Tool Path Generation for Rough Machining Using STEP

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the
degree of Master of Applied Science in
Mechanical Engineering

By
Syed Shafee Ahamed

Ottawa-Carleton Institute for
Mechanical and Aeronautical Engineering
University of Ottawa
Ottawa, Ontario, Canada

ISBN 0-612-04935-3

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

The production efficiency has been significantly improved over the years due to the rapid advancement in CAD and CAM. Very often, however, the production cycle is still unacceptably long. This has been frequently caused by the bottleneck link between CAD and CAM. The CAD design model cannot be directly machined by a CNC machine without an NC program, an NC program cannot be coded without a clearly defined tool path, and the CAD data are not readily useable for preparing tool path. A large amount of time is often spent on data exchange, NC tool path generation, and NC program preparation. To further reduce the production cycle, the CAD model has to be documented in a neutral format such that it can be readily used in CAM and the tool path should be automatically generated from the CAD model in a neutral format. The recently developed STEP data exchange standard provides such a neutral format that is independent of any software and hardware. However, none of the existing tool path generation systems is based on STEP.

In view of the above facts, a STEP based tool path generation system is developed for rough machining of planar surfaces. This system comprises four modules. The first one is the data extraction module used to obtain geometric data from STEP. The second module is the slicing module which is designed to convert the three dimensional tool path generation problem into a series of two dimensional problems. The tool path will be generated by the third module. The tool path is then translated into NC program by an NC code generator -- the last module.

The proposed system has been implemented in a UNIGRAPHICS CAD/CAM environment. The application of the system is illustrated using an example part.

Acknowledgements

I would like to express my sincere appreciation and gratitude to my supervisors, Dr. Ming Liang and Dr. Tet Yeap for their continued guidance and efforts throughout my graduate studies.

My special thanks to Mr. Bert van den Berg, Research Officer, National Research Council, Ottawa for his support and encouragement during the project. I am also thankful to all staff at NRC for their valuable suggestions during this research.

Also, I would like to thank the members of my defence committee, Dr. Dan Necsulescu and Dr. Geza Kardos, for their time and effort in reviewing and examining my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Dedication

to

My dear father

Chapter 1

INTRODUCTION

1.1 Background

Design and manufacturing are two major stages in the production process. The progress in computer-aided design (CAD) and computer-aided manufacturing (CAM) over the past few decades has significantly improved design and manufacturing efficiency. However, the total production cycle is still unacceptably long in many cases. The potential for further improving design or manufacturing efficiency separately is approaching its saturation point. With today's technology, a part designed by a CAD system cannot be directly machined on a CNC (computerized-numerically controlled) machine. A set of NC (numerical control) programs, i.e., numerical instructions, has to be prepared to drive the CNC machine. There exist alternative ways to machine a part. A specific tool path, the motion trajectory of the cutter center to be followed in the machining process, has to be clearly defined before preparing the NC programs. Traditionally, the tool path and NC code are manually prepared. Since every single cutting segment of a tool path and associated feedrate, depth of cut, spindle speed must be specified, an NC program can easily reach a few thousands lines. This process is thus very time-consuming, error-prone, and skill-dependent. Moreover, as the CAD design model is usually documented in a particular data format, the geometric and topological message may not be readily "visualized" or understood by a tool path generator (a system or a human programmer).

Therefore, the information needed for tool path generation has to be extracted from CAD file and converted into useable format. Often, the time needed in generating a tool path and preparing a NC (Numerical Control) program for a part is considerably longer than the actual machining time. Also, it is not unusual for a company to spend more time transferring data from a CAD system to a CAM system or communicating between different CAD systems than the actual design time (Vergeest 1991). These middle links between CAD and CAM are the bottleneck for further improving production efficiency.

During the last few years various automatic tool path methods have been developed. Several tool path generation systems are commercially available. However, our extensive literature survey and the market study conducted jointly with the Institute for Advanced Manufacturing Technology, National Research Council of Canada, indicate that most publications pertinent to tool path planning did not address data exchange issues and all of existing systems are based on "local" data exchange standards such as IGES (Initial Graphics Exchange Specifications). Since these local data exchange standards are not compatible, and mostly contain only design related information, several intermediate data exchanges may be needed between different systems. This is again error-prone and will unnecessarily prolong the production cycle.

A new international data exchange standard, STEP (STandard for Exchange of Product data), has been recently developed to gradually replace all the currently used data exchange standards. This new standard provides an efficient vehicle for transferring product data covering the entire life cycle. If the design or path planning data are documented in STEP format, the time needed for data communication process can be

considerably reduced since STEP standard has a neutral format which is independent of any software package and unrestricted to any particular hardware platform. Implementing STEP standard will undoubtedly facilitate the integration of CAD and CAM since they can virtually be treated as a single system. However, none of the existing tool path generation systems in today's market is based on STEP. A STEP based tool path generation system is also highly desirable for further reducing production cycle time.

1.2 Objective

The above background information clearly indicates the need for the development of a STEP based tool path generation system. However, since the generation of tool paths for various parts is a very complex issue and STEP standard is a huge document containing a great mass of information, it may not be realistic to develop a general tool path planning system within the limited period of time. Literature has shown that rough machining usually demands more machining time as compared with finish cutting. In most cases, the machining time for rough machining is five to ten times that spent in finish machining (Gunsekera 1989). Further, most of the three-dimensional rough machining problems can be converted into two-dimensional problems by slicing the machining volume into parallel planes. This thesis therefore will focus on STEP based tool path generation system for rough machining of planar surfaces. To achieve this goal, the following components will be developed and integrated:

. a data extraction module,

. a part slicing module,

. a tool path generation module, and

. an NC code generator.

1.3 Structure of the thesis

The rest of the thesis is organized as follows: Chapter 2 provides a brief review of tool path related research and some existing data exchange standards. A brief introduction to the STEP structure and the algorithms required for the data extraction from STEP, part slicing, machining area definition, tool path generation, and elimination of redundant data points from CL file are presented in Chapter 3. Chapter 4 explains the implementation of these algorithms. An application example is also included in Chapter 4. Chapter 5 concludes this thesis and recommend some future work.

Chapter 2

LITERATURE REVIEW

Tool path planning is an important link for CAD/CAM integration and has drawn increasing attention in both academic and industrial communities over the last decade. Traditionally, tool paths and required CNC codes are manually prepared based on the CAD design files. This is often very time-consuming, skill-dependent, and error-prone. The CAD/CAM integration and manufacturing productivity are thus severely hindered. Further, due to the lengthy programming process, alternative tool paths may not be considered. A well planned tool path pattern can significantly reduce cutting time for rough machining. Surface quality can be improved using a proper tool path in finish cutting. All of these, however, rely on the availability of a system that can quickly generate tool paths. During the past few years, various tool path generation approaches have been proposed. The following sections provide a brief review of the related work in the literature.

2.1 Tool Path Generation Methods

The existing tool path generation methods can be classified based on machining

approaches and tool path patterns. There are two types of machining approaches: offsetting and contour-map (Figure 2.1).

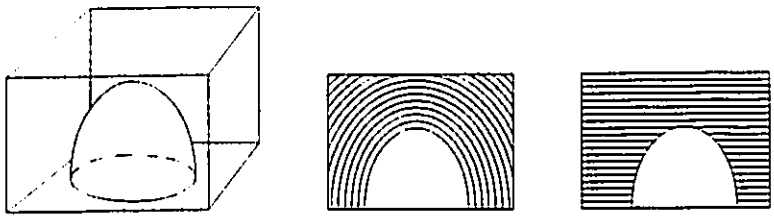


Figure 2.1 Two types of machining approaches

Most of existing tool path patterns fall into five categories, i.e., staircase, window-frame (or spiral), space-filling, Voronoi diagram, and isoparametric based patterns. Different tool path patterns may be used by a particular machining approach. They are reviewed as follows.

2.1.1 Offsetting and Contour-map Machining

1. Offsetting approach

The offsetting approach is mostly used for sculptured parts or free-form surfaces. With this method, the cutting surfaces are obtained by offsetting a certain distance from the sculptured part surface until the specified machining volume has been completely removed. Ball mills are often used. In addition to calculate cutter location (CL) data, cutter contact (CC) points or point trajectories, which represent the offset surfaces of the

sculptured part, also have to be determined (Choi and Lee 1988, Kim and Biegel 1988). The generation of this type of tool path is usually very time-consuming. Particularly, as Dong et al (1993) indicated, when the shape of the sculptured part and shape of the stock are significantly different, a considerable amount of cutting time may be wasted due to the blank cuts or over milling. Therefore, offsetting approach is primarily used for finish machining and is more concerned with cutting accuracy.

Bobrow (1985) reported a tool path generation scheme using the offsetting approach. The main component in this scheme is an algorithm for generation of gouge-free tool paths.

Along this line, Choi et al (1994) developed a unified CAM system for die and mould manufacturing based on the offsetting approach. To achieve the desired accuracy, several modules have been incorporated in their unified CAM system. One of the major functions of the modules is to ensure a gouge-free tool path.

Zhu (1991) proposed to improve surface accuracy by generating several alternative tool paths. By using Coons patch equations, several alternative tool paths can be generated and the best one is selected.

To improve cutting efficiency of the offsetting approach, Marshall and Griffiths (1994) proposed a modified zigzag tool path for machining the free-form surfaces. Particular attention was paid to the reduction of rapid traverses. Elber and Cohen (1994) suggested that the cutting efficiency be improved by reducing blank cuts. Adaptive isocurves are applied for this purpose. The implementation of this approach was also

demonstrated.

Vickers and Quan (1989) investigated the possibility of using end mill, rather than ball mill, to improve productivity. Further studies have been also carried out by their group for reducing machining time. A circular interpolation method was suggested (Vickers and Bradley 1992). They have demonstrated that the machining time can be reduced up to 67% by using their circular interpolation algorithm.

Hwang (1992) studied a special type of surfaces: parametric compound surfaces. In this case, the surface to be machined consists of a number of topologically unrelated parametric surfaces. His focus was on the generation of planar CL paths rather than planar CC paths whenever possible. The logic is that it is easier to control NC machine feedrates along the planar path.

In addition to the machining accuracy and efficiency, generation of tool path for complex parts is also of particular interest in offsetting approach. The related work is however quite limited in the literature. Recently, Kim and Jeong (1995) reported a path generation algorithm for machining free-form pockets with multiple islands. Since their focus was on the automatic generation of complex tool paths, the machining time and accuracy were not reported.

2. Contour map approach

This method logically slices a stock into a certain number of parallel layers. The slicing direction will depend on the cutting orientation. The layers are perpendicular to

the spindle direction. The thickness of each layer can be determined by the depth of cut. In this way the three-dimensional problem reduces to a two-dimensional problem and the tool path generation is simplified. Further, since the tool motion is on a two-dimensional plane, more efficient tool paths can be selected. Due to these advantages, contour map approach is mostly used to remove large volume of material in rough cutting. The main objective in this case is to maximize material removal rate or minimize cutting time. Since most of the studies pertinent to contour map cutting focus on the selection of tool path patterns, only a few publications will be cited below to avoid duplication with the review of tool path patterns.

Wang et al (1987) were probably among the first to systematically study the cutting efficiency problem in face milling, a special case of the contour map cutting. They examined the cutting efficiency using the two most commonly used path patterns: staircase milling and window frame milling. Some guidelines are suggested based on their findings.

Dong et al (1993) proposed to minimize manufacturing time by optimizing the feedrate and number of cutting layers. They have shown that the production time can be reduced up to 76% for a half-ellipsoid part and 47% for a ship hull.

An interesting issue to be addressed in slicing is how to handle the steps left after rough cutting. This is particularly important when surface curvature changes drastically. Selecting thicker layers in roughing can usually reduce machining time but will inevitably increase the burden of finish cutting. The trade-off between the saved rough cutting time

and increased finish cutting time should be examined. A good attempt has been made recently by Dolenc and Makela (1994). Though their approach cannot be used for general rough cutting problems, they have proposed a fairly efficient way to handle a class of problems such as the surfaces with steps.

The octree method (Figure 2.2) originally introduced by Yamaguchi (1984) and Yuen (1987) and recently reexamined by Lee et al (1994) represents another type of slicing technique. With this technique, a hexahedron is first created which completely contains the part to be cut. Then eight smaller hexahedra are obtained by equally dividing the length, width, and height of the hexahedron. A hexahedron is called a full octant if it is completely contained by the volume to be removed, empty octant if it is entirely outside of the volume and partial octant otherwise. The full and empty octants are not

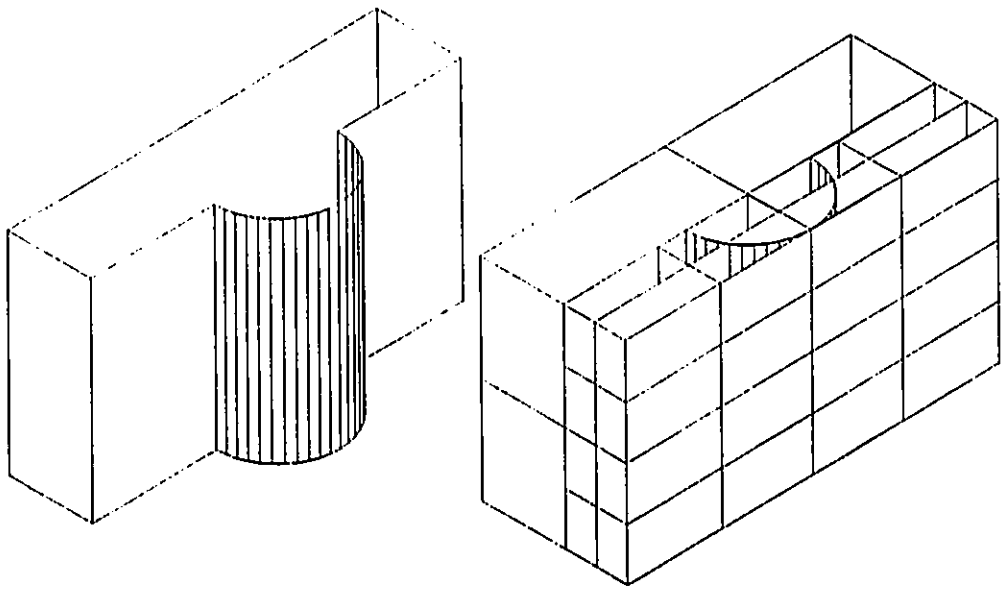


Figure 2.2 Octree method

split. The partial octants are further divided and the newly obtained smaller octants are again classified as full, empty and partial octants. This procedure is repeated until the desired resolution level has been reached. The tool, the width of cut, and the depth of cut will be determined by the full octant size. Since different octants are produced, the depth of cut may be different at different locations, rendering non-uniform thickness of a layer. The main benefit of this method is its adaptability to the surface shape. The large step leftover for the finish cutting may be avoided using the octree method. Further, the tools can be selected and changed according to the octant sizes. However, as pointed out by Bala and Chang (1991), it may not be a good manufacturing practice to make many tool changes while machining a feature.

2.1.2 Tool Path Patterns

Tool path pattern has a significant impact on the machining efficiency and surface quality. Much work has been carried out to investigate path pattern effects and to select the best patterns for various machining scenarios. This section presents a brief review of the five most widely cited and frequently used tool path patterns.

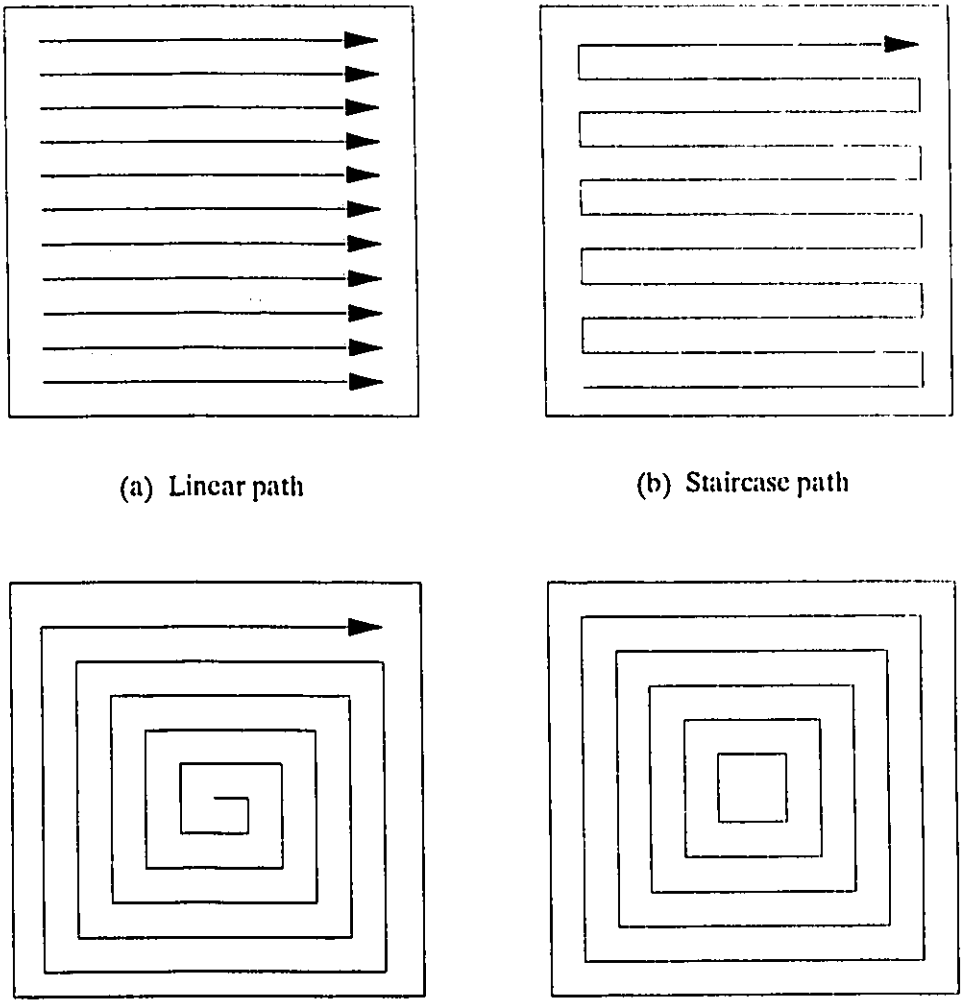
1. Staircase pattern (Figure 2.3 a, b)

This pattern is the simplest and yet most widely used due to its efficiency. In this pattern, the tool moves along parallel passes until the surface to be machined has been entirely covered. This pattern can be achieved by either uni-direction cut (linear path,

Figure 2.3 a) or dual-direction cut (zigzag path, Figure 2.3 b). In the case of uni-direction cut, material is removed in forward direction only and the cutter is retracted rapidly in the reverse direction. Though the rapid retraction results in additional travel distance, it does provide a smoother machining process and is also advantageous to tool life due to the lower contact impact with part surface (Kalpakjian 1992, Raman and Lakkaraju 1991, 1992). Recently, Weck et al (1994) reported that linear path yields shorter cutting time than zigzag and window frame paths.

The pattern obtained by the dual direction parallel cuts is also called zigzag pattern. A series of publications of Wang et al (1987, 1988) pioneered the analytical study of the efficiency problem in staircase milling. In these papers, they particularly investigated the effects of the cutting orientation on the total length of cut in face milling. Their analyses show that the cutting orientation has a significant impact on the total length of cut with an average variation of 5-10% and a maximum variation of 100%. They also observed that, for regular polygons, the optimal cutting orientation is normally parallel to the longest edge of a given polygon.

The above observations were confirmed by Lakkaraju et al (1992), and Raman and Lakkaraju (1993). They pointed out that, in face milling, the total length of cut is also a function of the tool diameter. This may be considered as an important supplement to the work of Wang et al (1987, 1988).



(a) Linear path

(b) Staircase path

(c) Spiral path

(d) Window frame path

Figure 2.3 Tool path patterns

Applications of the staircase pattern in planar surface milling have also been reported by Bala and Chang (1991), Lakkaraju and raman (1990), Deshmukh and Wang (1993), Tseng and Joshi (1994), among others.

Recently, several researchers have examined the possibility of using staircase pattern for machining complex surfaces. For example, Choi et al (1994) applied staircase pattern to die and mould manufacturing. Marshall and Griffiths (1994) generated tool path for free-form surface with several pockets and a single protrusion using the combined staircase and contour passes. Kim et al (1995) also employed the staircase path for three dimensional surface cutting. In their work, particular attention was paid to generating a gouge-free or overcut-free path.

2. Window frame pattern

Window frame milling is probably the second most popular tool path pattern. In this method, the tool passes are parallel to the contour of the periphery of the surface to be machined. The tool passes offset inward (or outward) equidistantly (Figure 2.3 c) or progress inward (or outward) in a spiral fashion (Figure 2.3 d, in this case, it is also called spiral milling) until the surface has been completely covered.

Wang et al (1987) developed an algorithm to generate window frame tool paths. The simulation results based on their algorithm show that the total length of cut is not

significantly affected by the starting point. The comparison between staircase and window frame patterns was also made. They found that the optimal length of cut generated by staircase pattern was better than that obtained using window frame milling but the average results from staircase milling may be worse than those from window frame pattern.

Raman and Lakkaraju (1993) indicated that the length of cut of a window frame path can be reduced by properly selecting machining parameters such as spindle speed and feedrate. Their study also implied that the window frame pattern may be good for tool life due to the reduced number of tool entrances and exits.

The window frame pattern has been mostly used in rough milling and pocketing. Related work includes Tseng and Joshi (1994), Tarng and Shyur (1993), Satyanarayana et al (1990), and Suh and Lee (1989).

3. Space-filling curves (Figure 2.4)

In this method, a particular curve pattern is repeatedly used to sweep a specified surface region. Different curve patterns may be applied to different regions of the same part. The coverage density of a curve pattern may vary depending on the curvature and resolution requirements. The space-filling curves are mostly applied in computer graphics. Earlier work in this field includes the studies by Null (1971), Goldschlager (1981), Witten and Wyvill (1983), and Griffiths (1985).

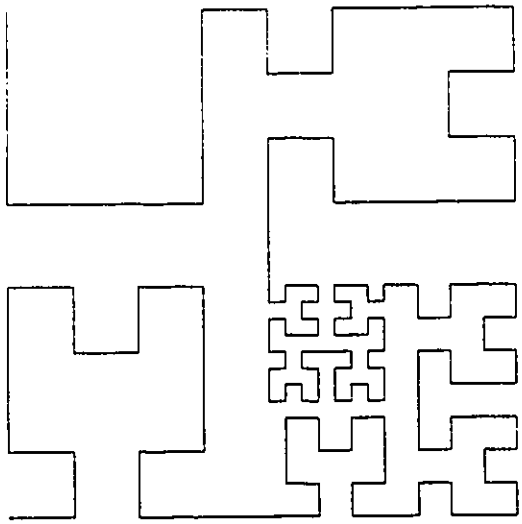


Figure 2.4 Space filling curves

Recently space-filling has been used as a class of NC tool path pattern. However, due to the complexity and the low machining efficiency, its application in NC machining has been very limited. Reported studies include Cox et al (1994) and Griffiths (1994). Cox et al (1994) successfully implemented several space-filling algorithms for NC tool path generation. They compared the quality of surfaces generated using space-filling method and conventional methods and the results seemed to be favourable for space-filling method. Griffiths (1994) suggested an adaptive approach to generate space-filling curves. The main idea is to intensify the density of the filling curves in the area where surface quality requirement is high and reduce the filling density in the region where

quality requirement is low. However, the time for preparing such a complicated tool path could be very high. In addition, the large number of stops and turns in a tool path generated using space-filling curves will inevitably increase cutting time. For these reasons, the industrial application of this type of path patterns is still limited.

4. Voronoi diagram (Figure 2.5)

The Voronoi diagram is used to divide a planar shape into several zones -- Voronoi regions. The planar is bounded by straight lines, curves, or their combination. These boundary straight lines or curves are called elements. The partition of the regions is performed in such a way: a) every region is the immediate neighbour of a part boundary element; and b) a point on the common boundary of two adjacent regions is defined such that the two regions boundary elements are equidistant from this point. Only two publications (Held 1993, Held et al 1994) are available in the accessible literature. Held et al (1994) generated tool paths based on the Voronoi diagram using window frame and local window frame patterns. They concluded that Voronoi diagram can significantly reduce computational time in tool generation. However the required cutting time is not reported.

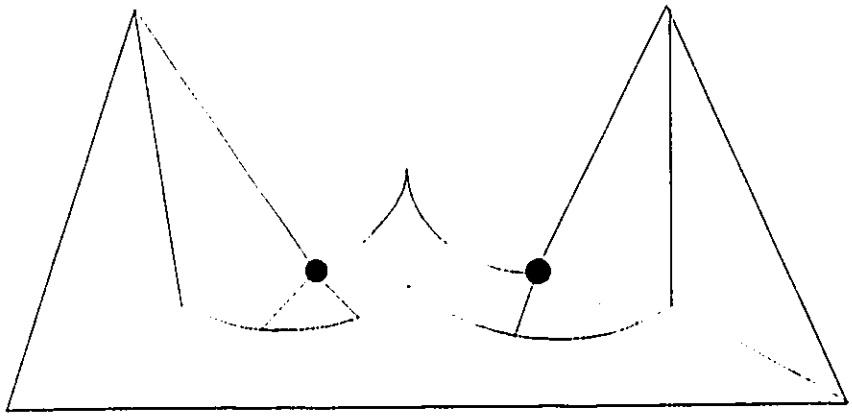


Figure 2.5 Voronoi diagram

5. Isoparametric path

In this case, the tool path segments are parametrically parallel to each other (Broomhead and Edkins 1986, Choi 1988, Marshall and Griffiths 1994). This type of tool path pattern is mainly used for free-form surface machining. Actually, the staircase path is a special case of isoparametric path when the surface is a plane. This path pattern is often criticized due to the overlapped tool path or excessive blank cuts.

To alleviate the blank cut problem, Elber and Cohen (1994) recently proposed to use adaptive isoparametric curves. The main idea is to reduce the density of tool path in

bottleneck-like area. The implementation of this method has been also reported.

The above path patterns may also be used jointly to form a combined pattern. The combined patterns are particularly useful for parts with islands. Research on this issue is however very scarce. Reported work includes the study by Vickers et al (1992). They applied the combined tool patterns for parts with single island. Six different combinations of staircase and window frame patterns were proposed. Some guidelines were also suggested for a particular machining problem.

2.1.3 Comments

Most of the existing tool path generation systems are based on DXF, IGES or other local data exchange standards. These standards, however, will gradually be replaced by an international standard -- STEP. Extensive search of the accessible literature indicates that the STEP based tool path planning system is not yet available. Development of a such tool path planning system is therefore highly desirable.

2.2 Data Exchange Standards and Related Research

There are many CAD/CAM systems commercially available in today's market. Different CAD/CAM systems may be used by the designer and the manufacturer in the same enterprise. The designers of different companies may also use different systems. Further, as indicated by Vergeest (1991), "different parts of one product may, for reasons of optimization, be designed with different types of CAD systems". To reduce unnecessary

duplication in design and drafting, the need for product data exchange between systems becomes indispensable. To this end, a neutral mechanism, or data exchange standard, has to be developed. The product data can be easily communicated between different systems if the product data are first documented in such a neutral format.

Over the past three decades, several data exchange standards have been developed (Mufti et al 1990, Brandli and Mittelstaedt 1989, Bloor and Owen 1991, Shah and Mathew 1991, and Laurance 1994). However, most of them have a limited application domain and are local in nature. For example, DXF (Data eXtract Format) proposed Autodesk Inc. is mostly used for transferring data between different versions of AutoCAD, IGES (Initial Graphics Exchange Specifications) initiated by the National Standards Institute and Technology can be used only for design data exchange between different CAD/CAM systems, PDDI (Product Definition Data Interface) by the U.S. Air Force is aerospace-oriented, PDES (Product Data Exchange Specifications) is popular in the U.S. automotive industry, SET (Standard d'Exchange er de Transfer) is used in France, VDAFS (Verband der Automobile Industrie Flachen Schnittstelle) is a German native, and CAD*I (CAD Interface) is mostly adopted by European countries. Further, most of the above data exchange standards are design-oriented and none of them can provide comprehensive information over the life cycle of a product. This will inevitably hamper the increasingly needed CAD/CAM integration and concurrent engineering processes. Moreover, these standards lack compatibility. Product data file in a particular standard format may not be or at least cannot be completely translated by

another standard. This has caused unnecessary confusion and duplicated data exchange effort. For the above reasons, STEP (STandard for Exchange of Product data) is developed.

STEP is intended to replace all other data exchange standards. The STEP standard provides a common mechanism representing the product model data throughout the life cycle of a product. The STEP standard has a neutral format which is independent of any application software that may be used to process or develop the product. STEP standard is also independent of the hardware platform on which it is used as long as it can be executed. Thus STEP provides an effective data exchange vehicle. The current version of STEP consists of a number of sections or parts as shown in Table 2.1. (ISO 1992)

Of these parts, Part 42 is of particular interest in tool path planning. The geometric data such as axis, point, vector, surface and topological information such as vertex, edge, loop, shell are included and defined in the EXPRESS (a language used in STEP) format in this part. More detailed reviews of STEP are available in (Bloor and Owen 1991, Vergeest et al 1991, and Laurance 1994). A brief description of STEP data structure will be provided in Chapter 3 of this thesis.

Some STEP related studies have been carried out in the recent years. However, with possible exception of Gu and Chan (1995), most of the studies conducted so far focus on the explanation of the STEP structure, comparison between STEP and other data exchange standards, and conversion of data of other standards into STEP format. The recent work of Gu and Chan (1995) is probably among the first that attempted the

application of STEP to production problem. In their paper, a STEP based product modelling system consisting of a generic product model, a model database and a modelling/design interface is reported.

PARTS	DESCRIPTION
1	Overview of the entire standard
11 to 20	Intent of STEP (Part 11, covers the EXPRESS language)
21 to 30	Implementation methods (part 21, represents the format for expressing information)
31 to 40	Testing implementations (part 31, an overview of testing)
41 to 100	Generic resources (part 42, for geometry and topology)
101 to 200	Application resources
201 to 1200	Application protocols (part 201, for drafting, part 203, for design of mechanical products)

Table 2.1 Parts/Sections in STEP standard

2.3 Motivation

Most tool path related studies focus on machining efficiency or machining surface quality. The time spent for data exchange from CAD to CAM system, however,

represents a sizeable portion of the entire production cycle. Reduction of this portion of time and avoidance of the error in the data transfer process have been traditionally overlooked. STEP data exchange standard has emerged as an efficient means to minimize data transfer time and reduce transfer errors. As reviewed above, none of the reported tool path generation systems is based on STEP. This fact is also revealed by our market survey conducted jointly with the Institute for Advanced Manufacturing Technology, National Research Council of Canada. The application of STEP, though still in its infant stage, is becoming indispensable as the STEP standard will gradually replace all the other data exchange standards (the 1995 version of the most popular standard IGES will be the last version (Conkoi 1994)). This thesis therefore aims at the development of a STEP based tool path generation system.

Chapter 3

PROPOSED PROCEDURE FOR TOOL PATH GENERATION

In this chapter, the proposed approach to generate tool paths from a CAD model via STEP data exchange standard will be elaborated. The chapter starts from a preliminary introduction to STEP data structure. The methodology and algorithm for extracting geometric data from STEP file are then presented. Next, a method used to slice the three dimensional CAD model is illustrated, followed by a 4-direction strategy employed to generate the tool paths for each slice. Finally, a procedure for eliminating redundant path definition points in a CL (cutter location) file is also discussed. The overall structure of the proposed tool path planning procedure is shown in Figure 3.1 and the details of each block are discussed in the following sections.

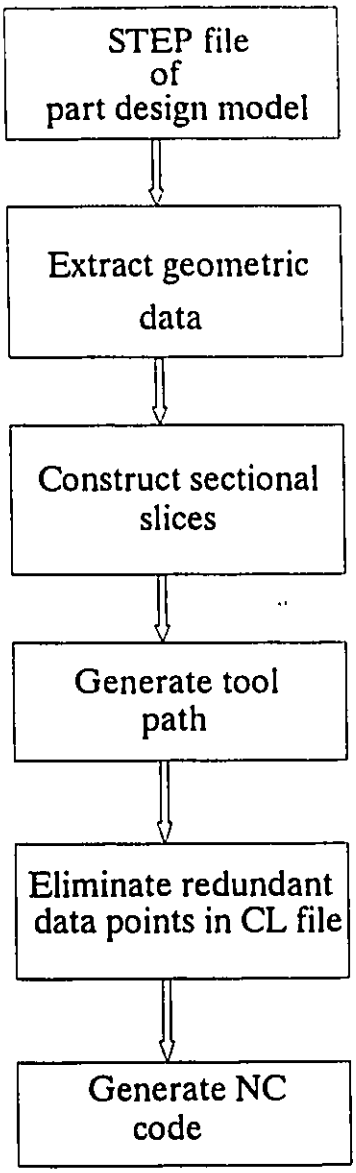


Figure 3.1 Overall structure of the proposed tool path planning

3.1 STEP Data Structure

STEP is a data exchange standard used to convert a given CAD model into an ASCII text file in a neutral format. The output file from STEP includes three major divisions: header, geometric section and topological section. They are arranged in a sequential order. Details of STEP translator version and CAD software used are included in the header section. The geometric section consists of Cartesian coordinates of all vertex points in the CAD model as well as other data such as planes, directions and axes. A geometric or topological entity such as a line, a loop, or a surface, is called a functional element in STEP. Many functional elements are used in STEP to accurately define the geometry and topology of a product. Information for connectivity of these functional elements is nested in the topological statements. This information is expressed as pointers which are essentially numbers preceded with # symbol. The pointers are arranged in an increasing order (ISO 1994). However, the value of a pointer has no topological or geometric meaning. A pointer is simply used as an address of a functional element. Further, it should be pointed out that the geometric section and topological section are not physically separated in a STEP file. In other words, both geometric and topological statements can co-exist in a STEP file segment.

Some STEP terms or functional elements that are of interest in this study are briefly explained below. More detailed definitions and explanations are provided in ISO/TC/184/SC4 N142 standard (ISO 1992).

CLOSED_SHELL	A collection of one or more faces which bounds a region in three dimensional space and divides the space into two regions, one finite and the other infinite.
FACE_SURFACE	A type of face in which the geometry is defined by the associated surface, boundary and vertices.
FACE_BOUND	A loop used for bounding a face.
EDGE_LOOP	A path in which the start and end vertices are the same.
ORIENTED_EDGE	An edge constructed from another (original) edge and contains a direction (orientation) information. The ORIENTED_EDGE will be equivalent to the original edge if the orientation information is not included.
EDGE_CURVE	A type of edge which has its geometry fully defined.
VERTEX_POINT	A point defining the geometry of a vertex.
CARTESIAN_POINT	Address of a point in Cartesian space.

The format of the above elements are as follows (note: only the information that is useful for tool path generation is included here):

CLOSED_SHELL ((pointer *1*, pointer *2*, ..., pointer *n1*));

where, the *n1* pointers are addresses of *n1* FACE_SURFACEs

FACE_SURFACE ((pointer *1*, pointer *2*, ..., pointer *n2*), pointer *n2+1*);

where, pointer *1* represents the address of the FACE_OUTER_BOUND, pointer

2, ..., pointer *n2* are the addresses of the *n2-1* internal face boundaries. Each FACE_SURFACE can have no more than one FACE_OUTER_BOUND. Pointer *n2+1* is the address of the associated surface, where, the surface type, plane or curved surface, is specified.

FACE_BOUND (pointer);

The only pointer in this case is the address of the EDGE_LOOP.

EDGE_LOOP ((pointer *1*, pointer *2*, ..., pointer *n3*);

The *n3* pointers are the addresses of the *n3* ORIENTED_EDGEs.

ORIENTED_EDGE (pointer);

The only pointer is the address of the EDGE_CURVE.

EDGE_CURVE (pointer *1*, pointer *2*, pointer *3*);

The first two pointers are the start and end points (vertices) of the edge. The third pointer specify the address where the curve type is defined, e.g., line or curve.

VERTEX_POINT (pointer);

The single pointer here is used to specify the address of a CARTESIAN_POINT.

CARTESIAN_POINT ((*X,Y,Z*));

Where *X,Y,Z* are the coordinates of the associated vertex.

In the above list only CARTESIAN_POINT is a geometric functional element and the others are topological functional elements. The STEP data structure is arranged in a top-down fashion. All the entities at the lowest level of a product hierarchy such as vertex points and their coordinates are first defined. The entities at the second lowest level such as lines and edges are defined next. Then the entities at the other levels are

sequentially defined according to their location in the product hierarchy. The entity at the top of the hierarchy, i.e., the entity representing the final product is defined last. The principle is that an entity can be defined only when all the its lower level entities have been defined. To illustrate, some statements excerpted from a STEP file are listed below:

#110 = CARTESIAN_POINT ((0.75, 1.5, 0.5));

#120 = VERTEX_POINT (#110);

#130 = CARTESIAN_POINT ((0.75, 1.5, 2.0));

#140 = VERTEX_POINT (#130);

#150 = CARTESIAN_POINT ((0.987308004315134, 3.53449314192417, 1.));

#160 = VERTEX_POINT (#150);

#170 = CARTESIAN_POINT ((0.280201223128585, 2.82738636073763, 1.));

....

#2510 = CARTESIAN_POINT ((0.987308004315134, 3.53449314192417, 0.75));

#2520 = DIRECTION ((0., 0., 1.));

#2530 = VECTOR (#2520, 1.);

#2540 = LINE (#2510, #2530);

#2550 = EDGE_CURVE (#160, #220, #2540, .T.);

....

#9390 = EDGE_LOOP ((#9350, #9360, #9370, #9380));

#9400 = FACE_OUTER_BOUND ((#9390, .T.));

#9410 = CARTESIAN_POINT ((0.625, 4.5, 1.9375));

#9420 = DIRECTION ((-1., 1.42125554512318E-15, 0.));

#9430 = DIRECTION ((0., 0., 1.));

#9440 = AXIS2_PLACEMENT_3D (#9410, #9420, #9430);

#9450 = PLANE (#9440);

#9460 = FACE_SURFACE ((#9400), #9450, .T.);

....

#12560 = FACE_SURFACE ((#12500), #12550, .F.);

#12570 = CLOSED_SHELL ((#6960, #7080, #7200, #7310, #7420, #7540, #7660, #7780, #7900, #8010, #8120, #8240, #8360, #8480, #8600, #8720, #8860, #8980, #9100, #9220, #9340, #9460, #9580, #9700, #9820, #9940, #10060, #10180, #10300, #10420, #10540, #10760, #11230, #11350, #11470, #11590, #11750, #11870, #11960, #12080, #12200, #12320, #12440, #12560));

It can be seen from the above, pointer #12570 is the address of the final product. In STEP it is topologically represented as a closed shell. A number of lower level pointers representing the connected faces are nested in this statement. Similarly, pointer #9460 is the address of a surface. This surface is further defined by pointers #9400 and #9450. In address #9450, the surface is specified as a plane and the boundary of the plane is defined by the information in address #9400. Further tracing to #9390, we can find four address pointers #9350, #9360, #9370, and #9380 from which the four coordinates can be located after a few more levels.

Therefore, the entire STEP file of a product is logically arranged in an inverted tree-like structure with a functional element CLOSED_SHELL at the top and CARTESIAN_POINT at the bottom. Other functional elements such as

FACE_SURFACE, FACE_BOUND, etc are located in between in a hierarchy manner. This structure is illustrated in Figure 3.2.

STEP standard provides a large amount of product information throughout the life cycle of a product, which includes the data needed for design, manufacturing and business applications. Thus it is essential to develop an efficient data extraction method for specific applications such as process planning and tool path planning. For this purpose, a hierarchy based approach is proposed for data extraction in this research. Based on this structure, the extraction of the required data from a STEP file can be performed starting from the CLOSED_SHELL, and moving down towards the CARTESIAN_POINT. In this process all the associated pointers are traced to ensure the proper connectivity among different functional elements. Details of this procedure are explained in the next section.

3.2 Geometric Data Extraction

The data extraction method proposed in this research uses the geometric and topological sections of the STEP file. As the first step, the number of pointers used to construct the CLOSED_SHELL is counted. Since each pointer represents one FACE_SURFACE, the number of pointers equals the number of FACE_SURFACEs in the model. Then each of these FACE_SURAFCE pointers is traced to get address of its FACE_BOUND pointer. The FACE_BOUND contains the pointer address of an EDGE_LOOP. This pointer is then traced to get the set of pointers required to define the associated ORIENTED_EDGEs. Each ORIENTED_EDGE provides the information of pointer

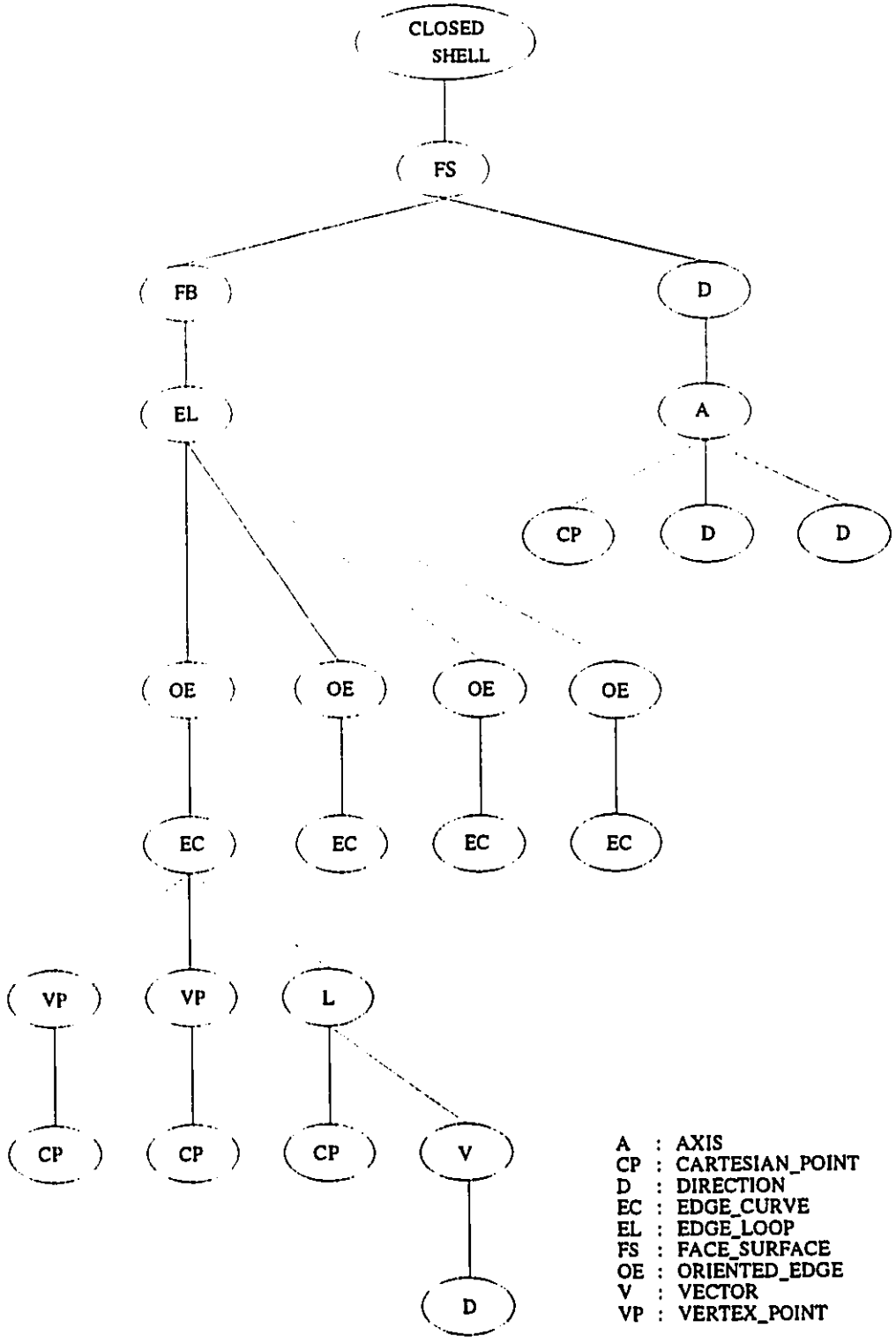


Figure 3.2 Hierarchy of functional elements in STEP

address for the EDGE_CURVE. This pointer leads to a VERTEX_POINT which in turn provides pointer address to the corresponding CARTESIAN_POINT. Finally the pointer address of CARTESIAN_POINT is traced to obtain the coordinates of a point in three dimensional space.

While tracing down the tree in search of pointers for the next hierarchy element, pointers associated with each element is stored. These pointers establish the connectivity in branching. For example, the number of pointers used to form a EDGE_LOOP determines the number of ORIENTED_EDGEs that have to be traced down. The number of associated pointers also indicates the shape of the boundary (e.g., a four-pointer EDGE_LOOP indicates a rectangle and a five-pointer EDGE_LOOP means a pentagon, etc.).

A rectangular box (Figure 3.3) is chosen to illustrate the data extraction process. The STEP file of the box is attached in Appendix 1. Note that the STEP file of this box is in old STEP format and the pointers are not necessarily arranged in an increasing order. The functional elements of the box such as CLOSE_SHELL, FACE_SURFACE, etc are explained in Figure 3.4. The data extraction procedure is depicted in Figure 3.5.

The box is represented as a CLOSED_SHELL shown in STEP file as "#145 = CLOSED_SHELL ((#100, #116, #132, #144, #80, #44))". Here #145 is the address of the box. The six pointers nested in the statement: #100, #116, #132, #144, #80, and #44, are the address of the six faces or FACE_SURFACEs of the box. The coordinates of the eight vertices of the box can be obtained by tracing down these six pointers. For example, along the left most branch, pointer #100 leads to Cartesian pointer #17. In

address #17, the coordinate values of a vertex are located: (100, 0, 0). #17 appears several times at the bottom of the tree. This is simply because the associated point is the vertex of several connected surfaces. The entire tree can be obtained by tracing the remaining five FACE_SURFACE pointers.

The pointers of the CARTESIAN_POINT's on the bottom of the completed hierarchy (Figure 3.5) are the addresses of the eight vertices of the box. Thus, the coordinates of these vertices can be easily obtained from the STEP file. The eight pointers and associated coordinates are as follows:

Pointer number	Coordinates X	Y	Z
17	100,	0,	0
19	100,	.50,	0
69	100,	50,	25
55	100,	0,	25
26	0,	50,	0
62	0,	50,	25
33	0,	0,	0
53	0,	0,	25

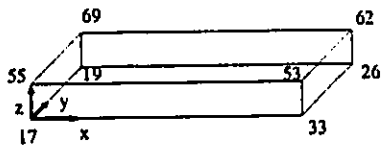


Figure 3.3 Example box

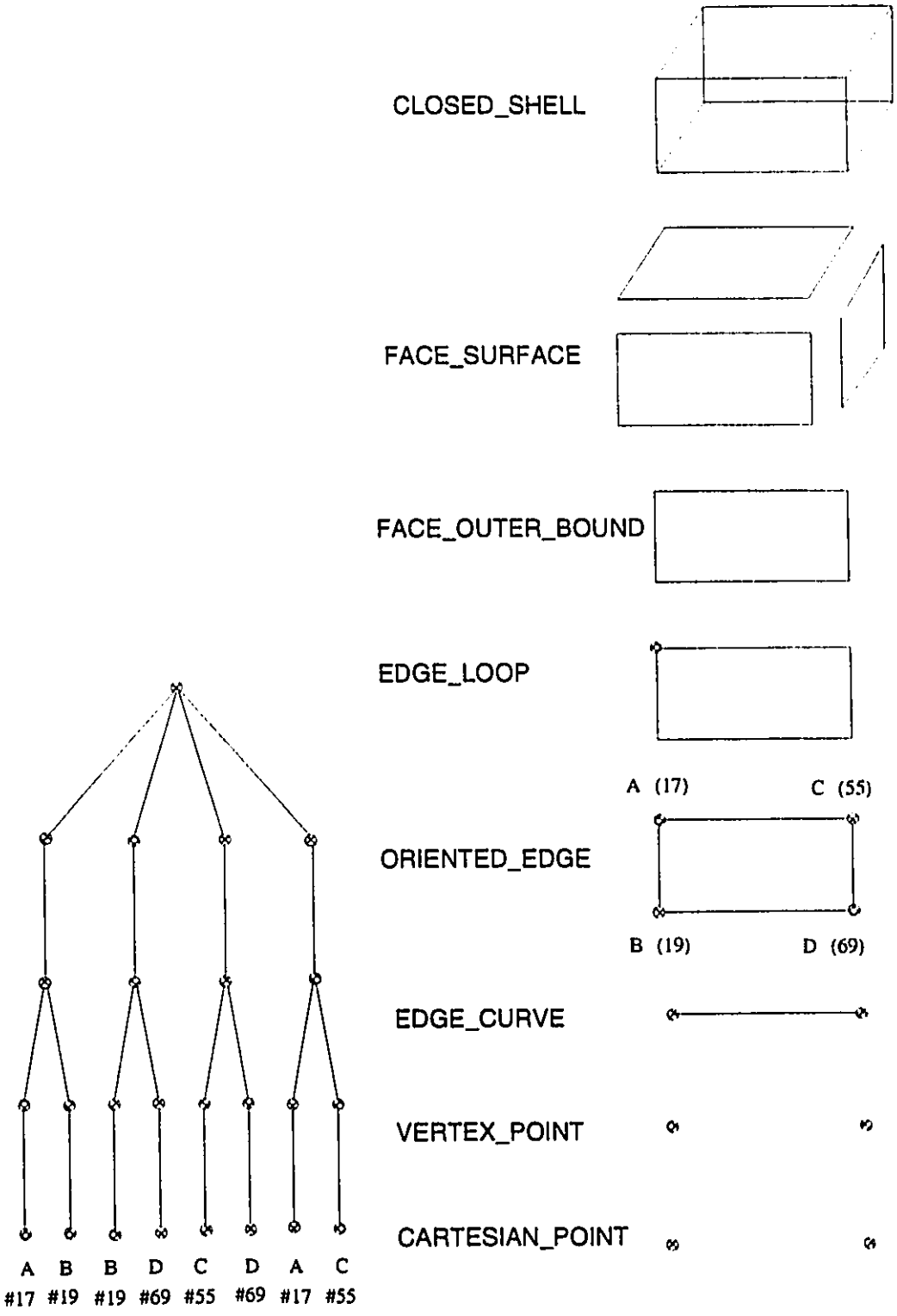


Figure 3.4 Hierarchy of functional elements for the example box

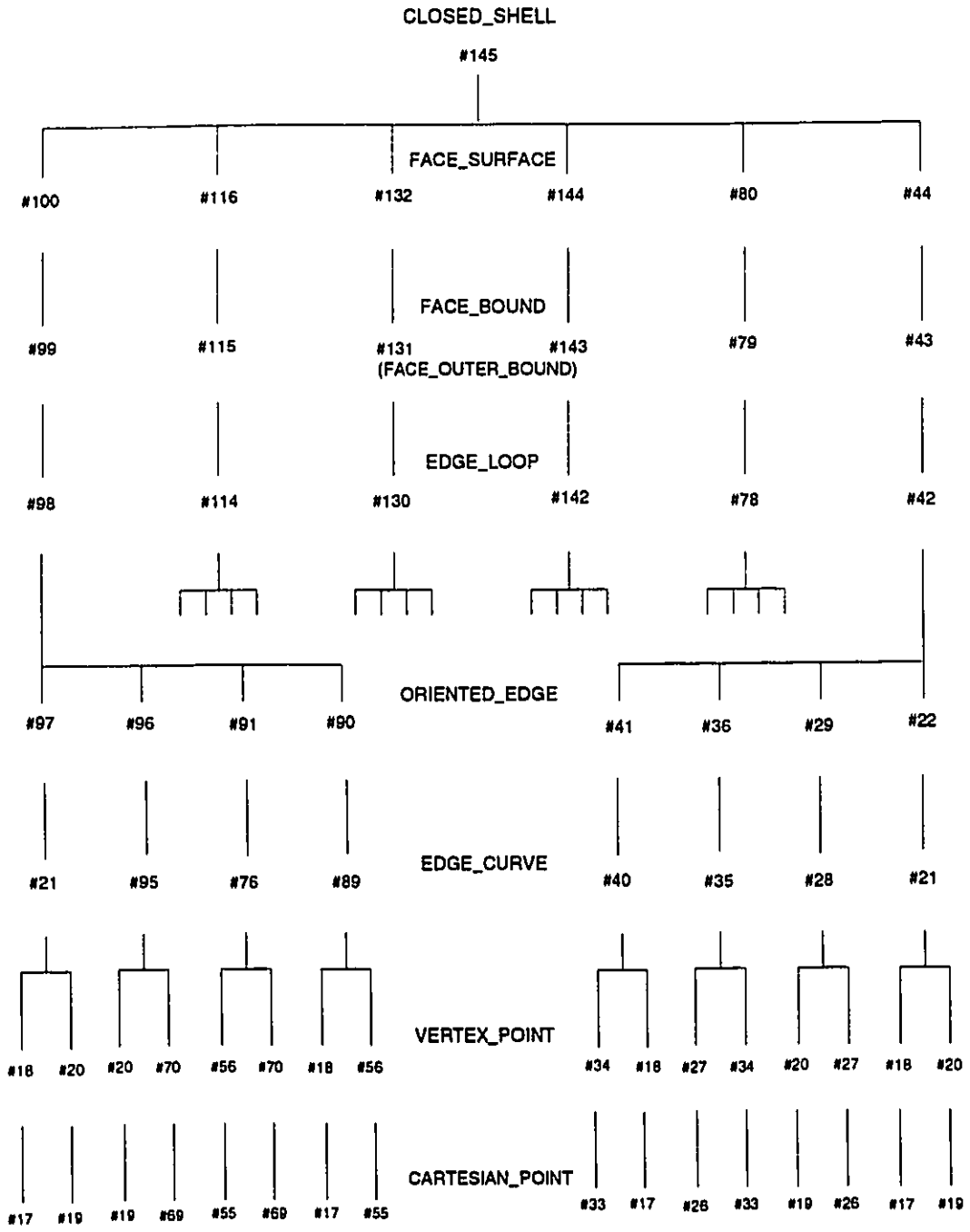


Figure 3.5 Tree structure of the data extraction procedure for the rectangular box

The above procedure is summarized as an algorithm to extract planar features from STEP files, as shown below.

Algorithm 1 (Extract geometric data from STEP part file)

Read STEP file.

If the number of elements in the CLOSED_SHELL > 0 *then*

record the pointer addresses of all elements (i.e., number of FACE_SURFACEs) in the CLOSED_SHELL and return number of elements found: *NS*, continue;

else return message "The drawing file is empty".

DO *i=1* to *NS*

record the pointer addresses of the elements (i.e., number of FACE_BOUNDs) in the FACE_SURFACE and return number of elements found: *NB*

DO *j=1* to *NB*

record the pointer address of the FACE_BOUND, the pointer address of the associated EDGE_LOOP, and the pointer addresses of the ORIENTED_EDGEs of the EDGE_LOOP, and return number of elements (i.e., number of ORIENTED_EDGEs) in the EDGE_LOOP: *NE*

DO *k=1* to *NE*

if the *3rd* element of the EDGE_LOOP is a pointer to LINE *then*

record the pointer address of the associated EDGE_CURVE and pointer addresses of VERTEX_POINTs in the EDGE_CURVE, return number of VERTEX_POINTs: *NP*

else continue

DO $l=1$ to *NP*

record the pointer address of the associated CARTESIAN_POINT and return its elements (X,Y,Z).

ENDDO

ENDDO

ENDDO

ENDDO

In the algorithm, *NS*, *NB*, *NE*, and *NP* are defined as follows:

NS=number of FACE_SURFACEs in this CLOSED_SHELL;

NB=number of FACE_BOUNDs in the current FACE_SURFACE;

NE=number of ORIENTED_EDGEs in the current EDGE_LOOP;

NP=number of EDGE_CURVEs in the current ORIENTED_EDGE.

The data obtained from the algorithm will be used to build boundary plane equations for all surfaces of the part.

3.3 Volume Slicing

To improve machining efficiency, it is recommended that the material be removed layer by layer in rough machining. The reason is that the planar tool path can be generated on each layer and the planar path is more advantageous to cutting efficiency. In doing so, the stock has to be sliced into a series of layers parallel to XY plane where, X and Y

represent the directions of width and length of the component, respectively. These layers are successively machined. Another advantage of this method is that it reduces a three-dimensional (3D) problem to a series of two-dimensional (2D) problems, thus simplifying tool path planning and machining process control. The detailed slicing procedure developed in this study is illustrated below.

Once the necessary data have been extracted from the STEP file, a set of equations representing the part boundary and feature boundaries can be obtained. These equations are called *boundary equations* thereafter. A layer, i.e., a cross sectional profile of the part for a particular Z value can be defined by its boundary edges. The boundary edges of a particular layer can be determined by substituting its Z value into the associated boundary equations. However, care should be taken to scan all valid boundary equations for a particular Z value. For example, slicing a part with pockets may yield a layer containing several "holes" (Figure 3.6). The layer is therefore defined by both outside boundary edges and internal boundary edges. Hence, the Z value should be substituted into all valid boundary equations, including the ones that define internal features. Following this procedure the stock can be completely sliced from the top to the bottom by specifying a proper Z increment, i.e., the thickness of each layer. In practice the layer thickness can be simply set to the depth of cut. The tool path can then be generated for each of these layers.

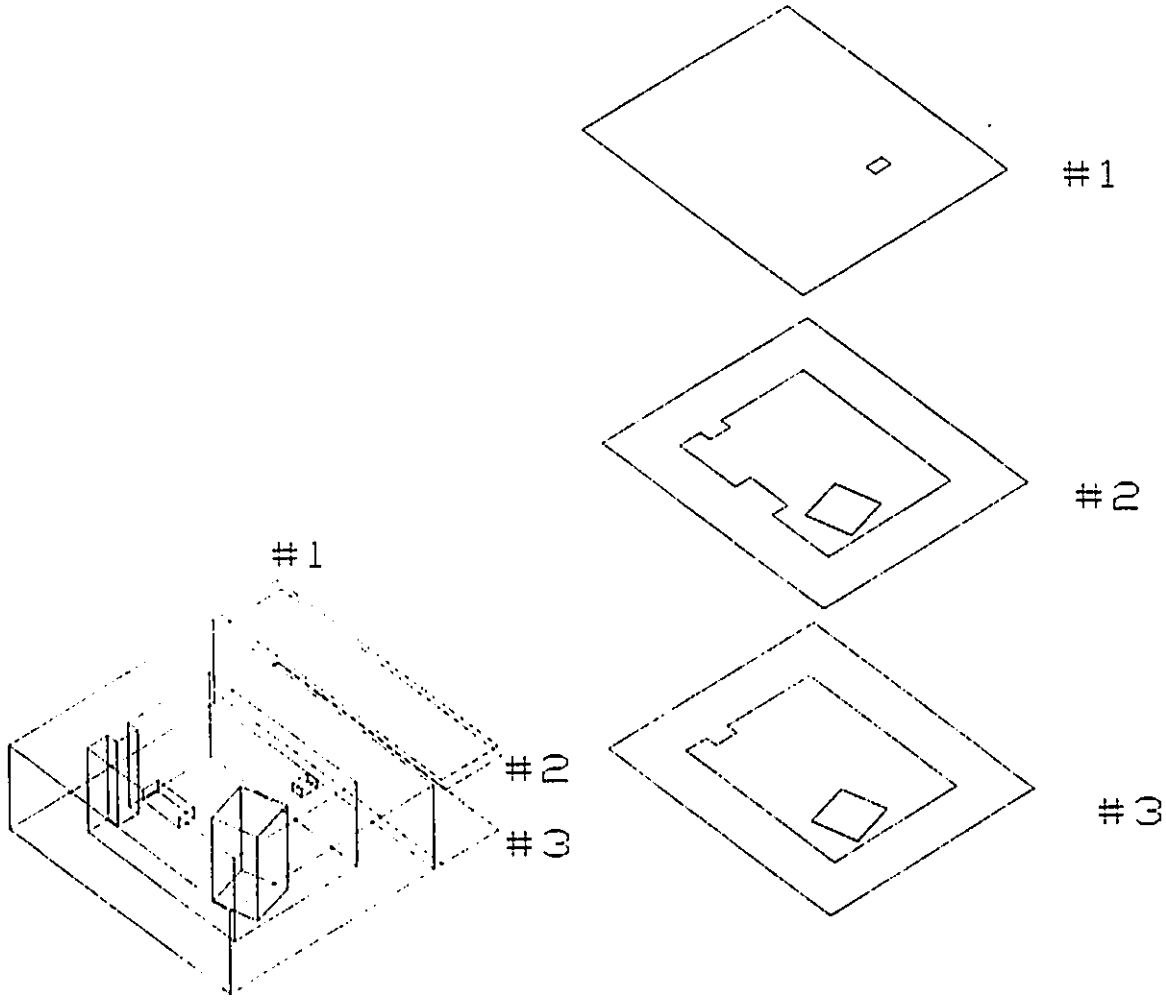


Figure 3.6 Block drawing showing layers at different Z heights

The procedure for deriving equations of planes for the given vertices, and for generating the required cross sections is outlined in the following two algorithms.

Algorithm 2 (construct boundary equations)

Step 1 Read vertex coordinates of all part and feature boundary planes from the output of algorithm 1.

Step 2 For each part or feature boundary plane, specify two vectors *V1* and *V2* based on any three vertices of the plane.

Step 3 Determine n, the normal vector of the concerned plane using *V1* and *V2*. The normal vector is $n = ai+bj+ck$, where, i, j, and k are the unit vectors in X, Y, and Z directions respectively, and a, b, c are components of n.

Step 4 Select any point on the plane, e.g., a vertex, $\mathbf{P}=(X_0, Y_0, Z_0)$. The plane equation is then obtained as follows

$$aX+bY+cZ = aX_0+bY_0+cZ_0$$

The profile of each layer and the boundary of each internal feature, if any, will be outlined using the following algorithm.

Algorithm 3 (generate cross sections)

Step 1 Read all boundary equations, vertices of the part and its features, set

ZMIN = Z coordinate of the stock bottom plane

ZMAX =Z coordinate of the stock top plane

ZS = slicing step length (layer thickness)

Step 2 Set $Z=Z-ZS$. *If* $Z>ZMIN$, go to next step, otherwise, go to step 5.

Step 3 Solve for the vertices of the plane associated with the current Z value and set

$XMIN = \min (X_i |\ i=1,\ldots,NZ)$

$XMAX = \max (X_i |\ i=1,\ldots,NZ)$

$YMIN = \min (Y_i |\ i=1,\ldots,NZ)$

$YMAX = \max (Y_i |\ i=1,\ldots,NZ)$

where NZ is the total number of vertices in the current plane.

Step 4 *DO* $Y=YMIN$ to $YMAX$, step $INCR$

DO $X=XMIN$ to $XMAX$, step $INCR$

if (X, Y, Z) satisfies a boundary equation of the plane, save (X,Y,Z)

else, continue

ENDDO

ENDDO

(where, $INCR$ is a small positive number, usually a fraction of the tool diameter).

Step 5 Stop

It should be pointed out that the above algorithm is used to avoid the complex feature recognition process. The accuracy or resolution of the defined boundary or profile will depend on step length, $INCR$. Since the scanning process in the algorithm is very fast, a sufficiently small $INCR$ can be selected. The layer profile or feature boundary outlined in this way is sufficiently accurate for rough milling.

Now, the cross sectional profiles of the part in XY plane can be obtained using these algorithms. Each of these boundary profiles will be used to generate the tool paths. The tool path generation method is explained in the following section.

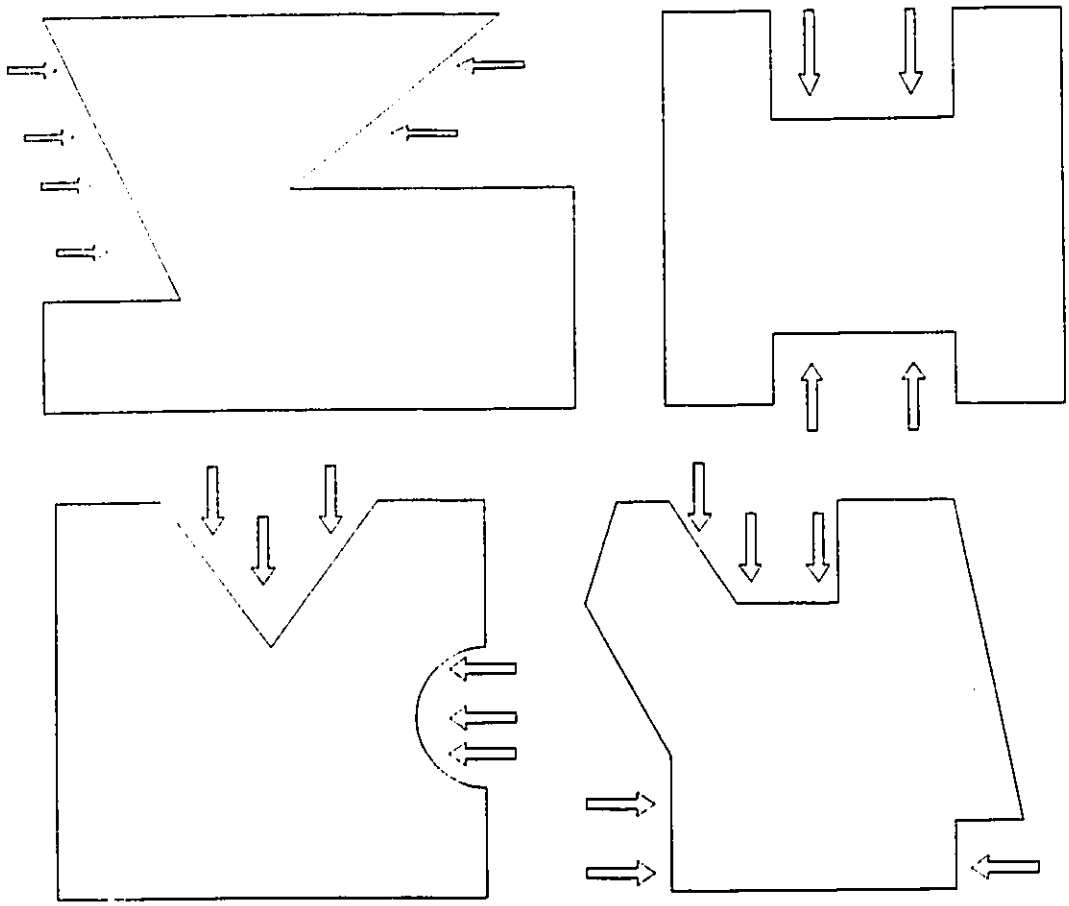
3.4 Tool Path Generation

An important concern in tool path planning is the computational time. To efficiently generate tool paths for rough cutting, a two-stage approach is proposed in this study. Before presenting the two-stage tool path generation approach, the following definitions are provided to simplify discussions.

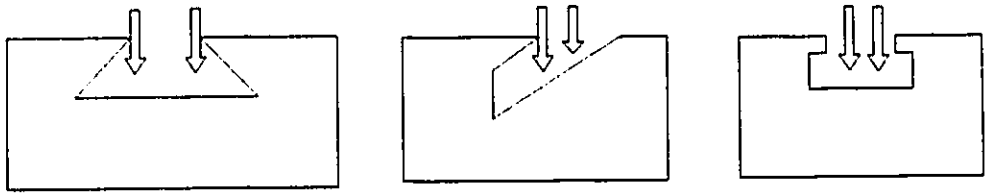
Orthogonally approachable area

The orthogonally approachable area is a machining area that can be reached in +X, -X, +Y, -Y directions (or from left, right, bottom and top) without crossing any part boundary or feature boundary.

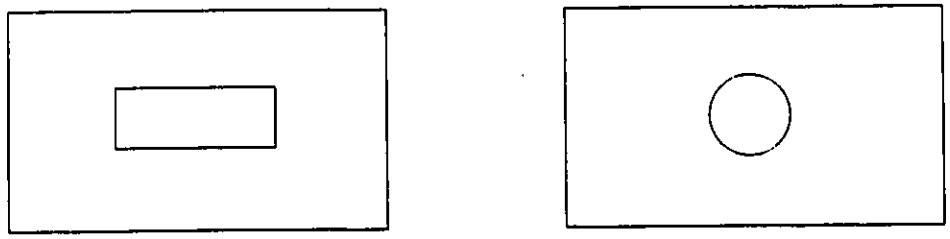
The orthogonally approachable area covers all material that is outside of the part profile and some external features or some portion of external features that is parallel to one of the four directions mentioned above. Some examples are illustrated in Figure 3.7(a).



(a) Features that can be handled orthogonally



(b) External features that cannot be handled orthogonally alone



(c) Internal features that cannot be handled orthogonally

Figure 3.7 Orthogonally and non-orthogonally approachable areas

Non-orthogonally approachable area

Any machining area that does not belong to the orthogonally approachable area is called non-orthogonally approachable area. This includes all internal features such as holes, internal islands, inner portion of dovetails and guideways. Some non-orthogonally approachable areas are shown in Figure 3.7(b) and Figure 3.7(c).

In the first stage, the orthogonally approachable area is defined based on a 4-directional method. The overdefined area is simplified and then the CL file is obtained. The second stage will generate tool paths for all non-orthogonally approachable areas. The logic behind this approach is that most material to be removed is often orthogonally approachable and thus represents a major portion of computational time for tool path generation. If the computational time can be reduced for the orthogonally approachable area, the total computational time can be significantly reduced.

3.4.1 Tool path Generation for Orthogonally Approachable Area

A 4-direction based method for rough milling is developed in this section. Various tool paths can be generated using the 4-direction based method. However, since the main purpose of this thesis is to develop a framework for generating tool path from STEP data files, tool path optimization is not considered. As such, only one type of tool path, linear tool path, is implemented though other alternative paths can be incorporated into this framework. With the linear tool path, the tool always cuts in one direction (X or Y). The tool retracts at the end of the workpiece and is then rapidly positioned to the beginning of the next path segment. The reason for implementing linear path is that, if the direction

is properly selected, e.g., up milling, it can avoid high contact impact between the cutter and part surface with scale (e.g., hot-worked metals and castings) and thus provides a smoother machining process (Kalpakjian 1992). Also, the tool life can be longer as compared to other tool paths. It might be legitimately argued that the linear tool path may prolong the cutting time due to the rapid retraction and re-positioning time. Surprisingly, however, Weck et al (1994) has recently reported that the machining time required by the linear tool path is significantly shorter than the most widely used window frame (Spiral) and dual-directional staircase (zigzag) tool paths. The details of the 4-direction based tool path generation method are discussed below.

Imagine that we first use some parallel light beams of a particular colour, e.g., red, to sweep all the area that can be approached from one direction (e.g.,+X direction or left), i.e., all the area that can be machined by moving the cutter in this direction without crossing any part or feature boundaries. Some areas that are to be removed may not be covered by this colour since the light beam may be blocked by some features. Then, some parallel light beams of another colour, e.g., green, are used to sweep the layer from another direction, say, +Y direction (up direction). Now some areas that are not covered by the red colour are filled with green colour. Some areas may have been covered by both red and green colours and some may remain uncovered. We then repeat this procedure from the other two directions, i.e., -X and -Y directions. Most of external areas to be machined can be covered in this fashion. The redundant coverage will be removed and then the tool path can be generated for the covered area. The 4-direction based method can quickly generate tool path for any convex profiles, such as polygons

and most external features, except for some internal dovetail or guideway type features. The area that cannot be covered by the 4-directional method is often a small portion of the total rough cutting volume and will be separately handled.

The actual tool path generation procedure is as follows. First, the tool path in each direction is generated by griding the area that can be covered from one direction into equal sized square cells. The cell size is determined by the tool diameter. These cells will be used to determine the cutter locations. The procedure is repeated for all the other three directions. In the process of griding two cases may arise:

(a) The current cell encounters a point on the part boundary. If this is the case, any further movement will violate the part boundary and hence such a cut is prohibited. Thus the value of tool diameter is offset from X value of the profile point. Then the Y value is updated and the next row is considered.

(b) The current cell reaches the stock boundary of the row. If so, Y value is updated and the next row is initiated.

In both cases the centroid point coordinates of every cell will be saved. The process is repeated until all the approachable area in this direction that is to be machined has been grided.

To remove the redundant coverage, a reference direction is selected. Every cell that is "approachable" in the reference direction is checked. If a cell has the reference direction's coverage, the coverage of other directions will be removed. Then the cells

that have been checked will be marked with the reference direction's symbol and saved in a separate file. Next, a new reference direction is selected and all the cells are checked. If a cell has been saved in the file associated with the previous reference direction, it will not be processed. Otherwise, it will be saved to the file associated with the current reference direction and will be removed from the data files of the remaining directions that have not been used as reference directions. The redundant coverage will be removed by repeating this procedure for all directions. The detailed steps are as follows.

Algorithm 4 (define orthogonally approachable area from left)

Step 1 Read Z coordinate, stock, part and feature boundary data of current layer from the output of algorithm 3, and specify tool diameter d.

Step 2 Let the coordinates of the lower left corner of the stock be (XL, YL), and the coordinates of the upper right corner of the stock be (XU, YU).

DO $Y=YL$ to YU, step $k\times d$

DO $X=XL$ to XU, step $k\times d$

if the current cell contains any data point belonging to part or feature boundary, then subtract $k\times d$ from the X coordinate of the profile point and save the coordinates of the boundary point.

else save the centroid coordinates of the current cell and continue.

ENDDO

ENDDO

In step 2, k is the cutter immersion ratio, $0<k<1$. It is recommended that k be set to 0.85 (Lee et al 1994) for rectangular stocks. The above algorithm only defines the machining area from the left. It can be used for defining the machining area from other three directions, i.e., bottom, right and top, with minor modification. The outputs are saved in File-L, File-B, File-R, and File-T, respectively. Once the four directions have been considered, the majority of the machining area is defined. However, some area may be over-defined due to the redundant coverage (Figure 3.8a). The redundant coverage can be removed using the following algorithm.

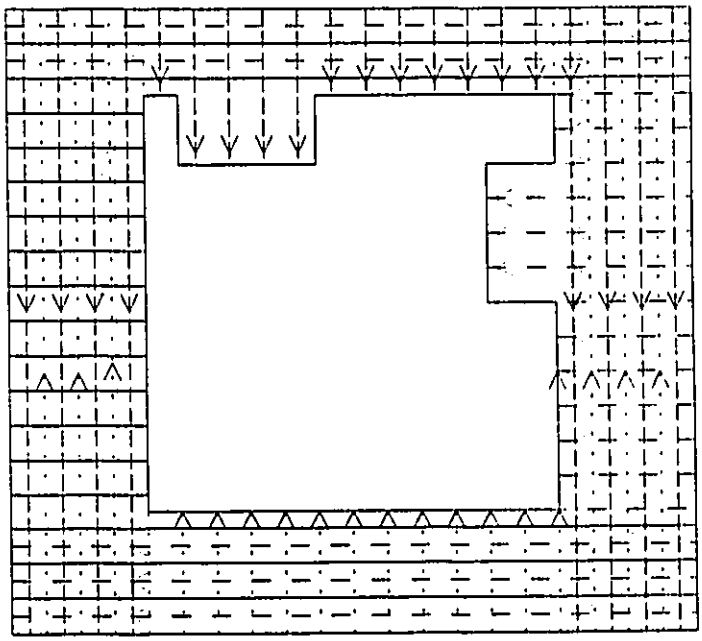


Figure 3.8 (a) Orthogonally defined machining areas with redundant coverage

Algorithm 5 (Remove redundant area coverage)

Step 1 Input File-L of the current layer and define the left direction as the reference direction.

Step 2 Input File-B and read a data record, i.e., (X,Y,Z) coordinates of a cell, from the top of the file. Scan File-L. If this data record also exists in File-L, eliminate it from File-B. Otherwise, keep it in File-B. Repeat this process until all data records in File-B have been checked.

Step 3 Repeat Step 2 for File-R and File-T. Now the machining area that is approachable from left is solely defined by File-L and the associated CL file will be generated based on File-L.

Step 4 Input the updated File-B and define File-B as the new reference file.

Step 5 Read a record from the top of File-R and scan File-B. If this record exists in File-B, then eliminate it from File-R; otherwise keep it in File-R. Repeat this procedure until all the records in File-R have been checked.

Step 6 Repeat step 5 for File-T. Now, all the machining area that can be reached from bottom but is not approachable from left is solely defined by the updated File-B.

Step 7 Define the updated File-R as the new reference file and read a record from the top of the updated File-T. If the record is also available in File-R, then remove it from File-T; otherwise keep it in File-T. Repeat this until all the records in File-T have been tested. Now, the machining area that can be reached from right but cannot be reached from left and bottom has been solely defined by the updated File-R. The area that can be reached from the top but is not approachable

from other directions is solely defined by File-T.

Once algorithm 5 is executed, all the machining areas that are approachable from the four directions are completely defined without redundant coverage (Figure 3.8b).

So far all orthogonally approachable machining areas have been defined. However, the internal features and the dovetail-like features, if existing, remain to be defined.

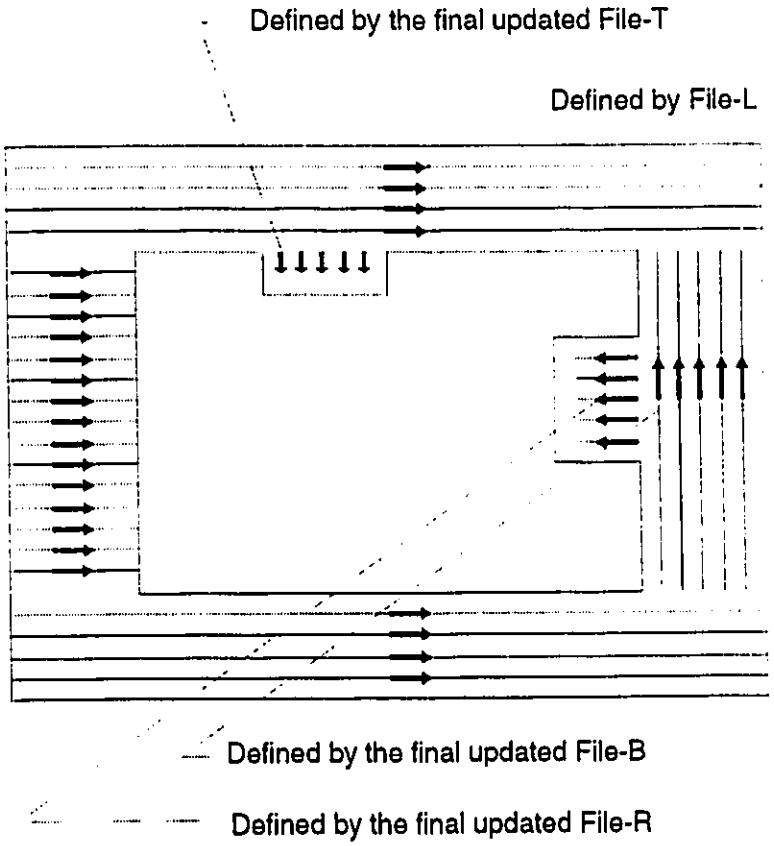


Figure 3.8(b) Orthogonally defined machining areas without redundant coverage

3.4.2 Tool Path Generation for Non-orthogonally Approachable Areas

If an internal feature is a primitive, the tool path can be generated by considering boundary equations that can be constructed based on the vertices extracted from the STEP file. If the non-orthogonally approachable feature is complex, the tool path will be generated in a "*divide-and-conquer*" fashion. A complex internal feature can be divided into several primitives such as rectangles and triangles. Each of these primitives are then treated separately. The algorithm is presented below. Note that a non-orthogonally approachable area could be an internal feature or a portion of an external feature.

Algorithm 6 (define non-orthogonally approachable areas)

Step 1 Read the vertices of a non-orthogonally approachable feature. If the feature is not a primitive in shape, divide it into several primitives. The dividing criterion is to obtain as few primitives as possible.

Step 2 Select a primitive and tool diameter, d, for the selected primitive.

Step 3 Define

(XUV, YUV) = coordinates of the highest vertex of the primitive

(XLV, YLV) = coordinates of the lowest vertex of the primitive

$XLF(Y)$ = left boundary function

$XRF(Y)$ = right boundary function

$L1, L2$ = right, left offset

$L3, L4$ = bottom, top offset

Step 4 *DO Y=YLV+L3* to *YUV-L4*, step $k \times d$

DO X=XLF(Y)+L2 to *XRF(Y)-L1*, step $k \times d$

save the centroid coordinates of the current square

ENDDO

ENDDO

(where, k=cutter immersion ratio, d=cutter diameter)

Step 5 Repeat steps 3 and 4 for all primitives.

The offset values L1, L2, L3 and L4 for different cases will be calculated as follows. Note K in the following equations is the distance from the center of the tool to the vertex point, $K = \frac{r}{\sin(a)}$, where r is the radius of the tool and a is the half angle subtended by the two joining sides.

Case a: (Figure 3.9 (a))

$$L1 = \frac{r}{\sin(b)}$$

$$L2 = r * \sec(c)$$

$$L3 = K * \sin(a+b)$$

Case b: (Figure 3.9 (b))

$$L1 = \frac{r}{\sin(b)}$$

$$L2 = \frac{r}{\sin(c)}$$

$$L3 = K * \sin(a+b)$$

Case c: (Figure 3.9 (c))

$$L1 = \frac{r}{\cos\ (b)}$$

$$L2 = \frac{r}{\sin\ (c)}$$

$$L4 = K * \cos\ (a + b)$$

Case d: (Figure 3.9 (d))

$$L1 = K * \sin\ (a - b) + \tan\ (b) * [\ K * \cos\ (a - b)\ ]$$

$$L2 = \frac{r}{\sin\ (c)}$$

$$L4 = K * \cos\ (a - b)$$

3.5 Elimination of Redundant Data Points from CL File

Now, all the areas that are to be machined have been defined. The data files obtained from algorithms 5 and 6 will be used as the CL files. However, as mentioned above, the data points are defined by incrementing a tool diameter length at each step. This may yield too many data points on a path segment. Thus many path segments are over-defined, rendering very large data files. Mathematically, if a path segment is a straight line, only two points are needed to completely define the segment. Therefore, the other points are redundant and should be eliminated. This is done using the algorithm 7.

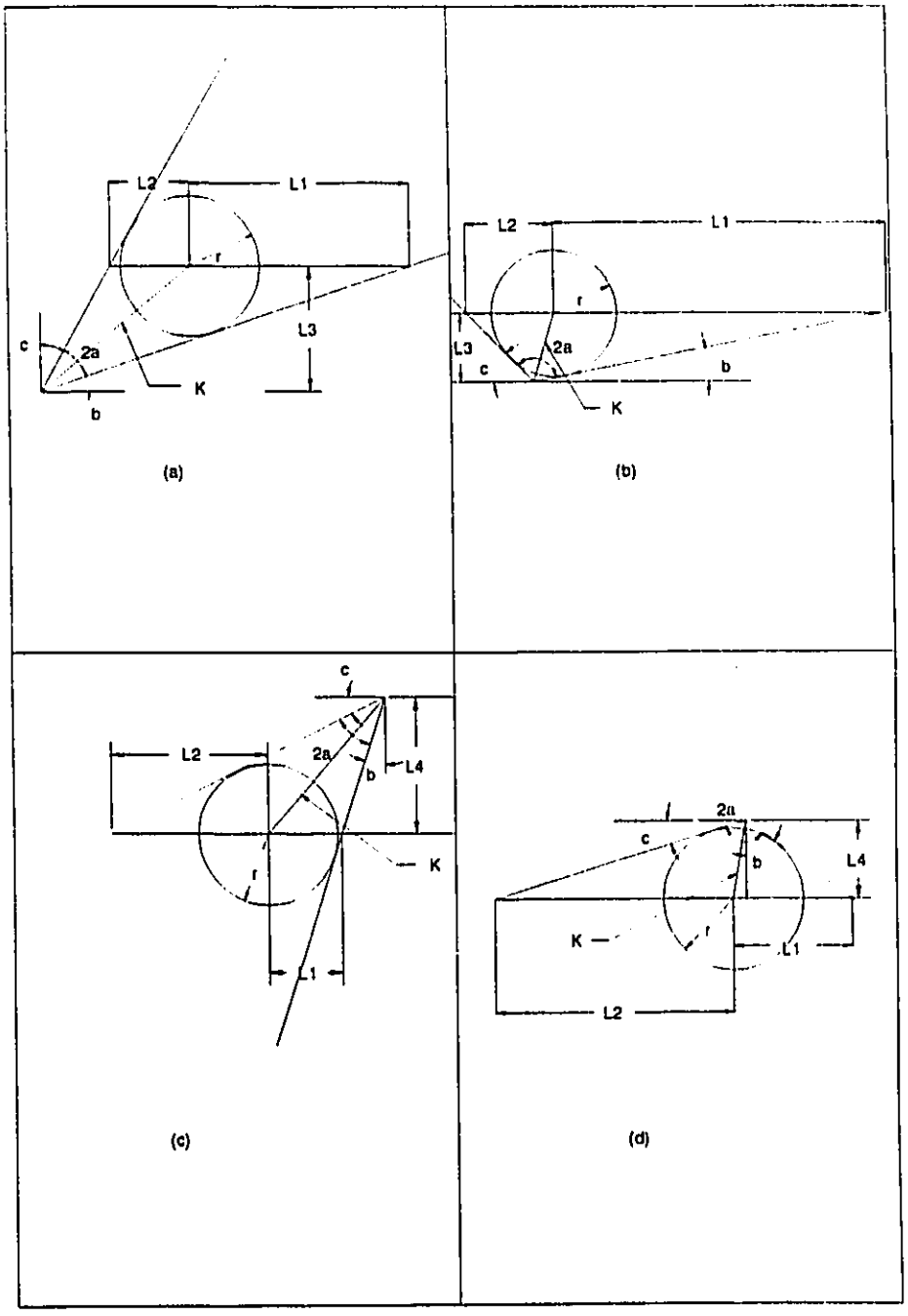


Figure 3.9 Offset distances for different cases

Algorithm 7 (eliminate redundant data points)

Step 1 Input a CL data file and define

(XLP, YLP) = coordinates of the lowest vertex of the machining area

(XUP, YUP) = coordinates of the highest vertex of the machining area

$XLB(Y)$ = left boundary function of the machining area

$XRB(Y)$ = right boundary function of the machining area

$YLB(X)$ = bottom boundary function of the machining area

$YRB(X)$ = top boundary function of the machining area

Step 2 *If* the current path is parallel to X axis, e.g., the path defined in File-L or File-R,

then

DO $Y=YLP$ to YUP, step $k \times d$ (tool diameter used in the current file)

Read a X value of a record. *If* $XLB(Y) < X < XRB(Y)$, discard this record, otherwise, keep it in the file.

ENDDO

else

DO $X=XLP$ to XUP, step $k \times d$ (tool diameter used in the current file)

Read a Y value of a record. *If* $YLB(X) < Y < YRB(X)$, discard this record, otherwise, keep it in the file.

ENDDO

Step 3 Repeat steps 1 and 2 for all data files.

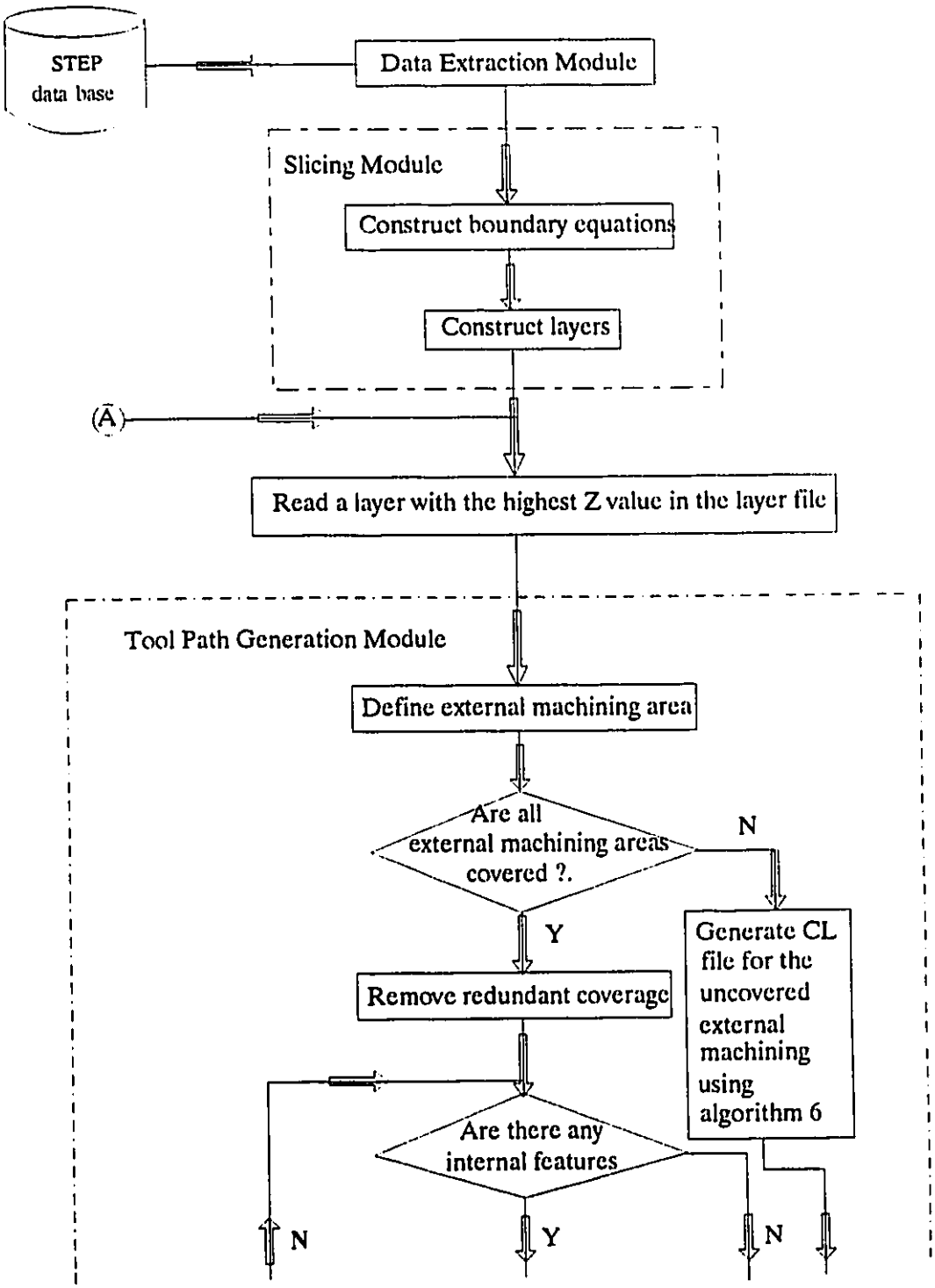
Other input data such as layer thickness, tool diameter, feedrate and spindle speed are decided by the user. In this thesis, these parameters are assumed to be constant.

Now, the algorithms for data extraction, slicing, tool path generation, elimination of redundant data points from the overdefined tool path have been developed. The implementation of the algorithms is illustrated in the next chapter.

Chapter 4

SYSTEM IMPLEMENTATION

The implementation of the algorithms developed in Chapter 3 will be presented in this chapter. Considering expendability and adaptability, the system is modulized following the scheme shown in Figure 3.1. The structure of this system is shown in Figure 4.1. All modules in Figure 4.1 are coded in C and the system is implemented in a UNIGRAPHICS environment in the Institute for Advance Manufacturing Technology, National Research Council of Canada, Ottawa. All the source codes are attached in Appendix 2. This system, however, can also be implemented in PCs provided that PC version CAD/CAM systems are available.


STEP
data base
Data Extraction Module
Slicing Module
Construct boundary equations
Construct layers
(Ã)
Read a layer with the highest Z value in the layer file
Tool Path Generation Module
Define external machining area
Are all external machining areas covered ?.
N
Y
Generate CL file for the uncovered external machining using algorithm 6
Remove redundant coverage
Are there any internal features
N
Y
N

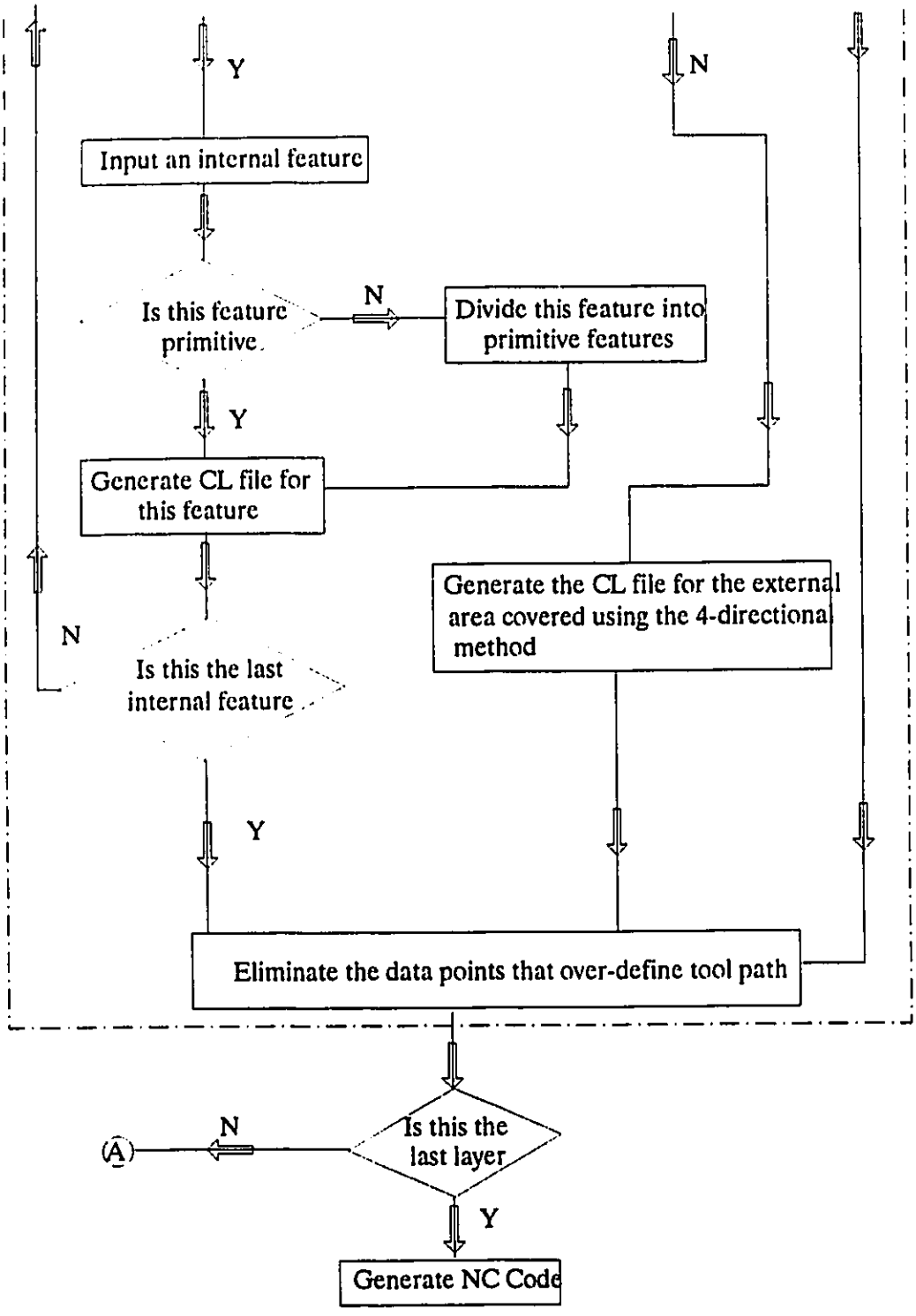


Figure 4.1 Block Diagram of the tool path planning system

4.1 Description of the System

Data extraction module

This module is written based on algorithm 1 as described in Chapter 3. The STEP files of part design models are available in UNIGRAPHICS system. The module starts with screening the STEP file of a particular part design model. The purpose of this process is to filter out the information that is not needed for tool path planning. Then the STEP file is processed following the procedure outlined in algorithm 1. The output of this module includes the coordinate values of all vertices of the part, its external features, and internal features. These data are saved in a file named PTS.DAT for the use of the next stage. The size of the file is significantly smaller than the original STEP file. Our computational experience shows that the size of the PTS.DAT file is normally less than 5% of that of the original STEP file.

Slicing module

This module consists of algorithms 2 and 3. The vertex data contained in PTS.DAT are used to construct the boundary equations for all planar surfaces of the part boundaries, external and internal features using algorithm 2. The equations are stored in file EQS.OUT. The cross sections or machining layers are generated using these equations following the procedure given in algorithm 3. The output will be saved in a separate file called PTS.TXT.

Tool path generation module

In this module, the tool paths will be generated for every layer obtained in the previous module starting from the one with the largest Z coordinate value. This module includes four algorithms, i.e., algorithms 4,5,6 and 7. The tool paths of the orthogonally approachable machining area can be generated based on the 4-directional method presented in algorithm 4. The redundant area coverage will be removed using algorithm 5.

Although the above 4-directional method is efficient in generating tool paths for most external areas, it cannot generate tool paths for internal features and cannot generate the complete tool paths for some dovetail or guideway-like external features. A divide-and-conquer based method as described in algorithm 6 is used for all machining areas that cannot be handled by the 4-directional method alone.

The data file generated in this module will be used as CL files. The cutting direction may be different depending on the resulting data file. For example, if the machining area is defined by File-L, the tool path will be horizontally oriented. If a linear tool path pattern is applied, the cutting direction is also defined, i.e., from left to right in this case. If, however, the machining area is defined by File-B, the orientation of the tool path will be vertical and the cutting direction is from bottom to top if the linear path pattern is selected. The tool path for the entire part can then be readily generated based on the above CL files.

In most cases, however, the size of the CL files generated following the above procedure are very large since many tool path segments are over-defined. The

redundant data points in the CL files are eliminated using a program based on algorithm 7. The final CL file will be stored as TOOLPATH.OUT.

NC code generator

The final CL file generated from the previous modules along with the corresponding tool diameter is processed to generate the NC codes for each machining layer. In addition to CL data, other specifications such as inch/metric units, incremental/absolute coordinate system, spindle speed, feedrate, should also be provided. The NC code is generated using a C program (Appendix 2).

4.2 Demonstration of System Application

Implementation of this system is demonstrated using an example part as shown in Figure 4.2. The dimensions of the part are given in Figure 4.3. This part has four features: a through slot on a side face, a blind slot on the front face, a through square hole, and a rectangular protrusion on the top. The relevant portion of the STEP file of this part is attached in Appendix 3. This part is to be machined from a rectangular stock as shown in Figure 4.2.

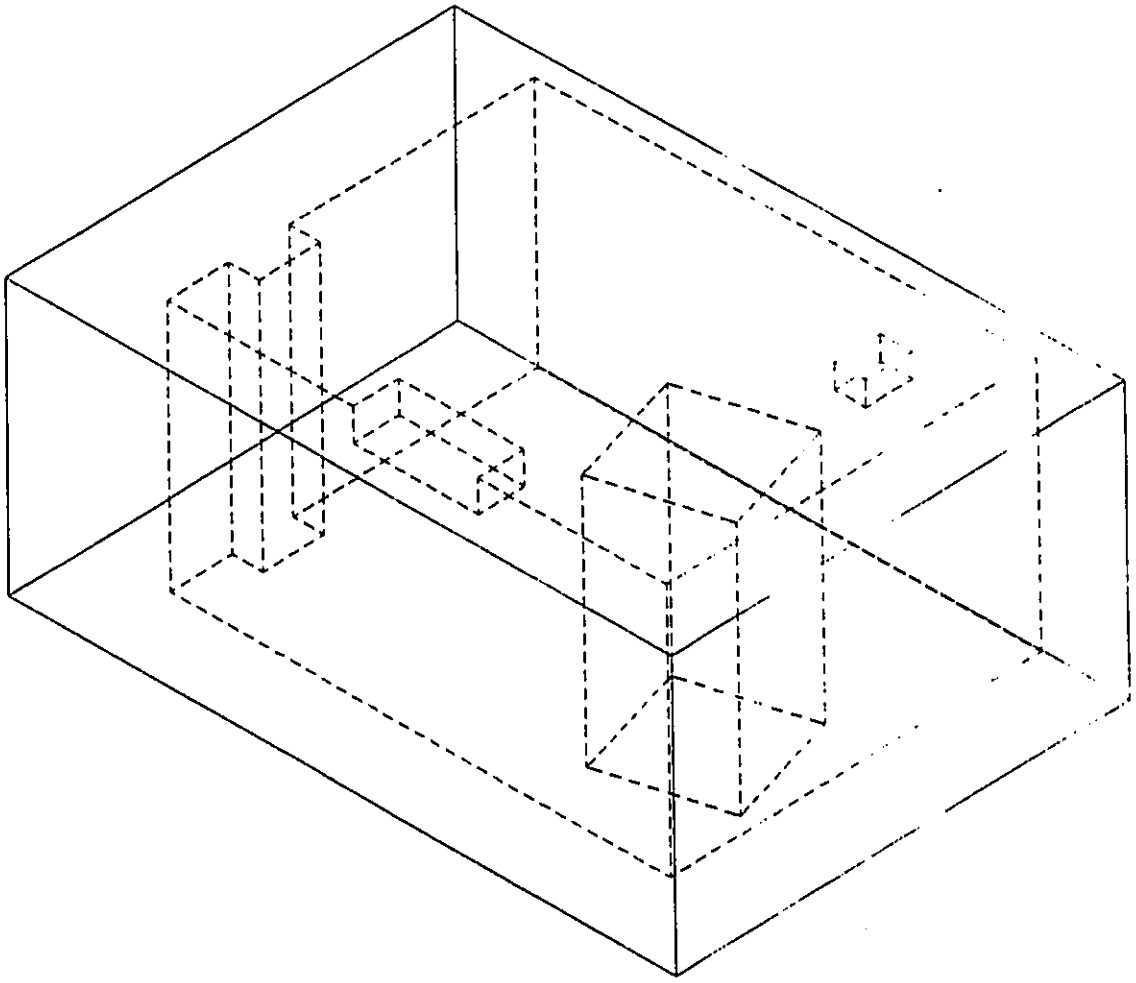


Figure 4.2 Drawing of the example part

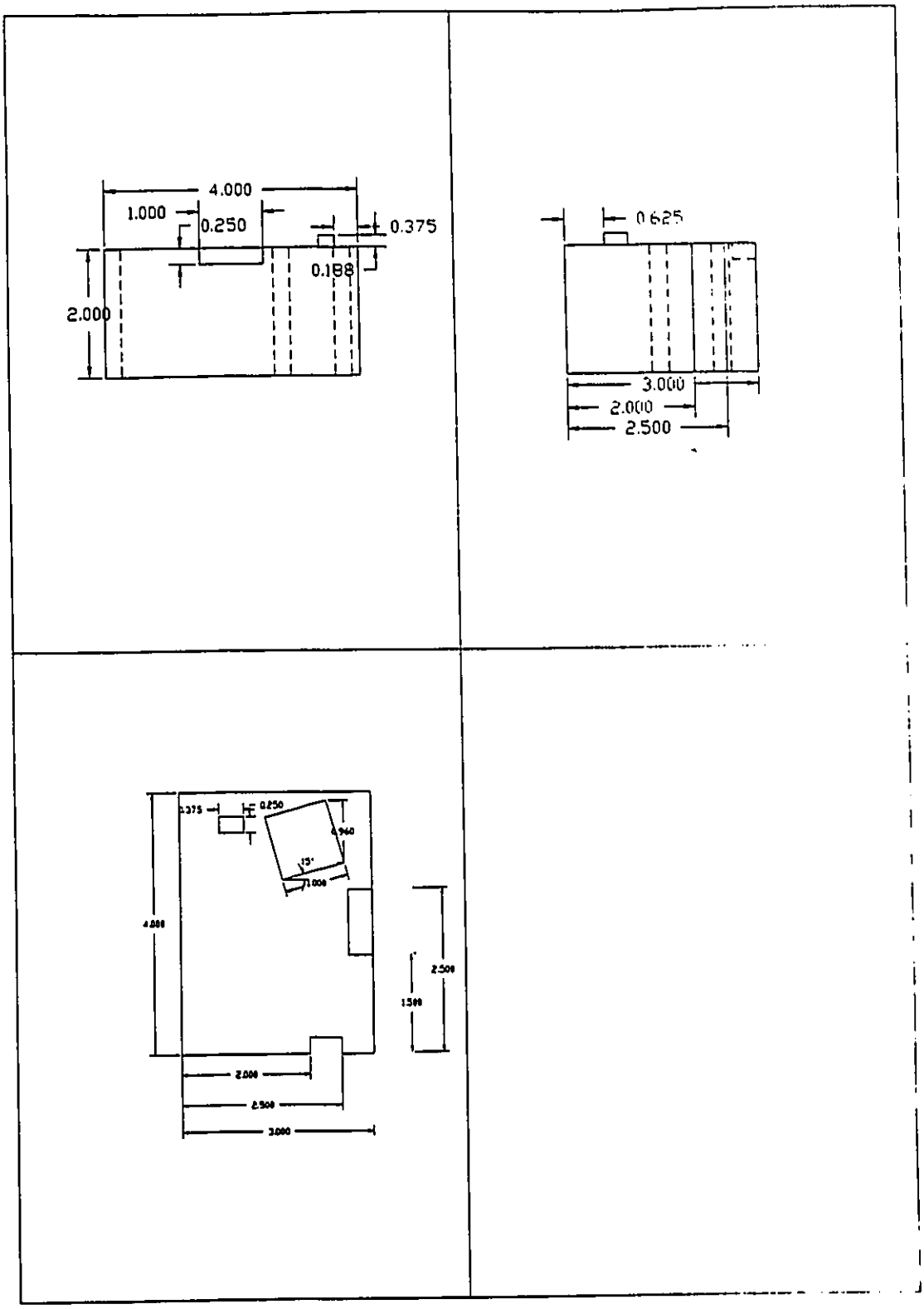


Figure 4.3 Dimension views of the example part

The portion of the STEP file containing the geometric and topological information is about 21 pages. As shown in Appendix 3, the pointer addresses are arranged in increasing order. The data extraction process starts from scanning the STEP file of the part design model. In the STEP file, the final part is defined using CLOSED_SHELL at pointer address #12570. At this address several pointer addresses of lower level functional elements, e.g., #6960, #7080, etc, are included. These pointers represent FACE_SURFACE's. The pointers of the next lower level, i.e., FACE_BOUND, can be traced from FACE_SURFACE pointers. The two levels, i.e., FACE_SURFACE and FACE_BOUND, have no direct use in tool path generation. They are simply used as links of the information chain to find the pointers for the vertices, i.e., VERTEX_POINT. For example, the tracing chain for plane I (shown in Figure 4.4) is as follows.

(CLOSED_SHELL) #12570 → (FACE_SURFACE) #9700 → (FACE_BOUND) #9640 → (EDGE_LOOP) #9630 → (ORIENTED_EDGE) (#9590, #9600, #9610, #9620) → (EDGE_CURVE) (#4970, #5370, #5020, #5220) → (VERTEX_POINT) ((#1020, #1080), (#1080, #1060), (#1060, #1000), (#1020, #1000)) → (CARTESIAN_POINT) ((#1010, #1070), (1070, #1050), (#1050, #990), (#1010, #990)) → (coordinate values) ((2.0, 1.25, 0) (2.5, 1.25, 0.0)), ((2.0,1.25,0.0) (2.5, 1.25, 2.0)), ((2.5, 1.25, 2.0) (2.0, 1.25, 2.0)), ((2.0, 1.25, 0.0) (2.0, 1.25, 2.0)).

Similarly, the vertices of other planes can be traced from the STEP file following the pointer addresses from top to bottom. The pointer addresses for all vertices, i.e.,

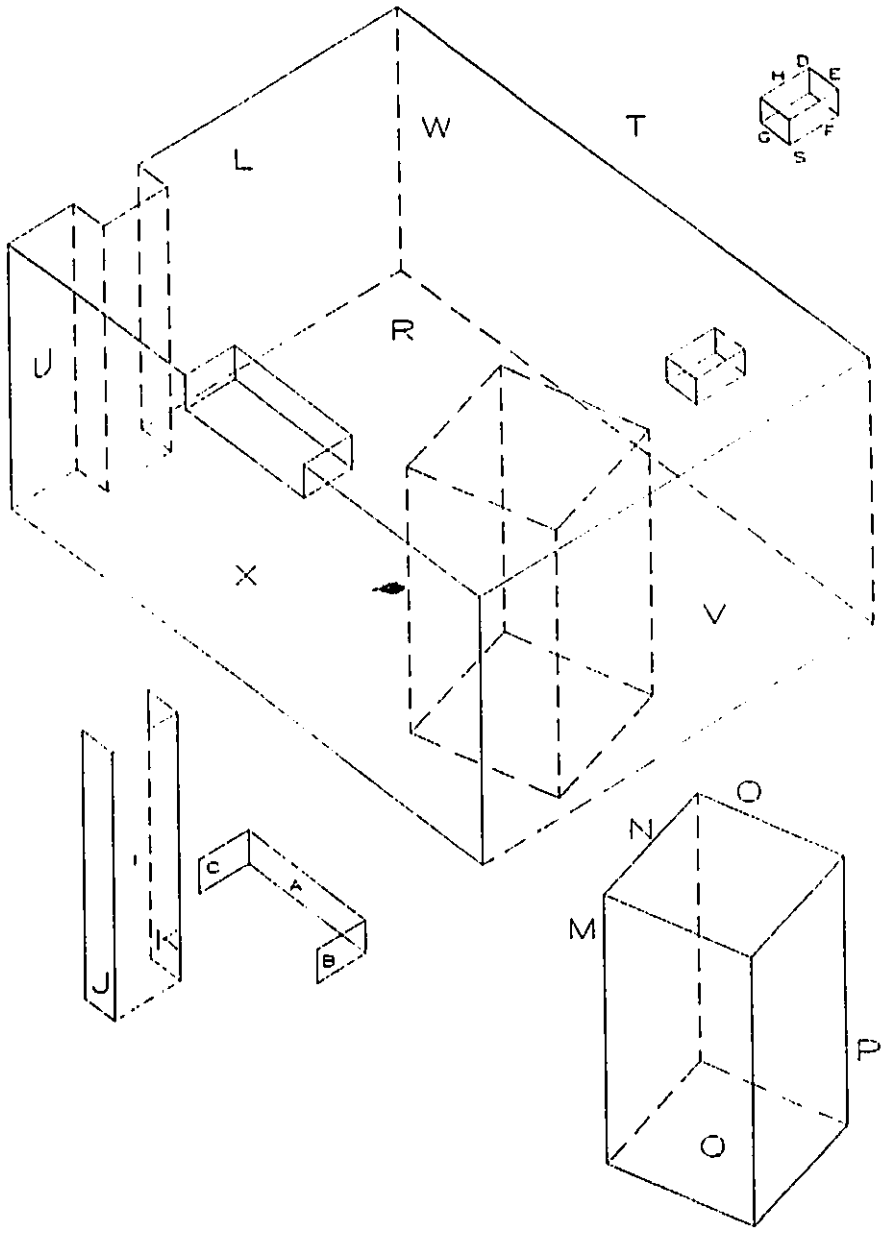


Figure 4.4 Example part shown with marked planes

CARTESIAN_POINTs, obtained using this procedure are summarised in Table 4.1. However, the following should be pointed out:

a) Some planes may have more than four pointers of CARTESIAN_POINTs. For instance, plane T has fourteen pointers for CARTESIAN_POINT because there are fourteen vertices on this plane, and there exists a small fillet at the intersection of two lines (not shown in Figure 4.4).

b) A pointer of CATESIAN_POINT may appear more than once. For example, pointer #850 appears in planes D, G, and H. This simply means that the Cartesian point represented by pointer #850 is a common vertex of the three planes.

Having obtained these pointers, the Cartesian coordinates of these vertices can be traced from the STEP file. The coordinate values associated with each plane of this part are summarised in Table 4.2.

PLANE NO's	POINTERS OF CARTESIAN_POINT
A	#650, #630, #690, #670
B	#570, #730, #550, #710
C	#750, #610, #590, #770
D	#870, #810, #850, #830
E	#890, #810, #930, #870
F	#910, #830, #890, #810
G	#790, #850, #910, #830
H	#930, #870, #790, #850
I	#1010, #1070, #1050, #990
J	#950, #1010, #990, #970
K	#1070, #1090, #1030, #1050
L	#970, #950, #1270, #1390
M	#1190, #1130, #1110, #1250
N	#1130, #1170, #1150, #1110
O	#1170, #1230, #1210, #1150
P	#1230, #1190, #1250, #1210
Q	#1190, #1130, #1230, #1170
R	#950, #1010, #1390, #1370, #1350, #1410, #1090, #1070
S	#890, #930, #910, #790
T	#550, #710, #670, #690, #610,#750, #1310, #1290, #1270, #970,#990, #1050, #1030, #1330
U	#1090, #1030, #1410, #1330
V	#1390, #1270, #1290, #1370
W	#1370, #1290, #1310, #1350
X	#710, #730, #1330, #1410, #1350, #1310, #750, #770

Table 4.1 Associated pointers for CARTESIAN_POINTs of all planes in the model

Surface	Vertex coordinates[1]
A (#8600)*	(#650: 2.500,3.375,1.750)**, (#630: 2.500,2.625,1.750) (#690: 2.500,3.375,2.000), (#670: 2.500,2.625,2.000)
B (#8720)	(#570: 2.625,2.500,1.750), (#730: 3.000,2.500,1.750) (#550: 2.625,2.500,2.000), (#710: 3.000,2.500,2.000)
C (#8980)	(#750: 3.000,3.500,2.000), (#610: 2.625,3.000,2.000) (#590: 2.625,3.500,1.750), (#770: 3.000,3.500,1.750)
D (#9100)	(#810: 1.000,4.375,2.188), (#870: 1.000,4.625,2.188) (#850: 0.625,4.625,2.188), (#830: 0.625,4.375,2.188)
E (#9220)	(#810: 1.000,4.375,2.188), (#870: 1.000,4.625,2.188) (#930: 1.000,4.625,2.000), (#890: 1.000,4.375,2.000)
F (#9340)	(#810: 1.000,4.375,2.188), (#830: 0.625,4.375,2.188) (#910: 0.625,4.375,2.000), (#890: 1.000,4.375,2.000)
G (#9460)	(#790: 0.625,4.625,2.000), (#850: 0.625,4.625,2.188) (#830: 0.625,4.375,2.188), (#910: 0.625,4.375,2.000)
H (#9580)	(#930: 1.000,4.625,2.000), (#870: 1.000,4.625,2.188) (#850: 0.625,4.625,2.188), (#790: 0.625,4.625,2.000)
I (#9700)	(#1010: 2.000,1.250,0.000), (#990: 2.000,1.250,2.000) (#1050: 2.500,1.250,2.000), (#1070: 2.500,1.250,0.000)
J (#9820)	(#970: 2.000,1.000,2.000), (#990: 2.000,1.250,2.000) (#950: 2.000,1.000,0.000), (#1010: 2.000,1.250,0.000)
K (#9940)	(#1030: 2.500,1.000,2.000), (#1050: 2.500,1.250,2.000) (#1070: 2.500,1.250,0.000), (#1090: 2.500,1.000,0.000)
L (#10060)	(#970: 2.000,1.000,2.000), (#1270: 0.000,1.000,2.000) (#1390: 0.000,1.000,0.000), (#950: 2.000,1.000,0.000)
M (#10180)	(#1130: 1.600,3.650,0.000), (#1190: 2.566,3.909,0.000) (#1250: 2.566,3.909,2.000), (#1110: 1.600,3.650,2.000)
N (#10300)	(#1150: 1.340,4.610,2.000), (#1110: 1.600,3.650,2.000) (#1130: 1.600,3.650,0.000), (#1170: 1.340,4.610,0.000)
O (#10420)	(#1210: 2.307,4.875,2.000), (#1150: 1.341,4.616,2.000) (#1170: 1.341,4.616,0.000), (#1230: 2.307,4.875,0.000)

continued....

Surface	Vertex coordinates
P (#10540)	(#1210: 2.307,4.875,2.000), (#1250: 2.566,3.909,2.000) (#1190: 2.566,3.909,0.000), (#1230: 2.307,4.874,0,000)
Q (#10760)	(#1190: 2.566,3.900,0.000), (#1130: 1.600,3.650,0.000) (#1170: 1.341,4.616,0.000), (#1230: 2.307,4.875,0.000)
R (#10760)	(#1090: 2.500,1.000,0.000), (#1410: 3.000,1.000,0.000) (#1350: 3.000,5.000,0.000), (#1370: 0.000,5.000,0.000) (#1390: 0.000,1.000,0.000), (#950: 2.000,1.000,0.000) (#1010: 2.000,1.250,0.000), (#1070: 2.500,1.250,0.000)
S (#11230)	(#890: 1.000,4.375,2.000), (#930: 1.000,4.625,2.000) (#790: 0.625,4.625,2.000), (#910: 0.625,4.375,2.000)
T (#11230)	(#1330: 3.000,1.000,2.000), (#1030: 2.500,1.000,2.000) (#1050: 2.500,1.250,2.000), (#990: 2.000,1.250,2.000) (#970: 2.000,1.000,2.000), (#1270: 0.000,1.000,2.000) (#1290: 0.000,5.000,2.000), (#1310: 3.000,5.000,2.000) (#750: 3.000,3.500,2.000), (#610: 2.625,3.500,2.000) (#690: 2.500,3.375,2.000), (#670: 2.500,2.625,2.000) (#550: 2.625,2.500,2.000), (#710: 3.000,2.500,2.000)
U (#11350)	(#1410: 3.000,1.000,0.000), (#1330: 3.000,1.000,2.000) (#1030: 2.500,1.000,2.000), (#1090: 2.500,1.000,0.000)
V (#11470)	(#1390: 0.000,1.000,0.000), (#1270: 0.000,1.000,2.000) (#1290: 0.000,5.000,2.000), (#1370: 0.000,5.000,0.000)
W (#11590)	(#1290: 0.000,5.000,2.000), (#1310: 3.000,5.000,2.000) (#1350: 3.000,5.000,0.000), (#1370: 0.000,5.000,0.000)
X (#11750)	(#1410: 3.000,1.000,0.000), (#1350: 3.000,5.000,0.000) (#1310: 3.000,5.000,2.000), (#750: 3.000,3.500,2.000) (#770: 3.000,3.500,1.750), (#730: 3.000,2.500,1.750) (#710: 3.000,2.500,2.000), (#1330: 3.000,1.000,2.000)

Note: * FACE_SURFACE pointers.

** In (#650: 2.500,3.375,1.750), #650 is pointer of CARTESIAN_POINT, 2.500,3.375,1.750 are the X,Y and Z coordinates of this vertex.

Table 4.2 Summary of the vertex coordinates traced from STEP file

The boundary equations can be constructed based on the coordinate values in Table 4.2, which are summarised in Table 4.3. Based on these equations, machining layers can be generated. Some typical layers are shown in Figure 4.5.

Plane No	Boundary equation
A	0.250x = 0.625
B	0.125y = - 0.312
C	0.094y = 0.328
D	0.094z = 0.205
E	0.047x = 0.047
F	0.070y = - 0.308
G	0.047x = 0.029
H	0.070y = 0.326
I	1.000y = 1.250
J	0.500x = 1.000
K	0.500x = 1.250
L	4.000y = 4.000
M	0.518x - 1.93y = - 6.223
N	1.920x + 0.520y = 4.970
O	0.518x - 1.932y = - 8.220
P	1.932x + 0.518y = 6.980
Q	-0.998z = 0.000
R	2.000z = 0.000
S	0.094z = 0.187
T	0.125z = 0.250
U	1.000y = 1.0000
V	-8.000x = 0.000
W	6.000y = 30.000
X	8.000x = 24.000

Table 4.3 Boundary equations of all planes in the model

The external machining area for each layer is then defined using the 4-directional method (algorithm 4). If some external areas cannot be covered using the 4-directional method, they will be handled in the next stage. For this part, however, all external machining area can be defined by the 4-directional method. Three typical layers are depicted in Figures 4.6.1, 4.6.2, and 4.6.3. It can be seen that the external machining area has been over-defined due to the redundant coverage from different directions. The redundant coverage is removed using algorithm 5 and the resulting external tool paths for the above three typical layers are respectively shown in Figures 4.7 a, b, and c.

Then the internal area of this layer is checked to see if any internal features such as pockets or holes exist. If any such features do exist, algorithm 6 is called to generate tool paths for internal features. The final tool paths including both internal and external tool paths corresponding to the above three layers are displayed in Figure 4.8. It should be pointed out that the data points over-defining tool paths will be removed. The associated CL files (with and without redundant data points) are included in Appendix 4. As can be seen in Appendix 4, the CL file has been reduced to 10% of the original size. The NC code generated based on the above CL file has been attached to Appendix 5. The machined part with internal and external features based on the tool paths generated using STEP data exchange is shown in Figure 4.9.

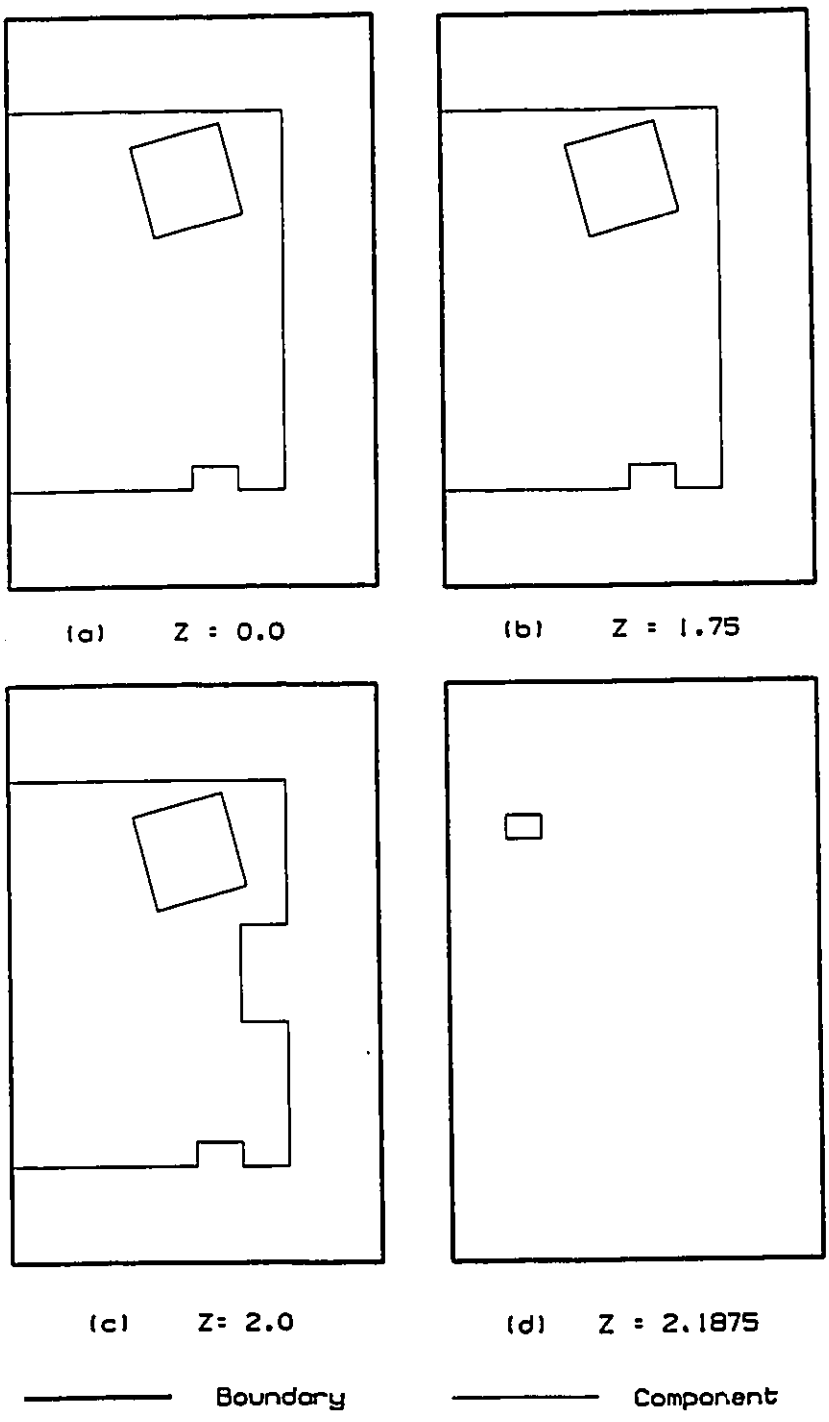


Figure 4.5 Typical layers of the example part

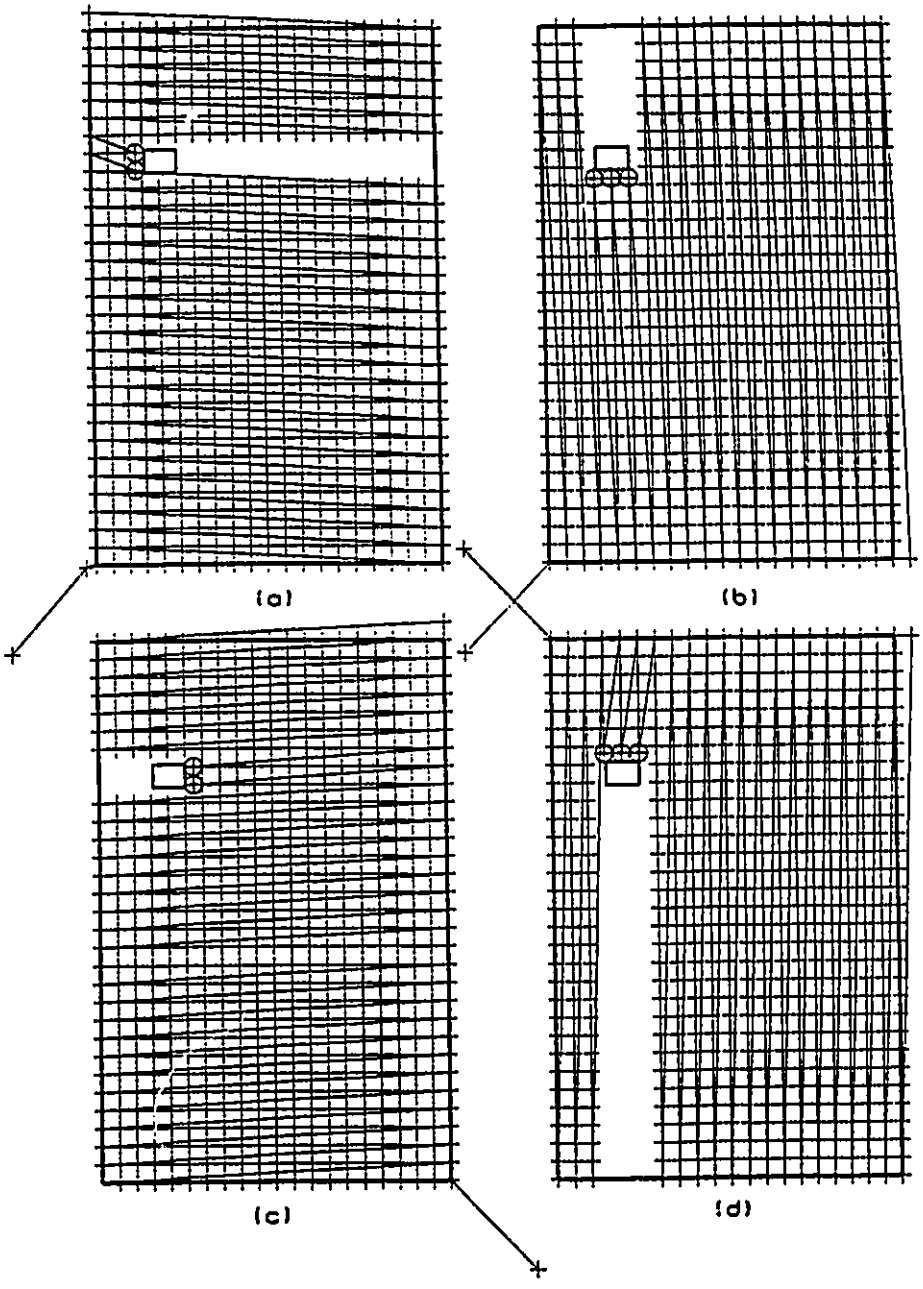


Figure 4.6.1 Orthogonally approachable machining areas for layer at Z = 2.1875

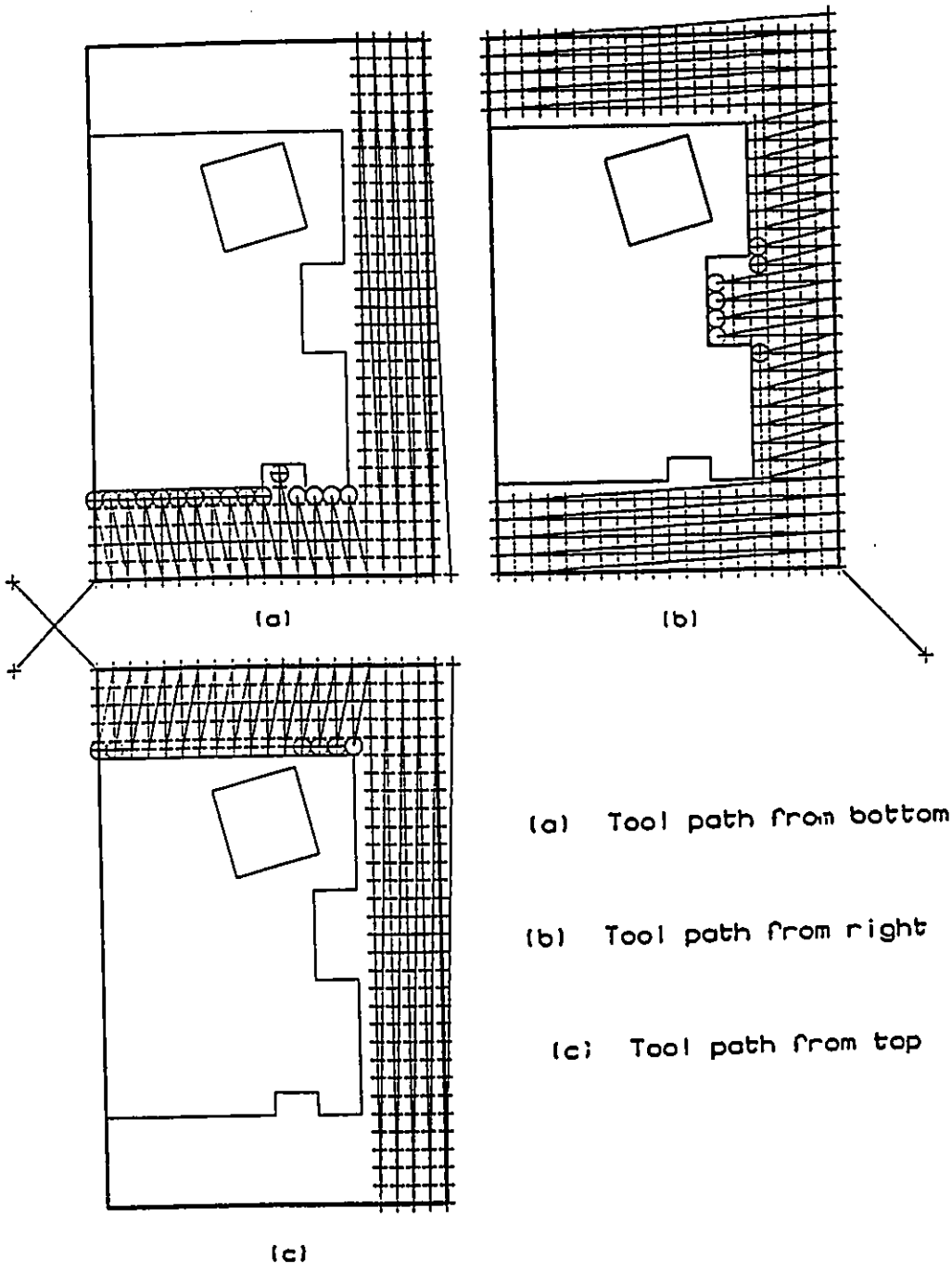


Figure 4.6.2 Orthogonally approachable machining areas for layer at Z = 2.00

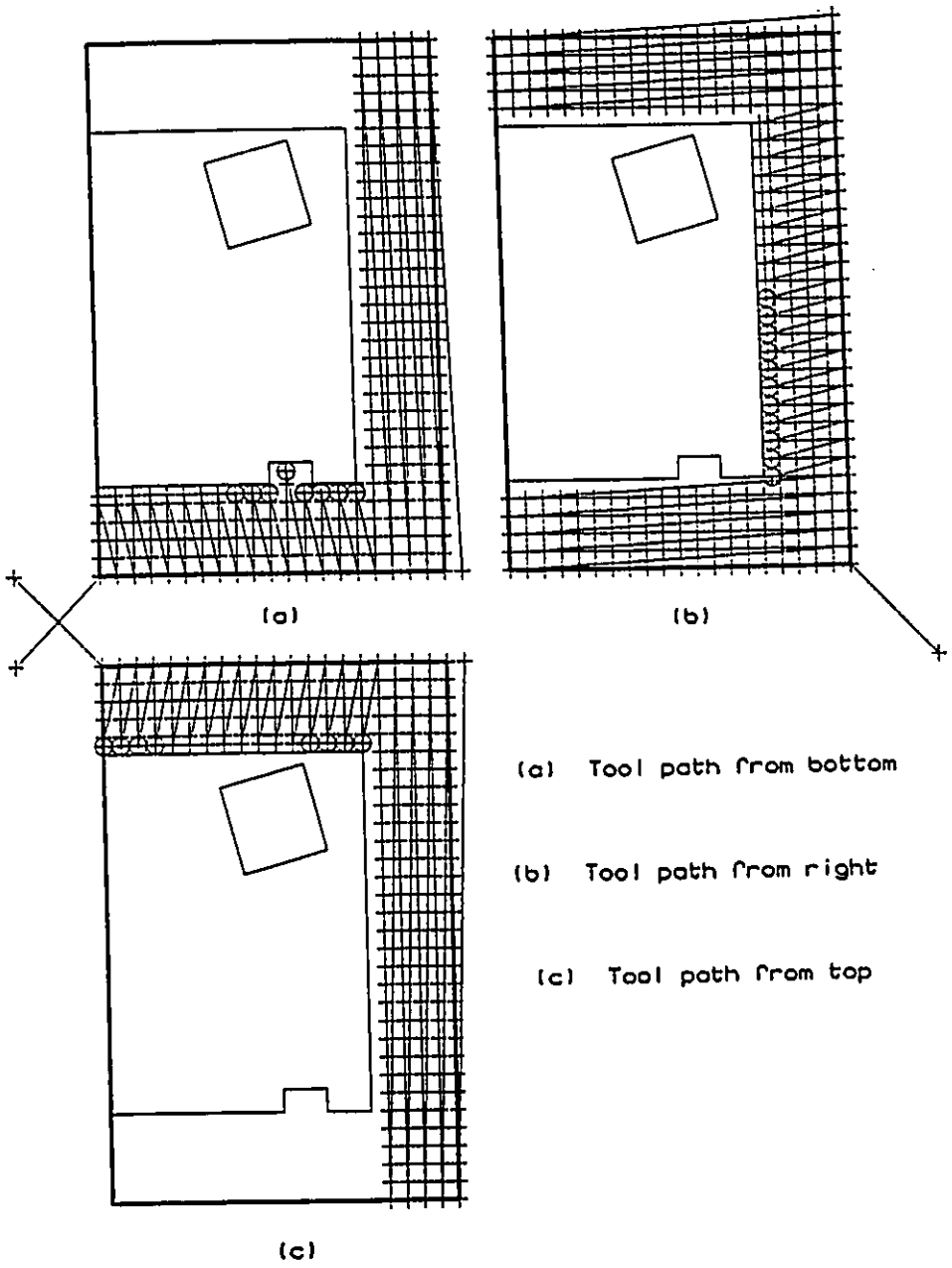


Figure 4.6.3 Orthogonally approachable machining areas for layer at Z = 1.75

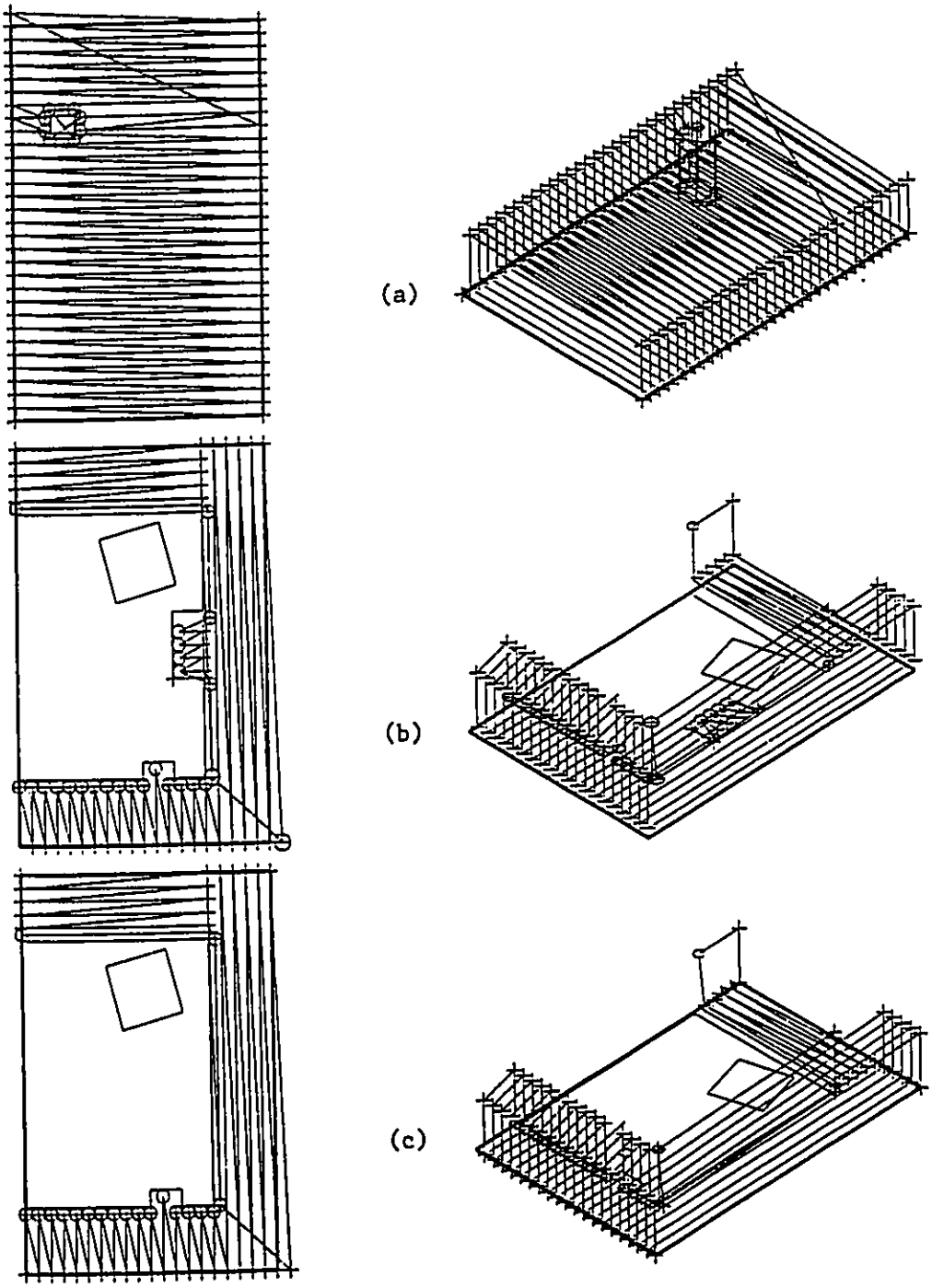


Figure 4.7 Machining areas without redundant coverage

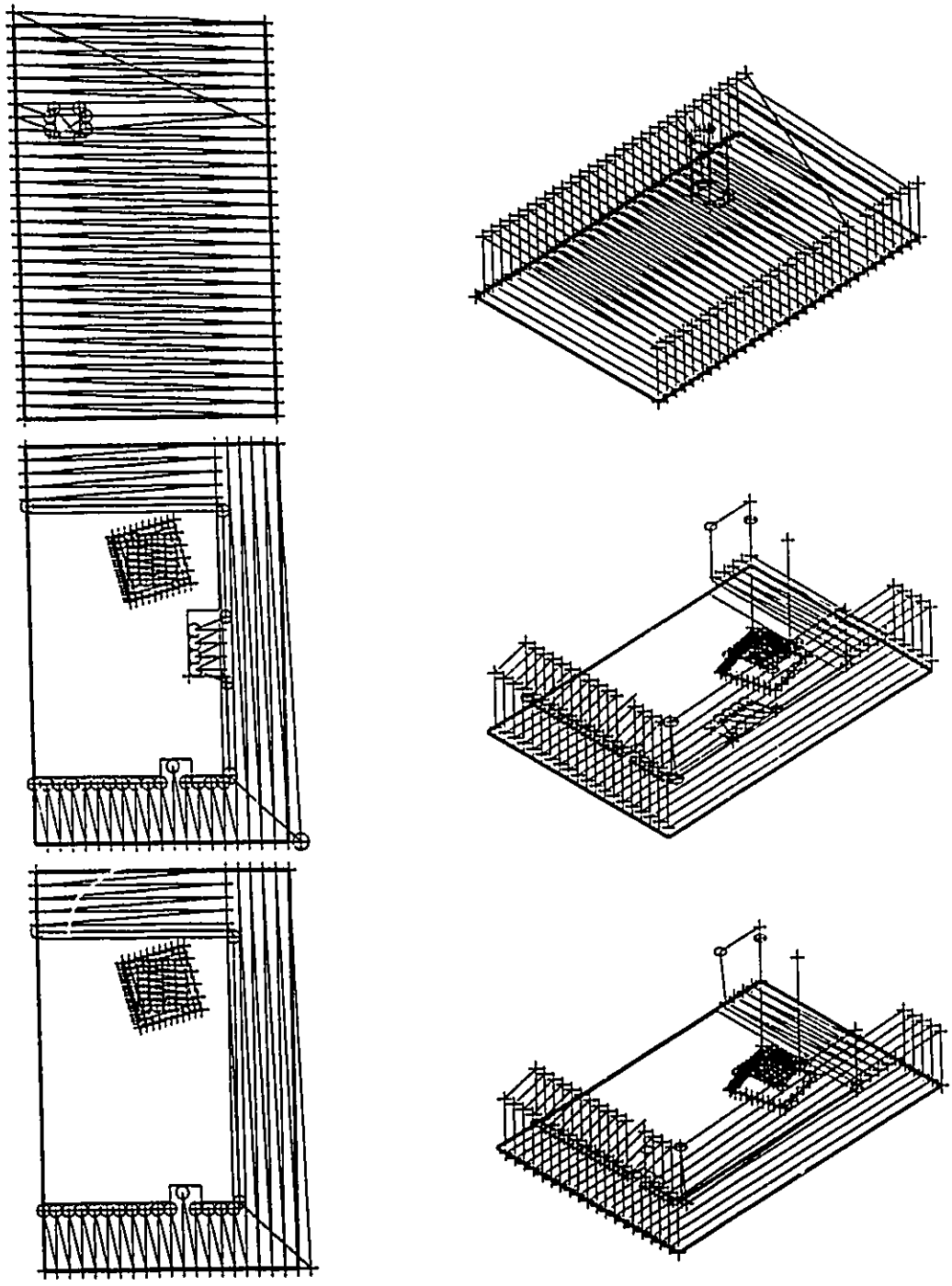


Figure 4.8 Final tool paths including internal and external features

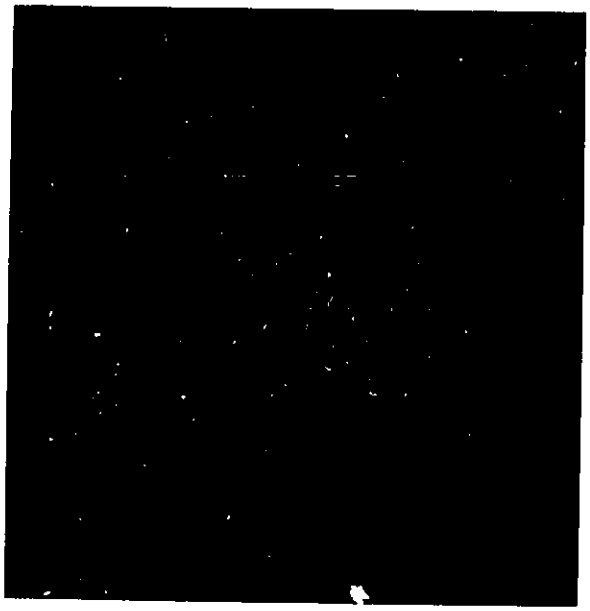


Figure 4.9 Machined part showing internal and external features

Chapter 5

CONCLUSIONS AND FUTURE RECOMMENDATIONS

5.1 Conclusions

A STEP based tool path generation system has been developed for rough machining of planar surfaces. The system is currently implemented in a UNIGRAPHICS environment. However, it can be easily implemented in PCs subject to the availability of PC version CAD/CAM software.

The tool path generation system consists of the following modules:

1. *Data extraction module*

   This module is designed to acquire geometric information from STEP part file. It is developed based on a hierarchical tracing procedure to utilize the top-down structure of STEP data files. The output data of the module are the coordinates of all surface vertices of the part boundary, internal and external features.

2. *Slicing module*

This module is used to section a part and a stock containing the part into a number of parallel layers. Each of these layers is perpendicular to the tool spindle. By using this module, the 3D problem can be reduced to a 2D problem. The slicing process is performed based on boundary equations which are constructed using the vertex information extracted from a STEP data file. The advantages of this approach include: a) tool path generation process is simplified; and b) implementation of the tool path is facilitated since feedrate control in a planar surface is easier than along a curved path.

3. *Tool path generation module*

Two methods are used to create CL files for different machining areas in this module. A 4-directional method is employed to create CL file for the orthogonally approachable machining are. A divide-and-conquer method is applied to the internal features and some external features that cannot be processed by the 4-directional method.

4. *NC code generator*

This generator is developed to automatically translate all the tool paths created by the previous module into NC programs. The NC programs can be used directly to drive an NC machine.

Since tool path is automatically generated and no intermediate data exchange is needed due to the application of STEP standard, a shorter manufacturing cycle can be expected. The application of the system has been demonstrated using an example part.

5.2 Recommendations for Future Work

This study is only a first step in applying STEP to tool path planning. Based on the framework proposed in this thesis, the following can be carried out in the future:

1. *Alternative tool path generation for rough machining*

   Since the purpose of this thesis is to develop a system framework, effort is not made to explore the optimality of the tool path. However, different tool path generators can be easily incorporated into this system. A module can also be designed to record the length of cut, number of tool lifts, number of plunges, and distance of total rapid traverses. The alternative tool paths can thus be generated and evaluated using the above criteria.

2. *Data extraction of curved surfaces*

   In the current system only planar surfaces can be handled. Extracting data for curved surface is a very interesting and yet challenging issue. The proposed system may be expanded for this purpose.

3 *Tool path generation for finish machining*

Another possible extension of the current work is to generate tool path for finish machining. However, since most finish machining involves curved surfaces and needs 3D cutting, this work will greatly rely on the success in data extraction of curved surfaces.

REFERENCES

Bala, M., and Chang, T. C., 1991, "Automatic Cutter Selection and Optimal Cutter Path Generation for Prismatic Parts", *International Journal of Production Research*, Vol. 29, No. 11, 2163-2176.

Bloor, M. S., and Owen, J., 1991, "CAD/CAM Product Data Exchange: The Next Step", *Computer-Aided Design*, Vol. 23, No. 4, 237-243.

Bobrow, J. E., 1985, "NC Machine Tool Path Generation from CSG part Representations", *Computer-Aided Design*, Vol. 17, No. 2, 69-75.

Brandli, N., and Mittelstaedt, M., 1989, "Exchange of Solid Models: Current State and Future Trends", *Computer-Aided Design*, Vol. 21, No. 2, 87-96.

Broomhead, P., and Edkins, M, 1986, "Generating NC data at the machine tool for the manufacture of freeform surfaces", *International Journal of Production Research*, Vol. 24, No. 1, 1-14.

Catania, G., 1992, "A Computer-aided Prototype system for NC Rough Milling of Free-form Shaped Mechanical Part-pieces", *Computers in Industry*, Vol. 20, 275-293.

Chen, Y. D., Ni, J., and Wu, S. M., 1993, "Real time CNC Tool Path Generation for Machining IGES Surfaces", *Transactions of the ASME*, Vol. 115, 480-486.

Choi, B. K., Chung, Y. C., Park, J. W., and Kim, D. H., 1994, "Unified CAM System Architecture for Die and Mould Manufacturing", *Computer-Aided Design*, Vol. 26, No. 3, 235-243.

Choi, B. K., and Lee, C. S., 1988, "Compound Surface Modelling and Machining", *Computer-Aided Design*, Vol. 20, No. 2, 127-136.

Conkol, G. K., 1994, "The Role of CAD/CAM in CIM: The User Perspective", SME Blue Book Series, December.

Cox, J. J., Takezaki, Y., Ferguson, H. R. P., Kohkonen, K. E., and Mulkay, E. L., 1994, "Space filling curves in tool path application", *Computer-Aided Design*, Vol. 26, No. 3, 215-224.

Davies, B. J., Joseph, A., and Kalta, M., 1991, "An Expert System to Generate NC Data from CAD Product Models", *Journal of Material Processing Technology*, Vol. 26, 227-236.

Deshmukh, A., and Wang, H., 1993, "Tool Path Planning for NC Milling of Convex Polygonal Faces", *International Journal of Advance Manufacturing Technology*, Vol. 8, 17-24.

Dolenc, A., and Makela, I., 1994, "Slicing Procedures for Layered Manufacturing Techniques", *Computer-Aided Design*, Vol. 26, No. 2, 119-127.

Dong, Z., Li, H., and Vickers, G. W., 1993, "Optimal Rough Machining of Sculptured Parts on a CNC Milling Machine", Transactions of the ASME, *Journal of Engineering industry*, Vol. 115, 425-431.

Elber, G., and Cohen, E, 1994, "Toolpath Generation for Freeform Surface Models", *Computer-Aided Design*, Vol. 26, No. 6, 491-496.

Encarnacao, J., Schuster.R., and Voge. E.,1986, "Product Data Interfaces in CAD/CAM Applications", Springler Verlag, Berlin.

Goldschlager, L. M., 1981, "Short algorithms for Space-filling Curves", *Software Practice and Experiments*, Vol. 11, 99-100.

Griffiths, J. G., 1985, "Table-driven Algorithms for Generating Space-filling Curves", *Computer-Aided Design*, Vol. 17, No. 1, 37-41.

Griffiths, J. G., 1993, "Toolpaths Based on Hilbert's Curve", *Computer-Aided Design*, Vol. 26, No. 11, 839-844.

Gu, P., and Chan, K., 1995, "Product Modelling Using STEP", *Computer-Aided Design*, Vol. 27, No. 3, 163-179.

Gunsekera, J. S., 1989, CAD/CAM of Dies, Ellis Harwood Limited, New York.

Held, M, 1993, "A Fast Incremental Algorithm for Computing the Voronoi Diagram of a Planar Shape", Communicating with Virtual World, Springler-Verlag, 318-329.

Held, M., Luckas, G., and Andor, L., 1994, "Pocket Machining Based on Contour Parallel Tool Path Generated by Means of Proximity Maps", *Computer-Aided Design*, Vol. 26, No. 3, 189-202.

Hwang, J. S., 1992, "Interference-free Tool-path Generation in the NC machining of Parametric Compound Surface", *Computer-Aided Design*, Vol. 24, No. 12, 667-674.

Ing. D. W., and Eversheim. W., 1989, "Requirements on Interface and Data Models for NC Data Transfer in view of Computer Integrated Manufacturing, *CIRP*, Vol. 38, No. 1, 443-446.

ISO., 1992, "Product Data Representation and Exchange Part 42: Integrated Resources: Geometric and Topological Representation", ISO CD 10303-42, issued by International Organization for Standardization ISO TC 184/SC4 N 141, dated 25th Aug. 1992.

ISO.,1992, "Product Data Representation and Exchange Part 204: Application protocol: Mechanical design using boundary representation", ISO CD 10303-204, issued by International Organization for Standardization ISO TC184/SC4/*WG3 N 277, dated 25th Aug. 1992.

ISO., 1994, "Application Protocol: Mechanical Design Using Boundary Representation, Product Data Representation and Exchange", ISO TC 184/SC4/*WG3 N 277, dated 29th April 1994.

Jayaram, S., and Myklebust, A., 1990, "Automatic Generation of Geometry Interface between Application Programs and CADCAM Systems", *Computer-Aided Design*, Vol. 22, No. 1, 50-56.

Kalpakjian, S, 1992, Manufacturing Engineering and Technology, 2nd edition, Addision-Wesley Publishing Company, Inc., New York.

Kim, K., and Biegel, J. E, 1988, "An Integrated Approach to sculptured Surface Design and Manufacture", *Computers in Engineering*, Vol. 14, No. 3, 271-280.

Kim, K., and Jeong, J., 1995, "Tool Path Generation for Machining Free-Form Pockets with Islands", *Computers in industrial Engineering*, Vol. 28, No. 2, 399-407.

Kim, C. B., Park, S., and Yang, M. Y., 1995, "Verification of NC Tool Path for Manual and Automatic Editing of NC code", *International Journal of Production Research*, Vol. 33, No. 3, 659-673.

Kuragano, T., "FRESDAM System for Design of Aesthetically Pleasing Free-form Objects and Generation of Collision-free Tool Paths", *Computer-Aided Design*, Vol. 24, No. 11, 573-581.

Lakkaraju, R., and Raman, S., 1990, "Optimal NC Path Planning: Is it Really Possible ?" , *Computers in Industrial Engineering*, Vol. 19, No. 1-4, 462-464.

Lakkaraju, R. K., Raman, S., and Irani, S. A, 1992, "An Analytical Model for Optimization of NC Tool Cutting Path", *International Journal of Production Research*, Vol. 30, No. 1, 109-127.

Laurance, N., 1994, "A High Level view of STEP", *Manufacturing Review*, Vol. 7, No. 1, 39-46.

Lee, K., Kim, T. J., and Hong, S. E, 1994, "Generation of Toolpath with Selection of Proper Tools for Rough Cutting Process", *Computer-Aided Design*, Vol. 26, No. 11, 822-830.

Lee, Y., and Chang, T. C, 1994, "Using Virtual Boundaries for the Planning and Machining of Protrusion Free-form Features", *Computers in Industry*, Vol. 25, 173-187.

Li, H., Dong, Z., and Vickers., 1994, "Optimal Toolpath Pattern Identification for Single Island, Sculptured Part Rough Machining Using Fuzzy Pattern Analysis", *Computer-Aided Design*, Vol. 26, No. 11, 787-795.

Marshall, S., and Griffiths, J. G., 1994, "A New Cutter Path Topology for Milling Machine", *Computer-Aided Design*, Vol. 26, No. 3, 204-214.

Mill, F. G., Naish, J. C., and Salmon, J. C., 1994, "Design for Machining with a Simultaneous-engineering Workstation", *Computer-Aided Design*, Vol. 26, No. 7, 521-527.

Mufti, A. A., Morris, M. L., and Spencer, W.B, 1990, "Data Exchange Standards for Computer Aided Engineering and Manufacturing", *International Journal of Computer Applications in Technology*, Vol. 3, No. 2, 70-80.

Nakamur, I., Kojima, T., Kugai, Y., and Kimura, F., 1993, "A CAD Database Interface based on STEP", Interfaces in Industrial systems for Production and Engineering", (B-10), Elsevier Science publishers, North Holland.

Null, A., 1971, "Space-filling Curves, or how to waste time with a plotter", *Software practice and Experiments*", Vol. 1, 403-410.

Perng, D., Chen, Z., and Li, R., 1990, "Automatic 3D Machining Feature Extraction from 3D CSG Solid Input", *Computer-Aided Design*, Vol. 22, No. 5, 285-295.

Raman, S., Lakkaraju, R., 1991, "Tool life and Other Process Constraints for NC Path Planning", *Computers and Engineering*, Vol. 21, No. 1-4, 471-475.

Raman, S., and Lakkaraju, R. K., 1992, "Incorporation of Tool-life Variables in NC Path Planning", *International Journal of Production Research*, Vol. 30, No. 11, 2545-2558.

Raman, S., and Lakkaraju, R., 1993, "The Effect of Tool Life and Other Process Variables in NC Path Planning", *Computer in Industrial Engineering*, Vol. 24, No. 2, 315-328.

Rayan, A. S., 1992, "PDES/STEP: A status report", *Engineering data management, ASME*, 43-47.

Satyanarayana, B., Rao, P. N., and Tewari, N. K., 1990, "An Interactive Programming System for Milling Contours and Pockets", *International Journal of Advance Manufacturing Technology*, Vol. 5, 188-213.

Scholz-Reiter. B., 1992, "CIM Interfaces", Chapman & Hall, Tokyo.

Shah, J. J., and Mathew, A., 1991, "Experimental Investigation of the STEP Form Feature Information Model", *Computer-Aided Design*, Vol. 23, No. 4, 282-296.

Srinivasan, R., Liu, C. R., and Fu, K. S., 1985, "Extraction of manufacturing details from Geometric Models", *Computer and Industrial Engineering*, Vol. 9, No. 2, 125-133.

Suh, Y. S., and Lee, K., 1989,"NC Milling Tool Path Generation for Arbitrary Pockets Defined by Sculptured Surfaces", *Computer-Aided Design*, Vol. 22, No. 5, 273-284.

Tseng, Y. J., and Joshi, S. B., 1994, "Recognizing Multiple Interpretations in 2.5D Machining of Pockets", *International Journal of Production Research*, Vol. 32, No. 5, 1063-1086.

Vergeest, J. S. M., 1991, "CAD Surface Data Exchange Using STEP", *Computer-Aided Design*, Vol. 23, No. 4, 269-281.

Vickers, G. W., and Quan, K. W, 1989., "Ball-mills Versus End-mills for Curved surface Milling", *ASME Journal of Engineering for industry*, Vol. 111, February, 22-26.

Vickers, G. W., Li, H., and Dong, Z., 1992, "Automated Rough Machining of Curved Surfaces", *Proceedings of the CSME Forum SCGM*, 593-597.

Vickers, G. W., and Bradley, C., 1992, "Curved Surface Machining Through Circular Arc Interpolation", *Computers in Industry*, Vol. 19, 329-327.

Vosniakos, G. C., and Davies, B. J., 1990, "An IGES Post Processor for Interfacing CAD and CAPP", *International Journal of Advance Manufacturing Technology*, Vol. 5, 135-164.

Wang, H., Chang, T. C., and Wysk, R. A, 1987, "Online Efficiency of NC Tool Path Planning for Face Milling Operations", *ASME Journal of Engineering for industry*, Vol. 109, November, 370-376.

Wang, H., Chang, H., and Wysk, R. A., 1988, "An Analytical Approach to Optimize NC Tool Path Planning for Face Milling Flat Convex Polygonal Surfaces", *IEE Transactions*, Vol. 20, No. 3, 325-332.

Weck, M., Altintas, Y., and Beer, C, 1994, "CAD Assisted Chater-free NC Tool Path Generation in Milling", *International Journal of Machine Tools Manufacture*, Vol. 34, No. 6, 879-891.

Witten, I. H., and Wyvill, B, 1983, "On the generation and use of space-filling curves", *Software Practice and Experiments*, Vol. 13, 519-525.

Yamaguchi, K, 1984, "Computer Integrated Manufacturing of Surfaces using Octree Encoding", *IEEE Computer Graph and Applications*, Vol. 17, No. 1, 60-65.

Yuen, M, 1987, "An Octree Approach to Rough Machining", *Proceedings of Institute of Mechanical Engineers*, Vol. 2, No. B3, 157-163.

Zhu, C., 1991, "Tool-path Generation in Manufacturing Sculptured Surfaces with a Cylindrical End-milling Cutter", *Computers in Industry*, Vol. 17, 385-389.

APPENDIX 1

(STEP file for a box)

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('BREP-file', 'compatible with DIS P41,P42,P43'),
                 'BREP VERSION Feb. 1993', 'updated 15-March-1993'),
                  'BREP_AP conformance class 2, rectangular box 100/50/25');

FILE_NAME('box100-50-25.stp', '1993-03-03T18:11:01',
         ('Hermann Ruess, Peter Schild'),
         ('HEWLETT-PACKARD GmbH, MDD R&D',
          'Herrenberger Str. 130, 7030 Boeblingen, Germany'),
          'STEP 1.x (AP 204)', 'Preprocessor Version: handcrafted ',
          'Precision Engineering ', 'Hermann J. Ruess');
FILE_SCHEMA(('part_204_brep_product_schema'));
ENDSEC;

DATA;

#1 = APPLICATION_CONTEXT('status-label', 'AIM-schemaname-label',
                          AP-yearnumber, 'appl-text');
#3 = PRODUCT_CONTEXT('name-label', #1, 'disciplinetype-label');
#4 = PRODUCT(id-identifier, 'name-label', 'description-text', #3);
#5 = PRODUCT_VERSION(id-identifier, 'descript-text', #4);
#6 = PRODUCT_DEFINITION_CONTEXT('name-label' ,#1, 'life_cycle_stage-label');
#7 = PRODUCT_DEFINITION('descript-text', #5, #6);
#8 = PRODUCT_DEFINITION_SHAPE(#7);

#17 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 0.00000000000));
#18 = VERTEX_POINT(#17);
#19 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 0.00000000000));
#20 = VERTEX_POINT(#19);
#14 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 0.00000000000));
#15 = DIRECTION((0.00000000000, 1.00000000000, 0.00000000000));
#16 = LINE(#14, #15);
#21 = EDGE_CURVE(#18, #20, #16, .T.);
#97 = ORIENTED_EDGE(*,*,#21, .T.);
#69 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 25.00000000000));
#70 = VERTEX_POINT(#69);
#92 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 0.00000000000));
#93 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#94 = LINE(#92, #93);
#95 = EDGE_CURVE(#20, #70, #94, .T.);
#96 = ORIENTED_EDGE(*,*,#95, .T.);
#55 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 25.00000000000));
#56 = VERTEX_POINT(#55);
#73 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 25.00000000000));
#74 = DIRECTION((0.00000000000, 1.00000000000, 0.00000000000));
#75 = LINE(#73, #74);
```

```
#76 = EDGE_CURVE(#56, #70, #75, .T.);
#91 = ORIENTED_EDGE(*,*,#76, .F.);
#86 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 0.00000000000));
#87 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#88 = LINE(#86, #87);
#89 = EDGE_CURVE(#18, #56, #88, .T.);
#90 = ORIENTED_EDGE(*,*,#89, .F.);
#98 = EDGE_LOOP((#97, #96, #91, #90));
#81 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 0.00000000000));
#82 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#83 = DIRECTION((0.00000000000, 1.00000000000, 0.00000000000));
#84 = AXIS2_PLACEMENT_3D(#81, #82, #83);
#85 = PLANE(#84);
#99= FACE_BOUND(#98 ,.T.);
#100 = FACE_SURFACE((#99), #85,.T.);
#26 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 0.00000000000));
#27 = VERTEX_POINT(#26);
#23 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 0.00000000000));
#24 = DIRECTION((-1.00000000000, 0.00000000000, 0.00000000000));
#25 = LINE(#23, #24);
#28 = EDGE_CURVE(#20, #27, #25, .T.);
#113 = ORIENTED_EDGE(*,*,#28, .T.);
#62 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 25.00000000000));
#63 = VERTEX_POINT(#62);
#108 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 0.00000000000));
#109 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#110 = LINE(#108, #109);
#111 = EDGE_CURVE(#27, #63, #110, .T.);
#112 = ORIENTED_EDGE(*,*,#111, .T.);
#66 = CARTESIAN_POINT((100.00000000000, 50.00000000000, 25.00000000000));
#67 = DIRECTION((-1.00000000000, 0.00000000000, 0.00000000000));
#68 = LINE(#66, #67);
#71 = EDGE_CURVE(#70, #63, #68, .T.);
#107 = ORIENTED_EDGE(*,*,#71, .F.);
#106 = ORIENTED_EDGE(*,*,#95, .F.);
#114 = EDGE_LOOP((#113, #112, #107, #106));
#101 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 0.00000000000));
#102 = DIRECTION((0.00000000000, 1.00000000000, -0.00000000000));
#103 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#104 = AXIS2_PLACEMENT_3D(#101, #102, #103);
#105 = PLANE(#104);
#115 = FACE_BOUND(#114, .T.);
#116 = FACE_SURFACE((#115), #105,.T.);
#33 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 0.00000000000));
#34 = VERTEX_POINT(#33);
#30 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 0.00000000000));
```

```
#31 = DIRECTION((0.00000000000, -1.00000000000, 0.00000000000));
#32 = LINE(#30, #31);
#35 = EDGE_CURVE(#27, #34, #32, .T.);
#129 = ORIENTED_EDGE(*,*,#35, .T.);
#53 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 25.00000000000));
#54 = VERTEX_POINT(#53);
#124 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 0.00000000000));
#125 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#126 = LINE(#124, #125);
#127 = EDGE_CURVE(#34, #54, #126, .T.);
#128 = ORIENTED_EDGE(*,*,#127, .T.);
#59 = CARTESIAN_POINT((0.00000000000, 50.00000000000, 25.00000000000));
#60 = DIRECTION((0.00000000000, -1.00000000000, 0.00000000000));
#61 = LINE(#59, #60);
#64 = EDGE_CURVE(#63, #54, #61, .T.);
#123 = ORIENTED_EDGE(*,*,#64, .F.);
#122 = ORIENTED_EDGE(*,*,#111, .F.);
#130 = EDGE_LOOP((#129, #128, #123, #122));
#117 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 0.00000000000));
#118 = DIRECTION((-1.00000000000, 0.00000000000, 0.00000000000));
#119 = DIRECTION((0.00000000000, 1.00000000000, 0.00000000000));
#120 = AXIS2_PLACEMENT_3D(#117, #118, #119);
#121 = PLANE(#120);
#131 = FACE_BOUND(#130,.T.);
#132 = FACE_SURFACE((#131), #121,.T.);
#37 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 0.00000000000));
#38 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#39 = LINE(#37, #38);
#40 = EDGE_CURVE(#34, #18, #39, .T.);
#141 = ORIENTED_EDGE(*,*,#40, .T.);
#140 = ORIENTED_EDGE(*,*,#89, .T.);
#50 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 25.00000000000));
#51 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#52 = LINE(#50, #51);
#57 = EDGE_CURVE(#54, #56, #52, .T.);
#139 = ORIENTED_EDGE(*,*,#57, .F.);
#138 = ORIENTED_EDGE(*,*,#127, .F.);
#142 = EDGE_LOOP((#141, #140, #139, #138));
#133 = CARTESIAN_POINT((100.00000000000, 0.00000000000, 0.00000000000));
#134 = DIRECTION((0.00000000000, -1.00000000000, 0.00000000000));
#135 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#136 = AXIS2_PLACEMENT_3D(#133, #134, #135);
#137 = PLANE(#136);
#143 = FACE_BOUND(#142,.T.);
#144 = FACE_SURFACE((#143),#137,.T.)
#77 = ORIENTED_EDGE(*,*,#76, .T.);
```

```
#72 = ORIENTED_EDGE(*,*,#71, .T.);
#65 = ORIENTED_EDGE(*,*,#64, .T.);
#58 = ORIENTED_EDGE(*,*,#57, .T.);
#78 = EDGE_LOOP((#77, #72, #65, #58));
#45 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 25.00000000000));
#46 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#47 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#48 = AXIS2_PLACEMENT_3D(#45, #46, #47);
#49 = PLANE(#48);
#79 = FACE_BOUND(#78, .T.);
#80 = FACE_SURFACE((#79),#49,.T.);
#41 = ORIENTED_EDGE(*,*,#40, .F.);
#36 = ORIENTED_EDGE(*,*,#35, .F.);
#29 = ORIENTED_EDGE(*,*,#28, .F.);
#22 = ORIENTED_EDGE(*,*,#21, .F.);
#42 = EDGE_LOOP((#41, #36, #29, #22));
#9 = CARTESIAN_POINT((0.00000000000, 0.00000000000, 0.00000000000));
#10 = DIRECTION((0.00000000000, 0.00000000000, 1.00000000000));
#11 = DIRECTION((1.00000000000, 0.00000000000, 0.00000000000));
#12 = AXIS2_PLACEMENT_3D(#9, #10, #11);
#13 = PLANE(#12);
#43 = FACE_BOUND(#42, .F.);
#44 = FACE_SURFACE((#43), #13,.F.);
#145 = CLOSED_SHELL((#100, #116, #132, #144, #80, #44));
#146 = MANIFOLD_SOLID_BREP(#145);

#2000= (LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI., .METRE.));

#147 = GEOMETRIC_REPRESENTATION_CONTEXT(3)
       GLOBAL_UNIT_ASSIGNED_CONTEXT(#2000))
       REPRESENTATION_CONTEXT('3D-mm','3D Cartesian, lengths in mm'));

#148 = SHAPE_REPRESENTATION((#146), #147);
#149 = SHAPE_DEFINITION_REPRESENTATION(#148, #8);
ENDSEC;
END-ISO-1030-21;
```

APPENDIX 2

(C Source listings)

```
/********* PROGRAM TO PROCESS STEP FILE FOR EXTRACTION OF PLANAR
SURFACES ***********/
#include <stdio.h>
#include <stdlib.h>
#define LINESIZE 80
h_key ()
{ printf("Hit any key to continue .....\n");
  getchar(); }
main()
{
char *result;
char *result1;
char line[LINESIZE];
char dum1[10],dum2[80];
int l_no;
int e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14;
int old,count;
int type[99];
int i;
float x,y,z;
char str1;
FILE *inp;
FILE *inp1;
FILE *out;
/* intialization  */
for (i=0; i<100; ++i)
type[i]=0;
printf("Processing FACE_SURFACE from STEP file:  \n\n\n ");
inp=fopen("shafee.step","r");
out=fopen("face_surface.dump","w");
  /*     do dump all face surface on file                */
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
if (sscanf(line,"#%d = FACE_SURFACE %s",&l_no,&dum2) == 2)
{
fprintf(out,"%s",line);
}
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
/*      to dump all FACE_SURFACE element numbers on file  */
printf("Writing FACE_SURFACE elemnts on face_surface.nos file:\n\n\n ");
inp=fopen("face_surface.dump","r");
out=fopen("face_surface.nos","w");
result=fgets(line,LINESIZE,inp);
```

```
while (result != NULL)
{
if (sscanf(line,"#%d = FACE_SURFACE ((#%d),  #%d, %s",&l_no,&e1,&e2,&dum2) == 4)

{
fprintf(out,"%d\n%d\n",e1,e2);
}
if (sscanf(line,"#%d = FACE_SURFACE ((#%d, #%d),  #%d, %s",&l_no,&e1,&e2,&e3,&dum2)
== 5)
{
fprintf(out,"%d\n%d\n%d\n",e1,e2,e3);
}
if  (sscanf(line,"#%d  =  FACE_SURFACE  ((#%d,  #%d,  #%d,  #%d,  #%d),     #%d,
%s",&l_no,&e1,&e2,&e3,&e4,&e5,&e6,&dum2) == 8)
{
fprintf(out,"%d\n%d\n%d\n%d\n%d\n%d\n",e1,e2,e3,e4,e5,e6);
}
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
printf("Processing FACE_BOUNDS & FACE_OUTER_BOUNDS for each FACE_SURFACE
\n ");
/*      To dum all face bound or face outer bounds on file */
inp=fopen("shafee.step","r");
inp1=fopen("face_surface.nos","r");
out=fopen("face_bound.nos","w");
result1=fgets(line,LINESIZE,inp1);
while (result1 != NULL)
{
 sscanf(line,"%d",&e1);
     result=fgets(line,LINESIZE,inp);
     while (result != NULL)
     {
       if (sscanf(line,"#%d = FACE_BOUND (#%d, %s",&l_no,&e2,&dum2) == 3)
       {
         if (e1 == l_no)
           {
            fprintf(out,"%d\n",e2);
           }
       }
       if (sscanf(line,"#%d = FACE_OUTER_BOUND (#%d, %s",&l_no,&e2,&dum2) == 3)
       {
         if (e1 == l_no)
           {
             fprintf(out,"%d\n",e2);
           }
```

```
    }
   result=fgets(line,LINESIZE,inp);
   }
fclose(inp);
inp=fopen("shafee.step","r");
result1=fgets(line,LINESIZE,inp1);
}
fclose(inp);
fclose(inp1);
fclose(out);
h_key ();
printf("Processing EDGE_LOOPS for each FACE_BOUNDS or  FACE_OUTER_BOUND\n ");
/*          to dump all edge loop on file          */
inp=fopen("shafee.step","r");
out=fopen("edge_loop.dump","w");
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
if (sscanf(line,"#%d = EDGE_LOOP %s",&l_no,&dum2) == 2)
{
fprintf(out,"%s",line);
}
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
printf("Writing EDGE_LOOPS to file  :\n ");
/* to dump all EDGE_LOOP elements in file  */
inp=fopen("edge_loop.dump","r");
out=fopen("oriented_edge.nos","w");
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
if (sscanf(line,"%s #%d #%d #%d",&dum2,&e1,&e2,&e3) == 4)
{
printf("%d %d %d  \n",e1,e2,e3);
}
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
printf("Processing ORIENTED_EDGE  for each EDGE_LOOP \n ");
/* to dump all oriented_edge in file  */
inp=fopen("shafee.step","r");
out=fopen("oriented_edge.dump","w");
result=fgets(line,LINESIZE,inp);
```

```
while (result != NULL)
{
if (sscanf(line,"#%d = ORIENTED_EDGE (*, *, #%d,",&l_no,&e1) == 2)
{
fprintf(out,"%d %d   \n",l_no,e1);
}

result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
printf("Writing ORIENTED_EDGE  elements to file  :\n ");
/* to pair  oriented_edge in file  */
count=1;
inp=fopen("oriented_edge.dump","r");
out=fopen("oriented_pair.dump","w");
result=fgets(line,LINESIZE,inp);
sscanf(line,"%d %d",&e1,&e2);
old=e1;
fprintf(out,"%d %d\n",e2,count);
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
sscanf(line,"%d %d",&e1,&e2);
  if (e1 == (old+10))
       {
        fprintf(out,"%d %d\n",e2,count);
        old=e1;
       }
    else
       {
       count=count+1;
        fprintf(out,"%d %d\n",e2,count);
        old=e1;
       }
 result=fgets(line,LINESIZE,inp);
 }
 fclose(inp);
 fclose(out);
 h_key ();
 printf("Processing EDGE_CURVE  for each ORIENTED_EDGE\n ");
 /* writng all elements of connected EDGE_CURVE on to file   */
 inp=fopen("shafee.step","r");
 inp1=fopen("oriented_pair.dump","r");
 out=fopen("edge_curve.ele","w");
 result1=fgets(line,LINESIZE,inp1);
 while (result1 != NULL)
```

```
{
sscanf(line,"%d %d",&e1,&e2);
   result=fgets(line,LINESIZE,inp);
   while (result != NULL)
   {
    if (sscanf(line,"#%d = EDGE_CURVE (#%d, #%d, #%d,%s",&l_no,&e3,&e4,&e5,&dum2)
== 5)
         {
          if(l_no == e1)
           {
          fprintf(out,"%d %d %d %d\n",e3,e4,e5,e2);
           }
         }
     result=fgets(line,LINESIZE,inp);
     }
fclose(inp);
inp=fopen("shafee.step","r");
result1=fgets(line,LINESIZE,inp1);
}
fclose(inp);
fclose(inp1);
fclose(out);
h_key 0;
printf("Processing LINE elements from STEP file: \n ");
/* writing line_nos & its elements to file */
inp=fopen("shafee.step","r");
out=fopen("line_nos.ele","w");
    result=fgets(line,LINESIZE,inp);
    while (result != NULL)
    {
     if (sscanf(line,"#%d = LINE (#%d, #%d)",&l_no,&e1,&e2) == 3)
           {
             fprintf(out,"%d %d %d\n",l_no,e1,e2);
           }
 result=fgets(line,LINESIZE,inp);
    }
 fclose(inp);
 fclose(out);
 h_key 0;
 printf("Processing EDGE_CURVE elements for each EDGE_LOOP \n ");
 /* picking the 1st element of edge curve for each edge loop    */
 inp=fopen("edge_curve.ele","r");
 out=fopen("check_line.dum","w");
 result=fgets(line,LINESIZE,inp);
 sscanf(line,"%d %d %d %d",&e1,&e2,&e3,&e4);
 fprintf(out,"%d %d\n",e3,e4);
 old=e4;
 result=fgets(line,LINESIZE,inp);
```

```
while (result != NULL)
{
sscanf(line,"%d %d %d %d",&e1,&e2,&e3,&e4);
   if (e4 != old) { fprintf(out,"%d %d\n",e3,e4); old=e4;      }
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key ();
printf("Checking EDGE_CURVE elements for each EDGE_LOOP for LINE segments\n ");

/* writing line elemnts to a file  */
inp=fopen("check_line.dum","r");
inp1=fopen("line_nos.ele","r");
count=0;
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
sscanf(line,"%d %d",&e1,&e2);
         result1=fgets(line,LINESIZE,inp1);
         while (result1 != NULL)
         {
          sscanf(line,"%d %d %d",&l_no,&e3,&e4);
          if (l_no == e1) { type[count]=e2; count=count+1; }
         result1=fgets(line,LINESIZE,inp1);
         }
fclose(inp1);
inp1=fopen("line_nos.ele","r");
result=fgets(line,LINESIZE,inp);
}
/*
for (i=0; i<100; ++i)
  {
    printf("%d\n",type[i]);
  }
*/
fclose(inp);
fclose(inp1);
h_key ();
printf("Forming planes by matched LINE segments\n ");
/*   picking all planes  formed by line segments  */
inp=fopen("edge_curve.ele","r");
out=fopen("cartesian.ele","w");
result=fgets(line,LINESIZE,inp);
while (result != NULL)
{
sscanf(line,"%d %d %d %d",&e1,&e2,&e3,&e4);
       for (i=0; i<100; ++i)
```

```
          {
           if (e4 == type[i]) {fprintf(out,"%d %d\n%d %d\n",(e1-10),e4,(e2-10),e4);}
          }
result=fgets(line,LINESIZE,inp);
}
fclose(inp);
fclose(out);
h_key 0;
printf("Writing vertex information of each plane \n ");
/* writing the final cartesian coordinates with plane nos  */
inp=fopen("shafee.step","r");
inp1=fopen("cartesian.ele","r");
out=fopen("plane.pts","w");
 result1=fgets(line,LINESIZE,inp1);
while (result1 != NULL)
{
sscanf(line,"%d %d",&e1,&e2);
   result=fgets(line,LINESIZE,inp);
   while (result != NULL)
   {
    if (sscanf(line,"#%d = CARTESIAN_POINT ((%f, %f,%f))"
        ,&l_no,&x,&y,&z) == 4)
            {
              if (e1 == l_no)
               {
                fprintf(out,"%f %f %f %d\n",x,y,z,e2);
               }
            }
    result=fgets(line,LINESIZE,inp);
    }
 fclose(inp);
 inp=fopen("shafee.step","r");
 result1=fgets(line,LINESIZE,inp1);
 }
 }
 **********TO GENERATE PLANES **********
 #include <stdio.h>
 #include <math.h>
 main()
 {
 double x1,x2,x3,y1,y2,y3,z1,z2,z3;
 double v11,v12,v13,w11,w12,w13,i,j,k,sum,ii;
 FILE *pts;
 FILE *out;
 pts=fopen("pts.dat","r");
 out=fopen("eqs.out","w");
 for (ii=1; ii<=30; ii=ii+1)
 {
```

```
fscanf(pts,"%lf %lf %lf %lf %lf %lf %lf %lf %lf",&x1,&y1,&z1,&x2,&y2,&z2,&x3,&y3,&z3);
/* printf("%f %f %f %f %f %f %f %f %f\n",x1,y1,z1,x2,y2,z2,x3,y3,z3);        */
v11=x2-x1;
v12=y2-y1;
v13=z2-z1;
w11=x3-x1;
w12=y3-y1;
w13=z3-z1;
/* printf("%f %f %f %f %f %f   \n",v11,v12,v13,w11,w12,w13);   */
i=((v12*w13)-(v13*w12));
j= (-1) * (  ((v11*w13)-(v13*w11)) );
k=((v11*w12)-(v12*w11));
/* printf("%f %f %f\n",i,j,k);      */
sum=x1*i+y1*j+z1*k;
/*  printf("  %f  \n", sum);        */
/*
printf(" %lf %lf %lf\n",x1,y1,z1);
printf(" %lf %lf %lf\n",x2,y2,z2);
printf(" %lf %lf %lf\n",x3,y3,z3);    */
printf("%lf %lf %lf %lf\n",i,j,k,sum);
fprintf(out,"%lf %lf %lf %lf\n",i,j,k,sum);
}
}
**********TO GENERATE CROSS SECTIONAL POINTS********
#include <stdio.h>
#include <math.h>
main()
{
int i,j,k;
int count=0;
FILE *pts;
pts=fopen("pts.txt","w");
/* for plane a*/
for(i=2500; i<=2500; i=i+100)
{
for(j=2500; j<=3500; j=j+100)
{
for (k=1750; k<= 2000; k = k+10)
{
if ( ( (225000*i) + (0*j) + (0*k) )== 562500000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
/* for plane b */
for(i=2500; i<=3000; i=i+100)
{
for(j=2500; j<=2500; j=j+100)
{
```

```
for (k=1750; k<= 2000; k = k+10)
{
if ( ( (0*i) - (125000*j) + (0*k) )== -312500000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
/* for plane c */
for(i=2625; i<=3000; i=i+100)
{
for(j=3500; j<=3500; j=j+100)
{
for (k=1750; k<= 2000; k = k+10)
{
if ( ( (0*i) - (93750*j) + (0*k) )== -328125000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
/* for plane d     */
/*
for(i=625; i<=1000; i=i+100)
{
for(j=4375; j<=4625; j=j+100)
{
for (k=21875; k<= 21875; k = k+100)
{
if ( ( (0*i) + (0*j) + (93750*k) )==
 2.05078125e09) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/10000.));
}
}}}
*/
/* for plane e */
for(i=1000; i<=1000; i=i+100)
{
for(j=4375; j<=4625; j=j+10)
{
for (k=20000; k<= 21875; k = k+25)
{
if ( ( (46875*i) - (0*j) + (0*k) )== 46875000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/10000.));
}
}}}
/* for plane f */
for(i=625; i<=1000; i=i+10)
{
for(j=4375; j<=4375; j=j+10)
{
for (k=20000; k<= 21875; k = k+25)
```

```
{
if ( ( (0*i) - (70312.5*j) + (0*k) )== -307617187.5) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/10000.));
}
}}}
/* for plane g */
for(i=625; i<=625; i=i+10)
{
for(j=4375; j<=4625; j=j+10)
{
for (k=20000; k<= 21875; k = k+25)
{
if ( ( (46875*i) + (0*j) + (0*k) )== 29296875) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/10000.));
}
}}}
/* for plane h */
for(i=625; i<=1000; i=i+10)
{
for(j=4625; j<=4625; j=j+10)
{
for (k=20000; k<= 21875; k = k+25)
{

if ( ( (0*i) - (70312.5*j) + (0*k) )== -325195312.5) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/10000.));
}
}}}
/* for plane i */
for(i=2000; i<=2500; i=i+100)
{
for(j=1250; j<=1250; j=j+100)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (0*i) - (1000000*j) + (0*k) )== -1250000000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
/* for plane j */
for(i=2000; i<=2000; i=i+100)
{
for(j=1000; j<=1250; j=j+100)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (500000*i) + (0*j) + (0*k) )== 1000000000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
```

```
}
}}}
/* for plane k */
for(i=2500; i<=2500; i=i+100)
{
for(j=1000; j<=1250; j=j+100)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (500000*i) + (0*j) + (0*k) )== 1250000000) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
/* for plane l */
for(i=0; i<=2000; i=i+100)
{
for(j=1; j<=1; j=j+1)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (0*i) + (4*j) + (0*k) )== 4) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1.),(k/1000.));
}
}}}
/* for plane m */
for(i=1600; i<=2570; i=i+1)
{
for(j=3650; j<=3910; j=j+1)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (5.2e05*i)-(1.94e06*j) + (0*k) )== -6.249e09) {
   fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
   /* for plane n */
for(i=1340; i<=1600; i=i+1)
{
for(j=3650; j<=4610; j=j+1)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (1.92e06*i)+(5.2e05*j) + (0*k) )== 4.97e09) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
   /* for plane o */
for(i=1340; i<=2310; i=i+1)
```

```
{
for(j=4620; j<=4870; j=j+1)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (5.0e05*i)-(1.94e06*j) - (0*k) )== -8.2928e09) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
  /* for plane p */
for(i=2310; i<=2570; i=i+1)
{
for(j=3910; j<=4875; j=j+1)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (1.93e06*i)+(5.2e05*j) + (0*k) )== 6.9933e09) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
    /* for plane u */
for(i=2500; i<=3000; i=i+100)
{
for(j=1000; j<=1000; j=j+100)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (0*i)-(1.0*j) + (0*k) )== -1.0e03) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
    /* for plane v */
for(i=0; i<=0; i=i+1)
{
for(j=1000; j<=5000; j=j+100)
{
for (k=0; k<= 2000; k = k+50)
{
if ( ( (8*i)+(0*j) + (0*k) )== 0.0) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
    /* for plane w */
for(i=0; i<=3000; i=i+100)
{
for(j=5000; j<=5000; j=j+100)
{
for (k=0; k<=   2000; k = k+50)
```

```
{
if ( ( (0*i)-(6.0e06*j) + (0*k) )== -30.0e09) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
   /* for plane x */
for(i=3000; i<=3000; i=i+100)
{
for(j=1000; j<=5000; j=j+100)
{
for (k=0; k<=2000; k = k+50)
{
if (j > 2500 && j < 3500 && k > 1750 && k < 2001)
if ( ( (8.0e06*i)+(0*j) + (0*k) )== 24.0e09) {
fprintf(pts,"%f %f %f\n",(i/1000.),(j/1000.),(k/1000.));
}
}}}
}

**********TO EXTRACT POINTS ON A GIVEN LAYER (Z)****
#include <stdio.h>
#include <math.h>
#define LINESIZE 80
main()
{
char *result;
char line[LINESIZE];
float x,y,z,my_z;
FILE *inp;
FILE *out;
inp=fopen("pts.txt","r");
out=fopen("z.out","w");
printf(" Please input the Zvalue:\n");
scanf("%f",&my_z);
printf("%f\n",my_z);
result=fgets(line,LINESIZE,inp);
while (result != NULL)
 {
sscanf(line,"%f %f %f",&x,&y,&z);
if (my_z == z)
   {
     fprintf(out,"%f %f %f\n",x,y,z);
     printf("%f %f %f\n",x,y,z);
   }
result=fgets(line,LINESIZE,inp);
 }
close(inp);
close(out);
```

```
}
*********TO GENERATE THE TOOL PATH FROM RIGHT SIDE ********
#include <stdio.h>
#include <math.h>
#define LINESIZE 80
main()
{
char line[LINESIZE];
char *result;
char file_1[25];
int llx,lly,lrx,lry,urx,ury,ulx,uly;
float sllx,slly,slrx,slry,surx,sury,sulx,suly;
float d,z,tx,ty;
int i,j,step_len_y;
float p1x,p1y,p2x,p2y,p3x,p3y,p4x,p4y;
float temp;
float x,y,zz;
int tog;
float incr_y;
float minx,ptx,pty;
int x_flag;
FILE *out;
FILE *inp;
FILE *tmp;
FILE *tmp1;
printf("\n Key in the input file name ");
printf("\n i.e. output of new_points.,c    : ");
scanf("%25s",&file_1);
inp=fopen(file_1,"r");
out=fopen("ncr.x","w");
tmp=fopen("y.y","w");
tmp1=fopen("a.a","w");
/* data of stock  */
/* give always one decimal accuracy value only */
sllx=0.0;
slly=0.0;
slrx=4.0;
slry=0.0;
surx=4.0;
sury=6.0;
sulx=0.0;
suly=6.0;
llx=sllx*10.0;
lly=slly*10.0;
lrx=slrx*10.0;
lry=slry*10.0;
urx=surx*10.0;
ury=sury*10.0;
```

```
ulx=sulx*10.0;
uly=suly*10.0;
/* data of tool */
printf("ENTER THE DIA Of TOOL:");
scanf("%f",&d);
  /* d=1.0;  */
d=d/2.0;
z=2.18;
tx=llx-1.0;
ty=lly-1.0;
fprintf(out,"%.5f %.5f %.5f\n",tx,ty,z);
  /*      printf("a %f %f %f\n",tx,ty,z);         */
p1x=llx-(1.5*d);
p1y=lly-(1.5*d);
p2x=llx-d;
p2y=lly-d;
p3x=llx-d;
p3y=lly+d;
p4x=llx-(1.5*d);
p4y=lly+d;
fprintf(tmp,"%f   %f   %f   \n%f   %f   %f   \n%f   %f   %f   \n%f   %f   %f
\n",p1x,p1y,z,p2x,p2y,z,p3x,p3y,z,p4x,p4y,z);
tx=(p3x+p1x)/2.0;
ty=(p3y+p1y)/2.0;
fprintf(out,"%.5f %.5f %.5f\n",tx,ty,z);
/* printf("b %f %f %f\n",tx,ty,z);       */
incr_y= 2.0*d;
temp=0.0;
tog=0;
x_flag=0;
for (j=lly; j<=uly; j=j+(2.0*d*10))
{
for (i=llx; i<=lrx; i=i+(2.0*d*10))
{
   if (tog == 0)
        {
         p1x=p2x;
         p1y=p2y;
         p4x=p3x;
         p4y=p3y;
         p2x=p1x+(2.0*d);
         p2y=p1y;
         p3x=p4x+(2.0*d);
         p3y=p4y;
        }
 fprintf(tmp,"%f %f %f\n",(p1x),(p1y),z);
 fprintf(tmp,"%f %f %f\n",(p2x),(p2y),z);
 fprintf(tmp,"%f %f %f\n",(p3x),(p3y),z);
```

```
fprintf(tmp,"%f %f %f\n",(p4x),(p4y),z);
  tog=0;
 result=fgets(line,LINESIZE,inp);
 while (result != NULL)
   {
   sscanf(line,"%f %f %f",&x,&y,&zz);
    if (x > p1x && x < p2x &&  y > p1y && y < p4y)
      {
/* given val of y to find start and end of x val  begin */
       minx=slrx;
       ptx=x;
       pty=y;
       rewind(inp);
       result=fgets(line,LINESIZE,inp);
            while (result != NULL)
                 {
                  sscanf(line,"%f %f %f",&x,&y,&zz);
                   if (pty == y && x < minx)
                       {
                        ptx=x;
                        pty=y;
                        minx=x;
                        x_flag=1;
                       }
                  result=fgets(line,LINESIZE,inp);
                 }
       rewind(inp);
       p3x=ptx;
       p1x=ptx-(2.0*d);
       tx=(p3x+p1x)/2.0;
       ty=(p3y+p1y)/2.0;
       fprintf(out,"%.5f %.5f %.5f \n",tx,ty,z);
      }
 /* given val of y to find start and end of x val end */
    result=fgets(line,LINESIZE,inp);
    if (x_flag == 1) break;
    }
    if (x_flag == 1) break;
    rewind(inp);
 tx=(p3x+p1x)/2.0;
 ty=(p3y+p1y)/2.0;
 fprintf(out,"%.5f %.5f %.5f \n",tx,ty,z);
 /*  printf("c %f %f %f \n",tx,ty,z);
 }
     x_flag=0;
     z=z+1.0;
   fprintf(out,"%.5f %.5f %.5f\n",tx,ty,z);
 /* printf("d %f %f %f\n",tx,ty,z);       */
```

```
    tx=llx;
    temp=lly+incr_y+temp;
    ty=temp;
   fprintf(out,"%.5f %.5f %.5f \n",tx,ty,z);
/* printf("c %f %f %f %f \n",tx,ty,z,d);     */
    z=z-1.0;
    tx=tx;
    ty=ty;
    p1x=tx-d;
    p1y=ty-d;
    p2x=tx+d;
    p2y=p1y;
    p3x=tx+d;
    p3y=ty+d;
    p4x=tx-d;
    p4y=ty+d;
    tog=1;
    }
}
*******TO CLUB TOOL PATHS FROM ANY TWO DIRECTIONS********
#include <stdio.h>
#define LINESIZE 80
main()
{
char line[LINESIZE];
char *res_x;
char *res_d;
char file1[25],file2[25],file3[25];
float x,y,z,x1,y1,z1;
float az;
int flag;
FILE *f1;
FILE *f2;
FILE *f3;
printf("input the base file name:  \n");
scanf("%25s",&file1);
printf("input the secondary file name: \n");
scanf("%25s",&file2);
printf("input the output file name:  \n");
scanf("%25s",&file3);
f1=fopen(file1,"r");
f2=fopen(file2,"r");
f3=fopen(file3,"w");
flag=0;
printf("enter the value of Z :\n");
scanf("%f",&az);
printf("%f\n",az);
/* comparing two file and eliminating the air toolpath */
```

```
res_x=fgets(line,LINESIZE,f2);
while (res_x != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
        res_d=fgets(line,LINESIZE,f1);
        while (res_d != NULL && flag == 0)
        {
        sscanf(line,"%f %f %f",&x1,&y1,&z1);

        if(x==x1 && y==y1 && az==z1)
                {
                flag = 1;
                }
              else
                {
                flag=0;
                }
       res_d=fgets(line,LINESIZE,f1);
       }
       if ( (flag == 0) && (z==az)  )
          fprintf(f3,"%f %f %f\n",x,y,z);
       if (flag == 1)
          printf("%f %f %f\n",x,y,z)
       flag=0;
       rewind(f1);
res_x=fgets(line,LINESIZE,f2);
       }
}
****TO INCORPORATE TOOL LIFT AND CLUBBING*********
#include <stdio.h>
#define LINESIZE 80
main()
{
char line[LINESIZE];
char *res;
char file_1[25],file_2[25],file_3[25],file_4[25],file_5[25];
float min_x,max_x,min_y,max_y;
float x,y,z;
int l_no;
FILE *f1;
FILE *f2;
FILE *f3;
FILE *f4;
FILE *f5;
printf("\n Enter the minimum X value = ");
scanf("%f",&min_x);
printf("\n %f",min_x);
printf("\n Enter the maximum X value = ");
```

```
scanf("%f",&max_x);
printf("\n %f",max_x);
printf("\n Enter the minimum Y value = ");
scanf("%f",&min_y);
printf("\n %f",min_y);
printf("\n Enter the maximum Y value = ");
scanf("%f",&max_y);
printf("\n %f",max_y);
printf("\n    input the 1 st file name \n");
printf(" i.e. renamed output of nc_r.c \n");
scanf("%25s",&file_1);
printf("\n    input the 2 nd file name \n");
printf(" i.e. output of club.c  \n");
scanf("%25s",&file_2);
printf("\n    input the 3 rd file name \n");
printf(" i.e. output of nc_l.c \n");
scanf("%25s",&file_3);
printf("\n    input the 4 th file name \n");
printf(" i.e. output of nc_t.c \n");
scanf("%25s",&file_4);
printf("\n    output file name \n");
scanf("%25s",&file_5);
f1=fopen(file_1,"r");
f2=fopen(file_2,"r");
f3=fopen(file_3,"r");
f4=fopen(file_4,"r");
f5=fopen(file_5,"w");
res=fgets(line,LINESIZE,f1);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
fprintf(f5,"%f %f %f\n",x,y,z);
res=fgets(line,LINESIZE,f1);
}
l_no=1;
res=fgets(line,LINESIZE,f2);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (x >= min_x && x <= max_x && y >= min_y && y <= max_y)
  {
      if (l_no == 1)
        {
          fprintf(f5,"%f %f %f\n",x,y,z+1.0);
          l_no=0;
        }
   fprintf(f5,"%f %f %f\n",x,y,z);
   }
```

```
res=fgets(line,LINESIZE,f2);
}
fprintf(f5,"%f %f %f\n",x,y,z+1.0);
l_no=1;
res=fgets(line,LINESIZE,f3);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (x >= min_x && x <= max_x && y >= min_y && y <= max_y)
  {
       if (l_no == 1)
        {
          fprintf(f5,"%f %f %f\n",x,y,z+1.0);
          l_no=0;
        }
  fprintf(f5,"%f %f %f\n",x,y,z);
  }
res=fgets(line,LINESIZE,f3);
}
fprintf(f5,"%f %f %f\n",x,y,z+1.0);
l_no=1;
res=fgets(line,LINESIZE,f4);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (x >= min_x && x <= max_x && y >= min_y && y <= max_y)
  {
          if (l_no == 1)
         {
          fprintf(f5,"%f %f %f\n",x,y,z+1.0);
          l_no=0;
         }
  fprintf(f5,"%f %f %f\n",x,y,z);
  }
res=fgets(line,LINESIZE,f4);
}
fprintf(f5,"%f %f %f\n",x,y,z+1.0);
close(f1);
close(f2);
close(f3);
close(f4);
close(f5);
****TO REMOVE REDUNDANT DATA**********
}
#include <stdio.h>
#define LINESIZE 80
main()
{
```

```
char line[LINESIZE];
char line1[LINESIZE];
char *res;
char *res1;
char file_1[25],file_2[25];
float x,y,z;
float x1,y1,z1;
float ox,oy,oz;
int flag=0;
int found=0;
FILE *f1;
FILE *f2;
FILE *f3;
/* program to reduce the intermediate tool points
   if they lie in the same straight line  */
printf("What is the input file name  :\n ");
printf(" ( old tool path file name  ) \n");
scanf("%25s",&file_1);
printf("What is the output file name  :\n ");
printf(" ( new tool path file name  ) \n");
scanf("%25s",&file_2);
f1=fopen(file_1,"r");
f3=fopen("temp.inp","w");
/*     Y _ DIRECTION   */
res=fgets(line,LINESIZE,f1);
sscanf(line,"%f %f %f",&x,&y,&z);
/*   fprintf(f3,"%f %f %f\n",x,y,z);       */
ox=x;
oy=y;
oz=z;
res=fgets(line,LINESIZE,f1);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (z != oz)
    {
     fprintf(f3,"%f %f %f\n",ox,oy,oz);
     fprintf(f3,"%f %f %f\n",x,y,z);
    }
 else
    {
      if (x != ox)
       {
       fprintf(f3,"%f %f %f\n",ox,oy,oz);
       fprintf(f3,"%f %f %f\n",x,y,z);
       }
    }
 ox=x;
```

```
oy=y;
oz=z;
res=fgets(line,LINESIZE,f1);
}
fclose(f1);
fclose(f3);
f1=fopen("temp.inp","r");
f3=fopen("temp.out","w");
/*     X _ DIRECTION   */
res=fgets(line,LINESIZE,f1);
sscanf(line,"%f %f %f",&x,&y,&z);
/*   fprintf(f3,"%f %f %f\n",x,y,z);       */
ox=x;
oy=y;
oz=z;
res=fgets(line,LINESIZE,f1);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (z != oz)
    {
     fprintf(f3,"%f %f %f\n",ox,oy,oz);
     fprintf(f3,"%f %f %f\n",x,y,z);
    }
else
    {
      if (y != oy)
       {
       fprintf(f3,"%f %f %f\n",ox,oy,oz);
       fprintf(f3,"%f %f %f\n",x,y,z);
       }
    }
ox=x;
oy=y;
oz=z;
res=fgets(line,LINESIZE,f1);
}
/* to remove duplicate lines produces from above */
fclose(f1);
fclose(f3);
f1=fopen("temp.out","r");
f2=fopen(file_2,"w");
res = fgets(line,LINESIZE,f1);
sscanf(line,"%f %f %f",&x,&y,&z);
fprintf(f2,"%f %f %f\n",x,y,z);
ox=x;
oy=y;
oz=z;
```

```
res = fgets(line,LINESIZE,f1);
while (res != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
 if (x == ox && y == oy &&  z == oz)
        {
         printf("%f %f %f\n",x,y,z);
        }
  else
        {
         fprintf(f2,"%f %f %f\n",x,y,z);
        }
ox=x;
oy=y;
oz=z;
res = fgets(line,LINESIZE,f1);
}
fclose(f1);
fclose(f2);
}
***************FOR REMOVING INTERNAL FEATURE ************
#include <stdio.h>
#include <math.h>
main()
{
float endx,endy;
int i,j;
float z;
float x,y;
float p1x,p1y,p2x,p2y,p3x,p3y,p4x,p4y;
float sp1x,sp1y,sp2x,sp2y,sp3x,sp3y,sp4x,sp4y;
FILE *out;
out=fopen("po.out","w");
/* pocket size x & y  */
sp1x=1.34;
sp1y=4.61;
sp2x=1.6;
sp2y=3.65;
sp3x=2.57;
sp3y=3.91;
sp4x=2.31;
sp4y=4.875;
/* offset points x & y  */
p1x=1.4626;
p1y=4.5398;
p2x=1.6704;
p2y=3.7724;
p3x=2.4473;
```

```
p3y=3.9806;
p4x=2.2384;
p4y=4.7518;
z=4.0;
fprintf(out,"%f %f %f\n",p1x,p1y,z);
z=2.0;
fprintf(out,"%f %f %f\n",p1x,p1y,z);
fprintf(out,"%f %f %f\n",p2x,p2y,z);
fprintf(out,"%f %f %f\n",p3x,p3y,z);
fprintf(out,"%f %f %f\n",p4x,p4y,z);
fprintf(out,"%f %f %f\n",p1x,p1y,z);
for (j=3772; j<=4751; j=j+100)
 {
for (i=146; i<= 244; i=i+10)
    {
    x=i/100.0;
    y=j/1000.0;
if (x > sp2x && x < sp4x && y > sp3y && y < sp1y)
            {
              fprintf(out,"%f %f %f\n",x,y,z);
              endx=x;
              endy=y;
            }
    }
 }
z=4.0;
fprintf(out,"%f %f %f\n",endx,endy,z);


**********TO GENERATE NC CODE*********
}
#include <stdio.h>
#define LINESIZE 80
main()
{
char line[LINESIZE];
char *result;
char cl_file[25],g_code[25];
int unit_mode;
int ordinate_mode;
float feed,speed;
float x,y,z;
float ox,oy,oz;
float z_level;
int block_no,incr;
FILE *file1;
FILE *file2;
/*****************************************************/
```

```
/*              POST PROCESSOR FOR                    */
/*     DEVELOPING GCODES FROM A TOOL MOTION FILE     */
/*****************************************************/
/*  Input: File contaning x,y,z values of all tool  */
/*          motion points                           */
/*         The value of Z coordinate at which actaul*/
/*          machining is done                       */
/*                                                  */
/*                                                  */
/*  Output: Gcode file for linear momvent only      */
printf(" What is the CL Filename   \n");
scanf("%25s",&cl_file);
printf(" What is the GCODE filename   \n");
scanf("%25s",&g_code);
printf("Are you machining in Inch/mm mode  :\n");
printf("     Press  1  for  Inch        \n");
printf("     Press  2  for  MM          \n");
printf("     Enter selection :          ");
scanf("%1d",&unit_mode);
printf("\n Are you machining in Incremental/absolute mode  :\n");
printf("     Press  1  for  Incremental        \n");
printf("     Press  2  for  Absolute           \n");
printf("     Enter selection :          ");
scanf("%1d",&ordinate_mode);
printf("\n What is the feed rate    \n");
if (unit_mode == 1)
  {
      printf("\n Feed in inches/min  = ");
  }
 if (unit_mode == 2)
     {
       printf("\n Feed in millimeters/min  = ");
     }
 scanf("%f",&feed);
 printf("\n What is the Spindle rpm  :  ");
 scanf("%f",&speed);
 printf("\n What is the z_level \n");
 scanf("%f",&z_level);
 file1=fopen(cl_file,"r");
 file2=fopen(g_code,"w");
 printf("%%\n");
 if (unit_mode == 1)
   {
    fprintf(file2," N0 G70\n");
   }
 if (unit_mode == 2)
   {
    fprintf(file2,"N0 G71\n");
```

```
  }
fprintf(file2," N10 G94 F%f\n",feed);
fprintf(file2," N20 G96 F%f\n",speed);
block_no=30;
incr=10;
result=fgets(line,LINESIZE,file1);
sscanf(line,"%f %f %f",&x,&y,&z);
ox=x;
oy=y;
oz=z;
fprintf(file2," N%d G00 %f %f %f   \n ",block_no,x,y,z);
block_no=block_no+incr;
result=fgets(line,LINESIZE,file1);
while (result != NULL)
{
sscanf(line,"%f %f %f",&x,&y,&z);
if (z != z_level)
   {
 fprintf(file2,"N%d G00 %f %f %f   \n ",block_no,x,y,z);
       block_no=block_no+incr;
   }
else
   {
 fprintf(file2,"N%d G01 %f %f %f   \n ",block_no,x,y,z);
       block_no=block_no+incr;
   }
ox=x;
oy=y;
oz=z;
result=fgets(line,LINESIZE,file1);
}
}
```

APPENDIX 3

(STEP file for an example part)

```
NOTE: The hole feature from this file is not used in this research.
HEADER;
/*----------------------------------------
 * Exchange File generated by ST-DEVELOPER v1.3
 * Conforms to ISO 10303-21
 */
FILE_DESCRIPTION (('1'), '1');
FILE_NAME ('shafee_solid', '1994-12-14T16:24:17-06:00', (''), (''),
 'ST-DEVELOPER v1.3', 'EDS - UNIGRAPHICS 10.3', '');
FILE_SCHEMA (('CONFIG_CONTROL_DESIGN'));
ENDSEC;
DATA;
#10 = DIMENSIONAL_EXPONENTS (1., 0., 0., 0., 0., 0., 0.);
#20 = DIMENSIONAL_EXPONENTS (0., 0., 0., 0., 0., 0., 0.);
#30 = SI_UNIT (*, .MILLI., .METRE.);
#40 = LENGTH_MEASURE_WITH_UNIT ( LENGTH_MEASURE(25.4), #30);
#50 = (
  CONVERSION_BASED_UNIT ('INCH', #40)
  LENGTH_UNIT ()
  NAMED_UNIT (#10) );
#60 = SI_UNIT (*, $, .RADIAN.);
#70 = PLANE_ANGLE_MEASURE_WITH_UNIT (
PLANE_ANGLE_MEASURE(0.0174532925), #60);
#80 = (
  CONVERSION_BASED_UNIT ('DEGREE', #70)
  NAMED_UNIT (#20)
  PLANE_ANGLE_UNIT () );
#90 = (
  NAMED_UNIT (#20)
  SI_UNIT ($, .STERADIAN.)
  SOLID_ANGLE_UNIT () );
#100 = (
  GEOMETRIC_REPRESENTATION_CONTEXT (3)
  GLOBAL_UNIT_ASSIGNED_CONTEXT ((#50, #80, #90))
  REPRESENTATION_CONTEXT ('A', 'B') );
#110 = CARTESIAN_POINT ((0.75, 1.5, 0.5));
#120 = VERTEX_POINT (#110);
#130 = CARTESIAN_POINT ((0.75, 1.5, 2.));
#140 = VERTEX_POINT (#130);
#150 = CARTESIAN_POINT ((0.987308004315134, 3.53449314192417, 1.));
#160 = VERTEX_POINT (#150);
#170 = CARTESIAN_POINT ((0.280201223128585, 2.82738636073763, 1.));
#180 = VERTEX_POINT (#170);
#190 = CARTESIAN_POINT ((0.280201223128585, 2.82738636073763, 2.));
#200 = VERTEX_POINT (#190);
#210 = CARTESIAN_POINT ((0.987308004315134, 3.53449314192417, 2.));
#220 = VERTEX_POINT (#210);
```

#230 = CARTESIAN_POINT ((0.280201223128585, 2.47383297014435, 1.));
#240 = VERTEX_POINT (#230);
#250 = CARTESIAN_POINT ((1.51763809020504, 1.23639610306789, 1.));
#260 = VERTEX_POINT (#250);
#270 = CARTESIAN_POINT ((1.51763809020504, 1.23639610306789, 2.));
#280 = VERTEX_POINT (#270);
#290 = CARTESIAN_POINT ((0.280201223128585, 2.47383297014435, 2.));
#300 = VERTEX_POINT (#290);
#310 = CARTESIAN_POINT ((2.57829826198486, 1.94350288425444, 1.));
#320 = VERTEX_POINT (#310);
#330 = CARTESIAN_POINT ((1.87119148079832, 1.23639610306789, 1.));
#340 = VERTEX_POINT (#330);
#350 = CARTESIAN_POINT ((2.57829826198486, 1.94350288425444, 2.));
#360 = VERTEX_POINT (#350);
#370 = CARTESIAN_POINT ((1.87119148079832, 1.23639610306789, 2.));
#380 = VERTEX_POINT (#370);
#390 = CARTESIAN_POINT ((2.57829826198486, 2.29705627484771, 1.));
#400 = VERTEX_POINT (#390);
#410 = CARTESIAN_POINT ((1.34086139490841, 3.53449314192417, 1.));
#420 = VERTEX_POINT (#410);
#430 = CARTESIAN_POINT ((1.34086139490841, 3.53449314192417, 2.));
#440 = VERTEX_POINT (#430);
#450 = CARTESIAN_POINT ((2.57829826198486, 2.29705627484771, 2.));
#460 = VERTEX_POINT (#450);
#470 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 0.75));
#480 = VERTEX_POINT (#470);
#490 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 0.75));
#500 = VERTEX_POINT (#490);
#510 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 0.75));
#520 = VERTEX_POINT (#510);
#530 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 0.75));
#540 = VERTEX_POINT (#530);
#550 = CARTESIAN_POINT ((2.625, 2.5, 2.));
#560 = VERTEX_POINT (#550);
#570 = CARTESIAN_POINT ((2.625, 2.5, 1.75));
#580 = VERTEX_POINT (#570);
#590 = CARTESIAN_POINT ((2.625, 3.5, 1.75));
#600 = VERTEX_POINT (#590);
#610 = CARTESIAN_POINT ((2.625, 3.5, 2.));
#620 = VERTEX_POINT (#610);
#630 = CARTESIAN_POINT ((2.5, 2.625, 1.75));
#640 = VERTEX_POINT (#630);
#650 = CARTESIAN_POINT ((2.5, 3.375, 1.75));
#660 = VERTEX_POINT (#650);
#670 = CARTESIAN_POINT ((2.5, 2.625, 2.));
#680 = VERTEX_POINT (#670);
#690 = CARTESIAN_POINT ((2.5, 3.375, 2.));
#700 = VERTEX_POINT (#690);

#710 = CARTESIAN_POINT ((3., 2.5, 2.));
#720 = VERTEX_POINT (#710);
#730 = CARTESIAN_POINT ((3., 2.5, 1.75));
#740 = VERTEX_POINT (#730);
#750 = CARTESIAN_POINT ((3., 3.5, 2.));
#760 = VERTEX_POINT (#750);
#770 = CARTESIAN_POINT ((3., 3.5, 1.75));
#780 = VERTEX_POINT (#770);
#790 = CARTESIAN_POINT ((0.625, 4.625, 2.));
#800 = VERTEX_POINT (#790);
#810 = CARTESIAN_POINT ((1., 4.375, 2.1875));
#820 = VERTEX_POINT (#810);
#830 = CARTESIAN_POINT ((0.625, 4.375, 2.1875));
#840 = VERTEX_POINT (#830);
#850 = CARTESIAN_POINT ((0.625, 4.625, 2.1875));
#860 = VERTEX_POINT (#850);
#870 = CARTESIAN_POINT ((1., 4.625, 2.1875));
#880 = VERTEX_POINT (#870);
#890 = CARTESIAN_POINT ((1., 4.375, 2.));
#900 = VERTEX_POINT (#890);
#910 = CARTESIAN_POINT ((0.625, 4.375, 2.));
#920 = VERTEX_POINT (#910);
#930 = CARTESIAN_POINT ((1., 4.625, 2.));
#940 = VERTEX_POINT (#930);
#950 = CARTESIAN_POINT ((2., 1., 0.));
#960 = VERTEX_POINT (#950);
#970 = CARTESIAN_POINT ((2., 1., 2.));
#980 = VERTEX_POINT (#970);
#990 = CARTESIAN_POINT ((2., 1.25, 2.));
#1000 = VERTEX_POINT (#990);
#1010 = CARTESIAN_POINT ((2., 1.25, 0.));
#1020 = VERTEX_POINT (#1010);
#1030 = CARTESIAN_POINT ((2.5, 1., 2.));
#1040 = VERTEX_POINT (#1030);
#1050 = CARTESIAN_POINT ((2.5, 1.25, 2.));
#1060 = VERTEX_POINT (#1050);
#1070 = CARTESIAN_POINT ((2.5, 1.25, 0.));
#1080 = VERTEX_POINT (#1070);
#1090 = CARTESIAN_POINT ((2.5, 1., 0.));
#1100 = VERTEX_POINT (#1090);
#1110 = CARTESIAN_POINT ((1.6, 3.65, 2.));
#1120 = VERTEX_POINT (#1110);
#1130 = CARTESIAN_POINT ((1.6, 3.65, 0.));
#1140 = VERTEX_POINT (#1130);
#1150 = CARTESIAN_POINT ((1.34118095489748, 4.61592582628907, 2.));
#1160 = VERTEX_POINT (#1150);
#1170 = CARTESIAN_POINT ((1.34118095489748, 4.61592582628907, 0.));
#1180 = VERTEX_POINT (#1170);

```
#1190 = CARTESIAN_POINT ((2.56592582628907, 3.90881904510252, 0.));
#1200 = VERTEX_POINT (#1190);
#1210 = CARTESIAN_POINT ((2.30710678118655, 4.87474487139159, 2.));
#1220 = VERTEX_POINT (#1210);
#1230 = CARTESIAN_POINT ((2.30710678118655, 4.87474487139159, 0.));
#1240 = VERTEX_POINT (#1230);
#1250 = CARTESIAN_POINT ((2.56592582628907, 3.90881904510252, 2.));
#1260 = VERTEX_POINT (#1250);
#1270 = CARTESIAN_POINT ((0., 1., 2.));
#1280 = VERTEX_POINT (#1270);
#1290 = CARTESIAN_POINT ((0., 5., 2.));
#1300 = VERTEX_POINT (#1290);
#1310 = CARTESIAN_POINT ((3., 5., 2.));
#1320 = VERTEX_POINT (#1310);
#1330 = CARTESIAN_POINT ((3., 1., 2.));
#1340 = VERTEX_POINT (#1330);
#1350 = CARTESIAN_POINT ((3., 5., 0.));
#1360 = VERTEX_POINT (#1350);
#1370 = CARTESIAN_POINT ((0., 5., 0.));
#1380 = VERTEX_POINT (#1370);
#1390 = CARTESIAN_POINT ((0., 1., 0.));
#1400 = VERTEX_POINT (#1390);
#1410 = CARTESIAN_POINT ((3., 1., 0.));
#1420 = VERTEX_POINT (#1410);
#1430 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 1.));
#1440 = DIRECTION ((0., 0., 1.));
#1450 = DIRECTION ((1., 0., 0.));
#1460 = AXIS2_PLACEMENT_3D (#1430, #1440, #1450);
#1470 = CIRCLE (#1460, 0.25);
#1480 = EDGE_CURVE (#260, #340, #1470, .T.);
#1490 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 1.));
#1500 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#1510 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#1520 = AXIS2_PLACEMENT_3D (#1490, #1500, #1510);
#1530 = CIRCLE (#1520, 0.25);
#1540 = EDGE_CURVE (#500, #260, #1530, .T.);
#1550 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 1.));
#1560 = DIRECTION ((0.707106781186547, -0.707106781186548,
 -2.65609883204994E-16));
#1570 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));
#1580 = AXIS2_PLACEMENT_3D (#1550, #1560, #1570);
#1590 = CIRCLE (#1580, 0.25);
#1600 = EDGE_CURVE (#240, #520, #1590, .T.);
#1610 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 1.));
#1620 = DIRECTION ((0., 0., 1.));
#1630 = DIRECTION ((1., 0., 0.));
#1640 = AXIS2_PLACEMENT_3D (#1610, #1620, #1630);
#1650 = CIRCLE (#1640, 0.25);
```

#1660 = EDGE_CURVE (#180, #240, #1650, .T.);
#1670 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 1.));
#1680 = DIRECTION ((-0.707106781186548, -0.707106781186547,
9.65854120745432E-17));
#1690 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#1700 = AXIS2_PLACEMENT_3D (#1670, #1680, #1690);
#1710 = CIRCLE (#1700, 0.25);
#1720 = EDGE_CURVE (#500, #340, #1710, .T.);
#1730 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 1.));
#1740 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#1750 = DIRECTION ((-0.707106781186548, 0.707106781186547, 0.));
#1760 = AXIS2_PLACEMENT_3D (#1730, #1740, #1750);
#1770 = CIRCLE (#1760, 0.25);
#1780 = EDGE_CURVE (#180, #520, #1770, .T.);
#1790 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 1.));
#1800 = DIRECTION ((0., 0., 1.));
#1810 = DIRECTION ((1., 0., 0.));
#1820 = AXIS2_PLACEMENT_3D (#1790, #1800, #1810);
#1830 = CIRCLE (#1820, 0.25);
#1840 = EDGE_CURVE (#320, #400, #1830, .T.);
#1850 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 1.));
#1860 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#1870 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#1880 = AXIS2_PLACEMENT_3D (#1850, #1860, #1870);
#1890 = CIRCLE (#1880, 0.25);
#1900 = EDGE_CURVE (#480, #320, #1890, .T.);
#1910 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 1.));
#1920 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#1930 = DIRECTION ((0.707106781186548, -0.707106781186548, 0.));
#1940 = AXIS2_PLACEMENT_3D (#1910, #1920, #1930);
#1950 = CIRCLE (#1940, 0.25);
#1960 = EDGE_CURVE (#160, #540, #1950, .T.);
#1970 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 1.));
#1980 = DIRECTION ((0., 0., 1.));
#1990 = DIRECTION ((1., 0., 0.));
#2000 = AXIS2_PLACEMENT_3D (#1970, #1980, #1990);
#2010 = CIRCLE (#2000, 0.25);
#2020 = EDGE_CURVE (#420, #160, #2010, .T.);
#2030 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 1.));
#2040 = DIRECTION ((0.707106781186547, -0.707106781186548,
9.65854120745432E-17));
#2050 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#2060 = AXIS2_PLACEMENT_3D (#2030, #2040, #2050);
#2070 = CIRCLE (#2060, 0.25);
#2080 = EDGE_CURVE (#480, #400, #2070, .T.);
#2090 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 1.));
#2100 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#2110 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));

#2120 = AXIS2_PLACEMENT_3D (#2090, #2100, #2110);
#2130 = CIRCLE (#2120, 0.25);
#2140 = EDGE_CURVE (#420, #540, #2130, .T.);
#2150 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 2.));
#2160 = DIRECTION ((0., 0., -1.));
#2170 = DIRECTION ((1., 0., 0.));
#2180 = AXIS2_PLACEMENT_3D (#2150, #2160, #2170);
#2190 = CIRCLE (#2180, 0.25);
#2200 = EDGE_CURVE (#380, #280, #2190, .T.);
#2210 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 2.));
#2220 = DIRECTION ((0., 0., -1.));
#2230 = DIRECTION ((1., 0., 0.));
#2240 = AXIS2_PLACEMENT_3D (#2210, #2220, #2230);
#2250 = CIRCLE (#2240, 0.25);
#2260 = EDGE_CURVE (#300, #200, #2250, .T.);
#2270 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 2.));
#2280 = DIRECTION ((0., 0., -1.));
#2290 = DIRECTION ((1., 0., 0.));
#2300 = AXIS2_PLACEMENT_3D (#2270, #2280, #2290);
#2310 = CIRCLE (#2300, 0.25);
#2320 = EDGE_CURVE (#220, #440, #2310, .T.);
#2330 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 2.));
#2340 = DIRECTION ((0., 0., -1.));
#2350 = DIRECTION ((1., 0., 0.));
#2360 = AXIS2_PLACEMENT_3D (#2330, #2340, #2350);
#2370 = CIRCLE (#2360, 0.25);
#2380 = EDGE_CURVE (#460, #360, #2370, .T.);
#2390 = CARTESIAN_POINT ((2.625, 2.625, 2.));
#2400 = DIRECTION ((0., 0., -1.));
#2410 = DIRECTION ((1., 0., 0.));
#2420 = AXIS2_PLACEMENT_3D (#2390, #2400, #2410);
#2430 = CIRCLE (#2420, 0.125);
#2440 = EDGE_CURVE (#560, #680, #2430, .T.);
#2450 = CARTESIAN_POINT ((2.625, 3.375, 2.));
#2460 = DIRECTION ((0., 0., -1.));
#2470 = DIRECTION ((1., 0., 0.));
#2480 = AXIS2_PLACEMENT_3D (#2450, #2460, #2470);
#2490 = CIRCLE (#2480, 0.125);
#2500 = EDGE_CURVE (#700, #620, #2490, .T.);
#2510 = CARTESIAN_POINT ((0.987308004315134, 3.53449314192417, 0.75));
#2520 = DIRECTION ((0., 0., 1.));
#2530 = VECTOR (#2520, 1.);
#2540 = LINE (#2510, #2530);
#2550 = EDGE_CURVE (#160, #220, #2540, .T.);
#2560 = CARTESIAN_POINT ((0.633754613721859, 3.1809397513309, 1.));
#2570 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));
#2580 = VECTOR (#2570, 1.);
#2590 = LINE (#2560, #2580);

#2600 = EDGE_CURVE (#180, #160, #2590, .T.);
#2610 = CARTESIAN_POINT ((0.280201223128585, 2.82738636073763, 0.75));
#2620 = DIRECTION ((0., 0., -1.));
#2630 = VECTOR (#2620, 1.);
#2640 = LINE (#2610, #2630);
#2650 = EDGE_CURVE (#200, #180, #2640, .T.);
#2660 = CARTESIAN_POINT ((0.976407431195481, 3.52359256880452, 2.));
#2670 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#2680 = VECTOR (#2670, 1.);
#2690 = LINE (#2660, #2680);
#2700 = EDGE_CURVE (#220, #200, #2690, .T.);
#2710 = CARTESIAN_POINT ((0.280201223128585, 2.47383297014435, 0.75));
#2720 = DIRECTION ((0., 0., 1.));
#2730 = VECTOR (#2720, 1.);
#2740 = LINE (#2710, #2730);
#2750 = EDGE_CURVE (#240, #300, #2740, .T.);
#2760 = CARTESIAN_POINT ((0.898919656666814, 1.85511453660612, 1.));
#2770 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#2780 = VECTOR (#2770, 1.);
#2790 = LINE (#2760, #2780);
#2800 = EDGE_CURVE (#260, #240, #2790, .T.);
#2810 = CARTESIAN_POINT ((1.51763809020504, 1.23639610306789, 0.75));
#2820 = DIRECTION ((0., 0., -1.));
#2830 = VECTOR (#2820, 1.);
#2840 = LINE (#2810, #2830);
#2850 = EDGE_CURVE (#280, #260, #2840, .T.);
#2860 = CARTESIAN_POINT ((0.627017096636468, 2.12701709663647, 2.));
#2870 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#2880 = VECTOR (#2870, 1.);
#2890 = LINE (#2860, #2880);
#2900 = EDGE_CURVE (#300, #280, #2890, .T.);
#2910 = CARTESIAN_POINT ((2.57829826198486, 1.94350288425444, 0.75));
#2920 = DIRECTION ((0., 0., -1.));
#2930 = VECTOR (#2920, 1.);
#2940 = LINE (#2910, #2930);
#2950 = EDGE_CURVE (#360, #320, #2940, .T.);
#2960 = CARTESIAN_POINT ((2.22474487139159, 1.58994949366117, 1.));
#2970 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#2980 = VECTOR (#2970, 1.);
#2990 = LINE (#2960, #2980);
#3000 = EDGE_CURVE (#320, #340, #2990, .T.);
#3010 = CARTESIAN_POINT ((1.87119148079832, 1.23639610306789, 0.75));
#3020 = DIRECTION ((0., 0., 1.));
#3030 = VECTOR (#3020, 1.);
#3040 = LINE (#3010, #3030);
#3050 = EDGE_CURVE (#340, #380, #3040, .T.);
#3060 = CARTESIAN_POINT ((2.56739768886521, 1.93260231113479, 2.));
#3070 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));

#3080 = VECTOR (#3070, 1.);
#3090 = LINE (#3060, #3080);
#3100 = EDGE_CURVE (#380, #360, #3090, .T.);
#3110 = CARTESIAN_POINT ((2.57829826198486, 2.29705627484771, 0.75));
#3120 = DIRECTION ((0., 0., 1.));
#3130 = VECTOR (#3120, 1.);
#3140 = LINE (#3110, #3130);
#3150 = EDGE_CURVE (#400, #460, #3140, .T.);
#3160 = CARTESIAN_POINT ((1.95957982844664, 2.91577470838594, 1.));
#3170 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#3180 = VECTOR (#3170, 1.);
#3190 = LINE (#3160, #3180);
#3200 = EDGE_CURVE (#420, #400, #3190, .T.);
#3210 = CARTESIAN_POINT ((1.34086139490841, 3.53449314192417, 0.75));
#3220 = DIRECTION ((0., 0., -1.));
#3230 = VECTOR (#3220, 1.);
#3240 = LINE (#3210, #3230);
#3250 = EDGE_CURVE (#440, #420, #3240, .T.);
#3260 = CARTESIAN_POINT ((1.68767726841629, 3.18767726841629, 2.));
#3270 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#3280 = VECTOR (#3270, 1.);
#3290 = LINE (#3260, #3280);
#3300 = EDGE_CURVE (#460, #440, #3290, .T.);
#3310 = CARTESIAN_POINT ((1.78280313315, 2.73899801308931, 0.75));
#3320 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#3330 = VECTOR (#3320, 1.);
#3340 = LINE (#3310, #3330);
#3350 = EDGE_CURVE (#480, #540, #3340, .T.);
#3360 = CARTESIAN_POINT ((2.04796817609495, 1.7667261889578, 0.75));
#3370 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));
#3380 = VECTOR (#3370, 1.);
#3390 = LINE (#3360, #3380);
#3400 = EDGE_CURVE (#500, #480, #3390, .T.);
#3410 = CARTESIAN_POINT ((1.07569635196345, 2.03189123190276, 0.75));
#3420 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#3430 = VECTOR (#3420, 1.);
#3440 = LINE (#3410, #3430);
#3450 = EDGE_CURVE (#520, #500, #3440, .T.);
#3460 = CARTESIAN_POINT ((0.810531309018496, 3.00416305603426, 0.75));
#3470 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#3480 = VECTOR (#3470, 1.);
#3490 = LINE (#3460, #3480);
#3500 = EDGE_CURVE (#540, #520, #3490, .T.);
#3510 = CARTESIAN_POINT ((2.625, 2.5, 1.75));
#3520 = DIRECTION ((0., 0., 1.));
#3530 = VECTOR (#3520, 1.);
#3540 = LINE (#3510, #3530);
#3550 = EDGE_CURVE (#580, #560, #3540, .T.);

#3560 = CARTESIAN_POINT ((1.5, 2.5, 2.));
#3570 = DIRECTION ((1., -1.42125554512318E-15, 0.));
#3580 = VECTOR (#3570, 1.);
#3590 = LINE (#3560, #3580);
#3600 = EDGE_CURVE (#560, #720, #3590, .T.);
#3610 = CARTESIAN_POINT ((2.625, 3.375, 1.75));
#3620 = DIRECTION ((0., 0., 1.));
#3630 = DIRECTION ((-1., 0., 0.));
#3640 = AXIS2_PLACEMENT_3D (#3610, #3620, #3630);
#3650 = CIRCLE (#3640, 0.125);
#3660 = EDGE_CURVE (#600, #660, #3650, .T.);
#3670 = CARTESIAN_POINT ((2.625, 2.625, 1.75));
#3680 = DIRECTION ((0., 0., 1.));
#3690 = DIRECTION ((-1., 0., 0.));
#3700 = AXIS2_PLACEMENT_3D (#3670, #3680, #3690);
#3710 = CIRCLE (#3700, 0.125);
#3720 = EDGE_CURVE (#640, #580, #3710, .T.);
#3730 = CARTESIAN_POINT ((3.25, 2.5, 1.75));
#3740 = DIRECTION ((1., -1.42125554512318E-15, 0.));
#3750 = VECTOR (#3740, 1.);
#3760 = LINE (#3730, #3750);
#3770 = EDGE_CURVE (#580, #740, #3760, .T.);
#3780 = CARTESIAN_POINT ((2.625, 3.5, 1.75));
#3790 = DIRECTION ((0., 0., -1.));
#3800 = VECTOR (#3790, 1.);
#3810 = LINE (#3780, #3800);
#3820 = EDGE_CURVE (#620, #600, #3810, .T.);
#3830 = CARTESIAN_POINT ((2.5, 3.375, 1.75));
#3840 = DIRECTION ((0., 0., 1.));
#3850 = VECTOR (#3840, 1.);
#3860 = LINE (#3830, #3850);
#3870 = EDGE_CURVE (#660, #700, #3860, .T.);
#3880 = CARTESIAN_POINT ((2.5, 2.625, 1.75));
#3890 = DIRECTION ((0., 0., -1.));
#3900 = VECTOR (#3890, 1.);
#3910 = LINE (#3880, #3900);
#3920 = EDGE_CURVE (#680, #640, #3910, .T.);
#3930 = CARTESIAN_POINT ((2.5, 3., 1.75));
#3940 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#3950 = VECTOR (#3940, 1.);
#3960 = LINE (#3930, #3950);
#3970 = EDGE_CURVE (#660, #640, #3960, .T.);
#3980 = CARTESIAN_POINT ((2.5, 3., 2.));
#3990 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#4000 = VECTOR (#3990, 1.);
#4010 = LINE (#3980, #4000);
#4020 = EDGE_CURVE (#700, #680, #4010, .T.);
#4030 = CARTESIAN_POINT ((3., 2.5, 0.));

#4040 = DIRECTION ((0., 0., -1.));
#4050 = VECTOR (#4040, 1.);
#4060 = LINE (#4030, #4050);
#4070 = EDGE_CURVE (#720, #740, #4060, .T.);
#4080 = CARTESIAN_POINT ((1.5, 3.5, 2.));
#4090 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#4100 = VECTOR (#4090, 1.);
#4110 = LINE (#4080, #4100);
#4120 = EDGE_CURVE (#760, #620, #4110, .T.);
#4130 = CARTESIAN_POINT ((3., 3.5, 0.));
#4140 = DIRECTION ((0., 0., 1.));
#4150 = VECTOR (#4140, 1.);
#4160 = LINE (#4130, #4150);
#4170 = EDGE_CURVE (#780, #760, #4160, .T.);
#4180 = CARTESIAN_POINT ((3.25, 3.5, 1.75));
#4190 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#4200 = VECTOR (#4190, 1.);
#4210 = LINE (#4180, #4200);
#4220 = EDGE_CURVE (#780, #600, #4210, .T.);
#4230 = CARTESIAN_POINT ((3., 3., 1.75));
#4240 = DIRECTION ((0., 1., 0.));
#4250 = VECTOR (#4240, 1.);
#4260 = LINE (#4230, #4250);
#4270 = EDGE_CURVE (#740, #780, #4260, .T.);
#4280 = CARTESIAN_POINT ((3., 3., 2.));
#4290 = DIRECTION ((0., -1., 0.));
#4300 = VECTOR (#4290, 1.);
#4310 = LINE (#4280, #4300);
#4320 = EDGE_CURVE (#720, #1340, #4310, .T.);
#4330 = CARTESIAN_POINT ((1., 4.375, 1.9375));
#4340 = DIRECTION ((0., 0., 1.));
#4350 = VECTOR (#4340, 1.);
#4360 = LINE (#4330, #4350);
#4370 = EDGE_CURVE (#900, #820, #4360, .T.);
#4380 = CARTESIAN_POINT ((0.624999999999998, 3., 2.));
#4390 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#4400 = VECTOR (#4390, 1.);
#4410 = LINE (#4380, #4400);
#4420 = EDGE_CURVE (#800, #920, #4410, .T.);
#4430 = CARTESIAN_POINT ((0.625, 4.375, 1.9375));
#4440 = DIRECTION ((0., 0., 1.));
#4450 = VECTOR (#4440, 1.);
#4460 = LINE (#4430, #4450);
#4470 = EDGE_CURVE (#920, #840, #4460, .T.);
#4480 = CARTESIAN_POINT ((1., 4.625, 1.9375));
#4490 = DIRECTION ((0., 0., 1.));
#4500 = VECTOR (#4490, 1.);
#4510 = LINE (#4480, #4500);

#4520 = EDGE_CURVE (#940, #880, #4510, .T.);
#4530 = CARTESIAN_POINT ((0.625, 4.625, 1.9375));
#4540 = DIRECTION ((0., 0., 1.));
#4550 = VECTOR (#4540, 1.);
#4560 = LINE (#4530, #4550);
#4570 = EDGE_CURVE (#800, #860, #4560, .T.);
#4580 = CARTESIAN_POINT ((1., 4.5, 2.1875));
#4590 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#4600 = VECTOR (#4590, 1.);
#4610 = LINE (#4580, #4600);
#4620 = EDGE_CURVE (#880, #820, #4610, .T.);
#4630 = CARTESIAN_POINT ((0.8125, 4.375, 2.1875));
#4640 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#4650 = VECTOR (#4640, 1.);
#4660 = LINE (#4630, #4650);
#4670 = EDGE_CURVE (#820, #840, #4660, .T.);
#4680 = CARTESIAN_POINT ((0.625, 4.5, 2.1875));
#4690 = DIRECTION ((1.42125554512318E-15, 1., 0.));
#4700 = VECTOR (#4690, 1.);
#4710 = LINE (#4680, #4700);
#4720 = EDGE_CURVE (#840, #860, #4710, .T.);
#4730 = CARTESIAN_POINT ((0.8125, 4.625, 2.1875));
#4740 = DIRECTION ((1., -1.42125554512318E-15, 0.));
#4750 = VECTOR (#4740, 1.);
#4760 = LINE (#4730, #4750);
#4770 = EDGE_CURVE (#860, #880, #4760, .T.);
#4780 = CARTESIAN_POINT ((0.999999999999998, 3., 2.));
#4790 = DIRECTION ((1.42125554512318E-15, 1., 0.));
#4800 = VECTOR (#4790, 1.);
#4810 = LINE (#4780, #4800);
#4820 = EDGE_CURVE (#900, #940, #4810, .T.);
#4830 = CARTESIAN_POINT ((1.5, 4.375, 2.));
#4840 = DIRECTION ((1., -1.42125554512318E-15, 0.));
#4850 = VECTOR (#4840, 1.);
#4860 = LINE (#4830, #4850);
#4870 = EDGE_CURVE (#920, #900, #4860, .T.);
#4880 = CARTESIAN_POINT ((1.5, 4.625, 2.));
#4890 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#4900 = VECTOR (#4890, 1.);
#4910 = LINE (#4880, #4900);
#4920 = EDGE_CURVE (#940, #800, #4910, .T.);
#4930 = CARTESIAN_POINT ((1.5, 1.25, 0.));
#4940 = DIRECTION ((1., 0., 0.));
#4950 = VECTOR (#4940, 1.);
#4960 = LINE (#4930, #4950);
#4970 = EDGE_CURVE (#1020, #1080, #4960, .T.);
#4980 = CARTESIAN_POINT ((1.5, 1.25, 2.));
#4990 = DIRECTION ((-1., 0., 0.));

#5000 = VECTOR (#4990, 1.);
#5010 = LINE (#4980, #5000);
#5020 = EDGE_CURVE (#1060, #1000, #5010, .T.);
#5030 = CARTESIAN_POINT ((2., 3., 0.));
#5040 = DIRECTION ((0., 1., 0.));
#5050 = VECTOR (#5040, 1.);
#5060 = LINE (#5030, #5050);
#5070 = EDGE_CURVE (#960, #1020, #5060, .T.);
#5080 = CARTESIAN_POINT ((2., 3., 2.));
#5090 = DIRECTION ((0., -1., 0.));
#5100 = VECTOR (#5090, 1.);
#5110 = LINE (#5080, #5100);
#5120 = EDGE_CURVE (#1000, #980, #5110, .T.);
#5130 = CARTESIAN_POINT ((2., 1., 0.));
#5140 = DIRECTION ((0., 0., -1.));
#5150 = VECTOR (#5140, 1.);
#5160 = LINE (#5130, #5150);
#5170 = EDGE_CURVE (#980, #960, #5160, .T.);
#5180 = CARTESIAN_POINT ((2., 1.25, -0.25));
#5190 = DIRECTION ((0., 0., 1.));
#5200 = VECTOR (#5190, 1.);
#5210 = LINE (#5180, #5200);
#5220 = EDGE_CURVE (#1020, #1000, #5210, .T.);
#5230 = CARTESIAN_POINT ((2.5, 3., 2.));
#5240 = DIRECTION ((0., 1., 0.));
#5250 = VECTOR (#5240, 1.);
#5260 = LINE (#5230, #5250);
#5270 = EDGE_CURVE (#1040, #1060, #5260, .T.);
#5280 = CARTESIAN_POINT ((2.5, 1., 0.));
#5290 = DIRECTION ((0., 0., 1.));
#5300 = VECTOR (#5290, 1.);
#5310 = LINE (#5280, #5300);
#5320 = EDGE_CURVE (#1100, #1040, #5310, .T.);
#5330 = CARTESIAN_POINT ((2.5, 1.25, -0.25));
#5340 = DIRECTION ((0., 0., 1.));
#5350 = VECTOR (#5340, 1.);
#5360 = LINE (#5330, #5350);
#5370 = EDGE_CURVE (#1080, #1060, #5360, .T.);
#5380 = CARTESIAN_POINT ((2.5, 3., 0.));
#5390 = DIRECTION ((0., -1., 0.));
#5400 = VECTOR (#5390, 1.);
#5410 = LINE (#5380, #5400);
#5420 = EDGE_CURVE (#1080, #1100, #5410, .T.);
#5430 = CARTESIAN_POINT ((1.5, 1., 0.));
#5440 = DIRECTION ((1., 0., 0.));
#5450 = VECTOR (#5440, 1.);
#5460 = LINE (#5430, #5450);
#5470 = EDGE_CURVE (#1100, #1420, #5460, .T.);

#5480 = CARTESIAN_POINT ((1.5, 1., 2.));
#5490 = DIRECTION ((-1., 0., 0.));
#5500 = VECTOR (#5490, 1.);
#5510 = LINE (#5480, #5500);
#5520 = EDGE_CURVE (#980, #1280, #5510, .T.);
#5530 = CARTESIAN_POINT ((1.34419872981078, 3.58145825622994, 0.));
#5540 = DIRECTION ((-0.965925826289068, -0.258819045102521, 0.));
#5550 = VECTOR (#5540, 1.);
#5560 = LINE (#5530, #5550);
#5570 = EDGE_CURVE (#1200, #1140, #5560, .T.);
#5580 = CARTESIAN_POINT ((1.75580127018922, 3.06854174377006, 0.));
#5590 = DIRECTION ((-0.258819045102521, 0.965925826289068, 0.));
#5600 = VECTOR (#5590, 1.);
#5610 = LINE (#5580, #5600);
#5620 = EDGE_CURVE (#1140, #1180, #5610, .T.);
#5630 = CARTESIAN_POINT ((1.75580127018922, 3.06854174377006, 2.));
#5640 = DIRECTION ((0.258819045102521, -0.965925826289068, 0.));
#5650 = VECTOR (#5640, 1.);
#5660 = LINE (#5630, #5650);
#5670 = EDGE_CURVE (#1160, #1120, #5660, .T.);
#5680 = CARTESIAN_POINT ((1.6, 3.65, -0.25));
#5690 = DIRECTION ((0., 0., 1.));
#5700 = VECTOR (#5690, 1.);
#5710 = LINE (#5680, #5700);
#5720 = EDGE_CURVE (#1140, #1120, #5710, .T.);
#5730 = CARTESIAN_POINT ((1.08537968470826, 4.54738408251901, 2.));
#5740 = DIRECTION ((-0.965925826289068, -0.258819045102521, 0.));
#5750 = VECTOR (#5740, 1.);
#5760 = LINE (#5730, #5750);
#5770 = EDGE_CURVE (#1220, #1160, #5760, .T.);
#5780 = CARTESIAN_POINT ((1.34118095489748, 4.61592582628907, -0.25));
#5790 = DIRECTION ((0., 0., 1.));
#5800 = VECTOR (#5790, 1.);
#5810 = LINE (#5780, #5800);
#5820 = EDGE_CURVE (#1180, #1160, #5810, .T.);
#5830 = CARTESIAN_POINT ((2.72172709647829, 3.32736078887258, 0.));
#5840 = DIRECTION ((0.258819045102521, -0.965925826289068, 0.));
#5850 = VECTOR (#5840, 1.);
#5860 = LINE (#5830, #5850);
#5870 = EDGE_CURVE (#1240, #1200, #5860, .T.);
#5880 = CARTESIAN_POINT ((2.72172709647829, 3.32736078887258, 2.));
#5890 = DIRECTION ((-0.258819045102521, 0.965925826289068, 0.));
#5900 = VECTOR (#5890, 1.);
#5910 = LINE (#5880, #5900);
#5920 = EDGE_CURVE (#1260, #1220, #5910, .T.);
#5930 = CARTESIAN_POINT ((2.56592582628907, 3.90881904510252, -0.25));
#5940 = DIRECTION ((0., 0., 1.));
#5950 = VECTOR (#5940, 1.);

```
#5960 = LINE (#5930, #5950);
#5970 = EDGE_CURVE (#1200, #1260, #5960, .T.);
#5980 = CARTESIAN_POINT ((2.30710678118655, 4.87474487139159, -0.25));
#5990 = DIRECTION ((0., 0., 1.));
#6000 = VECTOR (#5990, 1.);
#6010 = LINE (#5980, #6000);
#6020 = EDGE_CURVE (#1240, #1220, #6010, .T.);
#6030 = CARTESIAN_POINT ((1.08537968470826, 4.54738408251901, 0.));
#6040 = DIRECTION ((0.965925826289068, 0.258819045102521, 0.));
#6050 = VECTOR (#6040, 1.);
#6060 = LINE (#6030, #6050);
#6070 = EDGE_CURVE (#1180, #1240, #6060, .T.);
#6080 = CARTESIAN_POINT ((1.34419872981078, 3.58145825622994, 2.));
#6090 = DIRECTION ((0.965925826289068, 0.258819045102521, 0.));
#6100 = VECTOR (#6090, 1.);
#6110 = LINE (#6080, #6100);
#6120 = EDGE_CURVE (#1120, #1260, #6110, .T.);
#6130 = CARTESIAN_POINT ((3., 3., 0.));
#6140 = DIRECTION ((0., 1., 0.));
#6150 = VECTOR (#6140, 1.);
#6160 = LINE (#6130, #6150);
#6170 = EDGE_CURVE (#1420, #1360, #6160, .T.);
#6180 = CARTESIAN_POINT ((1.5, 1., 0.));
#6190 = DIRECTION ((1., 0., 0.));
#6200 = VECTOR (#6190, 1.);
#6210 = LINE (#6180, #6200);
#6220 = EDGE_CURVE (#1400, #960, #6210, .T.);
#6230 = CARTESIAN_POINT ((0.5, 1.5, 2.));
#6240 = DIRECTION ((0., 0., 1.));
#6250 = DIRECTION ((1., 0., 0.));
#6260 = AXIS2_PLACEMENT_3D (#6230, #6240, #6250);
#6270 = CIRCLE (#6260, 0.25);
#6280 = EDGE_CURVE (#140, #140, #6270, .T.);
#6290 = CARTESIAN_POINT ((1.5, 1., 2.));
#6300 = DIRECTION ((-1., 0., 0.));
#6310 = VECTOR (#6300, 1.);
#6320 = LINE (#6290, #6310);
#6330 = EDGE_CURVE (#1340, #1040, #6320, .T.);
#6340 = CARTESIAN_POINT ((0., 3., 2.));
#6350 = DIRECTION ((0., 1., 0.));
#6360 = VECTOR (#6350, 1.);
#6370 = LINE (#6340, #6360);
#6380 = EDGE_CURVE (#1280, #1300, #6370, .T.);
#6390 = CARTESIAN_POINT ((1.5, 5., 2.));
#6400 = DIRECTION ((1., 0., 0.));
#6410 = VECTOR (#6400, 1.);
#6420 = LINE (#6390, #6410);
#6430 = EDGE_CURVE (#1300, #1320, #6420, .T.);
```

#6440 = CARTESIAN_POINT ((3., 3., 2.));
#6450 = DIRECTION ((0., -1., 0.));
#6460 = VECTOR (#6450, 1.);
#6470 = LINE (#6440, #6460);
#6480 = EDGE_CURVE (#1320, #760, #6470, .T.);
#6490 = CARTESIAN_POINT ((0., 1., 0.));
#6500 = DIRECTION ((0., 0., 1.));
#6510 = VECTOR (#6500, 1.);
#6520 = LINE (#6490, #6510);
#6530 = EDGE_CURVE (#1400, #1280, #6520, .T.);
#6540 = CARTESIAN_POINT ((0., 5., 0.));
#6550 = DIRECTION ((0., 0., 1.));
#6560 = VECTOR (#6550, 1.);
#6570 = LINE (#6540, #6560);
#6580 = EDGE_CURVE (#1380, #1300, #6570, .T.);
#6590 = CARTESIAN_POINT ((3., 5., 0.));
#6600 = DIRECTION ((0., 0., 1.));
#6610 = VECTOR (#6600, 1.);
#6620 = LINE (#6590, #6610);
#6630 = EDGE_CURVE (#1360, #1320, #6620, .T.);
#6640 = CARTESIAN_POINT ((3., 1., 0.));
#6650 = DIRECTION ((0., 0., 1.));
#6660 = VECTOR (#6650, 1.);
#6670 = LINE (#6640, #6660);
#6680 = EDGE_CURVE (#1420, #1340, #6670, .T.);
#6690 = CARTESIAN_POINT ((1.5, 5., 0.));
#6700 = DIRECTION ((-1., 0., 0.));
#6710 = VECTOR (#6700, 1.);
#6720 = LINE (#6690, #6710);
#6730 = EDGE_CURVE (#1360, #1380, #6720, .T.);
#6740 = CARTESIAN_POINT ((0., 3., 0.));
#6750 = DIRECTION ((0., -1., 0.));
#6760 = VECTOR (#6750, 1.);
#6770 = LINE (#6740, #6760);
#6780 = EDGE_CURVE (#1380, #1400, #6770, .T.);
#6790 = CARTESIAN_POINT ((0.5, 1.5, 0.5));
#6800 = DIRECTION ((0., 0., 1.));
#6810 = DIRECTION ((1., 0., 0.));
#6820 = AXIS2_PLACEMENT_3D (#6790, #6800, #6810);
#6830 = CIRCLE (#6820, 0.25);
#6840 = EDGE_CURVE (#120, #120, #6830, .T.);
#6850 = ORIENTED_EDGE (*, *, #2200, .F.);
#6860 = ORIENTED_EDGE (*, *, #3050, .F.);
#6870 = ORIENTED_EDGE (*, *, #1480, .F.);
#6880 = ORIENTED_EDGE (*, *, #2850, .F.);
#6890 = EDGE_LOOP ((#6850, #6860, #6870, #6880));
#6900 = FACE_BOUND (#6890, .T.);
#6910 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 0.75));

```
#6920 = DIRECTION ((0., 0., 1.));
#6930 = DIRECTION ((1., 0., 0.));
#6940 = AXIS2_PLACEMENT_3D (#6910, #6920, #6930);
#6950 = CYLINDRICAL_SURFACE (#6940, 0.25);
#6960 = FACE_SURFACE ((#6900), #6950, .F.);
#6970 = ORIENTED_EDGE (*, *, #1600, .F.);
#6980 = ORIENTED_EDGE (*, *, #2800, .F.);
#6990 = ORIENTED_EDGE (*, *, #1540, .F.);
#7000 = ORIENTED_EDGE (*, *, #3450, .F.);
#7010 = EDGE_LOOP ((#6970, #6980, #6990, #7000));
#7020 = FACE_OUTER_BOUND (#7010, .T.);
#7030 = CARTESIAN_POINT ((1.07569635196345, 2.03189123190276, 1.));
#7040 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#7050 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#7060 = AXIS2_PLACEMENT_3D (#7030, #7040, #7050);
#7070 = CYLINDRICAL_SURFACE (#7060, 0.25);
#7080 = FACE_SURFACE ((#7020), #7070, .F.);
#7090 = ORIENTED_EDGE (*, *, #2260, .F.);
#7100 = ORIENTED_EDGE (*, *, #2750, .F.);
#7110 = ORIENTED_EDGE (*, *, #1660, .F.);
#7120 = ORIENTED_EDGE (*, *, #2650, .F.);
#7130 = EDGE_LOOP ((#7090, #7100, #7110, #7120));
#7140 = FACE_OUTER_BOUND (#7130, .T.);
#7150 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 0.75));
#7160 = DIRECTION ((0., 0., 1.));
#7170 = DIRECTION ((1., 0., 0.));
#7180 = AXIS2_PLACEMENT_3D (#7150, #7160, #7170);
#7190 = CYLINDRICAL_SURFACE (#7180, 0.25);
#7200 = FACE_SURFACE ((#7140), #7190, .F.);
#7210 = ORIENTED_EDGE (*, *, #1540, .T.);
#7220 = ORIENTED_EDGE (*, *, #1480, .T.);
#7230 = ORIENTED_EDGE (*, *, #1720, .F.);
#7240 = EDGE_LOOP ((#7210, #7220, #7230));
#7250 = FACE_OUTER_BOUND (#7240, .T.);
#7260 = CARTESIAN_POINT ((1.69441478550168, 1.41317279836453, 1.));
#7270 = DIRECTION ((0., 0., 1.));
#7280 = DIRECTION ((1., 0., 0.));
#7290 = AXIS2_PLACEMENT_3D (#7260, #7270, #7280);
#7300 = SPHERICAL_SURFACE (#7290, 0.25);
#7310 = FACE_SURFACE ((#7250), #7300, .F.);
#7320 = ORIENTED_EDGE (*, *, #1660, .T.);
#7330 = ORIENTED_EDGE (*, *, #1600, .T.);
#7340 = ORIENTED_EDGE (*, *, #1780, .F.);
#7350 = EDGE_LOOP ((#7320, #7330, #7340));
#7360 = FACE_BOUND (#7350, .T.);
#7370 = CARTESIAN_POINT ((0.456977918425222, 2.65060966544099, 1.));
#7380 = DIRECTION ((0., 0., 1.));
#7390 = DIRECTION ((1., 0., 0.));
```

#7400 = AXIS2_PLACEMENT_3D (#7370, #7380, #7390);
#7410 = SPHERICAL_SURFACE (#7400, 0.25);
#7420 = FACE_SURFACE ((#7360), #7410, .F.);
#7430 = ORIENTED_EDGE (*, *, #2380, .F.);
#7440 = ORIENTED_EDGE (*, *, #3150, .F.);
#7450 = ORIENTED_EDGE (*, *, #1840, .F.);
#7460 = ORIENTED_EDGE (*, *, #2950, .F.);
#7470 = EDGE_LOOP ((#7430, #7440, #7450, #7460));
#7480 = FACE_BOUND (#7470, .T.);
#7490 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 0.75));
#7500 = DIRECTION ((0., 0., 1.));
#7510 = DIRECTION ((1., 0., 0.));
#7520 = AXIS2_PLACEMENT_3D (#7490, #7500, #7510);
#7530 = CYLINDRICAL_SURFACE (#7520, 0.25);
#7540 = FACE_SURFACE ((#7480), #7530, .F.);
#7550 = ORIENTED_EDGE (*, *, #1720, .T.);
#7560 = ORIENTED_EDGE (*, *, #3000, .F.);
#7570 = ORIENTED_EDGE (*, *, #1900, .F.);
#7580 = ORIENTED_EDGE (*, *, #3400, .F.);
#7590 = EDGE_LOOP ((#7550, #7560, #7570, #7580));
#7600 = FACE_OUTER_BOUND (#7590, .T.);
#7610 = CARTESIAN_POINT ((2.04796817609495, 1.7667261889578, 1.));
#7620 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#7630 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#7640 = AXIS2_PLACEMENT_3D (#7610, #7620, #7630);
#7650 = CYLINDRICAL_SURFACE (#7640, 0.25);
#7660 = FACE_SURFACE ((#7600), #7650, .F.);
#7670 = ORIENTED_EDGE (*, *, #1780, .T.);
#7680 = ORIENTED_EDGE (*, *, #3500, .F.);
#7690 = ORIENTED_EDGE (*, *, #1960, .F.);
#7700 = ORIENTED_EDGE (*, *, #2600, .F.);
#7710 = EDGE_LOOP ((#7670, #7680, #7690, #7700));
#7720 = FACE_OUTER_BOUND (#7710, .T.);
#7730 = CARTESIAN_POINT ((0.810531309018496, 3.00416305603426, 1.));
#7740 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#7750 = DIRECTION ((0.707106781186548, -0.707106781186548, 0.));
#7760 = AXIS2_PLACEMENT_3D (#7730, #7740, #7750);
#7770 = CYLINDRICAL_SURFACE (#7760, 0.25);
#7780 = FACE_SURFACE ((#7720), #7770, .F.);
#7790 = ORIENTED_EDGE (*, *, #2320, .F.);
#7800 = ORIENTED_EDGE (*, *, #2550, .F.);
#7810 = ORIENTED_EDGE (*, *, #2020, .F.);
#7820 = ORIENTED_EDGE (*, *, #3250, .F.);
#7830 = EDGE_LOOP ((#7790, #7800, #7810, #7820));
#7840 = FACE_OUTER_BOUND (#7830, .T.);
#7850 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 0.75));
#7860 = DIRECTION ((0., 0., 1.));
#7870 = DIRECTION ((1., 0., 0.));

#7880 = AXIS2_PLACEMENT_3D (#7850, #7860, #7870);
#7890 = CYLINDRICAL_SURFACE (#7880, 0.25);
#7900 = FACE_SURFACE ((#7840), #7890, .F.);
#7910 = ORIENTED_EDGE (*, *, #1900, .T.);
#7920 = ORIENTED_EDGE (*, *, #1840, .T.);
#7930 = ORIENTED_EDGE (*, *, #2080, .F.);
#7940 = EDGE_LOOP ((#7910, #7920, #7930));
#7950 = FACE_OUTER_BOUND (#7940, .T.);
#7960 = CARTESIAN_POINT ((2.40152156668823, 2.12027957955108, 1.));
#7970 = DIRECTION ((0., 0., 1.));
#7980 = DIRECTION ((1., 0., 0.));
#7990 = AXIS2_PLACEMENT_3D (#7960, #7970, #7980);
#8000 = SPHERICAL_SURFACE (#7990, 0.25);
#8010 = FACE_SURFACE ((#7950), #8000, .F.);
#8020 = ORIENTED_EDGE (*, *, #2020, .T.);
#8030 = ORIENTED_EDGE (*, *, #1960, .T.);
#8040 = ORIENTED_EDGE (*, *, #2140, .F.);
#8050 = EDGE_LOOP ((#8020, #8030, #8040));
#8060 = FACE_BOUND (#8050, .T.);
#8070 = CARTESIAN_POINT ((1.16408469961177, 3.35771644662754, 1.));
#8080 = DIRECTION ((0., 0., 1.));
#8090 = DIRECTION ((1., 0., 0.));
#8100 = AXIS2_PLACEMENT_3D (#8070, #8080, #8090);
#8110 = SPHERICAL_SURFACE (#8100, 0.25);
#8120 = FACE_SURFACE ((#8060), #8110, .F.);
#8130 = ORIENTED_EDGE (*, *, #2080, .T.);
#8140 = ORIENTED_EDGE (*, *, #3200, .F.);
#8150 = ORIENTED_EDGE (*, *, #2140, .T.);
#8160 = ORIENTED_EDGE (*, *, #3350, .F.);
#8170 = EDGE_LOOP ((#8130, #8140, #8150, #8160));
#8180 = FACE_BOUND (#8170, .T.);
#8190 = CARTESIAN_POINT ((1.78280313315, 2.73899801308931, 1.));
#8200 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#8210 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));
#8220 = AXIS2_PLACEMENT_3D (#8190, #8200, #8210);
#8230 = CYLINDRICAL_SURFACE (#8220, 0.25);
#8240 = FACE_SURFACE ((#8180), #8230, .F.);
#8250 = ORIENTED_EDGE (*, *, #3720, .F.);
#8260 = ORIENTED_EDGE (*, *, #3920, .F.);
#8270 = ORIENTED_EDGE (*, *, #2440, .F.);
#8280 = ORIENTED_EDGE (*, *, #3550, .F.);
#8290 = EDGE_LOOP ((#8250, #8260, #8270, #8280));
#8300 = FACE_OUTER_BOUND (#8290, .T.);
#8310 = CARTESIAN_POINT ((2.625, 2.625, 1.75));
#8320 = DIRECTION ((0., 0., 1.));
#8330 = DIRECTION ((1., 0., 0.));
#8340 = AXIS2_PLACEMENT_3D (#8310, #8320, #8330);
#8350 = CYLINDRICAL_SURFACE (#8340, 0.125);

#8360 = FACE_SURFACE ((#8300), #8350, .F.);
#8370 = ORIENTED_EDGE (*, *, #3660, .F.);
#8380 = ORIENTED_EDGE (*, *, #3820, .F.);
#8390 = ORIENTED_EDGE (*, *, #2500, .F.);
#8400 = ORIENTED_EDGE (*, *, #3870, .F.);
#8410 = EDGE_LOOP ((#8370, #8380, #8390, #8400));
#8420 = FACE_OUTER_BOUND (#8410, .T.);
#8430 = CARTESIAN_POINT ((2.625, 3.375, 1.75));
#8440 = DIRECTION ((0., 0., -1.));
#8450 = DIRECTION ((-1., 0., 0.));
#8460 = AXIS2_PLACEMENT_3D (#8430, #8440, #8450);
#8470 = CYLINDRICAL_SURFACE (#8460, 0.125);
#8480 = FACE_SURFACE ((#8420), #8470, .F.);
#8490 = ORIENTED_EDGE (*, *, #3970, .F.);
#8500 = ORIENTED_EDGE (*, *, #3870, .T.);
#8510 = ORIENTED_EDGE (*, *, #4020, .T.);
#8520 = ORIENTED_EDGE (*, *, #3920, .T.);
#8530 = EDGE_LOOP ((#8490, #8500, #8510, #8520));
#8540 = FACE_OUTER_BOUND (#8530, .T.);
#8550 = CARTESIAN_POINT ((2.5, 3., 1.75));
#8560 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#8570 = DIRECTION ((0., 0., 1.));
#8580 = AXIS2_PLACEMENT_3D (#8550, #8560, #8570);
#8590 = PLANE (#8580);
#8600 = FACE_SURFACE ((#8540), #8590, .F.);
#8610 = ORIENTED_EDGE (*, *, #3770, .F.);
#8620 = ORIENTED_EDGE (*, *, #3550, .T.);
#8630 = ORIENTED_EDGE (*, *, #3600, .T.);
#8640 = ORIENTED_EDGE (*, *, #4070, .T.);
#8650 = EDGE_LOOP ((#8610, #8620, #8630, #8640));
#8660 = FACE_OUTER_BOUND (#8650, .T.);
#8670 = CARTESIAN_POINT ((3.25, 2.5, 1.75));
#8680 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#8690 = DIRECTION ((0., 0., 1.));
#8700 = AXIS2_PLACEMENT_3D (#8670, #8680, #8690);
#8710 = PLANE (#8700);
#8720 = FACE_SURFACE ((#8660), #8710, .F.);
#8730 = ORIENTED_EDGE (*, *, #4220, .T.);
#8740 = ORIENTED_EDGE (*, *, #3660, .T.);
#8750 = ORIENTED_EDGE (*, *, #3970, .T.);
#8760 = ORIENTED_EDGE (*, *, #3720, .T.);
#8770 = ORIENTED_EDGE (*, *, #3770, .T.);
#8780 = ORIENTED_EDGE (*, *, #4270, .T.);
#8790 = EDGE_LOOP ((#8730, #8740, #8750, #8760, #8770, #8780));
#8800 = FACE_OUTER_BOUND (#8790, .T.);
#8810 = CARTESIAN_POINT ((3.25, 3., 1.75));
#8820 = DIRECTION ((0., 0., -1.));
#8830 = DIRECTION ((-1.41553435639707E-15, -1., 0.));

#8840 = AXIS2_PLACEMENT_3D (#8810, #8820, #8830);
#8850 = PLANE (#8840);
#8860 = FACE_SURFACE ((#8800), #8850, .F.);
#8870 = ORIENTED_EDGE (*, *, #4120, .T.);
#8880 = ORIENTED_EDGE (*, *, #3820, .T.);
#8890 = ORIENTED_EDGE (*, *, #4220, .F.);
#8900 = ORIENTED_EDGE (*, *, #4170, .T.);
#8910 = EDGE_LOOP ((#8870, #8880, #8890, #8900));
#8920 = FACE_OUTER_BOUND (#8910, .T.);
#8930 = CARTESIAN_POINT ((3.25, 3.5, 1.75));
#8940 = DIRECTION ((1.42125554512318E-15, 1., 0.));
#8950 = DIRECTION ((0., 0., -1.));
#8960 = AXIS2_PLACEMENT_3D (#8930, #8940, #8950);
#8970 = PLANE (#8960);
#8980 = FACE_SURFACE ((#8920), #8970, .F.);
#8990 = ORIENTED_EDGE (*, *, #4620, .F.);
#9000 = ORIENTED_EDGE (*, *, #4770, .F.);
#9010 = ORIENTED_EDGE (*, *, #4720, .F.);
#9020 = ORIENTED_EDGE (*, *, #4670, .F.);
#9030 = EDGE_LOOP ((#8990, #9000, #9010, #9020));
#9040 = FACE_OUTER_BOUND (#9030, .T.);
#9050 = CARTESIAN_POINT ((0.8125, 4.5, 2.1875));
#9060 = DIRECTION ((0., 0., 1.));
#9070 = DIRECTION ((1.42247325030098E-15, 1., 0.));
#9080 = AXIS2_PLACEMENT_3D (#9050, #9060, #9070);
#9090 = PLANE (#9080);
#9100 = FACE_SURFACE ((#9040), #9090, .T.);
#9110 = ORIENTED_EDGE (*, *, #4370, .F.);
#9120 = ORIENTED_EDGE (*, *, #4820, .T.);
#9130 = ORIENTED_EDGE (*, *, #4520, .T.);
#9140 = ORIENTED_EDGE (*, *, #4620, .T.);
#9150 = EDGE_LOOP ((#9110, #9120, #9130, #9140));
#9160 = FACE_OUTER_BOUND (#9150, .T.);
#9170 = CARTESIAN_POINT ((1., 4.5, 1.9375));
#9180 = DIRECTION ((1., -1.42125554512318E-15, 0.));
#9190 = DIRECTION ((0., 0., -1.));
#9200 = AXIS2_PLACEMENT_3D (#9170, #9180, #9190);
#9210 = PLANE (#9200);
#9220 = FACE_SURFACE ((#9160), #9210, .T.);
#9230 = ORIENTED_EDGE (*, *, #4470, .F.);
#9240 = ORIENTED_EDGE (*, *, #4870, .T.);
#9250 = ORIENTED_EDGE (*, *, #4370, .T.);
#9260 = ORIENTED_EDGE (*, *, #4670, .T.);
#9270 = EDGE_LOOP ((#9230, #9240, #9250, #9260));
#9280 = FACE_OUTER_BOUND (#9270, .T.);
#9290 = CARTESIAN_POINT ((0.8125, 4.375, 1.9375));
#9300 = DIRECTION ((-1.42125554512318E-15, -1., 0.));
#9310 = DIRECTION ((0., 0., 1.));

#9320 = AXIS2_PLACEMENT_3D (#9290, #9300, #9310);
#9330 = PLANE (#9320);
#9340 = FACE_SURFACE ((#9280), #9330, .T.);
#9350 = ORIENTED_EDGE (*, *, #4570, .F.);
#9360 = ORIENTED_EDGE (*, *, #4420, .T.);
#9370 = ORIENTED_EDGE (*, *, #4470, .T.);
#9380 = ORIENTED_EDGE (*, *, #4720, .T.);
#9390 = EDGE_LOOP ((#9350, #9360, #9370, #9380));
#9400 = FACE_OUTER_BOUND (#9390, .T.);
#9410 = CARTESIAN_POINT ((0.625, 4.5, 1.9375));
#9420 = DIRECTION ((-1., 1.42125554512318E-15, 0.));
#9430 = DIRECTION ((0., 0., 1.));
#9440 = AXIS2_PLACEMENT_3D (#9410, #9420, #9430);
#9450 = PLANE (#9440);
#9460 = FACE_SURFACE ((#9400), #9450, .T.);
#9470 = ORIENTED_EDGE (*, *, #4520, .F.);
#9480 = ORIENTED_EDGE (*, *, #4920, .T.);
#9490 = ORIENTED_EDGE (*, *, #4570, .T.);
#9500 = ORIENTED_EDGE (*, *, #4770, .T.);
#9510 = EDGE_LOOP ((#9470, #9480, #9490, #9500));
#9520 = FACE_OUTER_BOUND (#9510, .T.);
#9530 = CARTESIAN_POINT ((0.8125, 4.625, 1.9375));
#9540 = DIRECTION ((1.42125554512318E-15, 1., 0.));
#9550 = DIRECTION ((0., 0., -1.));
#9560 = AXIS2_PLACEMENT_3D (#9530, #9540, #9550);
#9570 = PLANE (#9560);
#9580 = FACE_SURFACE ((#9520), #9570, .T.);
#9590 = ORIENTED_EDGE (*, *, #4970, .T.);
#9600 = ORIENTED_EDGE (*, *, #5370, .T.);
#9610 = ORIENTED_EDGE (*, *, #5020, .T.);
#9620 = ORIENTED_EDGE (*, *, #5220, .F.);
#9630 = EDGE_LOOP ((#9590, #9600, #9610, #9620));
#9640 = FACE_OUTER_BOUND (#9630, .T.);
#9650 = CARTESIAN_POINT ((2.25, 1.25, -0.25));
#9660 = DIRECTION ((0., 1., 0.));
#9670 = DIRECTION ((0., 0., 1.));
#9680 = AXIS2_PLACEMENT_3D (#9650, #9660, #9670);
#9690 = PLANE (#9680);
#9700 = FACE_SURFACE ((#9640), #9690, .F.);
#9710 = ORIENTED_EDGE (*, *, #5070, .T.);
#9720 = ORIENTED_EDGE (*, *, #5220, .T.);
#9730 = ORIENTED_EDGE (*, *, #5120, .T.);
#9740 = ORIENTED_EDGE (*, *, #5170, .T.);
#9750 = EDGE_LOOP ((#9710, #9720, #9730, #9740));
#9760 = FACE_OUTER_BOUND (#9750, .T.);
#9770 = CARTESIAN_POINT ((2., 1., -0.25));
#9780 = DIRECTION ((-1., 0., 0.));
#9790 = DIRECTION ((0., 0., 1.));

```
#9800 = AXIS2_PLACEMENT_3D (#9770, #9780, #9790);
#9810 = PLANE (#9800);
#9820 = FACE_SURFACE ((#9760), #9810, .F.);
#9830 = ORIENTED_EDGE (*, *, #5420, .T.);
#9840 = ORIENTED_EDGE (*, *, #5320, .T.);
#9850 = ORIENTED_EDGE (*, *, #5270, .T.);
#9860 = ORIENTED_EDGE (*, *, #5370, .F.);
#9870 = EDGE_LOOP ((#9830, #9840, #9850, #9860));
#9880 = FACE_OUTER_BOUND (#9870, .T.);
#9890 = CARTESIAN_POINT ((2.5, 1., -0.25));
#9900 = DIRECTION ((1., 0., 0.));
#9910 = DIRECTION ((0., 0., -1.));
#9920 = AXIS2_PLACEMENT_3D (#9890, #9900, #9910);
#9930 = PLANE (#9920);
#9940 = FACE_SURFACE ((#9880), #9930, .F.);
#9950 = ORIENTED_EDGE (*, *, #5170, .F.);
#9960 = ORIENTED_EDGE (*, *, #5520, .T.);
#9970 = ORIENTED_EDGE (*, *, #6530, .F.);
#9980 = ORIENTED_EDGE (*, *, #6220, .T.);
#9990 = EDGE_LOOP ((#9950, #9960, #9970, #9980));
#10000 = FACE_OUTER_BOUND (#9990, .T.);
#10010 = CARTESIAN_POINT ((1.5, 1., 0.));
#10020 = DIRECTION ((0., -1., 0.));
#10030 = DIRECTION ((0., 0., -1.));
#10040 = AXIS2_PLACEMENT_3D (#10010, #10020, #10030);
#10050 = PLANE (#10040);
#10060 = FACE_SURFACE ((#10000), #10050, .T.);
#10070 = ORIENTED_EDGE (*, *, #5570, .T.);
#10080 = ORIENTED_EDGE (*, *, #5720, .T.);
#10090 = ORIENTED_EDGE (*, *, #6120, .T.);
#10100 = ORIENTED_EDGE (*, *, #5970, .F.);
#10110 = EDGE_LOOP ((#10070, #10080, #10090, #10100));
#10120 = FACE_OUTER_BOUND (#10110, .T.);
#10130 = CARTESIAN_POINT ((2.08296291314453, 3.77940952255126, -0.25));
#10140 = DIRECTION ((0.258819045102521, -0.965925826289068, 0.));
#10150 = DIRECTION ((0., 0., -1.));
#10160 = AXIS2_PLACEMENT_3D (#10130, #10140, #10150);
#10170 = PLANE (#10160);
#10180 = FACE_SURFACE ((#10120), #10170, .F.);
#10190 = ORIENTED_EDGE (*, *, #5620, .T.);
#10200 = ORIENTED_EDGE (*, *, #5820, .T.);
#10210 = ORIENTED_EDGE (*, *, #5670, .T.);
#10220 = ORIENTED_EDGE (*, *, #5720, .F.);
#10230 = EDGE_LOOP ((#10190, #10200, #10210, #10220));
#10240 = FACE_OUTER_BOUND (#10230, .T.);
#10250 = CARTESIAN_POINT ((1.47059047744874, 4.13296291314453, -0.25));
#10260 = DIRECTION ((-0.965925826289068, -0.258819045102521, 0.));
#10270 = DIRECTION ((0., 0., 1.));
```

#10280 = AXIS2_PLACEMENT_3D (#10250, #10260, #10270);
#10290 = PLANE (#10280);
#10300 = FACE_SURFACE ((#10240), #10290, .F.);
#10310 = ORIENTED_EDGE (*, *, #6070, .T.);
#10320 = ORIENTED_EDGE (*, *, #6020, .T.);
#10330 = ORIENTED_EDGE (*, *, #5770, .T.);
#10340 = ORIENTED_EDGE (*, *, #5820, .F.);
#10350 = EDGE_LOOP ((#10310, #10320, #10330, #10340));
#10360 = FACE_OUTER_BOUND (#10350, .T.);
#10370 = CARTESIAN_POINT ((1.82414386804201, 4.74533534884033, -0.25));
#10380 = DIRECTION ((-0.258819045102521, 0.965925826289068, 0.));
#10390 = DIRECTION ((0., 0., 1.));
#10400 = AXIS2_PLACEMENT_3D (#10370, #10380, #10390);
#10410 = PLANE (#10400);
#10420 = FACE_SURFACE ((#10360), #10410, .F.);
#10430 = ORIENTED_EDGE (*, *, #5870, .T.);
#10440 = ORIENTED_EDGE (*, *, #5970, .T.);
#10450 = ORIENTED_EDGE (*, *, #5920, .T.);
#10460 = ORIENTED_EDGE (*, *, #6020, .F.);
#10470 = EDGE_LOOP ((#10430, #10440, #10450, #10460));
#10480 = FACE_OUTER_BOUND (#10470, .T.);
#10490 = CARTESIAN_POINT ((2.43651630373781, 4.39178195824705, -0.25));
#10500 = DIRECTION ((0.965925826289068, 0.258819045102521, 0.));
#10510 = DIRECTION ((0., 0., -1.));
#10520 = AXIS2_PLACEMENT_3D (#10490, #10500, #10510);
#10530 = PLANE (#10520);
#10540 = FACE_SURFACE ((#10480), #10530, .F.);
#10550 = ORIENTED_EDGE (*, *, #5070, .F.);
#10560 = ORIENTED_EDGE (*, *, #6220, .F.);
#10570 = ORIENTED_EDGE (*, *, #6780, .F.);
#10580 = ORIENTED_EDGE (*, *, #6730, .F.);
#10590 = ORIENTED_EDGE (*, *, #6170, .F.);
#10600 = ORIENTED_EDGE (*, *, #5470, .F.);
#10610 = ORIENTED_EDGE (*, *, #5420, .F.);
#10620 = ORIENTED_EDGE (*, *, #4970, .F.);
#10630 = EDGE_LOOP ((#10550, #10560, #10570, #10580, #10590, #10600, #10610, #10620));
#10640 = FACE_OUTER_BOUND (#10630, .T.);
#10650 = ORIENTED_EDGE (*, *, #5570, .F.);
#10660 = ORIENTED_EDGE (*, *, #5870, .F.);
#10670 = ORIENTED_EDGE (*, *, #6070, .F.);
#10680 = ORIENTED_EDGE (*, *, #5620, .F.);
#10690 = EDGE_LOOP ((#10650, #10660, #10670, #10680));
#10700 = FACE_BOUND (#10690, .T.);
#10710 = CARTESIAN_POINT ((1.5, 3., 0.));
#10720 = DIRECTION ((0., 0., -1.));
#10730 = DIRECTION ((-1., 0., 0.));
#10740 = AXIS2_PLACEMENT_3D (#10710, #10720, #10730);

```
#10750 = PLANE (#10740);
#10760 = FACE_SURFACE ((#10640, #10700), #10750, .T.);
#10770 = ORIENTED_EDGE (*, *, #3600, .F.);
#10780 = ORIENTED_EDGE (*, *, #2440, .T.);
#10790 = ORIENTED_EDGE (*, *, #4020, .F.);
#10800 = ORIENTED_EDGE (*, *, #2500, .T.);
#10810 = ORIENTED_EDGE (*, *, #4120, .F.);
#10820 = ORIENTED_EDGE (*, *, #6480, .F.);
#10830 = ORIENTED_EDGE (*, *, #6430, .F.);
#10840 = ORIENTED_EDGE (*, *, #6380, .F.);
#10850 = ORIENTED_EDGE (*, *, #5520, .F.);
#10860 = ORIENTED_EDGE (*, *, #5120, .F.);
#10870 = ORIENTED_EDGE (*, *, #5020, .F.);
#10880 = ORIENTED_EDGE (*, *, #5270, .F.);
#10890 = ORIENTED_EDGE (*, *, #6330, .F.);
#10900 = ORIENTED_EDGE (*, *, #4320, .F.);
#10910 = EDGE_LOOP ((#10770, #10780, #10790, #10800, #10810, #10820, #10830,
 #10840, #10850, #10860, #10870, #10880, #10890, #10900));
#10920 = FACE_BOUND (#10910, .T.);
#10930 = ORIENTED_EDGE (*, *, #3100, .F.);
#10940 = ORIENTED_EDGE (*, *, #2200, .T.);
#10950 = ORIENTED_EDGE (*, *, #2900, .F.);
#10960 = ORIENTED_EDGE (*, *, #2260, .T.);
#10970 = ORIENTED_EDGE (*, *, #2700, .F.);
#10980 = ORIENTED_EDGE (*, *, #2320, .T.);
#10990 = ORIENTED_EDGE (*, *, #3300, .F.);
#11000 = ORIENTED_EDGE (*, *, #2380, .T.);
#11010 = EDGE_LOOP ((#10930, #10940, #10950, #10960, #10970, #10980, #10990,
 #11000));
#11020 = FACE_OUTER_BOUND (#11010, .T.);
#11030 = ORIENTED_EDGE (*, *, #4820, .F.);
#11040 = ORIENTED_EDGE (*, *, #4870, .F.);
#11050 = ORIENTED_EDGE (*, *, #4420, .F.);
#11060 = ORIENTED_EDGE (*, *, #4920, .F.);
#11070 = EDGE_LOOP ((#11030, #11040, #11050, #11060));
#11080 = FACE_BOUND (#11070, .T.);
#11090 = ORIENTED_EDGE (*, *, #6120, .F.);
#11100 = ORIENTED_EDGE (*, *, #5670, .F.);
#11110 = ORIENTED_EDGE (*, *, #5770, .F.);
#11120 = ORIENTED_EDGE (*, *, #5920, .F.);
#11130 = EDGE_LOOP ((#11090, #11100, #11110, #11120));
#11140 = FACE_BOUND (#11130, .T.);
#11150 = ORIENTED_EDGE (*, *, #6280, .F.);
#11160 = EDGE_LOOP ((#11150));
#11170 = FACE_BOUND (#11160, .T.);
#11180 = CARTESIAN_POINT ((1.5, 3., 2.));
#11190 = DIRECTION ((0., 0., 1.));
#11200 = DIRECTION ((1., 0., 0.));
```

#11210 = AXIS2_PLACEMENT_3D (#11180, #11190, #11200);
#11220 = PLANE (#11210);
#11230 = FACE_SURFACE ((#10920, #11020, #11080, #11140, #11170), #11220, .T.);
#11240 = ORIENTED_EDGE (*, *, #5320, .F.);
#11250 = ORIENTED_EDGE (*, *, #5470, .T.);
#11260 = ORIENTED_EDGE (*, *, #6680, .T.);
#11270 = ORIENTED_EDGE (*, *, #6330, .T.);
#11280 = EDGE_LOOP ((#11240, #11250, #11260, #11270));
#11290 = FACE_BOUND (#11280, .T.);
#11300 = CARTESIAN_POINT ((1.5, 1., 0.));
#11310 = DIRECTION ((0., -1., 0.));
#11320 = DIRECTION ((0., 0., -1.));
#11330 = AXIS2_PLACEMENT_3D (#11300, #11310, #11320);
#11340 = PLANE (#11330);
#11350 = FACE_SURFACE ((#11290), #11340, .T.);
#11360 = ORIENTED_EDGE (*, *, #6530, .T.);
#11370 = ORIENTED_EDGE (*, *, #6380, .T.);
#11380 = ORIENTED_EDGE (*, *, #6580, .F.);
#11390 = ORIENTED_EDGE (*, *, #6780, .T.);
#11400 = EDGE_LOOP ((#11360, #11370, #11380, #11390));
#11410 = FACE_OUTER_BOUND (#11400, .T.);
#11420 = CARTESIAN_POINT ((0., 3., 0.));
#11430 = DIRECTION ((-1., 0., 0.));
#11440 = DIRECTION ((0., 0., 1.));
#11450 = AXIS2_PLACEMENT_3D (#11420, #11430, #11440);
#11460 = PLANE (#11450);
#11470 = FACE_SURFACE ((#11410), #11460, .T.);
#11480 = ORIENTED_EDGE (*, *, #6580, .T.);
#11490 = ORIENTED_EDGE (*, *, #6430, .T.);
#11500 = ORIENTED_EDGE (*, *, #6630, .F.);
#11510 = ORIENTED_EDGE (*, *, #6730, .T.);
#11520 = EDGE_LOOP ((#11480, #11490, #11500, #11510));
#11530 = FACE_OUTER_BOUND (#11520, .T.);
#11540 = CARTESIAN_POINT ((1.5, 5., 0.));
#11550 = DIRECTION ((0., 1., 0.));
#11560 = DIRECTION ((0., 0., 1.));
#11570 = AXIS2_PLACEMENT_3D (#11540, #11550, #11560);
#11580 = PLANE (#11570);
#11590 = FACE_SURFACE ((#11530), #11580, .T.);
#11600 = ORIENTED_EDGE (*, *, #4070, .F.);
#11610 = ORIENTED_EDGE (*, *, #4320, .T.);
#11620 = ORIENTED_EDGE (*, *, #6680, .F.);
#11630 = ORIENTED_EDGE (*, *, #6170, .T.);
#11640 = ORIENTED_EDGE (*, *, #6630, .T.);
#11650 = ORIENTED_EDGE (*, *, #6480, .T.);
#11660 = ORIENTED_EDGE (*, *, #4170, .F.);
#11670 = ORIENTED_EDGE (*, *, #4270, .F.);
#11680 = EDGE_LOOP ((#11600, #11610, #11620, #11630, #11640, #11650, #11660,

```
#11670));
#11690 = FACE_OUTER_BOUND (#11680, .T.);
#11700 = CARTESIAN_POINT ((3., 3., 0.));
#11710 = DIRECTION ((1., 0., 0.));
#11720 = DIRECTION ((0., 0., -1.));
#11730 = AXIS2_PLACEMENT_3D (#11700, #11710, #11720);
#11740 = PLANE (#11730);
#11750 = FACE_SURFACE ((#11690), #11740, .T.);
#11760 = ORIENTED_EDGE (*, *, #6840, .F.);
#11770 = EDGE_LOOP ((#11760));
#11780 = FACE_OUTER_BOUND (#11770, .T.);
#11790 = ORIENTED_EDGE (*, *, #6280, .T.);
#11800 = EDGE_LOOP ((#11790));
#11810 = FACE_BOUND (#11800, .T.);
#11820 = CARTESIAN_POINT ((0.5, 1.5, 3.));
#11830 = DIRECTION ((0., 0., -1.));
#11840 = DIRECTION ((-1., 0., 0.));
#11850 = AXIS2_PLACEMENT_3D (#11820, #11830, #11840);
#11860 = CYLINDRICAL_SURFACE (#11850, 0.25);
#11870 = FACE_SURFACE ((#11780, #11810), #11860, .F.);
#11880 = ORIENTED_EDGE (*, *, #6840, .T.);
#11890 = EDGE_LOOP ((#11880));
#11900 = FACE_BOUND (#11890, .T.);
#11910 = CARTESIAN_POINT ((0.5, 1.5, 0.5));
#11920 = DIRECTION ((0., 0., -1.));
#11930 = DIRECTION ((-1., 0., 0.));
#11940 = AXIS2_PLACEMENT_3D (#11910, #11920, #11930);
#11950 = PLANE (#11940);
#11960 = FACE_SURFACE ((#11900), #11950, .F.);
#11970 = ORIENTED_EDGE (*, *, #3100, .T.);
#11980 = ORIENTED_EDGE (*, *, #2950, .T.);
#11990 = ORIENTED_EDGE (*, *, #3000, .T.);
#12000 = ORIENTED_EDGE (*, *, #3050, .T.);
#12010 = EDGE_LOOP ((#11970, #11980, #11990, #12000));
#12020 = FACE_OUTER_BOUND (#12010, .T.);
#12030 = CARTESIAN_POINT ((2.22474487139159, 1.58994949366117, 0.75));
#12040 = DIRECTION ((0.707106781186547, -0.707106781186548, 0.));
#12050 = DIRECTION ((0., 0., -1.));
#12060 = AXIS2_PLACEMENT_3D (#12030, #12040, #12050);
#12070 = PLANE (#12060);
#12080 = FACE_SURFACE ((#12020), #12070, .F.);
#12090 = ORIENTED_EDGE (*, *, #2800, .T.);
#12100 = ORIENTED_EDGE (*, *, #2750, .T.);
#12110 = ORIENTED_EDGE (*, *, #2900, .T.);
#12120 = ORIENTED_EDGE (*, *, #2850, .T.);
#12130 = EDGE_LOOP ((#12090, #12100, #12110, #12120));
#12140 = FACE_OUTER_BOUND (#12130, .T.);
#12150 = CARTESIAN_POINT ((0.898919656666814, 1.85511453660612, 0.75));
```

```
#12160 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#12170 = DIRECTION ((0., 0., 1.));
#12180 = AXIS2_PLACEMENT_3D (#12150, #12160, #12170);
#12190 = PLANE (#12180);
#12200 = FACE_SURFACE ((#12140), #12190, .F.);
#12210 = ORIENTED_EDGE (*, *, #2600, .T.);
#12220 = ORIENTED_EDGE (*, *, #2550, .T.);
#12230 = ORIENTED_EDGE (*, *, #2700, .T.);
#12240 = ORIENTED_EDGE (*, *, #2650, .T.);
#12250 = EDGE_LOOP ((#12210, #12220, #12230, #12240));
#12260 = FACE_OUTER_BOUND (#12250, .T.);
#12270 = CARTESIAN_POINT ((0.633754613721859, 3.1809397513309, 0.75));
#12280 = DIRECTION ((-0.707106781186547, 0.707106781186548, 0.));
#12290 = DIRECTION ((0., 0., 1.));
#12300 = AXIS2_PLACEMENT_3D (#12270, #12280, #12290);
#12310 = PLANE (#12300);
#12320 = FACE_SURFACE ((#12260), #12310, .F.);
#12330 = ORIENTED_EDGE (*, *, #3200, .T.);
#12340 = ORIENTED_EDGE (*, *, #3150, .T.);
#12350 = ORIENTED_EDGE (*, *, #3300, .T.);
#12360 = ORIENTED_EDGE (*, *, #3250, .T.);
#12370 = EDGE_LOOP ((#12330, #12340, #12350, #12360));
#12380 = FACE_OUTER_BOUND (#12370, .T.);
#12390 = CARTESIAN_POINT ((1.95957982844664, 2.91577470838594, 0.75));
#12400 = DIRECTION ((0.707106781186548, 0.707106781186547, 0.));
#12410 = DIRECTION ((0., 0., -1.));
#12420 = AXIS2_PLACEMENT_3D (#12390, #12400, #12410);
#12430 = PLANE (#12420);
#12440 = FACE_SURFACE ((#12380), #12430, .F.);
#12450 = ORIENTED_EDGE (*, *, #3400, .T.);
#12460 = ORIENTED_EDGE (*, *, #3350, .T.);
#12470 = ORIENTED_EDGE (*, *, #3500, .T.);
#12480 = ORIENTED_EDGE (*, *, #3450, .T.);
#12490 = EDGE_LOOP ((#12450, #12460, #12470, #12480));
#12500 = FACE_OUTER_BOUND (#12490, .T.);
#12510 = CARTESIAN_POINT ((1.42924974255673, 2.38544462249603, 0.75));
#12520 = DIRECTION ((0., 0., -1.));
#12530 = DIRECTION ((-0.707106781186548, -0.707106781186547, 0.));
#12540 = AXIS2_PLACEMENT_3D (#12510, #12520, #12530);
#12550 = PLANE (#12540);
#12560 = FACE_SURFACE ((#12500), #12550, .F.);
#12570 = CLOSED_SHELL ((#6960, #7080, #7200, #7310, #7420, #7540, #7660, #7780,
 #7900, #8010, #8120, #8240, #8360, #8480, #8600, #8720, #8860, #8980, #9100,
 #9220, #9340, #9460, #9580, #9700, #9820, #9940, #10060, #10180, #10300,
 #10420, #10540, #10760, #11230, #11350, #11470, #11590, #11750, #11870, #11960,
 #12080, #12200, #12320, #12440, #12560));
#12580 = MANIFOLD_SOLID_BREP (#12570);
#12590 = ADVANCED_BREP_REPRESENTATION ((#12580), #100);
```

```
#12600 = APPLICATION_CONTEXT ('INTERNATIONAL_STANDARD',
'CONFIG_CONTROL_DESIGN',
 0, 'CONFIGURATION_MANAGEMENT');
#12610 = PRODUCT_CONTEXT ('CONFIGURATION_MANAGEMENT', #12600,
'MECHANICAL');
#12620 = PRODUCT_CATEGORY ('MECHANICAL', 'ALL MECHANICAL PARTS');
#12630 = PRODUCT ('NOT_KNOWN', 'NOT_KNOWN', 'NOT_KNOWN', #12610);
#12640 = PRODUCT_RELATED_PRODUCT_CATEGORY ('DETAIL', 'DETAIL PART',
(#12630));
#12650 = PRODUCT_CATEGORY_RELATIONSHIP ('ALL TO DETAIL', 'RELATIONSHIP',
#12620,
 #12640);
#12660 = PRODUCT_VERSION_WITH_SPECIFIED_SOURCE ('-', 'INITIAL RELEASE',
#12630,
 .NOT_KNOWN.);
#12670 = PRODUCT_DEFINITION_CONTEXT ('preliminary design', #12600,
 'for demo purposes');
#12680 = PRODUCT_DEFINITION ('DEFINITION OF NOT_KNOWN', #12660, #12670);
#12690 = PRODUCT_DEFINITION_SHAPE ('DESIGN SHAPE', 'SHAPE FOR
NOT_KNOWN',
 #12680);
#12700 = SHAPE_DEFINITION_REPRESENTATION (#12590, #12690);
ENDSEC;
END-ISO-10303-21;
```

APPENDIX 4

(CL file for an External and Internal feature)

-1.000000 -1.000000 2.180000
-0.125000 -0.025000 2.180000
0.000000 0.000000 2.180000
4.000000 0.000000 2.180000
4.000000 0.000000 3.180000
0.000000 0.200000 3.180000
0.000000 0.200000 2.180000
4.000000 0.200000 2.180000
4.000000 0.200000 3.180000
0.000000 0.400000 3.180000
0.000000 0.400000 2.180000
4.000000 0.400000 2.180000
4.000000 0.400000 3.180000
0.000000 0.600000 3.180000
0.000000 0.600000 2.180000
4.000000 0.600000 2.180000
4.000000 0.600000 3.180000
0.000000 0.800000 3.180000
0.000000 0.800000 2.180000
4.000000 0.800000 2.180000
4.000000 0.800000 3.180000
0.000000 1.000000 3.180000
0.000000 1.000000 2.180000
4.000000 1.000000 2.180000
4.000000 1.000000 3.180000
0.000000 1.200000 3.180000
0.000000 1.200000 2.180000
4.000000 1.200000 2.180000
4.000000 1.200000 3.180000
0.000000 1.400000 3.180000
0.000000 1.400000 2.180000
4.000000 1.400000 2.180000
4.000000 1.400000 3.180000
0.000000 1.600000 3.180000
0.000000 1.600000 2.180000
4.000000 1.600000 2.180000
4.000000 1.600000 3.180000
0.000000 1.800000 3.180000
0.000000 1.800000 2.180000
4.000000 1.800000 2.180000
4.000000 1.800000 3.180000
0.000000 2.000000 3.180000
0.000000 2.000000 2.180000
4.000000 2.000000 2.180000
4.000000 2.000000 3.180000
0.000000 2.200000 3.180000
0.000000 2.200000 2.180000
4.000000 2.200000 2.180000

4.000000 2.200000 3.180000
0.000000 2.400000 3.180000
0.000000 2.400000 2.180000
4.000000 2.400000 2.180000
4.000000 2.400000 3.180000
0.000000 2.600000 3.180000
0.000000 2.600000 2.180000
4.000000 2.600000 2.180000
4.000000 2.600000 3.180000
0.000000 2.800000 3.180000
0.000000 2.800000 2.180000
4.000000 2.800000 2.180000
4.000000 2.800000 3.180000
0.000000 3.000000 3.180000
0.000000 3.000000 2.180000
4.000000 3.000000 2.180000
4.000000 3.000000 3.180000
0.000000 3.200000 3.180000
0.000000 3.200000 2.180000
4.000000 3.200000 2.180000
4.000000 3.200000 3.180000
0.000000 3.400000 3.180000
0.000000 3.400000 2.180000
4.000000 3.400000 2.180000
4.000000 3.400000 3.180000
0.000000 3.600000 3.180000
0.000000 3.600000 2.180000
4.000000 3.600000 2.180000
4.000000 3.600000 3.180000
0.000000 3.800000 3.180000
0.000000 3.800000 2.180000
4.000000 3.800000 2.180000
4.000000 3.800000 3.180000
0.000000 4.000000 3.180000
0.000000 4.000000 2.180000
4.000000 4.000000 2.180000
4.000000 4.000000 3.180000
0.000000 4.200000 3.180000
0.000000 4.200000 2.180000
4.000000 4.200000 2.180000
4.000000 4.200000 3.180000
0.000000 4.400000 3.180000
0.000000 4.400000 2.180000
0.525000 4.400000 2.180000
0.525000 4.400000 3.180000
0.000000 4.600000 3.180000
0.000000 4.600000 2.180000
0.525000 4.600000 2.180000

0.525000 4.600000 3.180000
0.000000 4.800000 3.180000
0.000000 4.800000 2.180000
4.000000 4.800000 2.180000
4.000000 4.800000 3.180000
0.000000 5.000000 3.180000
0.000000 5.000000 2.180000
4.000000 5.000000 2.180000
4.000000 5.000000 3.180000
0.000000 5.200000 3.180000
0.000000 5.200000 2.180000
4.000000 5.200000 2.180000
4.000000 5.200000 3.180000
0.000000 5.400000 3.180000
0.000000 5.400000 2.180000
4.000000 5.400000 2.180000
4.000000 5.400000 3.180000
0.000000 5.600000 3.180000
0.000000 5.600000 2.180000
4.000000 5.600000 2.180000
4.000000 5.600000 3.180000
0.000000 5.800000 3.180000
0.000000 5.800000 2.180000
4.000000 5.800000 2.180000
4.000000 5.800000 3.180000
0.000000 6.000000 3.180000
0.000000 6.000000 2.180000
4.000000 6.000000 2.180000
4.000000 6.000000 3.180000
0.000000 6.200000 3.180000
0.600000 4.275000 3.180000
0.600000 4.275000 2.180000
1.000000 4.275000 2.180000
1.000000 4.275000 3.180000
1.100000 4.400000 3.180000
1.100000 4.400000 2.180000
1.100000 4.600000 2.180000
1.100000 4.600000 3.180000
0.600000 4.725000 3.180000
0.600000 4.725000 2.180000
1.000000 4.725000 2.180000
1.000000 4.725000 3.180000
1.462600 4.539800 4.000000
1.462600 4.539800 2.000000
1.670400 3.772400 2.000000
2.447300 3.980600 2.000000
2.238400 4.751800 2.000000
1.462600 4.539800 2.000000

1.660000 3.972000 2.000000
1.760000 3.972000 2.000000
1.860000 3.972000 2.000000
1.960000 3.972000 2.000000
2.060000 3.972000 2.000000
2.160000 3.972000 2.000000
2.260000 3.972000 2.000000
1.660000 4.072000 2.000000
1.760000 4.072000 2.000000
1.860000 4.072000 2.000000
1.960000 4.072000 2.000000
2.060000 4.072000 2.000000
2.160000 4.072000 2.000000
2.260000 4.072000 2.000000
1.660000 4.172000 2.000000
1.760000 4.172000 2.000000
1.860000 4.172000 2.000000
1.960000 4.172000 2.000000
2.060000 4.172000 2.000000
2.160000 4.172000 2.000000
2.260000 4.172000 2.000000
1.660000 4.272000 2.000000
1.760000 4.272000 2.000000
1.860000 4.272000 2.000000
1.960000 4.272000 2.000000
2.060000 4.272000 2.000000
2.160000 4.272000 2.000000
2.260000 4.272000 2.000000
1.660000 4.372000 2.000000
1.760000 4.372000 2.000000
1.860000 4.372000 2.000000
1.960000 4.372000 2.000000
2.060000 4.372000 2.000000
2.160000 4.372000 2.000000
2.260000 4.372000 2.000000
1.660000 4.472000 2.000000
1.760000 4.472000 2.000000
1.860000 4.472000 2.000000
1.960000 4.472000 2.000000
2.060000 4.472000 2.000000
2.160000 4.472000 2.000000
2.260000 4.472000 2.000000
1.660000 4.572000 2.000000
1.760000 4.572000 2.000000
1.860000 4.572000 2.000000
1.960000 4.572000 2.000000
2.060000 4.572000 2.000000
2.160000 4.572000 2.000000

2.260000 4.572000 2.000000
2.260000 4.572000 4.000000

APPENDIX 5

(NC code for an External and Internal feature)

```
N0 G70
N10 G94 F0.500000
N20 G96 F1000.000000
N30 G00 -1.000000 -1.000000 2.180000
N40 G01 -0.125000 -0.025000 2.180000
N50 G01 0.000000 0.000000 2.180000
N60 G01 4.000000 0.000000 2.180000
N70 G00 4.000000 0.000000 3.180000
N80 G00 0.000000 0.200000 3.180000
N90 G01 0.000000 0.200000 2.180000
N100 G01 4.000000 0.200000 2.180000
N110 G00 4.000000 0.200000 3.180000
N120 G00 0.000000 0.400000 3.180000
N130 G01 0.000000 0.400000 2.180000
N140 G01 4.000000 0.400000 2.180000
N150 G00 4.000000 0.400000 3.180000
N160 G00 0.000000 0.600000 3.180000
N170 G01 0.000000 0.600000 2.180000
N180 G01 4.000000 0.600000 2.180000
N190 G00 4.000000 0.600000 3.180000
N200 G00 0.000000 0.800000 3.180000
N210 G01 0.000000 0.800000 2.180000
N220 G01 4.000000 0.800000 2.180000
N230 G00 4.000000 0.800000 3.180000
N240 G00 0.000000 1.000000 3.180000
N250 G01 0.000000 1.000000 2.180000
N260 G01 4.000000 1.000000 2.180000
N270 G00 4.000000 1.000000 3.180000
N280 G00 0.000000 1.200000 3.180000
N290 G01 0.000000 1.200000 2.180000
N300 G01 4.000000 1.200000 2.180000
N310 G00 4.000000 1.200000 3.180000
N320 G00 0.000000 1.400000 3.180000
N330 G01 0.000000 1.400000 2.180000
N340 G01 4.000000 1.400000 2.180000
N350 G00 4.000000 1.400000 3.180000
N360 G00 0.000000 1.600000 3.180000
N370 G01 0.000000 1.600000 2.180000
N380 G01 4.000000 1.600000 2.180000
N390 G00 4.000000 1.600000 3.180000
N400 G00 0.000000 1.800000 3.180000
N410 G01 0.000000 1.800000 2.180000
N420 G01 4.000000 1.800000 2.180000
N430 G00 4.000000 1.800000 3.180000
N440 G00 0.000000 2.000000 3.180000
N450 G01 0.000000 2.000000 2.180000
N460 G01 4.000000 2.000000 2.180000
N470 G00 4.000000 2.000000 3.180000
```

N480 G00 0.000000 2.200000 3.180000
N490 G01 0.000000 2.200000 2.180000
N500 G01 4.000000 2.200000 2.180000
N510 G00 4.000000 2.200000 3.180000
N520 G00 0.000000 2.400000 3.180000
N530 G01 0.000000 2.400000 2.180000
N540 G01 4.000000 2.400000 2.180000
N550 G00 4.000000 2.400000 3.180000
N560 G00 0.000000 2.600000 3.180000
N570 G01 0.000000 2.600000 2.180000
N580 G01 4.000000 2.600000 2.180000
N590 G00 4.000000 2.600000 3.180000
N600 G00 0.000000 2.800000 3.180000
N610 G01 0.000000 2.800000 2.180000
N620 G01 4.000000 2.800000 2.180000
N630 G00 4.000000 2.800000 3.180000
N640 G00 0.000000 3.000000 3.180000
N650 G01 0.000000 3.000000 2.180000
N660 G01 4.000000 3.000000 2.180000
N670 G00 4.000000 3.000000 3.180000
N680 G00 0.000000 3.200000 3.180000
N690 G01 0.000000 3.200000 2.180000
N700 G01 4.000000 3.200000 2.180000
N710 G00 4.000000 3.200000 3.180000
N720 G00 0.000000 3.400000 3.180000
N730 G01 0.000000 3.400000 2.180000
N740 G01 4.000000 3.400000 2.180000
N750 G00 4.000000 3.400000 3.180000
N760 G00 0.000000 3.600000 3.180000
N770 G01 0.000000 3.600000 2.180000
N780 G01 4.000000 3.600000 2.180000
N790 G00 4.000000 3.600000 3.180000
N800 G00 0.000000 3.800000 3.180000
N810 G01 0.000000 3.800000 2.180000
N820 G01 4.000000 3.800000 2.180000
N830 G00 4.000000 3.800000 3.180000
N840 G00 0.000000 4.000000 3.180000
N850 G01 0.000000 4.000000 2.180000
N860 G01 4.000000 4.000000 2.180000
N870 G00 4.000000 4.000000 3.180000
N880 G00 0.000000 4.200000 3.180000
N890 G01 0.000000 4.200000 2.180000
N900 G01 4.000000 4.200000 2.180000
N910 G00 4.000000 4.200000 3.180000
N920 G00 0.000000 4.400000 3.180000
N930 G01 0.000000 4.400000 2.180000
N940 G01 0.525000 4.400000 2.180000
N950 G00 0.525000 4.400000 3.180000

```
N960 G00 0.000000 4.600000 3.180000
N970 G01 0.000000 4.600000 2.180000
N980 G01 0.525000 4.600000 2.180000
N990 G00 0.525000 4.600000 3.180000
N1000 G00 0.000000 4.800000 3.180000
N1010 G01 0.000000 4.800000 2.180000
N1020 G01 4.000000 4.800000 2.180000
N1030 G00 4.000000 4.800000 3.180000
N1040 G00 0.000000 5.000000 3.180000
N1050 G01 0.000000 5.000000 2.180000
N1060 G01 4.000000 5.000000 2.180000
N1070 G00 4.000000 5.000000 3.180000
N1080 G00 0.000000 5.200000 3.180000
N1090 G01 0.000000 5.200000 2.180000
N1100 G01 4.000000 5.200000 2.180000
N1110 G00 4.000000 5.200000 3.180000
N1120 G00 0.000000 5.400000 3.180000
N1130 G01 0.000000 5.400000 2.180000
N1140 G01 4.000000 5.400000 2.180000
N1150 G00 4.000000 5.400000 3.180000
N1160 G00 0.000000 5.600000 3.180000
N1170 G01 0.000000 5.600000 2.180000
N1180 G01 4.000000 5.600000 2.180000
N1190 G00 4.000000 5.600000 3.180000
N1200 G00 0.000000 5.800000 3.180000
N1210 G01 0.000000 5.800000 2.180000
N1220 G01 4.000000 5.800000 2.180000
N1230 G00 4.000000 5.800000 3.180000
N1240 G00 0.000000 6.000000 3.180000
N1250 G01 0.000000 6.000000 2.180000
N1260 G01 4.000000 6.000000 2.180000
N1270 G00 4.000000 6.000000 3.180000
N1280 G00 0.000000 6.200000 3.180000
N1290 G00 0.600000 4.275000 3.180000
N1300 G01 0.600000 4.275000 2.180000
N1310 G01 1.000000 4.275000 2.180000
N1320 G00 1.000000 4.275000 3.180000
N1330 G00 1.100000 4.400000 3.180000
N1340 G01 1.100000 4.400000 2.180000
N1350 G01 1.100000 4.600000 2.180000
N1360 G00 1.100000 4.600000 3.180000
N1370 G00 0.600000 4.725000 3.180000
N1380 G01 0.600000 4.725000 2.180000
N1390 G01 1.000000 4.725000 2.180000
N1400 G00 1.000000 4.725000 3.180000
 N0 G70
N10 G94 F0.500000
N20 G96 F500.000000
```

N30 G00 1.462600 4.539800 4.000000
N40 G00 1.462600 4.539800 2.000000
N50 G00 1.670400 3.772400 2.000000
N60 G00 2.447300 3.980600 2.000000
N70 G00 2.238400 4.751800 2.000000
N80 G00 1.462600 4.539800 2.000000
N90 G00 1.660000 3.972000 2.000000
N100 G00 1.760000 3.972000 2.000000
N110 G00 1.860000 3.972000 2.000000
N120 G00 1.960000 3.972000 2.000000
N130 G00 2.060000 3.972000 2.000000
N140 G00 2.160000 3.972000 2.000000
N150 G00 2.260000 3.972000 2.000000
N160 G00 1.660000 4.072000 2.000000
N170 G00 1.760000 4.072000 2.000000
N180 G00 1.860000 4.072000 2.000000
N190 G00 1.960000 4.072000 2.000000
N200 G00 2.060000 4.072000 2.000000
N210 G00 2.160000 4.072000 2.000000
N220 G00 2.260000 4.072000 2.000000
N230 G00 1.660000 4.172000 2.000000
N240 G00 1.760000 4.172000 2.000000
N250 G00 1.860000 4.172000 2.000000
N260 G00 1.960000 4.172000 2.000000
N270 G00 2.060000 4.172000 2.000000
N280 G00 2.160000 4.172000 2.000000
N290 G00 2.260000 4.172000 2.000000
N300 G00 1.660000 4.272000 2.000000
N310 G00 1.760000 4.272000 2.000000
N320 G00 1.860000 4.272000 2.000000
N330 G00 1.960000 4.272000 2.000000
N340 G00 2.060000 4.272000 2.000000
N350 G00 2.160000 4.272000 2.000000
N360 G00 2.260000 4.272000 2.000000
N370 G00 1.660000 4.372000 2.000000
N380 G00 1.760000 4.372000 2.000000
N390 G00 1.860000 4.372000 2.000000
N400 G00 1.960000 4.372000 2.000000
N410 G00 2.060000 4.372000 2.000000
N420 G00 2.160000 4.372000 2.000000
N430 G00 2.260000 4.372000 2.000000
N440 G00 1.660000 4.472000 2.000000
N450 G00 1.760000 4.472000 2.000000
N460 G00 1.860000 4.472000 2.000000
N470 G00 1.960000 4.472000 2.000000
N480 G00 2.060000 4.472000 2.000000
N490 G00 2.160000 4.472000 2.000000
N500 G00 2.260000 4.472000 2.000000

```
N510 G00 1.660000 4.572000 2.000000
N520 G00 1.760000 4.572000 2.000000
N530 G00 1.860000 4.572000 2.000000
N540 G00 1.960000 4.572000 2.000000
N550 G00 2.060000 4.572000 2.000000
N560 G00 2.160000 4.572000 2.000000
N570 G00 2.260000 4.572000 2.000000
N580 G00 2.260000 4.572000 4.000000
```