
scanf("%d",&In_Rec_n);
In_Rec_n++;

printf("Enter name of surface feature file \n");
scanf("%s",name2);

if((f1ptr = fopen(name1, "r")) ==NULL)
   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }

if((f2ptr = fopen(name2, "w+")) ==NULL)
   {
   printf("Can't write to file %s \n", name2);
   exit(1);
   }

/* READS POINTS IN REDUCED DATA FILE TO STRUCTURES */

for(p=1;p<In_Rec_n;p++)
   {
   fscanf(f1ptr,"%d",&parallel_Y[p].SEG_N);
        for (q=1; q<=parallel_Y[p].SEG_N;q++)
        {
        fscanf(f1ptr,"%f %f %f %f %f %f",
        &prev_X,&prev_Y,&prev_Z,&X,&Y,&Z);

parallel_Y[p].segment[q].start_pt.X=prev_X;parallel_Y[p].segment[q].start_pt.
Y=prev_Y;
parallel_Y[p].segment[q].start_pt.Z=prev_Z;parallel_Y[p].segment[q].end_pt.X
=X;

parallel_Y[p].segment[q].end_pt.Y=Y;parallel_Y[p].segment[q].end_pt.Z=Z;
      }
```

```
   }
   fclose(f1ptr);


/* PRINTS DATA JUST READ IN ON THE SCREEN */

for(p=1;p<In_Rec_n;p++)
   {
   printf("%d\n",parallel_Y[p].SEG_N);
        for (q=1; q<=parallel_Y[p].SEG_N;q++)
        {
        printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",
        parallel_Y[p].segment[q].start_pt.X,parallel_Y[p].segment[q].start_pt.Y,
        parallel_Y[p].segment[q].start_pt.Z,parallel_Y[p].segment[q].end_pt.X,
        parallel_Y[p].segment[q].end_pt.Y,parallel_Y[p].segment[q].end_pt.Z);
        }

    }

      q=1;
      {
      p=1;
      X1 = parallel_Y[p].segment[q].start_pt.X;
      Y1 = parallel_Y[p].segment[q].start_pt.Y;
      Z1 = parallel_Y[p].segment[q].start_pt.Z;
      X2 = parallel_Y[p].segment[q].end_pt.X;
      Y2 = parallel_Y[p].segment[q].end_pt.Y;
      Z2 = parallel_Y[p].segment[q].end_pt.Z;

      /*
      printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);
      */

      SURF_Y[j].point_1.X = X1;
      SURF_Y[j].point_1.Y = Y1;
      SURF_Y[j].point_1.Z = Z1;
      SURF_Y[j].point_2.X = X2;
      SURF_Y[j].point_2.Y = Y2;
      SURF_Y[j].point_2.Z = Z2;


    for(p=2;p<In_Rec_n;p++)
      {
      q=1;
```

```
    SLOPE_X1 = fabs(X2-X1);
    SLOPE_Z1 = fabs(Z2-Z1);


/* Comparing points of q segment of the p+1 record */

    X3 = parallel_Y[p].segment[q].start_pt.X;
    Y3 = parallel_Y[p].segment[q].start_pt.Y;
    Z3 = parallel_Y[p].segment[q].start_pt.Z;
    X4 = parallel_Y[p].segment[q].end_pt.X;
    Y4 = parallel_Y[p].segment[q].end_pt.Y;
    Z4 = parallel_Y[p].segment[q].end_pt.Z;

    /*
    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
    */

    SLOPE_X2 = fabs(X4-X3);
    SLOPE_Z2 = fabs(Z4-Z3);

  if (SLOPE_Z1==0)
     {
     SLOPE_1=0;
     }
  else
       {
       SLOPE_1 = SLOPE_X1/SLOPE_Z1;
       }

 if (SLOPE_Z2==0)
     {
     SLOPE_2=0;
     }
  else
       {
       SLOPE_2 = SLOPE_X2/SLOPE_Z2;
       }



  if (SLOPE_X1==0)
     {
     SLOPE_1=0;
     }
```

```
else
     {
     SLOPE_1 = SLOPE_X1/SLOPE_Z1;
     }

if (SLOPE_X2==0)
     {
     SLOPE_2=0;
     }
else
    {
    SLOPE_2 = SLOPE_X2/SLOPE_Z2;
    }

if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
     {
     temp_X1 = X3;
     temp_Y1 = Y3;
     temp_Z1 = Z3;
     temp_X2 = X4;
     temp_Y2 = Y4;
     temp_Z2 = Z4;
     }
 /*
     printf("%2.3f   %2.3f   %2.3f\n%2.3f   %2.3f
     %2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
 printf("p is %d  In rec is %d\n\n",p,In_Rec_n);
 */

if ( p == (In_Rec_n)-1)

     {
     SURF_Y[j].point_3.X = temp_X1;
     SURF_Y[j].point_3.Y = temp_Y1;
     SURF_Y[j].point_3.Z = temp_Z1;
     SURF_Y[j].point_4.X = temp_X2;
     SURF_Y[j].point_4.Y = temp_Y2;
     SURF_Y[j].point_4.Z = temp_Z2;
     SURF_Y[j].SURF_N=surf_n;
     j++;
     surf_n ++;
     }

if ((fabs(SLOPE_1-SLOPE_2)) > 5)
```

```
      {
      SURF_Y[j].point_3.X = temp_X1;
      SURF_Y[j].point_3.Y = temp_Y1;
      SURF_Y[j].point_3.Z = temp_Z1;
      SURF_Y[j].point_4.X = temp_X2;
      SURF_Y[j].point_4.Y = temp_Y2;
      SURF_Y[j].point_4.Z =temp_Z2;
      SURF_Y[j].SURF_N=surf_n;
      j++;
      surf_n ++;
      }
      }
    }

    q=2;
    {
    p=1;
    X1 = parallel_Y[p].segment[q].start_pt.X;
    Y1 = parallel_Y[p].segment[q].start_pt.Y;
    Z1 = parallel_Y[p].segment[q].start_pt.Z;
    X2 = parallel_Y[p].segment[q].end_pt.X;
    Y2 = parallel_Y[p].segment[q].end_pt.Y;
    Z2 = parallel_Y[p].segment[q].end_pt.Z;

    /*
    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);
    */

    SURF_Y[j].SURF_N=surf_n;
    SURF_Y[j].point_1.X = X1;
    SURF_Y[j].point_1.Y = Y1;
    SURF_Y[j].point_1.Z = Z1;
    SURF_Y[j].point_2.X = X2;
    SURF_Y[j].point_2.Y = Y2;
    SURF_Y[j].point_2.Z = Z2;

  for(p=2;p<In_Rec_n;p++)
    {
    q=2;
    SLOPE_X1 = fabs(X2-X1);
    SLOPE_Y1 = fabs(Y2-Y1);

/* Comparing points of q segment of the p+1 record */
```

```
X3 = parallel_Y[p].segment[q].start_pt.X;
Y3 = parallel_Y[p].segment[q].start_pt.Y;
Z3 = parallel_Y[p].segment[q].start_pt.Z;
X4 = parallel_Y[p].segment[q].end_pt.X;
Y4 = parallel_Y[p].segment[q].end_pt.Y;
Z4 = parallel_Y[p].segment[q].end_pt.Z;

/*
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
*/

SLOPE_X2 = fabs(X4-X3);
SLOPE_Y2 = fabs(Y4-Y3);

if (SLOPE_Y1==0)
  {
  SLOPE_1=0;
  }
else
    {
    SLOPE_1 = SLOPE_X1/SLOPE_Y1;
    }

if (SLOPE_Y2==0)
  {
  SLOPE_2=0;
  }
else
    {
    SLOPE_2 = SLOPE_X2/SLOPE_Y2;
    }

if (SLOPE_X1==0)
  {
  SLOPE_1=0;
  }
else
    {
    SLOPE_1 = SLOPE_X1/SLOPE_Y1;
    }

if (SLOPE_X2==0)
    {
    SLOPE_2=0;
```

```
    }
else
   {
   SLOPE_2 = SLOPE_X2/SLOPE_Y2;
   }

if ((fabs(SLOPE_1-SLOPE_2) <= 0.5) && (fabs(Z3-Z2)<=3))
    {
    temp_X1 = X3;
    temp_Y1 = Y3;
    temp_Z1 = Z3;
    temp_X2 = X4;
    temp_Y2 = Y4;
    temp_Z2 = Z4;
    }
 /*
    printf("%2.3f   %2.3f   %2.3f\n%2.3f   %2.3f
    %2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
 printf("p is %d  In rec is %d\n\n",p,In_Rec_n);
 */
    else
    {
    SURF_Y[j].point_3.X = temp_X1;
    SURF_Y[j].point_3.Y = temp_Y1;
    SURF_Y[j].point_3.Z = temp_Z1;
    SURF_Y[j].point_4.X = temp_X2;
    SURF_Y[j].point_4.Y = temp_Y2;
    SURF_Y[j].point_4.Z = temp_Z2;
    SURF_Y[j].SURF_N=surf_n;
    j++;
    surf_n ++;

    X1 = X3;
    Y1 = Y3;
    Z1 = Z3;
    X2 = X4;
    Y2 = Y4;
    Z2 = Z4;
    SURF_Y[j].point_1.X = X1;
    SURF_Y[j].point_1.Y = Y1;
    SURF_Y[j].point_1.Z = Z1;
    SURF_Y[j].point_2.X = X2;
    SURF_Y[j].point_2.Y = Y2;
    SURF_Y[j].point_2.Z = Z2;
```

```
        }

 if ( p == (In_Rec_n)-1)

        {
        SURF_Y[j].point_3.X = temp_X1;
        SURF_Y[j].point_3.Y = temp_Y1;
        SURF_Y[j].point_3.Z = temp_Z1;
        SURF_Y[j].point_4.X = temp_X2;
        SURF_Y[j].point_4.Y = temp_Y2;
        SURF_Y[j].point_4.Z = temp_Z2;
        SURF_Y[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        }

if ((fabs(SLOPE_1-SLOPE_2)) > 0.5)
       {
        SURF_Y[j].point_3.X = temp_X1;
        SURF_Y[j].point_3.Y = temp_Y1;
        SURF_Y[j].point_3.Z = temp_Z1;
        SURF_Y[j].point_4.X = temp_X2;
        SURF_Y[j].point_4.Y = temp_Y2;
        SURF_Y[j].point_4.Z =temp_Z2;
        SURF_Y[j].SURF_N=surf_n;
        j++;
        surf_n ++;
        SURF_Y[j].point_1.X = X3;
        SURF_Y[j].point_1.Y = Y3;
        SURF_Y[j].point_1.Z = Z3;
        SURF_Y[j].point_2.X = X4;
        SURF_Y[j].point_2.Y = Y4;
        SURF_Y[j].point_2.Z = Z4;
        SURF_Y[j].SURF'_N=surf_n;
        }
        }
    }
   q=3;
    {
    p=1;
    X1 = parallel_Y[p].segment[q].start_pt.X;
    Y1 = parallel_Y[p].segment[q].start_pt.Y;
    Z1 = parallel_Y[p].segment[q].start_pt.Z;
    X2 = parallel_Y[p].segment[q].end_pt.X;
```

```
    Y2 = parallel_Y[p].segment[q].end_pt.Y;
    Z2 = parallel_Y[p].segment[q].end_pt.Z;

    /*
    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",X1,Y1,Z1,X2,Y2,Z2);
    */
    SURF_Y[j].point_1.X = X1;
    SURF_Y[j].point_1.Y = Y1;
    SURF_Y[j].point_1.Z = Z1;
    SURF_Y[j].point_2.X = X2;
    SURF_Y[j].point_2.Y = Y2;
    SURF_Y[j].point_2.Z = Z2;


  for(p=2;p<In_Rec_n;p++)
    {
    SLOPE_X1 = fabs(X2-X1);
    SLOPE_Z1 = fabs(Z2-Z1);


/* Comparing points of q segment of the p+1 record */

    X3 = parallel_Y[p].segment[q].start_pt.X;
    Y3 = parallel_Y[p].segment[q].start_pt.Y;
    Z3 = parallel_Y[p].segment[q].start_pt.Z;
    X4 = parallel_Y[p].segment[q].end_pt.X;
    Y4 = parallel_Y[p].segment[q].end_pt.Y;
    Z4 = parallel_Y[p].segment[q].end_pt.Z;

    /*
    printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n\n",X3,Y3,Z3,X4,Y4,Z4);
    */

    SLOPE_X2 = fabs(X4-X3);
    SLOPE_Z2 = fabs(Z4-Z3);

  if (SLOPE_Z1==0)
    {
    SLOPE_1=0;
    }
  else
      {
      SLOPE_1 = SLOPE_X1/SLOPE_Z1;
      }
```

```
if (SLOPE_Z2==0)
    {
    SLOPE_2=0;
    }
  else
      {
      SLOPE_2 = SLOPE_X2/SLOPE_Z2;
      }

  if (SLOPE_X1==0)
    {
    SLOPE_1=0;
    }
  else
      {
      SLOPE_1 = SLOPE_X1/SLOPE_Z1;
      }

  if (SLOPE_X2==0)
      {
      SLOPE_2=0;
      }
  else
     {
     SLOPE_2 = SLOPE_X2/SLOPE_Z2;
     }

  if (fabs(SLOPE_1-SLOPE_2) <= 0.5)
      {
      temp_X1 = X3;
      temp_Y1 = Y3;
      temp_Z1 = Z3;
      temp_X2 = X4;
      temp_Y2 = Y4;
      temp_Z2 = Z4;
      }

   /*
      printf("%2.3f  %2.3f  %2.3f\n%2.3f  %2.3f
%2.3f\n\n",temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2);
   printf("p is %d  In rec is %d\n\n",p,In_Rec_n);
   */

 if ( p == (In_Rec_n)-1)
```

```
        {
        SURF_Y[j].point_3.X = temp_X1;
        SURF_Y[j].point_3.Y = temp_Y1;
        SURF_Y[j].point_3.Z = temp_Z1;
        SURF_Y[j].point_4.X = temp_X2;
        SURF_Y[j].point_4.Y = temp_Y2;
        SURF_Y[j].point_4.Z = temp_Z2;
        SURF_Y[j].SURF_N=surf_n;

        }

if ((fabs(SLOPE_1-SLOPE_2)) > 5)
        {
        SURF_Y[j].point_3.X = temp_X1;
        SURF_Y[j].point_3.Y = temp_Y1;
        SURF_Y[j].point_3.Z = temp_Z1;
        SURF_Y[j].point_4.X = temp_X2;
        SURF_Y[j].point_4.Y = temp_Y2;
        SURF_Y[j].point_4.Z =temp_Z2;
        SURF_Y[j].SURF_N=surf_n;
        j++;
        surf_n ++;
       SURF_Y[j].point_3.X = X3;
        SURF_Y[j].point_3.Y = Y3;
        SURF_Y[j].point_3.Z = Z3;
        SURF_Y[j].point_4.X = X4;
        SURF_Y[j].point_4.Y = Y4;
        SURF_Y[j].point_4.Z = Z4;
        SURF_Y[j].SURF_N=surf_n;
        }
        }
    }

    for ( j=1;j<= surf_n;j++)
    {
    printf("\nSURFACE_%d\n",j);

    printf("P1 X %2.3f Y %2.3f Z %2.3f\n",
          SURF_Y[j].point_1.X,SURF_Y[j].point_1.Y,SURF_Y[j].point_1.Z);

    printf("P2 X %2.3f Y %2.3f Z %2.3f\n",
          SURF_Y[j].point_2.X,SURF_Y[j].point_2.Y,SURF_Y[j].point_2.Z);

    printf("P3 X %2.3f Y %2.3f Z %2.3f\n",
```

```
        SURF_Y[j].point_3.X,SURF_Y[j].point_3.Y,SURF_Y[j].point_3.Z);

printf("P4 X %2.3f Y %2.3f Z %2.3f\n",
        SURF_Y[j].point_4.X,SURF_Y[j].point_4.Y,SURF_Y[j].point_4.Z);
 }


fprintf(f2ptr,"%d\n",surf_n);
for ( j=1;j<= surf_n;j++)
{
fprintf(f2ptr,"%d\n",j);
fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_Y[j].point_1.X,SURF_Y[j].point_1.Y,SURF_Y[j].point_1.Z);

fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_Y[j].point_2.X,SURF_Y[j].point_2.Y,SURF_Y[j].point_2.Z);

fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_Y[j].point_3.X,SURF_Y[j].point_3.Y,SURF_Y[j].point_3.Z);

fprintf(f2ptr,"%2.3f %2.3f %2.3f\n",
        SURF_Y[j].point_4.X,SURF_Y[j].point_4.Y,SURF_Y[j].point_4.Z);
 }

fclose(f2ptr);
/*
for ( j=1;j<= surf_n;j++)
{
printf ("\nSURFACE_%d P1 X %2.3f Y %2.3f Z %2.3f\n",


j,SURF_Y[j].point_1.X,SURF_Y[j].point_1.Y,SURF_Y[j].point_1.Z);


printf ("\nSURFACE_%d POINT_2 X %2.3f Y %2.3f Z %2.3f\n",
j,SURF_Y[j].point_2.X,SURF_Y[j].point_2.Y,SURF_Y[j].point_2.Z);


printf ("\nSURFACE_%d POINT_3 X %2.3f Y %2.3f Z %2.3f\n",
j,SURF_Y[j].point_3.X,SURF_Y[j].point_3.Y,SURF_Y[j].point_3.Z);

printf ("\nSURFACE_%d POINT_4 X %2.3f Y %2.3f Z %2.3f\n",
j,SURF_Y[j].point_4.X,SURF_Y[j].point_4.Y,SURF_Y[j].point_4.Z);
} /*}
```

APPENDIX G

NC CODE GENERATION STAGE

NC CODE GENERATION STAGE

```
/* NUMERICAL CODE GENERATION STAGE */

/* Function Prototype */


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>



/*  Global Variable Declaration  */

int In_Rec_n=1,Rec_n=1,Seg_n=1,n_pts_bet=1,pt=0;

float prev_X,prev_Y,prev_Z,PT1;

float X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,X4,Y4,Z4;

float XX1,XY1,XZ1,XX2,XY2,XZ2,XX3,XY3,XZ3,XX4,XY4,XZ4;

float YX1,YY1,YZ1,YX2,YY2,YZ2,YX3,YY3,YZ3,YX4,YY4,YZ4;

float temp_X1,temp_Y1,temp_Z1,temp_X2,temp_Y2,temp_Z2;



FILE *f1ptr, *f2ptr, *f3ptr;


struct XYZ
{
  float X;
  float Y;
  float Z;
```

```
};

struct Segment
{
        struct XYZ start_pt;
        struct XYZ end_pt;
        int n_pts_bet;
};


/*structure for each line */

struct RecordY
{
 struct  Segment segment[10];
 int SEG_N;
 int SURF_N;
}parallel_Y[31];


struct POINT
{
float X;
float Y;
float Z;
};

/* Structure for each surface */

struct  SURFY_XYZ
{
 float X;
 float Y;
 float Z;
};



struct SURFACE_Y
{
 struct SURFY_XYZ point_1;
 struct SURFY_XYZ point_2;
 struct SURFY_XYZ point_3;
```

```
 struct SURFY_XYZ point_4;
 int SURF_N,surf_nY;
}SURF_Y[15];


struct SURFACE_X
{
 struct SURFY_XYZ point_1;
 struct SURFY_XYZ point_2;
 struct SURFY_XYZ point_3;
 struct SURFY_XYZ point_4;
 int SURF_N,surf_nX;
}SURF_X[15];

/* STRUCTURE FOR FINAL SURFACE DEFINITION */

struct SURFACE_FINAL
{
 struct SURFY_XYZ point_1;
 struct SURFY_XYZ point_2;
 struct SURFY_XYZ point_3;
 struct SURFY_XYZ point_4;
 int FSURF_N;
}FIN_SURF[15];



main()
{

int j=1,k=1,p,q,surf_n,FSURF_n=1,p1,p2,p3;
char name1[35], name2[35], name3[35];

printf("Enter filename - surface feature file for parallel to X-axis \n");
scanf("%s", name1);

printf("Enter filename - surface feature file for parallel to Y-axis \n");
scanf("%s", name2);


printf("Enter name of NC code file to write the APT statements\n");
scanf("%s",name3);
```

```
if((f1ptr = fopen(name1, "r")) ==NULL)
   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }

if((f2ptr = fopen(name2, "r")) ==NULL)
   {
    printf("Can't open %s to read \n", name1);
    exit(1);
   }


if((f3ptr = fopen(name3, "w+")) ==NULL)

   {
   printf("Can't write to file %s \n", name2);
   exit(1);
   }




/* READS POINTS IN SURFACE DATA FILE TO STRUCTURES */

    fscanf(f1ptr,"%d",&surf_n);
    /*printf("Surf_n %d\n",surf_n);*/
    /*surf_nX = SURF_X[j].SURF_N.surf_nX;*/
    for ( j=1;j<= surf_n;j++)
    {
     fscanf(f1ptr,"%d",&k);
     /*printf("k %d\n",k);*/

     fscanf(f1ptr,"%f %f %f",
          &SURF_X[j].point_1.X,&SURF_X[j].point_1.Y,&SURF_X[j].point_1.Z);
     /*printf("%2.3f %2.3f %2.3f\n",
           SURF_X[j].point_1.X,SURF_X[j].point_1.Y,SURF_X[j].point_1.Z);
     */

     fscanf(f1ptr,"%f %f %f",
          &SURF_X[j].point_2.X,&SURF_X[j].point_2.Y,&SURF_X[j].point_2.Z);
     /*printf("%2.3f %2.3f %2.3f\n",
           SURF_X[j].point_2.X,SURF_X[j].point_2.Y,SURF_X[j].point_2.Z);
```

```
*/
fscanf(f1ptr,"%f %f %f",
      &SURF_X[j].point_3.X,&SURF_X[j].point_3.Y,&SURF_X[j].point_3.Z);
/*printf("%2.3f %2.3f %2.3f\n",
      SURF_X[j].point_3.X,SURF_X[j].point_3.Y,SURF_X[j].point_3.Z);
*/
fscanf(f1ptr,"%f %f %f",
      &SURF_X[j].point_4.X,&SURF_X[j].point_4.Y,&SURF_X[j].point_4.Z);
/*printf("%2.3f %2.3f %2.3f\n",
      SURF_X[j].point_4.X,SURF_X[j].point_4.Y,SURF_X[j].point_4.Z);
*/
}

fclose(f1ptr);


fscanf(f2ptr,"%d",&surf_n);
/*printf("Surf_n %d\n", surf_n);*/
for ( j=1;j<= surf_n;j++)
{
fscanf(f2ptr,"%d",&k);
/*printf("k %d\n",k);*/

fscanf(f2ptr,"%f %f %f",
      &SURF_Y[j].point_1.X,&SURF_Y[j].point_1.Y,&SURF_Y[j].point_1.Z);
/*printf("%2.3f %2.3f %2.3f\n",
      SURF_Y[j].point_1.X,SURF_Y[j].point_1.Y,SURF_Y[j].point_1.Z);
*/
fscanf(f2ptr,"%f %f %f",
      &SURF_Y[j].point_2.X,&SURF_Y[j].point_2.Y,&SURF_Y[j].point_2.Z);
/*printf("%2.3f %2.3f %2.3f\n",
      SURF_Y[j].point_2.X,SURF_Y[j].point_2.Y,SURF_Y[j].point_2.Z);
*/
fscanf(f2ptr,"%f %f %f",
      &SURF_Y[j].point_3.X,&SURF_Y[j].point_3.Y,&SURF_Y[j].point_3.Z);
/*printf("%2.3f %2.3f %2.3f\n",
      SURF_Y[j].point_3.X,SURF_Y[j].point_3.Y,SURF_Y[j].point_3.Z);
*/
 fscanf(f2ptr,"%f %f %f",
      &SURF_Y[j].point_4.X,&SURF_Y[j].point_4.Y,&SURF_Y[j].point_4.Z);
 /*printf("%2.3f %2.3f %2.3f\n",
      SURF_Y[j].point_4.X,SURF_Y[j].point_4.Y,SURF_Y[j].point_4.Z);
 */
```

```
    }

    fclose(f2ptr);

     j=1;
     {
     /* READING DATA FROM STRUCTURES OF SURFACES - PARALLEL TO
X-AXIS */
     XX1 = SURF_X[j].point_1.X;
     XY1 = SURF_X[j].point_1.Y;
     XZ1 = SURF_X[j].point_1.Z;
     XX2 = SURF_X[j].point_2.X;
     XY2 = SURF_X[j].point_2.Y;
     XZ2 = SURF_X[j].point_2.Z;
     XX3 = SURF_X[j].point_3.X;
     XY3 = SURF_X[j].point_3.Y;
     XZ3 = SURF_X[j].point_3.Z;
     XX4 = SURF_X[j].point_4.X;
     XY4 = SURF_X[j].point_4.Y;
     XZ4 = SURF_X[j].point_4.Z;
     /*
     printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",XX1,XY1,XZ1,XX2,XY2,XZ2);
     printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",XX3,XY3,XZ3,XX4,XY4,XZ4);
     */

    /* READING DATA FROM STRUCTURES OF SURFACES - PARALLEL TO
Y-AXIS */


     YX1 = SURF_Y[j].point_1.X;
     YY1 = SURF_Y[j].point_1.Y;
     YZ1 = SURF_Y[j].point_1.Z;
     YX2 = SURF_Y[j].point_2.X;
     YY2 = SURF_Y[j].point_2.Y;
     YZ2 = SURF_Y[j].point_2.Z;
     YX3 = SURF_Y[j].point_3.X;
     YY3 = SURF_Y[j].point_3.Y;
     YZ3 = SURF_Y[j].point_3.Z;
     YX4 = SURF_Y[j].point_4.X;
     YY4 = SURF_Y[j].point_4.Y;
     YZ4 = SURF_Y[j].point_4.Z;

     /*
     printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",YX1,YY1,YZ1,YX2,YY2,YZ2);
```

```
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",YX3,YY3,YZ3,YX4,YY4,YZ4);
*/
if(fabs(XZ1-YZ4) < 5)
{
k = 1;
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;
FIN_SURF[k].point_2.X = XX2;
FIN_SURF[k].point_2.Y = XY2;
FIN_SURF[k].point_2.Z = XZ2;
FIN_SURF[k].point_3.X = XX3;
FIN_SURF[k].point_3.Y = XY3;
FIN_SURF[k].point_3.Z = XZ3;
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = XY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
else
{
k=1;
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;
FIN_SURF[k].point_2.X = XX2;
FIN_SURF[k].point_2.Y = XY2;
FIN_SURF[k].point_2.Z = XZ2;
FIN_SURF[k].point_3.X = XX3;
FIN_SURF[k].point_3.Y = XY3;
FIN_SURF[k].point_3.Z = XZ3;
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = XY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;

k++;
FSURF_n ++;

FIN_SURF[k].point_1.X = YX1;
FIN_SURF[k].point_1.Y = YY1;
FIN_SURF[k].point_1.Z = YZ1;
```

```
FIN_SURF[k].point_2.X = YX2;
FIN_SURF[k].point_2.Y = YY2;
FIN_SURF[k].point_2.Z = YZ2;
FIN_SURF[k].point_3.X = YX3;
FIN_SURF[k].point_3.Y = YY3;
FIN_SURF[k].point_3.Z = YZ3;
FIN_SURF[k].point_4.X = YX4;
FIN_SURF[k].point_4.Y = YY4;
FIN_SURF[k].point_4.Z = YZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
}

j=2;
{
XX1 = SURF_X[j].point_1.X;
XY1 = SURF_X[j].point_1.Y;
XZ1 = SURF_X[j].point_1.Z;
XX2 = SURF_X[j].point_2.X;
XY2 = SURF_X[j].point_2.Y;
XZ2 = SURF_X[j].point_2.Z;
XX3 = SURF_X[j].point_3.X;
XY3 = SURF_X[j].point_3.Y;
XZ3 = SURF_X[j].point_3.Z;
XX4 = SURF_X[j].point_4.X;
XY4 = SURF_X[j].point_4.Y;
XZ4 = SURF_X[j].point_4.Z;
/*
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",XX1,XY1,XZ1,XX2,XY2,XZ2);
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",XX3,XY3,XZ3,XX4,XY4,XZ4);
*/
YX1 = SURF_Y[j].point_1.X;
YY1 = SURF_Y[j].point_1.Y;
YZ1 = SURF_Y[j].point_1.Z;
YX2 = SURF_Y[j].point_2.X;
YY2 = SURF_Y[j].point_2.Y;
YZ2 = SURF_Y[j].point_2.Z;
YX3 = SURF_Y[j].point_3.X;
YY3 = SURF_Y[j].point_3.Y;
YZ3 = SURF_Y[j].point_3.Z;
YX4 = SURF_Y[j].point_4.X;
YY4 = SURF_Y[j].point_4.Y;
```

```
YZ4 = SURF_Y[j].point_4.Z;

/*
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",YX1,YY1,YZ1,YX2,YY2,YZ2);
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",YX3,YY3,YZ3,YX4,YY4,YZ4);
*/
if(fabs(XZ1-YZ4) < 5 )
{
  if ((XX1 > YX1) && (XY1 < YY1))
  FIN_SURF[k].point_1.X = YX1;
  FIN_SURF[k].point_1.Y = XY1;
  FIN_SURF[k].point_1.Z = XZ1;

  if ((XX2 > YX2) && (XY2 < YY2))
  FIN_SURF[k].point_2.X = YX2;
  FIN_SURF[k].point_2.Y = YY2;
  FIN_SURF[k].point_2.Z = XZ2;

  if((XX3 < YX3) && (XY3 > YY3))
  FIN_SURF[k].point_3.X = YX3;
  FIN_SURF[k].point_3.Y = YY3;
  FIN_SURF[k].point_3.Z = XZ3;

  if((XX4 < YX4) && (XY4 < YY4))
  FIN_SURF[k].point_4.X = YX4;
  FIN_SURF[k].point_4.Y = XY4;
  FIN_SURF[k].point_4.Z = XZ4;
  FIN_SURF[k].FSURF_N = FSURF_n;
  k++;
  FSURF_n ++;
}

else
{
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;
FIN_SURF[k].point_2.X = XX2;
FIN_SURF[k].point_2.Y = XY2;
FIN_SURF[k].point_2.Z = XZ2;
FIN_SURF[k].point_3.X = XX3;
FIN_SURF[k].point_3.Y = XY3;
FIN_SURF[k].point_3.Z = XZ3;
FIN_SURF[k].point_4.X = XX4;
```

```
FIN_SURF[k].point_4.Y = XY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;

k++;
FSURF_n ++;

FIN_SURF[k].point_1.X = YX1;
FIN_SURF[k].point_1.Y = YY1;
FIN_SURF[k].point_1.Z = YZ1;
FIN_SURF[k].point_2.X = YX2;
FIN_SURF[k].point_2.Y = YY2;
FIN_SURF[k].point_2.Z = YZ2;
FIN_SURF[k].point_3.X = YX3;
FIN_SURF[k].point_3.Y = YY3;
FIN_SURF[k].point_3.Z = YZ3;
FIN_SURF[k].point_4.X = YX4;
FIN_SURF[k].point_4.Y = YY4;
FIN_SURF[k].point_4.Z = YZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
}

j = 3;
{
XX1 = SURF_X[j].point_1.X;
XY1 = SURF_X[j].point_1.Y;
XZ1 = SURF_X[j].point_1.Z;
XX2 = SURF_X[j].point_2.X;
XY2 = SURF_X[j].point_2.Y;
XZ2 = SURF_X[j].point_2.Z;
XX3 = SURF_X[j].point_3.X;
XY3 = SURF_X[j].point_3.Y;
XZ3 = SURF_X[j].point_3.Z;
XX4 = SURF_X[j].point_4.X;
XY4 = SURF_X[j].point_4.Y;
XZ4 = SURF_X[j].point_4.Z;

YX1 = SURF_Y[j].point_1.X;
YY1 = SURF_Y[j].point_1.Y;
YZ1 = SURF_Y[j].point_1.Z;
YX2 = SURF_Y[j].point_2.X;
```

```
YY2 = SURF_Y[j].point_2.Y;
YZ2 = SURF_Y[j].point_2.Z;
YX3 = SURF_Y[j].point_3.X;
YY3 = SURF_Y[j].point_3.Y;
YZ3 = SURF_Y[j].point_3.Z;
YX4 = SURF_Y[j].point_4.X;
YY4 = SURF_Y[j].point_4.Y;
YZ4 = SURF_Y[j].point_4.Z;

if(fabs(XZ1-YZ4) < 5 )
{
if ((YX1 > XX1) && (YY1 > XY1))
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;

if (( YX2 < XX2 ) && ( YY2 > XY2 ))
FIN_SURF[k].point_2.X = YX2;
FIN_SURF[k].point_2.Y = YY2;
FIN_SURF[k].point_2.Z = XZ2;

if (( YX3 > XX3 ) && (YY3 < XY3))
FIN_SURF[k].point_3.X = YX3;
FIN_SURF[k].point_3.Y = YY3;
FIN_SURF[k].point_3.Z = XZ3;

if (( XX4 > YX4) && ( YY4 > XY4))
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = YY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
else
{
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;
FIN_SURF[k].point_2.X = XX2;
FIN_SURF[k].point_2.Y = XY2;
FIN_SURF[k].point_2.Z = XZ2;
FIN_SURF[k].point_3.X = XX3;
FIN_SURF[k].point_3.Y = XY3;
```

```
FIN_SURF[k].point_3.Z = XZ3;
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = XY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;

k++;
FSURF_n ++;

FIN_SURF[k].point_1.X = YX1;
FIN_SURF[k].point_1.Y = YY1;
FIN_SURF[k].point_1.Z = YZ1;
FIN_SURF[k].point_2.X = YX2;
FIN_SURF[k].point_2.Y = YY2;
FIN_SURF[k].point_2.Z = YZ2;
FIN_SURF[k].point_3.X = YX3;
FIN_SURF[k].point_3.Y = YY3;
FIN_SURF[k].point_3.Z = YZ3;
FIN_SURF[k].point_4.X = YX4;
FIN_SURF[k].point_4.Y = YY4;
FIN_SURF[k].point_4.Z = YZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
}


j = 4;
{
XX1 = SURF_X[j].point_1.X;
XY1 = SURF_X[j].point_1.Y;
XZ1 = SURF_X[j].point_1.Z;
XX2 = SURF_X[j].point_2.X;
XY2 = SURF_X[j].point_2.Y;
XZ2 = SURF_X[j].point_2.Z;
XX3 = SURF_X[j].point_3.X;
XY3 = SURF_X[j].point_3.Y;
XZ3 = SURF_X[j].point_3.Z;
XX4 = SURF_X[j].point_4.X;
XY4 = SURF_X[j].point_4.Y;
XZ4 = SURF_X[j].point_4.Z;

YX1 = SURF_Y[j].point_1.X;
```

```
YY1 = SURF_Y[j].point_1.Y;
YZ1 = SURF_Y[j].point_1.Z;
YX2 = SURF_Y[j].point_2.X;
YY2 = SURF_Y[j].point_2.Y;
YZ2 = SURF_Y[j].point_2.Z;
YX3 = SURF_Y[j].point_3.X;
YY3 = SURF_Y[j].point_3.Y;
YZ3 = SURF_Y[j].point_3.Z;
YX4 = SURF_Y[j].point_4.X;
YY4 = SURF_Y[j].point_4.Y;
YZ4 = SURF_Y[j].point_4.Z;

if(fabs(XZ1-YZ4) < 5)
{
if ((YX1 > XX1) && (YY1 > XY1))
FIN_SURF[k].point_1.X = XX1;
FIN_SURF[k].point_1.Y = XY1;
FIN_SURF[k].point_1.Z = XZ1;

if (( YX2 < XX2 ) && ( YY2 > XY2 ))
FIN_SURF[k].point_2.X = YX2;
FIN_SURF[k].point_2.Y = YY2;
FIN_SURF[k].point_2.Z = XZ2;

if (( YX3 > XX3 ) && (YY3 < XY3))
FIN_SURF[k].point_3.X = YX3;
FIN_SURF[k].point_3.Y = YY3;
FIN_SURF[k].point_3.Z = XZ3;

if (( XX4 > YX4) && ( YY4 > XY4))
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = YY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;
k++;
FSURF_n ++;
}
else
{
  if(fabs(XZ1-YZ1) > 4)
  FIN_SURF[k].point_1.X = XX1;
  FIN_SURF[k].point_1.Y = XY1;
  FIN_SURF[k].point_1.Z = XZ1;
  FIN_SURF[k].point_2.X = XX2;
```

```
FIN_SURF[k].point_2.Y = XY2;
FIN_SURF[k].point_2.Z = XZ2;
FIN_SURF[k].point_3.X = XX3;
FIN_SURF[k].point_3.Y = XY3;
FIN_SURF[k].point_3.Z = XZ3;
FIN_SURF[k].point_4.X = XX4;
FIN_SURF[k].point_4.Y = XY4;
FIN_SURF[k].point_4.Z = XZ4;
FIN_SURF[k].FSURF_N = FSURF_n;

k++;
FSURF_n ++;

j = 5;
XX1 = SURF_X[j].point_1.X;
XY1 = SURF_X[j].point_1.Y;
XZ1 = SURF_X[j].point_1.Z;
XX2 = SURF_X[j].point_2.X;
XY2 = SURF_X[j].point_2.Y;
XZ2 = SURF_X[j].point_2.Z;
XX3 = SURF_X[j].point_3.X;
XY3 = SURF_X[j].point_3.Y;
XZ3 = SURF_X[j].point_3.Z;
XX4 = SURF_X[j].point_4.X;
XY4 = SURF_X[j].point_4.Y;
XZ4 = SURF_X[j].point_4.Z;

if(fabs(XZ1-YZ4) < 5 )
  {
  if((XX1 > YX1 ) && (YY1 > XY1))
  FIN_SURF[k].point_1.X = YX1;
  FIN_SURF[k].point_1.Y = XY1;
  FIN_SURF[k].point_1.Z = XZ1;

  if((XX2 > YX2) && (YY2 > XY2))
  FIN_SURF[k].point_2.X = YX2;
  FIN_SURF[k].point_2.Y = YY2;
  FIN_SURF[k].point_2.Z = XZ2;

  if((YX3 > XX3) && (XY3 > YY3))
  FIN_SURF[k].point_3.X = YX3;
  FIN_SURF[k].point_3.Y = YY3;
  FIN_SURF[k].point_3.Z = XZ3;
```

```
    if((YX4 > XX4 ) && (YY4 > XY4 ))
    FIN_SURF[k].point_4.X = YX4;
    FIN_SURF[k].point_4.Y = YY4;
    FIN_SURF[k].point_4.Z = XZ4;
   FIN_SURF[k].FSURF_N = FSURF_n;

    k++;
    FSURF_n ++;
    }
     }
}

j = 6;
{
XX1 = SURF_X[j].point_1.X;
XY1 = SURF_X[j].point_1.Y;
XZ1 = SURF_X[j].point_1.Z;
XX2 = SURF_X[j].point_2.X;
XY2 = SURF_X[j].point_2.Y;
XZ2 = SURF_X[j].point_2.Z;
XX3 = SURF_X[j].point_3.X;
XY3 = SURF_X[j].point_3.Y;
XZ3 = SURF_X[j].point_3.Z;
XX4 = SURF_X[j].point_4.X;
XY4 = SURF_X[j].point_4.Y;
XZ4 = SURF_X[j].point_4.Z;
j = 5;
YX1 = SURF_Y[j].point_1.X;
YY1 = SURF_Y[j].point_1.Y;
YZ1 = SURF_Y[j].point_1.Z;
YX2 = SURF_Y[j].point_2.X;
YY2 = SURF_Y[j].point_2.Y;
YZ2 = SURF_Y[j].point_2.Z;
YX3 = SURF_Y[j].point_3.X;
YY3 = SURF_Y[j].point_3.Y;
YZ3 = SURF_Y[j].point_3.Z;
YX4 = SURF_Y[j].point_4.X;
YY4 = SURF_Y[j].point_4.Y;
YZ4 = SURF_Y[j].point_4.Z;

if(fabs(XZ1-YZ4) > 5)
   {
    FIN_SURF[k].point_1.X = XX1;
    FIN_SURF[k].point_1.Y = XY1;
```

```
    FIN_SURF[k].point_1.Z = XZ1;
    FIN_SURF[k].point_2.X = XX2;
    FIN_SURF[k].point_2.Y = XY2;
    FIN_SURF[k].point_2.Z = XZ2;
    FIN_SURF[k].point_3.X = XX3;
    FIN_SURF[k].point_3.Y = XY3;
    FIN_SURF[k].point_3.Z = XZ3;
    FIN_SURF[k].point_4.X = XX4;
    FIN_SURF[k].point_4.Y = XY4;
    FIN_SURF[k].point_4.Z = XZ4;
    FIN_SURF[k].FSURF_N = FSURF_n;

    k++;
    FSURF_n ++;

    FIN_SURF[k].point_1.X = YX1;
    FIN_SURF[k].point_1.Y = YY1;
    FIN_SURF[k].point_1.Z = YZ1;
    FIN_SURF[k].point_2.X = YX2;
    FIN_SURF[k].point_2.Y = YY2;
    FIN_SURF[k].point_2.Z = YZ2;
    FIN_SURF[k].point_3.X = YX3;
    FIN_SURF[k].point_3.Y = YY3;
    FIN_SURF[k].point_3.Z = YZ3;
    FIN_SURF[k].point_4.X = YX4;
    FIN_SURF[k].point_4.Y = YY4;
    FIN_SURF[k].point_4.Z = YZ4;
    FIN_SURF[k].FSURF_N = FSURF_n;
    k++;

   }


for (k=1;k<=FSURF_n;k++)
{
printf("%d\n",FIN_SURF[k].FSURF_N);
printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",
FIN_SURF[k].point_1.X, FIN_SURF[k].point_1.Y, FIN_SURF[k].point_1.Z,
FIN_SURF[k].point_2.X, FIN_SURF[k].point_2.Y, FIN_SURF[k].point_2.Z);

printf("%2.3f %2.3f %2.3f\n%2.3f %2.3f %2.3f\n",
FIN_SURF[k].point_3.X, FIN_SURF[k].point_3.Y, FIN_SURF[k].point_3.Z,
FIN_SURF[k].point_4.X, FIN_SURF[k].point_4.Y, FIN_SURF[k].point_4.Z);
}
```

```
printf("\n\n");
printf("PART NAME - SAMPLE PART\n");
q =1;
for (k=1;k<=FSURF_n;k++)
{
/*printf("q_is %d\n",q);*/
printf("P%1d = POINT/%2.3f,%2.3f,%2.3f\n",
q,FIN_SURF[k].point_1.X,FIN_SURF[k].point_1.Y,FIN_SURF[k].point_1.Z);
p1=q;
q++;
printf("P%1d = POINT/%2.3f,%2.3f,%2.3f\n",q,FIN_SURF[k].point_2.X,
FIN_SURF[k].point_2.Y, FIN_SURF[k].point_2.Z);
p2 =q;
q++;
printf("P%1d = POINT/%2.3f,%2.3f,%2.3f\n",q,FIN_SURF[k].point_3.X,
FIN_SURF[k].point_3.Y, FIN_SURF[k].point_3.Z);
p3 =q;
q++;
/*printf("PL%d = PLANE/%2.3f,%2.3f,%2.3f\n",FIN_SURF[k].FSURF_N,
FIN_SURF[k].point_1.X, FIN_SURF[k].point_1.Y, FIN_SURF[k].point_1.Z);*/
printf("PL%d = PLANE/P%d,P%d,P%d\n",FIN_SURF[k].FSURF_N,p1,p2,p3);
}
printf("SP = POINT/0,0,0\n");
printf("\n");
printf("$$\n");
printf("FEDRAT/120\n");
printf("SPINDL/1200,CCW\n");
printf("INTOL/0.005\n");
printf("OUTOL/0.005\n");
printf("CUTTER/20\n");
printf("COOLNT/ON\n");
printf("\n");
printf("$$\n");
printf("FROM/SP\n");
printf("THICK/0.02\n");
for (k=1;k<=FSURF_n;k++)
 {
  printf("GO/TO,PL%d\n",k);

printf("GOFWD,PL%d,PAST,%2.3f,%2.3f,%2.3f\n",k,FIN_SURF[k].point_3.X,FIN_S
URF[k].point_3.Y,FIN_SURF[k].point_3.Z);
  }
printf("GO/TO,SP\n");
printf("THICK/0.00\n");
```

```
      for (k=1;k<=FSURF_n;k++)
       {
         printf("GO/TO,PL%d\n",k);

printf("GOFWD,PL%d,PAST,%2.3f,%2.3f,%2.3f\n",k,FIN_SURF[k].point_3.X,FIN_S
URF[k].point_3.Y,FIN_SURF[k].point_3.Z);
       }
      printf("GO/TO,SP\n");
      printf("COOLNT/OFF\n");
      printf("SPINDL/OFF\n");
      printf("FINI\n");

      fprintf(f3ptr,"PART NAME - SAMPLE PART\n");
       q =1;
       for (k=1;k<=FSURF_n;k++)
       {
       /*printf("q_is %d\n",q);*/
       fprintf(f3ptr,"P%1d = POINT/%2.3f,%2.3f,%2.3f\n",
       q,FIN_SURF[k].point_1.X,FIN_SURF[k].point_1.Y,FIN_SURF[k].point_1.Z);
       p1=q;
       q++;
       fprintf(f3ptr,"P%1d = POINT/%2.3f,%2.3f,%2.3f\n",q,FIN_SURF[k].point_2.X,
FIN_SURF[k].point_2.Y, FIN_SURF[k].point_2.Z);
       p2 =q;
       q++;
       fprintf(f3ptr,"P%1d = POINT/%2.3f,%2.3f,%2.3f\n",q,FIN_SURF[k].point_3.X,
FIN_SURF[k].point_3.Y, FIN_SURF[k].point_3.Z);
       p3 =q;
       q++;
       /*printf("PL%d = PLANE/%2.3f,%2.3f,%2.3f\n",FIN_SURF[k].FSURF_N,
FIN_SURF[k].point_1.X, FIN_SURF[k].point_1.Y, FIN_SURF[k].point_1.Z);*/
                  fprintf(f3ptr,"PL%d = PLANE/P%d,P%d,P%d\n",FIN_SURF[k].FSURF_N,p1,p2,p3);
       }
       fprintf(f3ptr,"SP = POINT/0,0,0\n");
       fprintf(f3ptr,"\n");
       fprintf(f3ptr,"$$\n");
       fprintf(f3ptr,"FEDRAT/120\n");
       fprintf(f3ptr,"SPINDL/1200,CCW\n");
       fprintf(f3ptr,"INTOL/0.005\n");
       fprintf(f3ptr,"OUTOL/0.005\n");
       fprintf(f3ptr,"CUTTER/20\n");
       fprintf(f3ptr,"COOLNT/ON\n");
       fprintf(f3ptr,"\n");
```

```
      fprintf(f3ptr,"$$\n");
      fprintf(f3ptr,"FROM/SP\n");
      fprintf(f3ptr,"THICK/0.02\n");
      for (k=1;k<=FSURF_n;k++)
       {
        fprintf(f3ptr,"GO/TO,PL%d\n",k);

fprintf(f3ptr,"GOFWD,PL%d,PAST,%2.3f,%2.3f,%2.3f\n",k,FIN_SURF[k].point_3.X,
FIN_SURF[k].point_3.Y,FIN_SURF[k].point_3.Z);
        }
      fprintf(f3ptr,"GO/TO,SP\n");
      fprintf(f3ptr,"THICK/0.00\n");
      for (k=1;k<=FSURF_n;k++)
       {
        fprintf(f3ptr,"GO/TO,PL%d\n",k);

fprintf(f3ptr,"GOFWD,PL%d,PAST,%2.3f,%2.3f,%2.3f\n",k,FIN_SURF[k].point_3.X,
FIN_SURF[k].point_3.Y,FIN_SURF[k].point_3.Z);
        }
     fprintf(f3ptr,"GO/TO,SP\n");
     fprintf(f3ptr,"COOLNT/OFF\n");
     fprintf(f3ptr,"SPINDL/OFF\n");
     fprintf(f3ptr,"FINI\n");



}
}
```

REFERENCES

Bezier, P., "UNISURF System : Principles, Program Language." Proceedings of the Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLOMAT '73, Budapest, April 10-13 1973 : 417-426.

Boulanger, P., Evans, K. B., Rioux, M., and Ruhlmann, L. "Interface between a 3-D Laser Scanner and a CAD/CAM System." Proceedings of the 5th CAD/CAM and Robotics Conference June 1986, Toronto, Canada : 731.1-731.7

Boulanger, P., Rioux, M., Taylor, J., Livingstone, F., "Automatic Replication and Recording of Museum Artifacts." Proceedings of the 12th International Symposium on the Conservation and Restoration of Cultural Property, Tokyo, Japan 1988: 131-147.

Chang, Chao-Hwa and Melkanoff, A.Michael. NC Programming and Software Design, Prentice Hall 1989.

Domey, J., Rioux, M., and Blais, F. "3-D Sensing for Robot Vision.", NATO ASI Series, Vol F64, Sensory Robotics for the Handling of limp materials: 159-192.

Duncan, J. P., Law, K. K., Computer-Aided Sculpture, Cambridge University Press 1989.

Duncan, J. P., Mair, S. G., "The Anti-Interference features of Polyhedral Machining." Proceedings of the 3rd International IFIP/IFAC Conference on Programming Languages for Machine Tools, PROLOMAT '76, Stirling, Scotland, 15-18 June 1976 : 181-195.

Duncan, J., and Mair, S. Sculptured Surfaces in Engineering and Medicine. Cambridge University Press, 1983.

Encarnacao, J., Schuster, R., Voge, E., Product Data Interfaces in CAD/CAM Applications, Design Implementation and Experiences, Springer Verlag 1986.

Flutter, A., "The POLYSURF System." Proceedings of the Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLOMAT '73, Budapest, April 10-13 1973 : 403-416.

Kanade, Takeo. Three Dimensional Machine Vision, Kluwer Academic Publishers 1987.

Livingstone, F. R., and Rioux, M. "Development of a Large Field of View 3-D Vision System." SPIE Proceedings 665 June 1986: 188-194.

Marshall, F. Gerald, Laser Beam Scanning, Marcel Dekker Inc 1985.

Mortenson, Michael. Geometric Modelling, John Wiley and Sons 1985.

Nasser., Daniel. "Non-Contact, Three Dimensional Object Digitizing Systems." Thesis, University of Central Florida, Fall 1989.

Nicolo, V., and Piccini M., "Interactive Curve Fitting." Proceedings of the Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLOMAT '73, Budapest, April 10-13 1973 : 427-438.

Rioux, M., Blais, F., Beraldin, J. A., Boulanger, P., "Range Imaging Sensors Development at NRC Laboratories." Proceedings of the Workshop on Interpretation of 3D Scenes, Austin, Texas Nov 27-29 1989: 154-160.

Sanz, L. C., Advances in Machine Vision, Springer Verlag 1988.

Taylor, J. M., et al., "Application of a Laser Scanner to Recording and Replication of Museum Objects." 8th Triennial Meeting of the ICOM Committe for Conservation, Sydney, Australia, September 6-11 1987 : 93-97.

Hosni.,Yasser, Hwang.,Jueng-Shing, and Ferreira, Labiche. "Tool Path Generation from Surface Mapping of an Object." Proceedings of PROCIEM '90, Tampa, Florida, November 11-13 1990 : 23-27.

Yoshiaki Shirai, Three Dimensional Computer Vision, Springer Verlag 1987.