GRAPHICAL VERIFICATION OF MULTI-AXIS NUMERICALLY CONTROLLED MACHINING PROGRAMS FOR SCULPTURED SURFACE PARTS

By

Ki-Yin Chang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1991

ABSTRACT

GRAPHICAL VERIFICATION OF MULTI-AXIS NUMERICALLY CONTROLLED MACHINING PROGRAMS FOR SCULPTURED SURFACE PARTS

By

Ki-Yin Chang

Verification of multi-axis numerically controlled (NC) machining programs is a costly procedure in the manufacturing process, especially when one uses the side of the cutter to do milling operations. A new object-space-based (surface-based) verification system has been developed for multi-axis NC milling of sculptured surfaces. The dissertation describes an algorithm which discretizes the nominal sculptured surfaces and directly computes the possible interference between these surface points and the moving tool without explicitly creating the bounding surface of the tool motion (the tool swept volume). The geometric model uses the ruled surface defined by the axis of the cutting tool to define the center of the tool envelope. This result requires far less computation under typical conditions than would the use of direct solid modeling. In contrast to image-based methods, the algorithm is view independent (except for final graphical display), which means that displaying another view of the part does not require a rerun of the program, and, within broad limits, accurate displays of resulting gouging and undercutting with altered tolerances are possible without rerunning the verification algorithm. The approach utilizes positional (true position) tolerances on the desired part surfaces, and outputs a colored-coded display of the as-designed surfaces which depicts regions within tolerance, gouged, and undercut.

ACKNOWLEDGEMENTS

I would like to extend my special thanks and gratitude to Professor Erik D. Goodman, my major professor, for his constant guidance, support and willingness to share his time and knowledge through this research.

Special thanks are due to the other members of my Doctoral Guidance Committee for taking time to serve on the examining committee and observing this work.

Thanks also are given to my parents and my wife, Ayshiou, whose support during this study are greatly useful and inspirational.

Finally, sincere thanks should be conveyed to my friends, Leslie Hoppensteadt, Bao Ming, Brett Harper, and Jackie Carlson, for providing many sources of my ideas and being there when I need you.

TABLE OF CONTENTS

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Numerical control (NC) can be defined as a technique allowing automatic control of a machine tool with coded information which consists of numbers, letters and symbols. NC technology is used extensively in the metal-removing milling process, but it has also been applied to a wide variety of other machines and processes. Advanced NC systems such as computer numerical control (CNC) and direct numerical control (DNC), combined with other computer technology, opened the door for computer-aided manufacturing (CAM) to do automatic control of manufacturing processes and systems.

The automated NC machining process used to make contoured aircraft and automotive dies and parts has been applied in industry since the 1950's. The instructions for NC systems can be prepared either manually or with computer assistance, and instructions are often written using the APT (Automatically Programmed Tool) or Compact languages to relieve the part programmer from manually entering all of the geometric data. Among all the common NC languages, APT is the most widespread and the most comprehensive one. The program uses statements to identify the machine parameters, the tool shape and the part shape, then specifies the way the tool should move relative to various surfaces.

1.1 NC Milling Machines

Numerically controlled milling machines are used to progressively remove material from a workpiece. The tool used to do the milling is called a milling cutter, and is advanced at a slow rate of feed into the workpiece while the cutter (mill) rotates at relatively high speed. The material is removed in the form of small chips produced by each of the teeth of the milling cutter. There are may different types of milling cutters, such as the ball-end mill, the flat-end mill, the fillet-end mill, etc. The main purpose of the milling machine is to mill flat or contoured configurations. A machine tool is characterized by the motions it can perform. According to the changes of the relative position of the tool and workpiece allowed, as shown in Figure 1.1, it can be classified into the following different types of milling machine.

1. Two-axis Milling Machine. Two-axis milling indicates that the contouring capability of the machine tool is limited to motions of the X and Y axes (i.e., in a fixed Z plane). This mode of operation is frequently referred to as two-dimensional operation.

2. Three-axis Milling Machine. Three-axis milling refers to a cutting tool moving simultaneously in the X, Y and Z axes under complete control of the NC program. This type of NC machine tool is the most commonly used today for milling. The tool axis orientation of the three-axis milling machine does not change relative to the workpiece during the entire tool motion.

3. Multi-axis Milling. When more complicated parts with complex configurations are designed, it is often not enough to have the X, Y and Z movements alone; in addition, rotation about at least one of the axes, X, Y, or Z is needed. (Of course, the rotation about the cutter axis to perform the cutting action is not counted.) In the aerospace industry there has long been a need for machines that have 360^{o} contouring with simultaneous control of the Z axis; this usually is accomplished by a five-axis milling machine which has two rotary axes. The five-axis milling machine can continuously re-orient the axis of the tool as it

follows the contour of the part, or alternatively, re-orient the part. For the multi-axis milling process, the milling axis often changes with almost every motion of the mill.

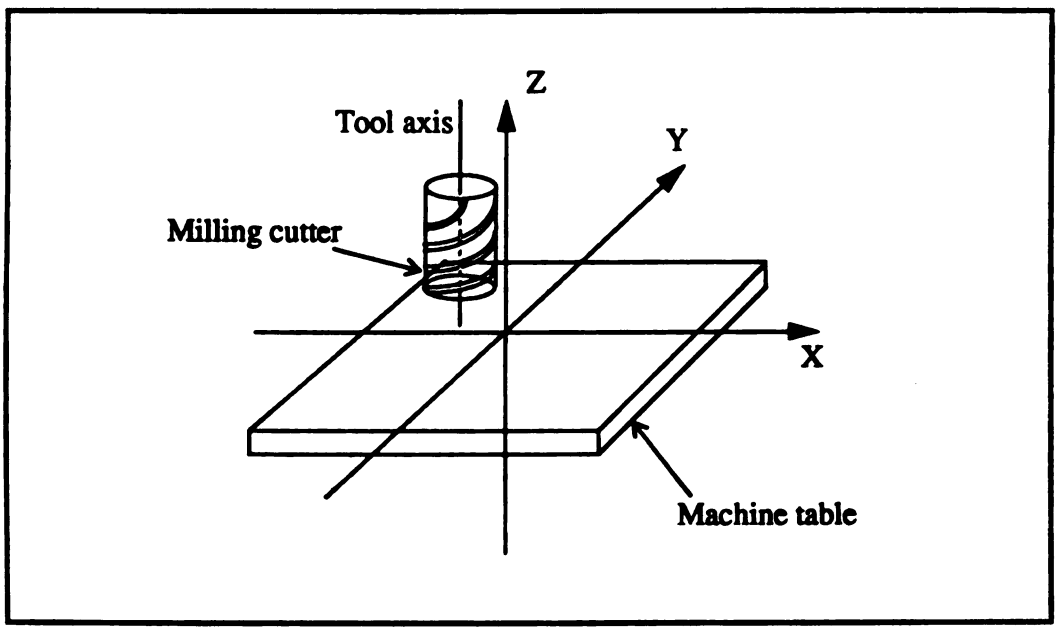

Figure 1.1 NC milling machine tool axis system

A three-dimensional surface can be formed by having a three-axis machine use a ball-end or fillet-end mill to form the contour, for instance, by making a series of small steps to approximate a twisted and warped surface. However, using a five-axis machine, if the side of the tool, say, must be tilted during the cut, the machine can make the sculptured surface to close tolerances with the side of the cutter in one pass and with an extremely satisfactory result. In addition, for high accuracy machining of sculptured surfaces, the five-axis milling machine is capable of operating the mill axis normal to the surface as the mill progresses along the surface, providing tight control of cusp heights.

1.2 NC Program Verification

For true position tolerancing, as is employed here, overcutting and undercutting of surfaces are relatively easy to define. Overcutting or gouging of a point on the part surface

happens when an NC cutting tool cuts beyond the tolerance limit inside the desired part surface [1][2]. Undercutting happens when an NC cutting tool does not cut the tolerance limit outside the desired part surface. If the cutting tool cuts between the tolerance limit outside the desired part and the tolerance limit inside the desired part surface, the point is said to be cut within tolerance. An example is shown in Figure 1.2. The machined surface point A is cut within tolerance, B and C are undercut, and D is gouged.

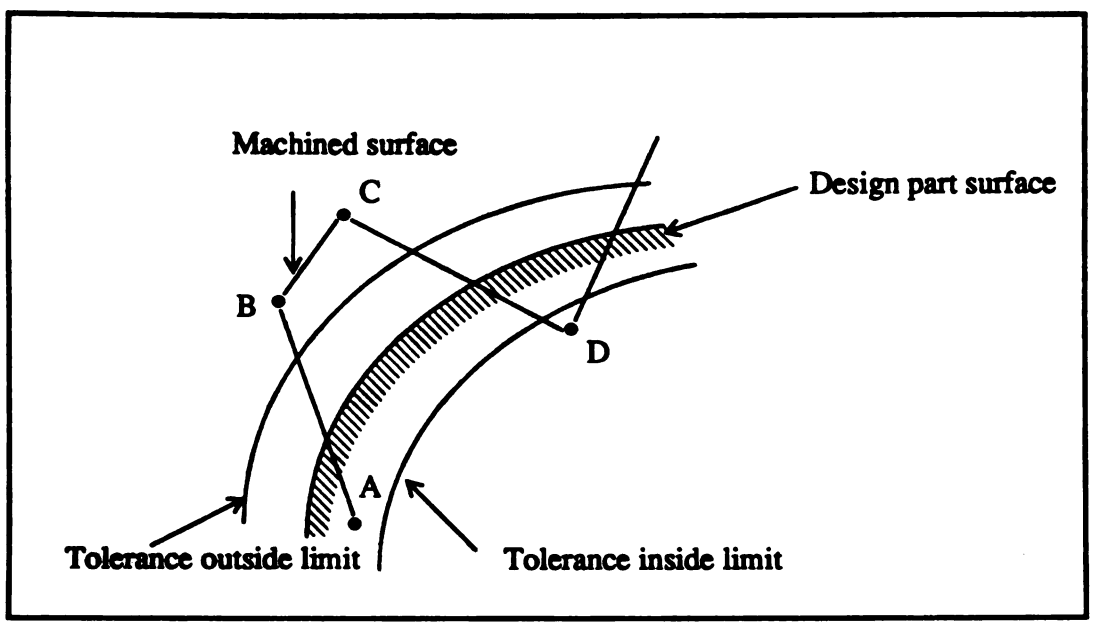

Figure 1.2 Tolerance for the NC machined surface

NC programs are commonly verified by milling a material softer than the actual desired surface, creating wooden, wax, plastic or foam prototypes. To reduce the cost of the process, verification of the NC program is sometimes used before these prototypes or the actual part material is machined. The verification of NC programs typically computes the difference between the design part model (*i.e.*, the model of the part as designed) and the workpiece part model (*i.e.*, the model of the part as it would be manufactured according to the NC program). Among all the surface types machined by NC programs, sculptured surfaces which are made up of arbitrary, nonanalytic contours comprise one of the most

challenging areas. In a NC machining process, it is easy to overcut the part surfaces. Therefore, how to avoid overcutting the part surface is one of the most crucial problems in NC machining of sculptured surfaces. Computer-assisted verification systems provide a process by which the errors between the desired part with the specified tolerances and the part as milled can be calculated and traced to specific tool motions. This process, using graphical computer models without expensive physical models, is a feasible and economical way to check the CL data (cutter location data).

In NC verification algorithms, the term 'cutter interference' is used to mean overcutting in a sculptured surface machining process [3]. A satisfactory handling of cutter interference requires not only removing overcutting, but also avoiding undercutting. Of course, during the "roughing" portion, undercutting is purposely programmed, but even there, verification software can be useful in pointing out the areas requiring additional cutting to achieve the finished part. The task of verifying an NC program for 3-D machining is tedious work. Recent research has tried to improve computational efficiency and reduce the running time with various approaches.

1.3 Objectives of the Study

Most NC verification software can model the three-axis NC milling process, and a few of them can model the five-axis NC milling process. Most of the current five-axis NC verification programs generate swept volumes using a geometric model. Boolean operations are calculated using a frame buffer. Therefore, the output display is view dependent, which means that if another view is needed, then the whole simulation must be rerun. It also cannot accurately provide true position tolerance calculations, because depths are calculated along sight lines of a fixed orientation. The above two disadvantages cause that type of verification algorithm not to be very effective for many real NC milling processes. This is because the five-axis milling process was originally designed for contoured surfaces and high precision processes. The objective of this study is to create a

geometric model and computational algorithm to do multi-axis NC verification and generate a view-independent data structure which enables a variety of final displays in a cost-effective way, while using true position tolerance for precise verification.

CHAPTER II

LITERATURE REVIEW OF NC VERIFICATION TECHNIQUES

In the last decade, many researchers have worked on NC verification and simulation using various approaches, such as solid modeling with Boolean operations, image-space methods, and object-space (surface-based) methods. An overview of these approaches is presented below.

2.1 Solid Modeling Techniques

Solid modeling is a useful approach to both simulation and verification. There are two notable schemes used in solid geometric modeling -- namely, constructive solid geometry (CSG) and boundary representation (B-rep) [4][5][6]. The CSG scheme is a constructive representation in which primitive solids such as cylinders, spheres, blocks, cones and other completely surface-bounded solids, appropriately positioned and combined via Boolean operations, are used to define an object of complicated shape. A tree structure with Boolean operators at the non-terminal nodes and primitives at the terminal nodes easily represents such a structure. The root node represents the complete solid. Therefore, a complicated solid can use the simple Boolean set operators to do the union, difference, intersection and assembly of the primitive solid. However, the CSG representation is an implicit representation, in the sense that the active regions bounding a complex solid are not represented explicitly in the data structure, but must be computed by means of the definitions of the primitives and the effects of the Boolean operations stored

in the tree structure. In contrast, in the B-rep scheme, a solid is viewed as bounded by the union of its bounding faces, for which the definitions are stored explicitly. A construction operation in a B-rep scheme uses the explicit representations of the boundaries of the solids and uses the topological relations among faces, edges, and vertices to evaluate and store the new boundary that is the result of the operation.

For a $2^1/_2$-axis flat-end mill motion, and using the CSG representation, the swept volume of a toolpath can be represented as a union of two cylinders and one block, as shown in Figure 2.1. Therefore, simulation can be done by sequentially subtracting models of the swept volumes of tool motions from the model of the workpiece [7][8]. Verification can be obtained by Boolean subtraction of the model of the workpiece from the desired part model. For a multi-axis tool motion, the simple primitive solids cannot represent a twisting tool motion. One needs either to generate more complex primitive solids for the complex multi-axis tool motion, or to approximate it as the union of a great many simpler primitives. Some modelers (such as I-DEAS from SDRC) are so-called *hybrid* modelers, using a representation containing both CSG and boundary constituents [9][10]. These can, of course, be used to implement the type of verification described above for CSG modelers.

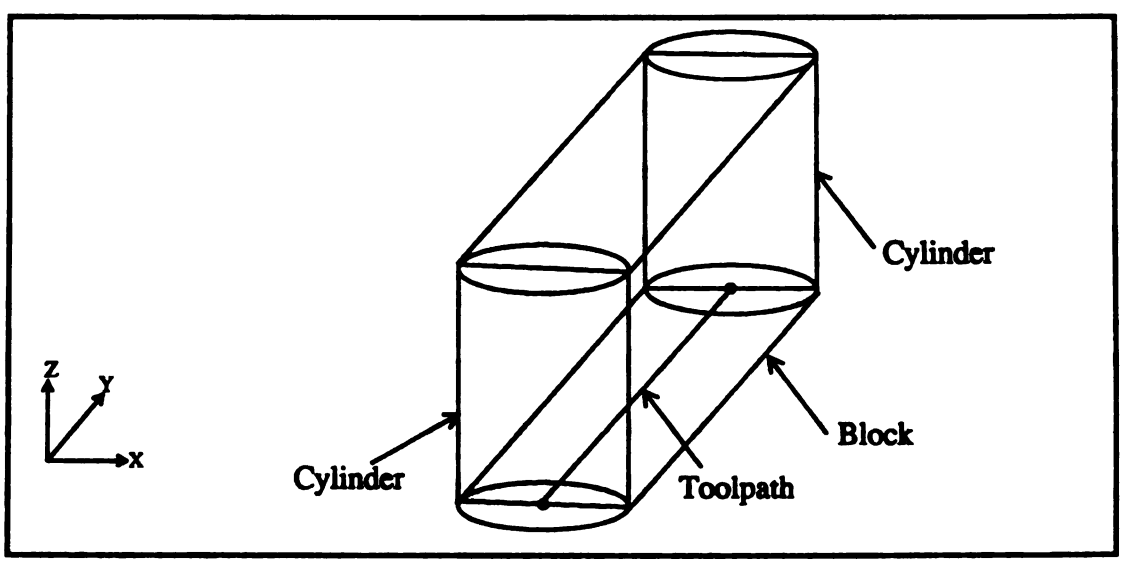

Figure 2.1 $2^1/_2$-axis swept volume representation in CSG scheme

Fridshal [11] enhanced and modified TIPS-1 [12], a solid modeling program, to do NC simulation. TIPS-1 does intersection calculations using a searching method based on a penalty function to find a series of points on the boundary of the intersection between two solids. These points can be interpolated as a spline. In GDTIPS, one generates the trajectory volume of the tool motion to form various NC trajectory volume primitives, and then subtracts the trajectory volumes from the workpiece and the holding fixture. Therefore, the "collision" of the tool, driven by the part program during the machining process, with the design geometry to be verified, is detected as verification proceeds. The problem with this approach is the large amount of computing resources it requires [13]. Five-axis swept volumes are composed of mathematical primitives which are extremely complex, and Boolean operations on them are very compute-intensive, thus reducing the cost-effectiveness of the technique.

2.2 Image-Space Approach

In the image-space method, Boolean operations are computed during image rendering. The three-dimensional Boolean operation could be reduced to a one-dimensional problem by considering the intersections of rays from each image pixel through a CSG solid model. Van Hook [14] developed an extended Z-buffer data structure called a dexel. The Z-buffer is a common approach for performing hidden surface removal for display of an interactive shaded solid, and contains a real number Z, or depth value, associated with each (X,Y) screen pixel. In contrast to a Z-buffer, each dexel contains not a single Z value, but several entries for each (X,Y) element: a pointer, a color, the Z value (depth) of the furthest surface, and the Z value of the nearest surface. The dexel structure is directly displayed just like a normal frame buffer, since the color of a dexel at the (X,Y) screen coordinate is the visible surface color at that pixel location. A pixel image of the cutting tool is precomputed. After scan conversion of a cutting tool, the tool is "moved" through the dexel structure, using Boolean operations along each ray and the stored depth

values to compare with the boundaries of the tool in each location along its path. It is performed like a CSG algorithm. The cutting tool is considered as a negative solid which is Boolean subtracted from the workpiece model (considered as a positive solid) as a program steps along a toolpath. After the tool is subtracted from the workpiece, the far surface of the tool typically becomes the new near surface of the block, and the inversely shaded tool color is the properly shaded new workpiece color. The mathematical model for the swept volume of the tool motion has not been described. Van Hook chose linear interpolation to divide the toolpath segments into typically 10~20 steps each, depending on the distance between the positions and a programmable interpolation tolerance. The selected view of the shaded image after verifying the milling path is easily displayed, but cannot be redisplayed from another view point without recomputing the entire problem. Also, the shaded image is an image-based model that does not provide tolerancing verification, and all distances are calculated along the view-dependent sight lines. The cutting function will not entirely replace object-space intersection calculations on general purpose computers, or test runs on the actual milling machine, because their algorithm for dexel-based computation is much less accurate than the vector/solid intersections employing surface normals. If a boundary representation of the tool motion is developed, it could be easily converted into a dexel structure. Then it would have a better result than the approximated boundary of the tool motion used by Van Hook, but would not resolve the problem that errors are measured along sight lines, rather than along surface normals.

The direct NC geometric verification technique for five-axis milling applications developed by Wang [13][15][16][17] was originally designed to create a solid model for a swept volume. The geometric model of boundary surface of the swept volume or the composition of the envelope surface is clearly described by its parametric form equation. The envelope consists of two categories of surface: (1) the subset of the boundary of the generator at the initial position and the final position, and (2) the new surfaces created by the generator during the motion. For example, in the swept volume of a cylindrical tool, two

side envelope faces are generated by a set of profile edges on the cylinder; the top and bottom envelope faces are generated by the points on the bounding edges of the cylinder. The method was later modified to perform graphical verification of NC toolpaths. This is a novel view-dependent method for five-axis NC verification. The algorithm uses standard Z-buffers and converts CSG part data into pixel data stored in the Z-buffer for subsequent sight line Boolean operations with other models. The input for modeling a toolpath swept volume includes cutter information and the CL data file, which consists of the tool control points and orientations. The swept volume is also converted into pixel data, which will be compared with those of the workpiece and fixtures. The Boolean subtraction removes the material from the workpiece. The interference between the tool swept volume and the workpiece or fixture can be shown using different colors to highlight various error situations. To deal with tolerance specification, the tool surface is offset. The result is approximately equivalent to the result of offsetting the part surface. He has implemented his geometric modeling scheme at the General Electric Company and has developed NC simulation software. Similarly, Saito and Takahashi [18][19] developed the G-buffer, another extension of the idea of a Z-buffer, and applied it to NC toolpath generation and verification using graphics or image processing hardware. This image-based method, as developed by Hook and Wang, is view dependent, which allows undetected errors or false indications of error because of the chosen viewing direction. Displaying another view of the part requires running the entire simulation again. In addition, the Z-buffer approaches are inherently limited in accuracy to the resolution of the Z-buffer, which is often only 16 or 24 bits. Thus, higher resolution is obtainable using the technique described here, particularly when the range of the depths to be represented is large. Many commercial milling verification programs, such as Vericut [20], are image-based algorithms.

2.3 Object-space Approach

In this approach, the verification is accomplished by calculating intersections of direction vectors with tool movement envelopes. The method can work for any part composed of a set of surfaces for which surface points and their corresponding normal vectors can be defined. That is, the data need not comprise a solid model. Chappel [21] described a method of using vectors to represent excess material removed by numerical control milling. The part surface is approximated by a set of points. This method gives a detailed description for calculating the intersection between a normal vector at a surface point and a randomly oriented cylinder cutter. This algorithm is possibly more efficient than solid modeling approaches since the intermediate simulation step is simplified considerably. The mathematical model derived in that paper is used to simulate cutting processes by moving the cylinder in discrete steps and calculating the intersection at each position. This technique also allows true position toleranced dimensional verification, which measures distances along part surface normal vectors to their intersections with the cylinder model. However, this algorithm is not very general because it treats only side surfaces of cylindrical cutters. For an end-mill cutter, to detect whether a part surface is machined by the bottom of the cutter or the cylindrical part of the cutter is a crucial problem for multi-axis tool motions. It is possible that in a multi-axis tool motion, the cylindrical surface of the cutter is cutting during part of the toolpath, and the flat bottom is cutting during another part. Both may be cutting simultaneously. Thus, computations for cutting of a normal vector of a point on a part surface must be made with both tool surfaces, and the correct (minimum) computation chosen for each portion of the toolpath. These are very common and important situations in verifications when one might have unexpected cutter interference [22][23]. Thus, this algorithm is not widely used.

Oliver and Goodman [24][25] developed an NC verification technique which extends Chappel's surface normal vector idea, but also incorporates solutions to the problems which limited Chappel's approach to relatively simple parts. The technique

involves intersection of surface normal vectors with milling tool swept volumes. This is a direct dimensional NC verification technique in which the minimum distance is computed along a surface vector to the swept volume of each tool motion. The algorithm first discretizes the desired part surface into surface points and their corresponding normals, using a point density determined by the screen resolution and current view of the part. Then, for each tool motion, it creates six planes which bound the tool swept volume for a primitive test of each surface point. If a surface point passes the primitive test -- that is, its normal vector intersects the bounding volume within a certain specified "range of interest" -- then the vector/solid intersection must be done. The swept volume is first approximated as a parallelepiped. This model is refined in areas where intersection is determined likely to occur by adding cylindrical and/ or spherical surfaces. Results of intermediate calculations are used to determine if further, more sophisticated, swept volume intersections are required. This ensures that redundant or superfluous calculation of vector/ solid intersection is minimized. The postprocessing procedure is also described in which the cut value and cosine of the normal vector with the sight line are used to assign hue and intensity to generate graphical output. This algorithm developed by them is a viable solution to the problem of accurate and efficient geometric verification of NC milling programs. It offers distinct advantages over the existing solid geometric modeling approaches. However, the entire portion of this algorithm dealing with mill axis space would probably not be useful for five-axis tool motion, since the five-axis tool axis changes at almost every motion of the mill.

Jerard [26][27][28] developed a surface-based technique for verification of NC programs used to machine sculptured surfaces. He chose a set of points on the surface of the object and used the sample as a discrete approximation to the actual surface. The surface points are calculated from parametric space. Later these surface data are placed into 2-D buckets with regular (x,y) spacing. Tool motion is projected onto the buckets. Only points in the buckets need to be examined. Inaccuracies in the cutting errors he calculates are

caused by three factors: 1) deviation between the actual surface and its polyhedral approximation, 2) the protrusion of the tool between the surface point because of the curvature of the tool and the non-zero distance between surface points, and 3) the fact that measurements are made along the axis of the tool, rather than normal to the part surface. The maximum error for this technique is the sum of the surface error, the protrusion error, and the error caused by the angle between the tool axis and the surface normal. When the method uses relatively few points for a given level of desired accuracy, it may indicate that some points are out of tolerance when in fact they are not. To resolve this problem, Jerard implemented a post-simulation analysis to calculate accurately the closest distance from the cut point to the desired surface. The system outputs a color-coded graphics display of the design surface which shows out-of-tolerance machined areas. This system can generate another view of a part without rerunning the simulation. To gain efficiency, surface curvature and cutting-tool size are used as inputs to a surface discretization algorithm, which guarantees that a user-defined level of simulation accuracy is achieved. The simulation time grows linearly with the number of tool motions and the number of surface points used. However, this method, like that of Oliver and Goodman, is only used for three-axis machining.

Fundamentally, direct solid modeling is expected to have more accurate representations of the milling process than the other techniques described, and to be able to do both verification and simulation. But the worst-case time complexity reported for direct solid modeling approaches, according to the report of Hunt[8] is $O(N^4)$, or $O(N^3 \log N)$ under certain special circumstances, where N is the number of the CL points in the toolpath. Since a multi-axis NC program contains many thousands of toolpath points, this can cause such algorithms to take many hours of execution time in a mainframe computer for some realistic verification problems. Image-space approaches take an advantage of specialized computer hardware (Z-buffers). They can use scan-line algorithms to do computation in

the frame buffer. This therefore reduces the execution time. But the resolution of the verification depends on the size of the screen pixels. View-dependent calculations and final display cause this algorithm to be unable to completely replace other approaches. The object-space (surface-based) algorithms discretize the surface points and compute the distance between the surface points and the tool swept volumes. The proper localization techniques for the surface points can reduce the time complexity to O(N) [24]. These algorithms can replace solid modeling in the verification of the NC milling process. However, prior object-space algorithms have been limited to three-axis NC milling. This dissertation presents such an algorithm to do multi-axis verification.

CHAPTER III

A GEOMETRIC MODEL FOR NC VERIFICATION

3.1 Historical Overview of Various Models for NC Verification

NC toolpath verification began with direct solid modeling approaches. NC verification via direct solid modeling is based on the principles of set theory, which provide that a simulated machined part can be represented by a series of Boolean subtractions of the swept volumes of the tool motions from the workpiece, or equivalently, by subtraction of the union of all of the swept volumes from the workpiece. The result of these operations is a simulated machined part, but not the solution to the verification problem, which seeks to show the discrepancies between the simulated machined part and the design part. Thus, to solve the verification problem, direct solid modeling needs to perform another subtraction to come up with the difference between the relatively simple design part model and the generally very complex simulated machined part. Of course, this difference is not necessarily in the form in which the user desires to see the discrepancies -- more likely, an identification of which toolpath segment was responsible for a gouge, for example, is desired. This would require still other computations than those generally performed by solid modeling systems. The use of this approach for solving problems with high complexity has been difficult, because such problems tend to require very intensive computations. Therefore, some researchers have sought to develop other techniques designed more specifically for solving this problem. For example, those using the "image

space" approach described in Chapter 2 employ hardware originally developed for surface shading to solve verification problems. The swept volume of the tool motion is converted by a scan line rendering processor into screen pixels which are comparable with the screen pixels of the workpiece and fixtures along a sight line. The Boolean subtraction is performed in the image space and is view-specific. The resolution for detecting errors is limited by the resolution of the view and the Z-buffer. The final verification produces the same type of result as did the direct solid modeling approach, but only for the chosen view. Disadvantages of this method were mentioned in Chapter 2.

This chapter introduces a different approach to geometric modeling of the NC verification process. Most of the current multi-axis NC verification systems need to create the boundary of the tool motion. The verification problem is handled only as a sort of post-process, beginning after a full simulation of the NC process is completed. But in the object-space algorithm described here, the verification is handled without many of the calculations required for a full simulation of the NC process. This is a real advantage, since multi-axis NC milling toolpaths often contain many thousands of CL points. Here we introduce a geometric model which can be considered to define the swept volume of a tool moving between CL points. However, the algorithm which uses this model detects interferences between sculptured surfaces, namely between part surfaces and tool swept volumes, without using Boolean operations. This chapter only discusses mathematical models and geometric models for the NC verification, leaving the discussion of methods for discretizing the part surface and localizing the surface points in 3-D space for the next chapter.

3.2 Characterization of Tool Motions

There are two types of cutter motions commonly employed in NC programming: point-to-point motion and contouring motion. Point-to-point motion is tool motion from the start point to the destination point, without specific requirements on the toolpath, and a

roughly linear motion over a fairly short segment is generally assumed, but not tightly specified. Contouring motion is motion of the tool along a designated path with, in the case of multi-axis toolpaths, designated orientations. Such complex tool motions are often reduced, instead, to a sequence of simpler, point-to-point motions, maintaining the deviation of the tool from the path within a specified tolerance of the given contour. Such conversion often occurs in either a postprocessor or a machine controller. For example, in APT contour motion, each step of the cutter motion can be defined by three surfaces: a drive surface, a part surface, and a check surface [29], as shown in Figure 3.1. A drive surface is a surface that guide the side of the cutter motion, and often does not represent any surface in the part model. A part surface is the surface that limits the tool motion along the tool axis or the depth of the cut. A check surface is the surface that defines the end point of the motion. Since these three must be defined for an APT contouring motion, some people use these three surfaces in implementing NC milling verification for this type of APT-specific code. But this approach turns out to be quite intractable for multi-axis tool motions. Alternatively, we will use a specification of the motion of the centerline of the tool to define the translation and rotation of the tool through 3-dimensional space. This is a key concept of the new algorithm developed here. If a cutter location file is specified in some manner other than the (x, y, z) for a control point on the tool and the cosines of the angles of the tool axis from the coordinate axes, then external software must be used to convert the CL file into this sort of representation, in order to use the algorithms described below.

3.3 Ruled Surface of A Tool Motion

An NC toolpath typically consists of tool axis statements and a list of tool positions and orientations which control a milling machine driving along the part surface. In a 3-axis NC machine tool, the tool moves from one position to another position and the orientation

of the tool axis does not change. For a multi-axis NC machine tool, the tool motion becomes very complicated because there is at least one rotary axis.

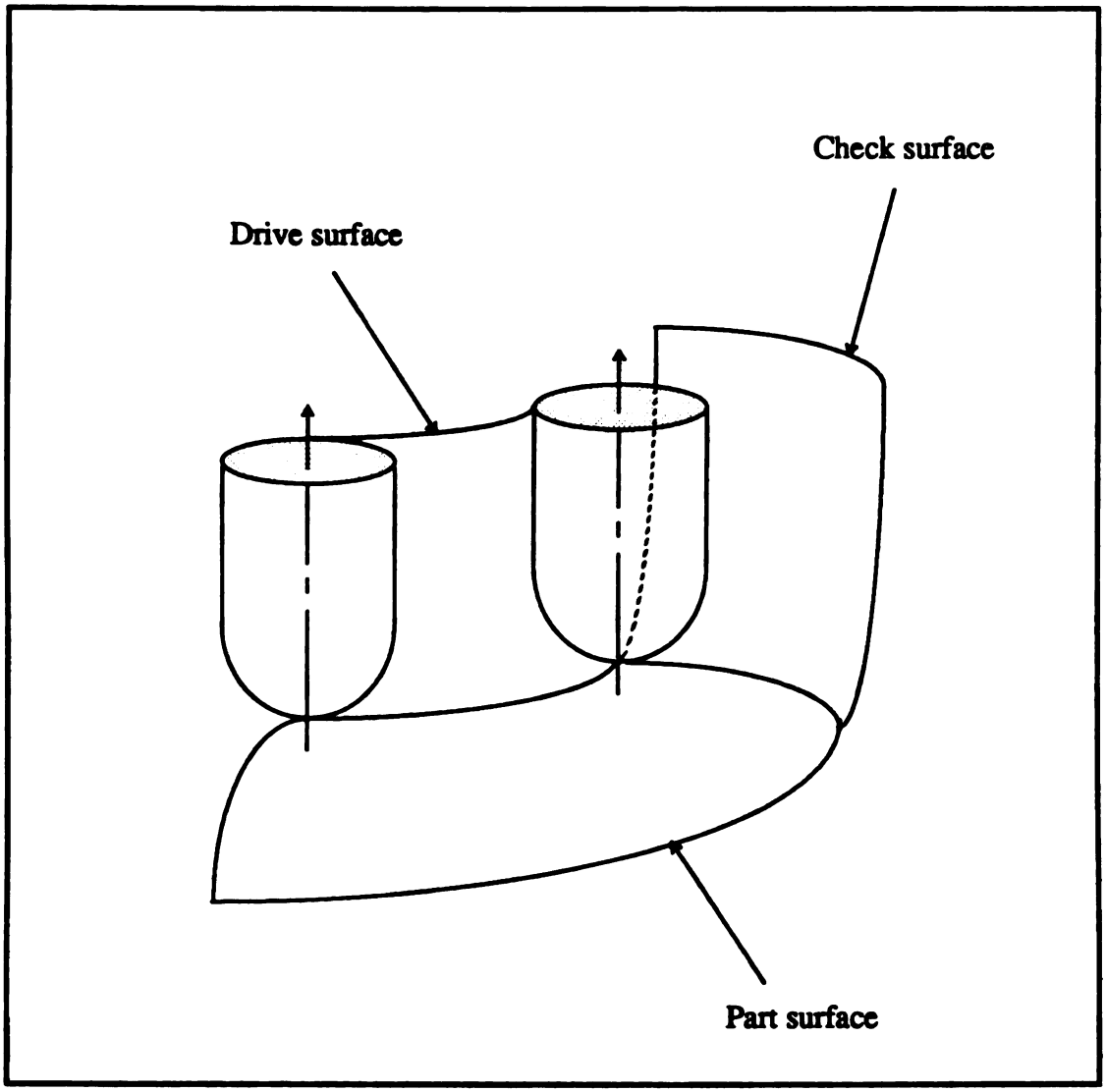

Figure 3.1 Three surfaces in APT contouring motions

The analysis of the geometric model presented here does not include any effects of tool deflection, tool wear, or part distortion. However, the model is at least theoretically extensible to represent such effects, although with a very significant time penalty for computations.

In order to analyze the problem, certain properties of the cutter are assumed:

1. For the purposes of the verification, the cutter may be represented as a (symmetrical) surface of revolution, which means that the cross section at any specific position along the cutter axis has a given radius. Of course, this is not actually true of milling cutters -- they have teeth. In order to make the assumption tenable, the speed of revolution of the cutter relative to its feed rate must be large, so that the effects of the individual teeth are not large relative to the tolerances being checked. This is the intent in milling operations, in general, so does not cause a particular hardship for the verification process. However, if a soft material were cut with a high feed rate, it is conceivable that this assumption could be violated to such an extent that this verification process, and any others we have discussed, would not detect some out of tolerance areas.
2. The envelope of the cutter can be expressed as a function of position (height) along the cutter axis. This is true of any 7-parameter APT cutter except one with a flat bottom. A flat bottom has a radius, but the cutter envelope is actually the plane bounded by the circle of radius R(0). However, this verification procedure handles such a flat-bottom cutter, as a special case, and also provides a flat "top" surface for all cutters.
3. Each step of the tool motion is continuous.

The motion of the tool is defined by the control point $\overline{C}(t)$ and the orientation of the tool axis (axis of revolution) $\overline{N}(t)$. The trajectory of the control point usually is rendered by linear interpolation between the previous position and the current position [15]. A series of linear motions is also frequently used to approximate more complex trajectories, such as circular motion between CL points, by using a chordal tolerance to determine the step sizes for a series of short linear segments. This process can occur in either postprocessors or machine controllers.

There are a variety of possible ways to interpolate the orientation of the tool axis between the orientations specified in successive CL points. The scheme used here, and in many other analyses, to interpolate the orientation of the tool axis (axis of revolution) is that the previous tool axis and current tool axis are treated as two vectors on a unit sphere [15], and interpolation is conducted along the great circle joining them. Other axis interpolation models would require revision of the mathematical description given below (see Equation 3.1) of the ruled surface $\bar{r}(t,h)$. Alternatively, a user could, for purposes of verification, approximate the behavior of an arbitrary machine simply by generating a higher density of CL points (each with its specified axis orientation), thereby making the error caused by a different interpolation scheme become arbitrarily small.

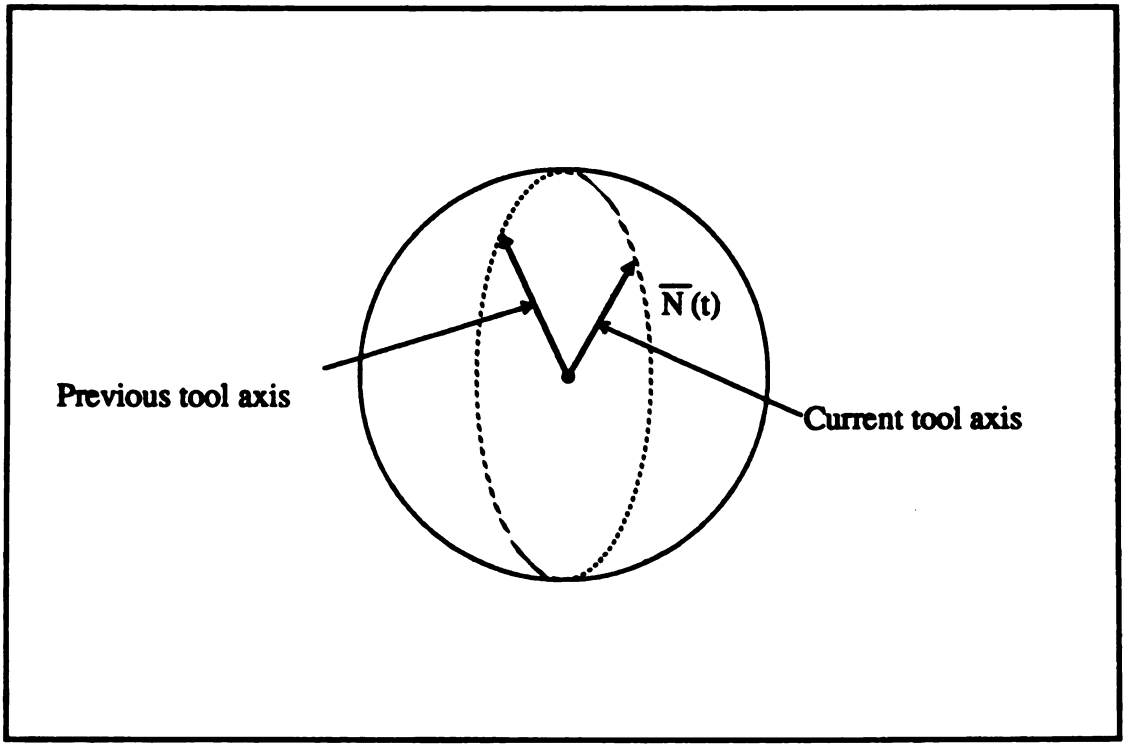

Figure 3.2 The orientation of the tool axis $\overline{N}(t)$

Thus, for this procedure, the vector function $\overline{N}(t)$ for tool orientation is defined by an arc on the great circle of a unit sphere passing through the two vectors, with motion at a constant velocity, as shown in Figure 3.2. The unit sphere is determined by normalizing the vectors and translating one of the vectors so that the two have a common origin (at the center of the unit sphere). However, as noted above, the actual interpolation for the motion of the tool axis relies on the NC controller and the postprocessor. If the function for the motion of the tool axis were known to be different from the great circle function used below, then $\overline{N}(t)$ could be specified without approximation.

The locus of the tool centerline moving with one degree of freedom during a multi-axis tool move constitutes a ruled surface [30][31]. For a given $\overline{C}(t) = ((x(t), y(t), z(t))$ and $\overline{N}(t) = (N_X(t), N_Y(t), N_Z(t))$ in Cartesian coordinates, the ruled surface for a tool motion can be parameterized as:

$$\begin{aligned}\overline{r}(t, h) &= \overline{C}(t) + h\,\overline{N}(t) \\ &= [x(t) + h\,N_X(t)]\,\vec{i} + [y(t) + h\,N_Y(t)]\,\vec{j} + [z(t) + h\,N_Z(t)]\,\vec{k} \qquad (3.1)\end{aligned}$$

where $0 \le h \le L$, $0 \le t \le 1$, and L is the cutter height from the control point of the cutter.

For a general APT (Automatically Programmed Tool) cutter (see Appendix A), the profile at any specific position along the cutter axis has a given radius. The radius of the cutter can be expressed as a function $R(h)$ of position (h = height) along the cutter axis. The function $R(h)$ for a general APT cutter can have up to three different functional forms for subintervals of $0 \le h \le L$.

Two methods are described below to deal with points on part surfaces, or on offset surfaces, as described also by Jerard[27]:

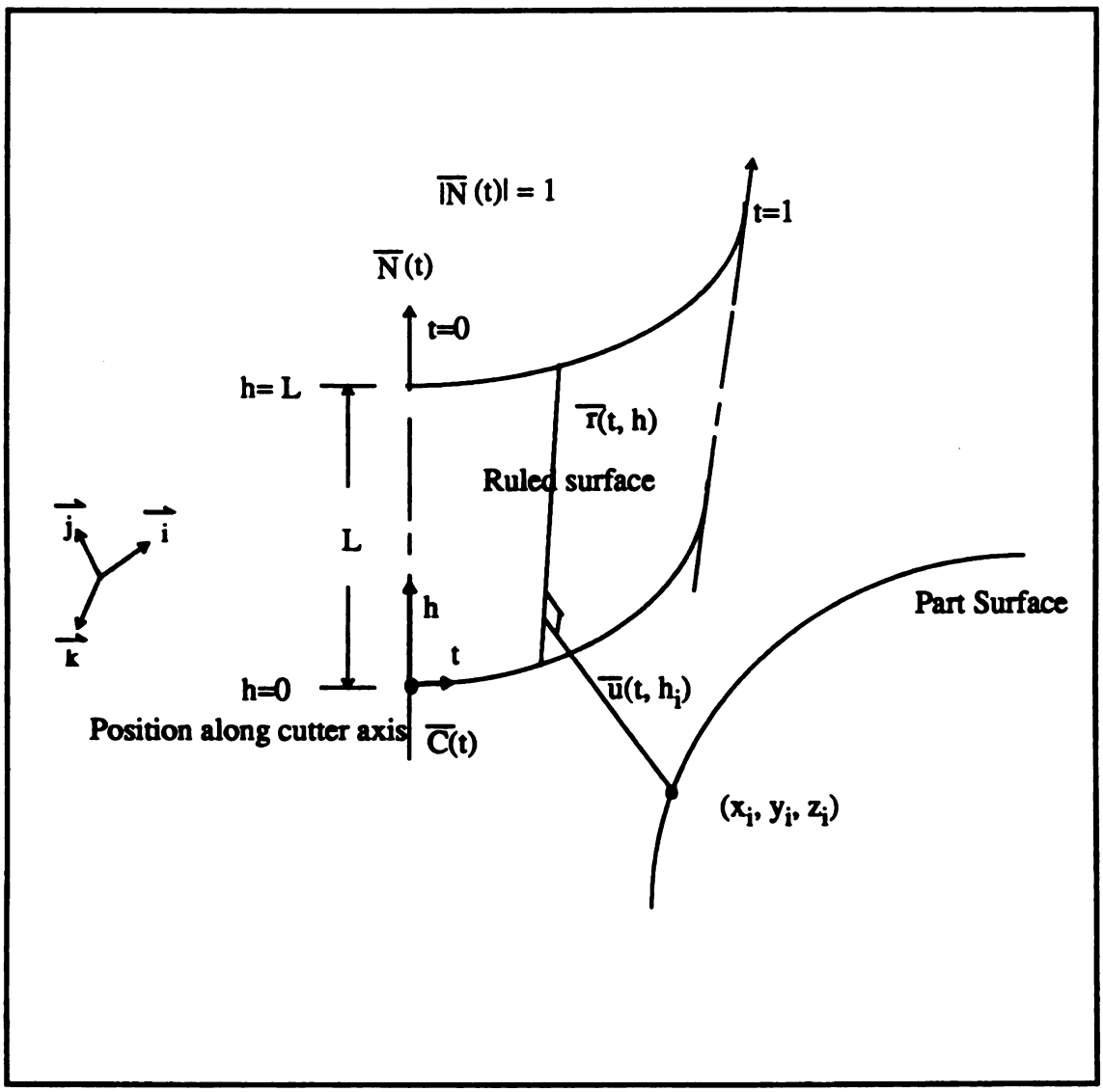

Figure 3.3 Ruled surface and potential connecting vectors $\overline{u}(t, h_i)$ of a toolpath segment

1. Ignore the specific normal direction at each surface point, which means that one does not consider the normal vectors of the surface at all. This method can be used, for example, for interference detection between a surface point and the envelope of the tool motion, and will be described in section 3.3.

2. Evaluate the surface point and the outward unit normal vector [21] [23], then calculate the cut value along this outward unit normal vector. This will be discussed in Appendix B.

3.4 Determination of whether or not a point is inside the envelope of the tool motion

In this section, a mathematical model will be presented for checking whether or not a point is inside the envelope of a specified tool motion. There are two parametric variables for the ruled surface. For any point on the part surface, its relationship with a ruled surface defined in terms of these two parametric variables can be found through the locus of potential connecting vectors. Assume $\overline{u}(t, h_i(t))$ is the locus of potential connecting vectors between a point (x_i, y_i, z_i) and a parametric ruled surface. By the properties of a connecting vector, it is the shortest vector connecting a point (x, y, z) to the tool axis $r(t_j,h)$ at any time t_j, $0 \leq t_j \leq 1$, and it is perpendicular to the line $r(t_j, h)$. For the specific point (x_i, y_i, z_i), the locus of connecting vectors is specified as:

$$\overline{u}(t, h_i(t)) = (x_i, y_i, z_i) - \overline{r}(t, h_i(t)) \text{ or}$$

$$\overline{u}(t, h_i(t)) = [x_i - x(t) - h_i(t)N_X(t)]\vec{i} + [y_i - y(t) - h_i(t)N_Y(t)]\vec{j} + [z_i - z(t) - h_i(t)N_Z(t)]\vec{k}$$

For a given point, the local coordinate (the closest point) on an infinite ruled surface must lie on a connecting vector $\overline{u}(t, h_i(t))$ which by definition must be perpendicular to the orientation of the tool axis $\overline{N}(t)$.

Thus, $\overline{u}(t, h_i(t)) \cdot \overline{N}(t) = 0$, and substituting for $\overline{u}(t, h_i(t))$ from the previous equation, we have:

$$[x_i - x(t) - h_i(t)N_X(t)]N_X(t) + [y_i - y(t) - h_i(t)N_Y(t)]N_Y(t) + [z_i - z(t) - h_i(t)N_Z(t)]N_Z(t) = 0$$

$$[x_i - x(t)]N_X(t) + [y_i - y(t)]N_Y(t) + [z_i - z(t)]N_Z(t) = h_i(t)\,[N_X^2(t) + N_Y^2(t) + N_Z^2(t)]$$

$$h_i(t) = [x_i - x(t)]N_X(t) + [y_i - y(t)]N_Y(t) + [z_i - z(t)]N_Z(t) \qquad (3.2)$$

The physical meaning of equation (3.2) is that every point in space has a corresponding value $h_i(t)$ for any moment t representing the closest position on the cutter axis for a particular tool motion. The value $h_i(t)$ might not be in the range of [0, L]. (Note that this point does not necessarily correspond to the closest point on the tool at that time, but to the closest position on the tool axis.)

For a given point (x_i, y_i, z_i) and a tool motion $(\overline{C}(t), \overline{N}(t))$, $0 \le t \le 1$, a determination whether or not a point is inside the envelope of the tool motion can be made from the following expression:

The point (x_i, y_i, z_i) has local coordinates $(t, h_i(t))$, *i.e.*, it is closest to the ruled surface at the point $\overline{r}(t, h_i(t))$, and the minimum distance between the point (x_i, y_i, z_i) and the ruled surface is the difference between the point and the ruled surface point $\overline{r}(t, h_i(t))$. So if $0 \le h_i(t) \le L$, that is, if the local h coordinate of the point falls within the length bounds of the cutter, then whether a point is inside the tool motion can be simplified to whether the point is inside a circle with center $\overline{r}(t, h_i(t)) = ([x(t) + h_i(t)N_X(t)], [y(t) + h_i(t)N_Y(t)], [z(t) + h_i(t)N_Z(t)])$ and radius $R(h_i(t))$.

The requirement that the point (x_i, y_i, z_i) be within the circle can be written as:

$$[x_i - x(t) - h_i(t)N_X(t)]^2 + [y_i - y(t) - h_i(t)N_X(t)]^2 + [z_i - z(t) - h_i(t)N_Z(t)]^2 \le R^2(h_i(t))$$

Expanding the above equation yields:

$$[x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - 2h_i(t)\{[x_i - x(t)]N_X(t) + [y_i - y(t)]N_Y(t) + [z_i - z(t)]N_Z(t)\} + h_i^2(t)[N_X^2(t) + N_Y^2(t) + N_Z^2(t)] \leq R^2(h_i(t))$$

Then, applying Equation 3.2, and because $\overline{N}$ is a unit vector:

$$[x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - 2h_i(t)h_i(t) + h_i^2(t) \leq R^2(h_i(t))$$

$$[x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \leq 0 \qquad (3.3)$$

That is, subject to the constraints, Equation 3.3 provides the condition that a point be contained within the envelope of a tool motion. This can be expressed as a simple optimization problem with one variable.

$$\begin{aligned} &\text{minimize:} \quad \underset{t \in R}{f(t)} = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \\ &\text{subject to :} \\ &\qquad 0 \leq t \leq 1 \\ &\qquad 0 \leq h_i(t) \leq L \end{aligned} \qquad (3.4)$$

If f(t) is less than or equal to 0, we can say that this surface point is inside or on the envelope of the tool motion at t or the surface point is inside the swept volume of the tool motion at t.

Let us consider two examples of familiar cutters. The first example is a flat-end cutter. Let the control point $\overline{C}(t)$ of the tool motion be specified as the center of the bottom of the cutter as shown in Figure 3.4. Its mathematical model can be expressed as follows:

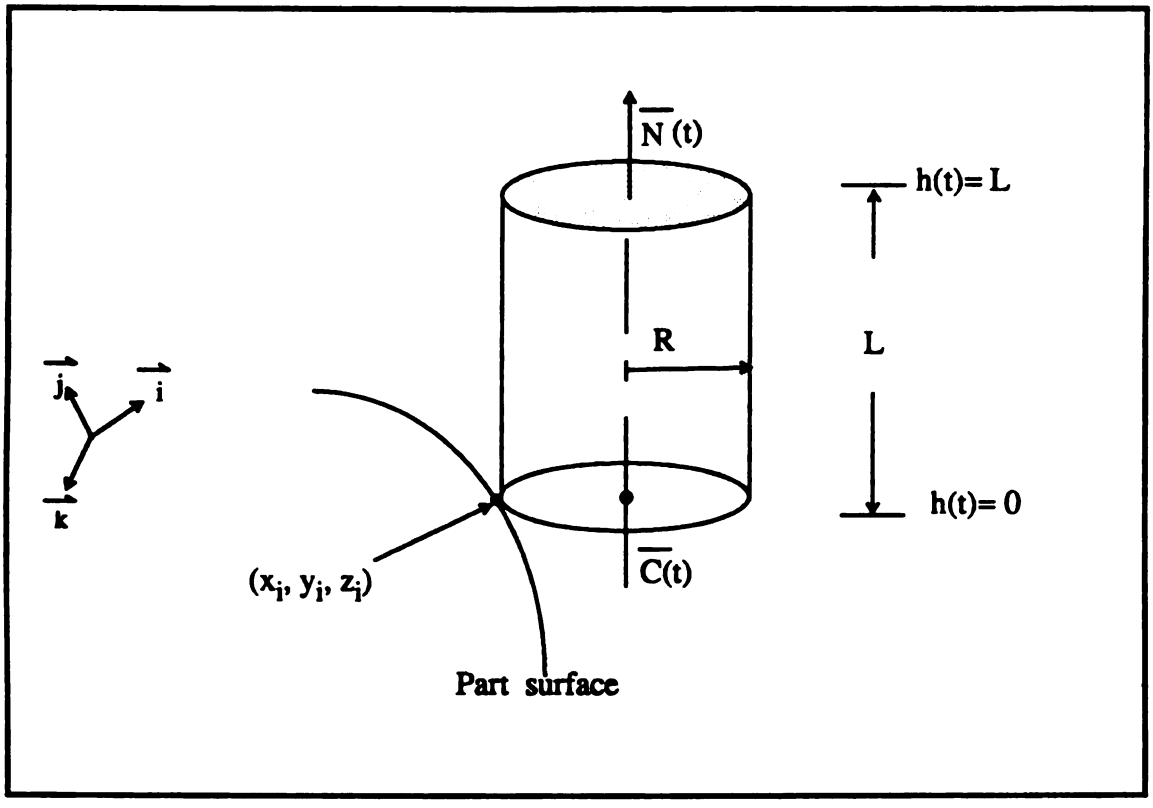

Figure 3.4 Interference detection for a flat-end cutter

Given a tool motion $(\overline{C}(t), \overline{N}(t))$ and a point (x_i, y_i, z_i),

minimize: $\underset{t \in R}{f(t)} = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2$

subject to :

$$0 \leq t \leq 1$$

$$0 \leq h_i(t) \leq L$$

If f(t) is less than or equal to 0, then the surface point is inside or on the envelope of the tool motion.

Let us consider another example -- that of the familiar ball-end cutter. Its mathematical model can be specified as follows, given a definition of the control point of the tool motion be specified as the center of the spherical cutter as shown in Figure 3.5:

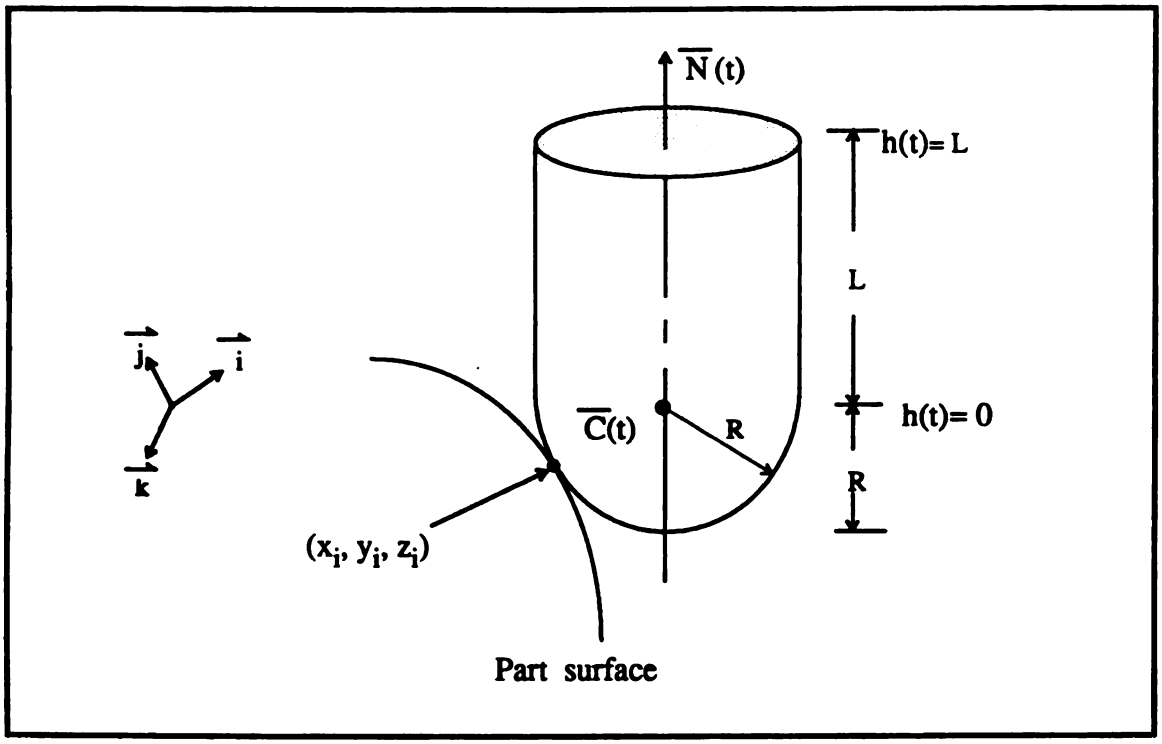

Figure 3.5 Interference detection for a ball-end cutter

Given a tool motion $(\overline{C}(t), \overline{N}(t))$ and point (x_i, y_i, z_i),

$$\text{minimize:} \quad \underset{t \in R}{f(t)} = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2$$

subject to :

$$0 \le t \le 1$$

$$0 < h_i(t) \le L$$

and

$$\text{minimize:} \quad \underset{t \in R}{f(t)} = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - R^2$$

subject to :

$$0 \le t \le 1$$

$$-R \le h_i(t) \le 0$$

If either f(t) is less than or equal to 0, then the surface point is inside or on the envelope of the tool motion.

This interference detection algorithm is developed based on the ruled surface. One important assumption in developing the algorithm is that the cutter is a symmetric surface of revolution. Currently the APT cutter only supports convex cutters. However, the algorithm can also be implemented if the cutter is concave, and even if the tool swept volume is self-intersecting.

Implementation issues such as tolerance specification for NC verification using this algorithm will be discussed in later chapters. The remainder of the body of this dissertation will present various methods for applying the interference calculation of a point in space with the envelope of the cutter to the task of NC verification. A geometric model for vector/ solid intersection which evaluates the cut value of a surface point along its outward unit normal vector, which is used by some others [25] for verification, is discussed in Appendix B. However, the cut value, or depth of cut, which is the minimum distance from the surface point along its surface normal vector to the moving tool, (equation B.5) cannot be solved with a simple interpolation method, since s(t) may have more than two minimum solutions under some conditions. Thus, exhaustive search is necessary to find a minimum cut value in equation B.5. Therefore, despite its more natural direct relationship to cut values, whether or not they are close to the tolerance specified, this algorithm was not implemented as part of this work, for reasons of computational efficiency.

CHAPTER IV

NC VERIFICATION PROCEDURE

There are several steps in implementing NC verification. In the NC verification software described here, one must load surface data, cutter information, and CL data, and must discretize the surface before doing the verification. This chapter will introduce the necessary input data and procedures of the NC verification software.

4.1 Workpiece Surface Discretization

The desired part surfaces and associated holding fixtures are read as rational (nonuniform or uniform) B-spline surfaces, in their Initial Graphics Exchange Specification (IGES) standard formats. This type of surface is very commonly used by CAD/CAM solid modelers, since the first derivative and second derivative are continuous on the B-spline surface, and many common analytical surfaces have exact representations using NURBS (Non-Uniform Rational B-Spline) surfaces. Sculptured surfaces allow accurate representation of complex surfaces, and allow interpolating techniques to meet a variety of design criteria. The desired part surfaces are usually directly produced by a CAD modeling package, such as the I-DEAS^{TM} software from Structural Dynamics Research Corporation (SDRC). The technique used to discretize untrimmed NURBS surfaces was developed by other researchers in the Case Center for Computer-Aided Engineering and Manufacturing (Correns [32]) to evaluate NURBS surfaces. The software can read and discretize untrimmed NURBS (IGES type 128), trimmed NURBS (IGES type 144) and

bounded NURBS (IGES type 143). Trim boundaries may currently include NURBS curves, composite curves, or copious data types.

In this approach, the first step is to discretize a workpiece and associated holding fixtures into a set of points, depending on the size and curvature of the surface. This data structure is then used as a discrete approximation to the actual surface. This object-space-based method shares some characteristics with the methods of Oliver [24] and Jerard [27] -- specifically, the intersection of a moving tool with vectors passing through points on the surface of the desired part. However, it differs in the method of surface discretization from the methods of Oliver and Jerard (and differs in its core simulation method, as well). The method presented is general, and can work for any set of surfaces on which surface points and normal vectors can be defined. Each surface is discretized into a triangular grid of points, in which the resolution depends on user-entered values for the smaller of maximum distances between points (in the s and t parametric directions) and maximum allowable chordal deviation [32 - 36] (in both s and t directions).

4.1.1 Chordal Deviation and Subdivision of Curves between Two Points

The maximum chordal deviation is a vital factor determining the accuracy of the NC verification. If the surfaces are not discretized sufficiently finely, it is likely that areas overcut or undercut between the discrete points can go undetected, if the overcut or undercut region does not include adjacent discrete points. The probability of this depends not only on the surface shape, but also on the tool geometry. As the maximum chordal deviation grows, relative to the allowed tolerance limit, the number of tolerance violations undetected in the verification process tends to increase. Therefore, the chordal deviation can be considered to be a practical lower bound on the accuracy of the verification process, as illustrated in Figure 4.1. When tolerances to be verified are much smaller than the chordal tolerance selected, some tolerance violations are likely to go undetected. (To save

time in the discretization process, squares of the distance are compared with the square of the criterion.)

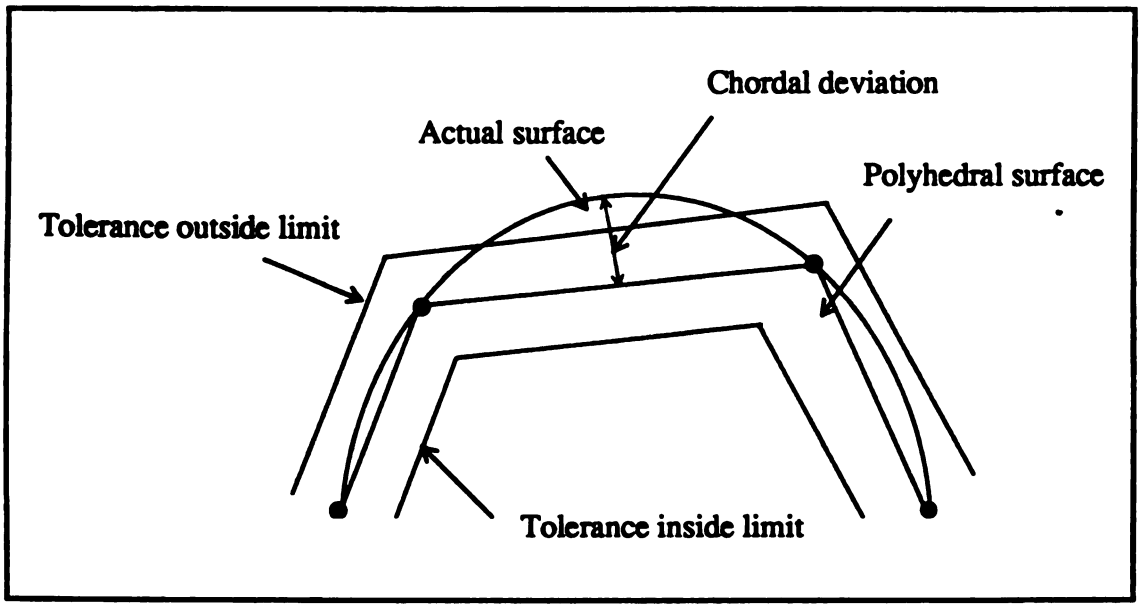

Figure 4.1 The chordal deviation selected limits the potential accuracy of the verification process.

The chordal deviation is approximated as the distance between the geometric midpoint P_m of two points and the point P as shown in Figure 4.2 :

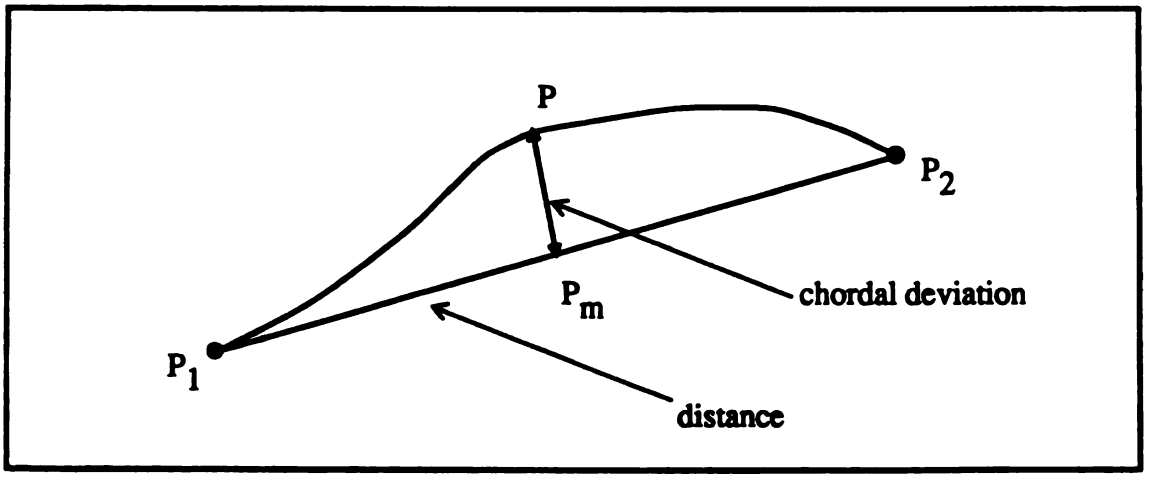

Figure 4.2 Chordal deviation of two surface points

The points are evaluated either in the s or t parametric direction. Therefore, one of the parameters is constant. For fixed s, for example, the value P is evaluated as $P(s, (t_1 + t_2)/2)$. This may not lie on the normal through P_m, but this fact is ignored. To speed up the program, two similar algorithms for s and t parametric directions are employed. The algorithm used to calculate the square of the chordal deviation in the s direction is shown below:

set $s = s_1$

set $t = (t_1 + t_2) / 2$

evaluate point P (s , t)

$$\text{chordal deviation} = (P_x - (P_{1,x} + P_{2,x}) / 2)^2 + (P_y - (P_{1,y} + P_{2,y}) / 2)^2 + (P_z - (P_{1,z} + P_{2,z}) / 2)^2$$

If the distance between two points or the chordal deviation of two points P_1 and P_2 exceed the user-entered maxima, the curve needs to be subdivided. A new point P is created with

$$s_m = (s_1 + s_2) / 2$$

or

$$t_m = (t_1 + t_2) / 2$$

depending on the direction of the evaluation. This is repeated until the user's specifications are met.

The points and triangles created in this process are stored in linked-list structures shown in Figure 4.3, which defines structures SURFACE, TRIANGLE, POINT, and TRI_LIST. The triangles are used to display the surface graphically. The accuracy of the NC verification algorithm depends, of course, on the level of refinement of the surface discretization. Verification error is caused by deviation between the actual surface and the

triangle approximation (the deviation is related to the chordal deviation specified in this approach). A data structure is created containing the (x, y, z) coordinate of each point, the outward normal unit vector, and minimum distance from the surface point to the boundary of the tool motion which cuts this vector most deeply to date (initialized to a large value).

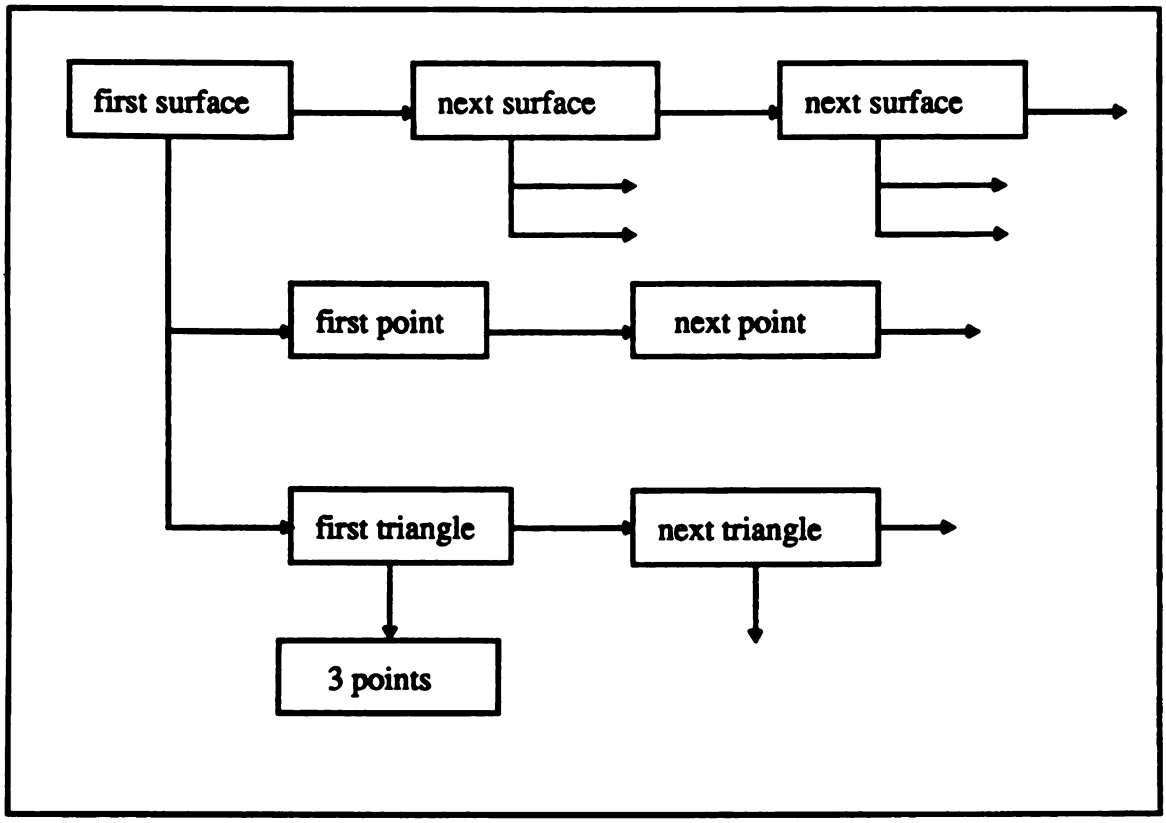

Figure 4.3 Data structure of surfaces

The sculptured surfaces needed for verification are usually only some of the boundaries of a solid, and some modelers do not assure a consistent orientation of surface normals for their boundary representations. Of course, the surfaces might not have come from a solid modeler at all. Therefore, since the software now implemented uses the surface normal direction of the workpiece model, the software also provides visualization tools which allow the user to reorient any surfaces which are not oriented properly. It is necessary to obtain surface normals which are all outward-directed before the verification is initiated.

4.1.2 Spatial Subdivision

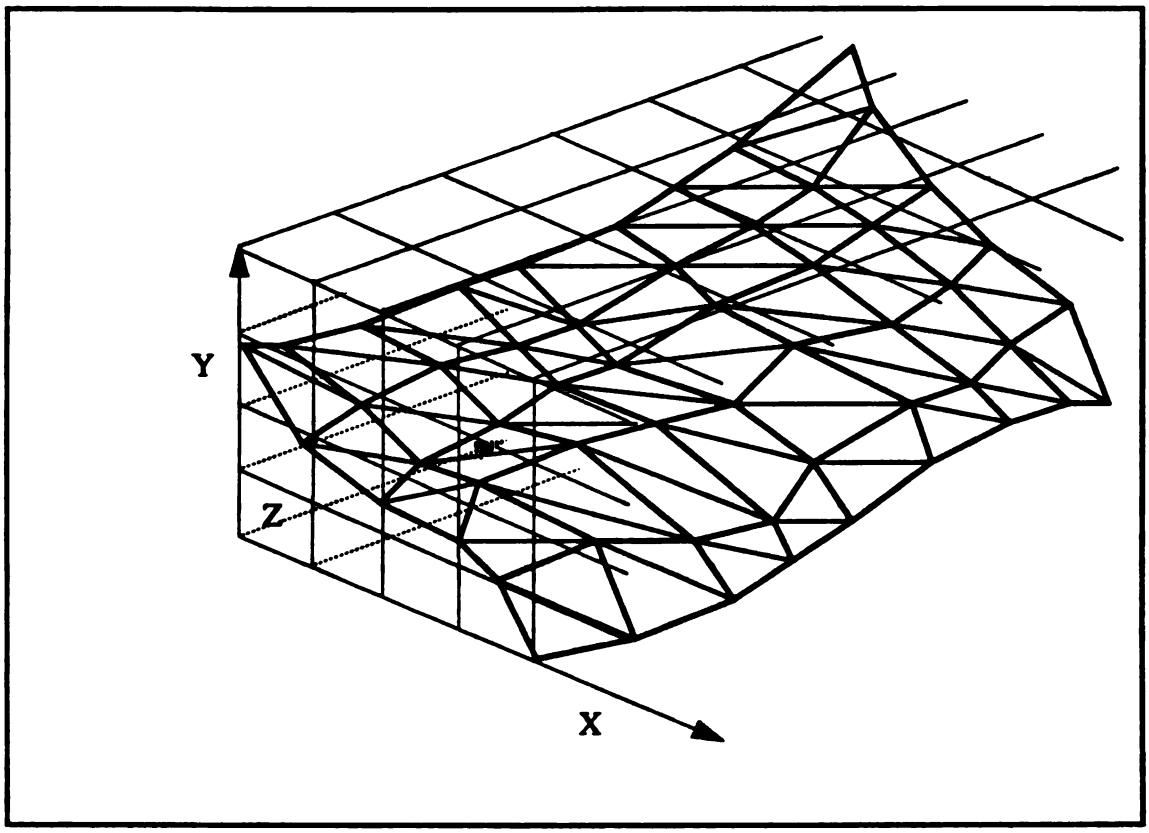

Figure 4.4 Space subdivision of a discretized NURBS surface

The subdivision method described here creates a set of uniform cubic 3-D voxels. A voxel (volume element) is defined as a rectangular solid element in 3-D space. For example, a 1 m^3 volume could be considered to contain 10^6 individual 1 cm^3 cells. A translation and scaling transformation is determined and applied which will move all surface points into a positive working space of a size such that the integer parts of each point's coordinates can serve as the index of the voxel containing the point. That is, after the transformation, a point's 3-dimensional voxel indices are the truncated real parts of its

translated (x, y, z) coordinates. A linked list of all surface points within each voxel is established. This makes it easy to search 3-D voxels and obtain the surface points possibly affected by a given tool motion. If the size of the voxel is small, then the total number of surface points for which to do the further computation is also small. But empty voxels become a burden for 3-D searching, so there is a trade-off to make between the size of the voxels and the number of voxels to be searched. It is suggested that one choose the size of the voxel to be on the order of the length of the median toolpath segment. Therefore, each tool movement may move into some voxels and out of others.

4.2. Toolpath Processing

In addition to the desired part surface model, some other input data are required for verification, including the cutter information, the cutter location data (CL-data), the tolerance of the desired surfaces, the range of interest for display, and limits for discretization of the toolpath. The cutter information includes seven parameters which are defined by APT. Therefore, the type of cutter, the height of the cutting tool, the radius of the cutter, *etc.*, are all defined. There are three types of common cutting tools: the ball-end, flat-end, and fillet-end cutters, which all can be specified by a general APT cutter definition as shown in Figure 4.5. The most important assumption about the tool motion is that the toolpath segments represent linear point-to-point motions. The control point of the cutting tool moves along a straight line from point to point. If the user programs for circular interpolation between points, for example, then our software depends on a postprocessor to be run first to generate a linear approximation to the circular path between control points. (In contrast, however, this software does internally allow for great-circle interpolation between tool axis angles of sequential control points.) Therefore, the objective function of the optimization problem for tool motion is only a second-order polynomial equation, which can be solved without using penalty functions.

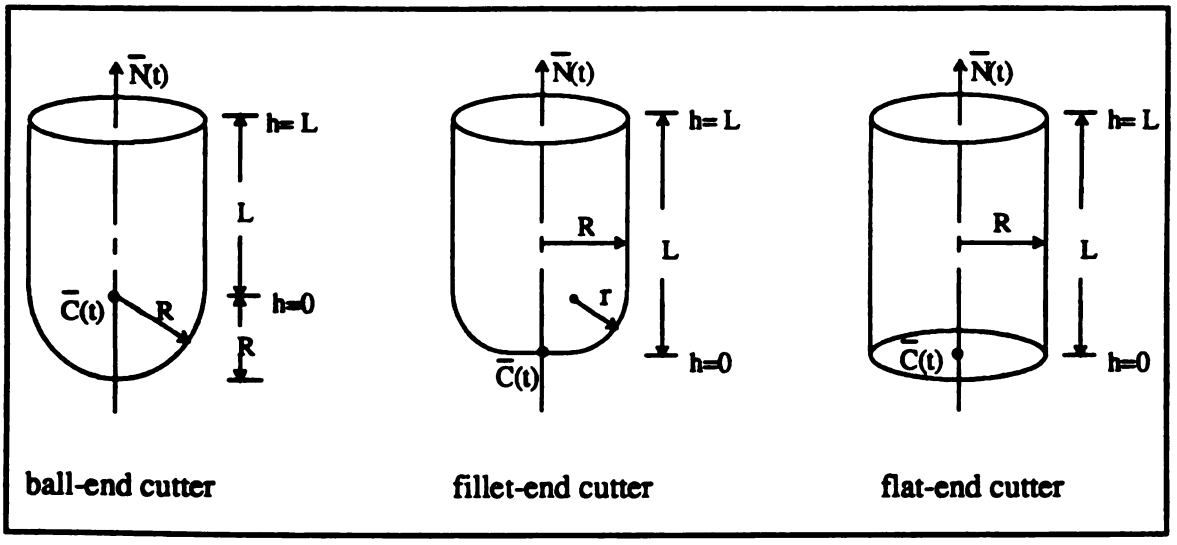

Figure 4.5 Various cutter shapes

For a multi-axis tool motion, the $h_i(t)$ value in equation (3.2) for a surface point may have more than two maximum or two minimum values under some circumstances. Therefore, interference detection becomes very complicated. To avoid this situation, the size of the toolpath discretization is regulated, via a maximum angle between the start point orientation and the end point orientation, as described in Chapter V. The program makes additional "internal" CL points for these intermediate tool positions and orientations.

4.3. Localization of A Tool Motion

A sculptured surface is typically discretized into thousands of points, depending on the desired accuracy, size, and curvature of the surface. Since a given tool motion will interfere with only a small percentage of the surface points, it is very desirable to eliminate all the surface points which cannot possibly cause interference. The calculation time is directly proportional to the number of interference calculations. Therefore, one needs to determine a loose boundary of a given tool motion. For any ruled surface $\bar{r}(t, h)$ within $0 \le h \le L$, $0 \le t \le 1$, one can easily use equation (3.1) to evaluate the maximum and

minimum values of the coordinate x_{max}, x_{min}, y_{max}, y_{min}, z_{max} and z_{min}. Then the coordinates of any surface point which might be affected by this tool motion satisfy: $x_{min} - R_{max} \le x \le x_{max}+R_{max}$, $y_{min} - R_{max} \le y \le y_{max}+R_{max}$, and $z_{min} - R_{max} \le z \le z_{max}+R_{max}$, where R_{max} is the maximum radius of the cutter plus range of interest R_{int}. The range of interest, R_{int}, is user-selected to provide a maximum offset around the surface, outside of which the algorithm need not calculate cut distance, (and later, outside of which the depth display can be uniformly colored). Therefore, the indices of the voxels containing those points are contained in the rectangular solid volume defined by $\lfloor x_{min} - R_{max} \rfloor \le x \le \lfloor x_{max}+R_{max} \rfloor$, $\lfloor y_{min} - R_{max} \rfloor \le y \le \lfloor y_{max}+R_{max} \rfloor$ and $\lfloor z_{min} - R_{max} \rfloor \le z \le \lfloor z_{max}+R_{max} \rfloor$. The linked list of surface points for each of those voxels is then available for further consideration.

4.4. Toleranced NC Verification

The minimum distance along the surface normal vector to the boundary of the tool motion can be evaluated by the method given in the Appendix B. Once all of the distances between the surface points and boundaries of the tool motions have been evaluated, the user can freely specify in- and out- tolerances about the desired part surface, as long as both values are less than R_{int}, and one can view color-coded displays of the results from arbitrary viewpoints.

The final machined part has to be compared with the desired part surface. For NC programs with tolerances specified by the INTOL and OUTTOL concept of APT, if the machined part is between the INTOL and OUTTOL surfaces, then the machined part surface meets the tolerance (positional tolerance) specification.

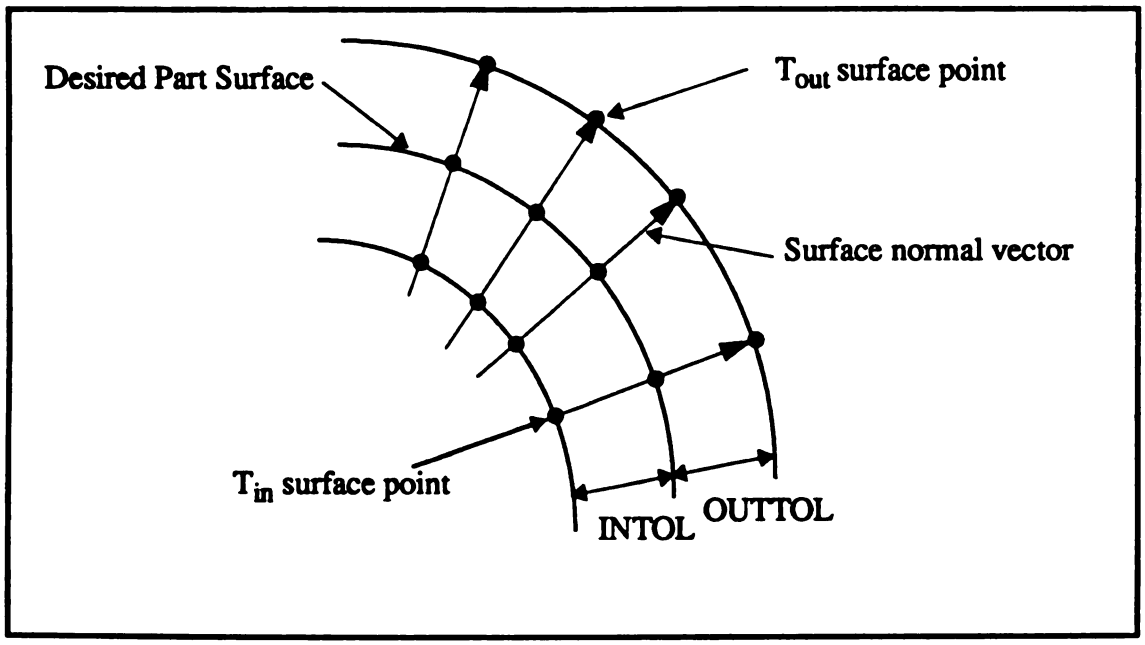

Figure 4.6 T_{out} and T_{in} surface points for tolerance specification

An alternative way, which greatly reduces the cost of the minimum distance calculation, is to use two surfaces offset from the part surfaces. Rather than to solve for the equations of the offset surfaces, the normals to the part surface are simply used to generate a T_{in} offset point = (x_{in}, y_{in}, z_{in}) and a T_{out} offset point = $(x_{out}, y_{out}, z_{out})$ in the data structure corresponding to each discretized surface point. T_{in} is offset along the negative of the normal vector by INTOL, and T_{out} is offset along the positive normal vector by OUTTOL. By definition, these points are on the corresponding offset surfaces [37][38]. Then for each surface point, the algorithm must calculate the signed distance to T_{in} and T_{out}, using the nominal tool radius function in equation (3.4). If the T_{out} surface point is inside the envelope of the tool motion and the T_{in} surface point is not, then the corresponding surface point is within the tolerance specification. The equivalence of this method to calculating the distance along the surface normal unit vector to the boundary of the tool motion is exact only when the depth of cut is at either T_{in} or T_{out}. However, we are also guaranteed that the cut is within tolerance at the surface point if the depth of cut is

anywhere between T_{in} and T_{out}, even though the level of deepest cut is only approximated.

A goal of interference detection is to allow subsequent adjustment of the toolpath to eliminate undercut or overcut regions. Therefore, using this method, it is necessary to approximate how deep the gouge is or how great the undercut is for any surface point outside the tolerance specifications. The approximate distance between a surface point and the side surface of the tool motion is evaluated as follows:

First, evaluate the approximate cut values $S_{in}(t)$ and $S_{out}(t)$ for the T_{in} and T_{out} offset points, respectively.

$$S_{in}(t) = \sqrt{[x_{in} - x(t)]^2 + [y_{in} - y(t)]^2 + [z_{in} - z(t)]^2 - h_{in}(t)^2} - R(h_{in}(t))$$

$$S_{out}(t) = \sqrt{[x_{out} - x(t)]^2 + [y_{out} - y(t)]^2 + [z_{out} - z(t)]^2 - h_{out}(t)^2} - R(h_{out}(t))$$

If $S_{in}(t)$ and $S_{out}(t)$ are positive values, which means T_{in} and T_{out} do not interfere with the envelope of the tool motion, then the approximate cut value is $newS(t) = S_{out}(t)$ + OUTTOL.

If $S_{in}(t)$ is a positive value and $S_{out}(t)$ is a negative value, which means T_{in} does not interfere with the envelope of the tool motion and T_{out} does, then the approximate cut value is $newS(t) = S_{out}(t)$ + OUTTOL.

If $S_{in}(t)$ and $S_{out}(t)$ are negative values, which means T_{in} and T_{out} interfere with the envelope of the tool motion, then the approximate cut value is $newS(t) = S_{in}(t)$ - INTOL.

If $S_{in}(t)$ is a negative value and $S_{out}(t)$ is a positive value, then we need to evaluate the surface point itself, since the surface normal and tool axis are pointing in radically different directions. If the approximate cut value of this surface point is also a negative value, then the approximate cut value is $newS(t) = S_{in}(t)$ - INTOL. Otherwise, this describes a case in which a surface point is being approached by the cutter from behind (inside) the outward-directed surface, and this anomaly can be considered either as an error (a deep gouge, for example), or as not cutting the normal at the surface point, leaving its

former value unchanged. This unusual configuration between the surface normal vector and the tool axis will be discussed as a special case in the Section 5.9.

Regardless of which of the four above cases holds, the cut value S(t) is then set to min (S(t), newS(t)), representing the deepest cut to date.

If the surface point is cut by a flat bottom surface of a cutter, the approximate depth of cut is calculated using $h_i(t)$ instead of S(t), which can be evaluated from Part 2 of Appendix B. If the "top" surface of a cutter cuts anything, it represents a collision of a non-cutting surface of a tool or tool holder with the part or fixture geometry, and is reported as an error condition.

This two-point algorithm can verify exactly whether the surface point is within the tolerance specification. Depth of undercut or overcut, when not exactly equal to the tolerance specifications, is calculated as an approximate value. Because the program tracks the index of the toolpath segment causing the deepest cut value S(t) (*i.e.*, the deepest gouge or closest undercut) to date, the user always knows at the end of the verification which segments need to be corrected, and by approximately how much, on each region of the part.

4.5 Postprocessing

The output image depicts the desired sculptured surface using color coding to show the areas within tolerance and out of tolerance in different colors under X-window systems [39 - 45]. Green represents the areas cut within tolerance. Gouges deeper than INTOL vary from red to yellow for distances from INTOL to $-R_{int}$. Undercuts greater than OUTTOL are shown in hues from dark blue to light blue for distances from OUTTOL to R_{int}. For intelligibility of the output, intensities of all hues are varied according to the angle between the eyepoint and the part surface normal.

Arbitrary views may be displayed without redoing any verification calculations, and without changing the answers coded for any points. Within the limits of the distance

approximations used for all points except those exactly at the specified tolerance limits, the user may also choose to display results with different tolerance limits, without needing to recalculate cut values. Such rapid exploration of the depth of cut values is very useful, for example, in identifying problems via visualizing cusps and near-gouges.

CHAPTER V

THE ALGORITHM FOR THE NC VERIFICATION SOFTWARE

In this chapter, the algorithm implemented in the NC verification software used as proof of concept for this work is presented and discussed via pseudo code [46]. Each of the specific functions will be discussed based on the mathematical model already presented. Testing the correctness of software is a very important phase of software development. Therefore, we introduce some geometric models as test cases with known results, to test the accuracy of the program. This chapter will also present a performance analysis of the algorithm in which an order of computational complexity is discussed.

5.1 Pseudo code for the system software

The NC verification algorithm can be depicted in pseudo-code as follows. The algorithm is divided into three tasks. The surface discretization and output display tasks will be discussed only briefly, since they are not the topic of this research. Each function or subroutine in the toolpath processing and the verification will be comprehensively discussed below.

Task1 : Object-Space Surface Discretization

{

Load the surface data and determine desired discretization parameters.

Discretize NURBS (Non-uniform Rational B-Spline) surfaces and store points, triangles and normals.

Classify each surface point into a uniform discretization of 3-D space (voxels), and set up a linked list including all the points in a voxel.

}

Task2 : Toolpath Processing and Verification

{

Load the cutter data and CL file, then link the CL points to make the struct tool_vect.

For every CL-point from 1 to N

{

Calculate the parameters for the tool axis motion, implicitly defining a ruled surface.

Further discretize the toolpath segment if necessary, if the tool axis changes within the segment.

Determine which voxels may contain this toolpath segment.

For every surface point in every voxel possibly interfering with this tool-path segment

{

Simulate the motion along the toolpath and get the cut value for the surface point.

Update the cut_value in the data structure of POINT.

}

}

}

Task3 : Display the verification output as a pseudocolor raster on the surface.

{

Use the updated cut_value from the data structure of POINT after the verification is complete.

The color of the discretized surface is decided by the magnitude of the cut_value. The boundary between regions of varying hues between the discretized surface points is determined by linear interpolation of cut_values. Intensity is based on angle between sight line and surface normal.

}

5.2. Loading Surface Data

The desired part surfaces used in this method consist of NURBS surfaces, which are defined by a designer and typically are directly produced by a CAD surface or solid modeling package. NURBS, trimmed NURBS, and bounded NURBS may all be read and discretized in their standard IGES representations. Trim and boundary curves may be either NURBS curves, composite curves or copious data entities. Three user-supplied values control the level of discretization performed on these surfaces. They are maximum chordal deviation and maximum distance (in world coordinates) between sequential points generated in each of the two parametric directions. After the surfaces are evaluated and discretized, the surface outline is displayed. An implementation developed by Correns[32] of NURBS evaluation algorithms is used here to evaluate untrimmed NURBS surfaces (points, chordal deviations and normals). Program flow for the surface loading task is illustrated in Figure 5.1.

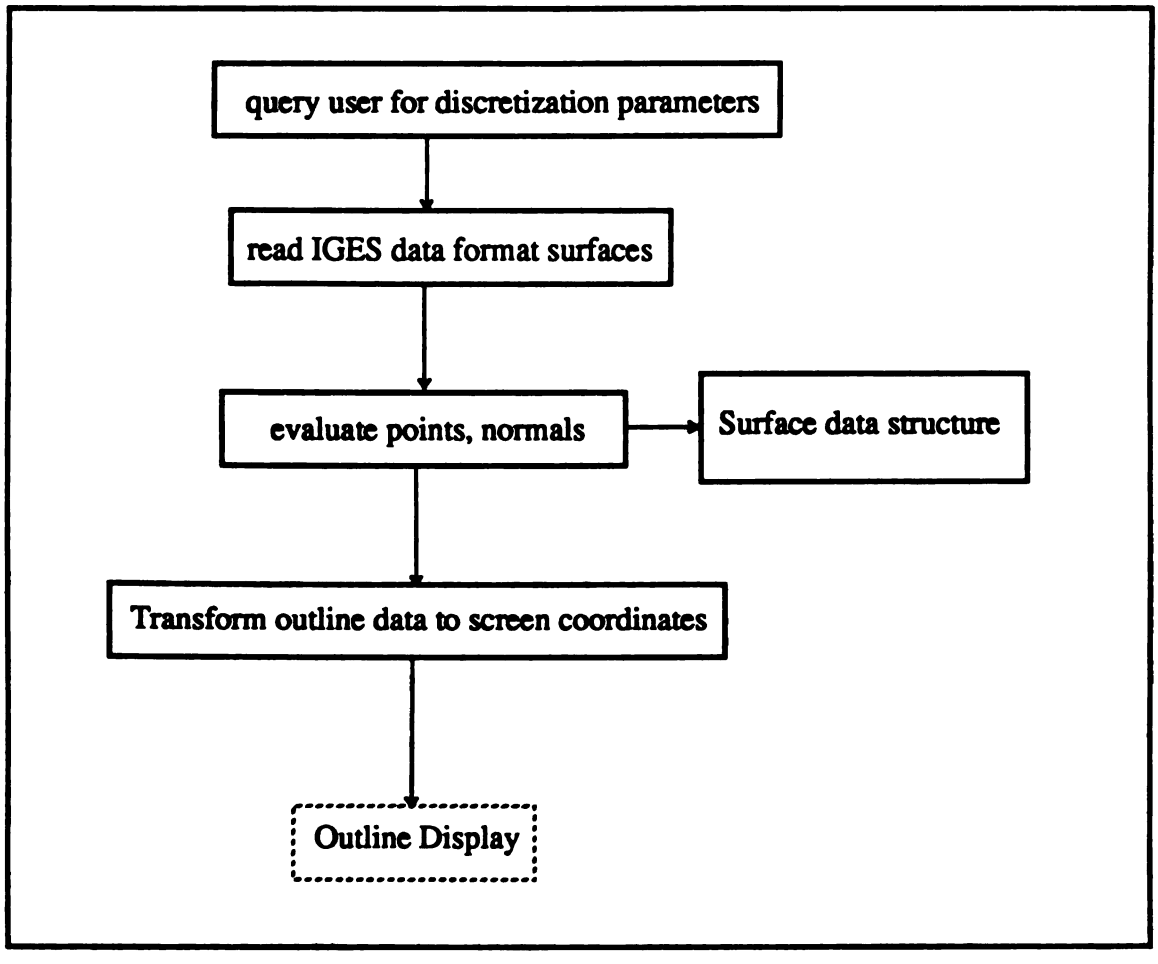

Figure 5.1 Program Flow in Loading Surface Data

5.3 Create voxel space

All the surface points in the data structure of the surface are classified into a uniform discretization of 3-D voxel space, and a linked list of all the points in a voxel is built.

set up the index for the voxel space

While (surface != NULL)

{

Translate all the surface points into positive quadrant of a coordinate system based on a box bounding all surfaces.

Classify each surface point into its corresponding voxel.

Transfer the surface corner points, surface maximum and surface minimum points into corresponding voxels.

}

5.4 Loading Toolpath data and cutter data

The data structure of the toolpath needs to be built as a linked list. The toolpath is considered to define a **linear motion** of the tool's control point between CLpoints. The toolpath data read from the CLfile are in groups of six elements, which are (x, y, z) coordinates of a CLpoint and ($\cos\alpha$, $\cos\beta$, $\cos\gamma$) where α, β, and γ are the angle of the tool axis with the x, y, and z axes at that point, respectively. The data structure, called *tool_vect*, has six data elements: the toolpath start point, start point orientation, end point, end point orientation, toolpath number, and a link to the cutter information for this tool movement. Dynamic allocations were used for all structures having a size dependent on the number of points in the toolpath, so the program has no upper limit on toolpath size except for the virtual memory capacity of the workstation.

Let P denote the start point of a toolpath segment; P_{next} , the end point of a toolpath segment.

Let P_orient denote the start point orientation; P_{next}_orient, the end point orientation.

allocate memory space for a toolpath segment

read two CL data points

Transform the CL data points using the voxel space transformation

store toolpath data and set toolpath pointer to space for next transformed CL data

set $P = P_{next}$ *and* $P\_orient = P_{next\_orient}$

while (CL data != EOF)

{

allocate memory space for a toolpath segment

read another CL point

Transform CL point using the voxel space transformation

store toolpath data and set toolpath pointer to space for next transformed CL data

set $P = P_{next}$ *and* $P\_orient = P_{next\_orient}$

}

The cutter information is defined by the general APT cutter in this program. Seven parameters are defined according to the definition in Appendix A. The data structure for the cutter contains cutter type (calculated from the APT parameters), distance from control point to top of cutter, the seven APT parameters, h values for three points bounding the three cutter regions, two precomputed coefficients of radius-dependent functions (to avoid the need to recalculate them for each move), and the range of the interest. The three most common cutter types are treated as special cases, based on the contents of the APT parameter file. In all cases but the ball-end cutter, the control point is considered to be at the tip of the cutter. Definitions of these cutter types in terms of the seven APT parameters are as follows:

ball-end cutter : CUTTER/ d, d/2, 0, d/2, 0, 0, L

The ball-end cutter has radius d/2 and total length L (but the distance from the control point to the top of the cutter is L-d/2)

flat-end cutter : CUTTER/ d, 0, d/2, 0, 0, 0, L

The flat-end cutter has radius d/2 and length L

fillet-end cutter : CUTTER/ d, r, d/2-r, r, 0, 0, L

The fillet-end cutter has radius d/2, corner circle radius r and length L

The cutter profile is displayed at the first two CLpoints when the toolpath is loaded, allowing a visual check of tool correctness. This is very important for the general APT cutter, since the shape of the cutter is not trivial to see from the seven segments parameters.

allocate memory space for a cutter data structure

store seven segments cutter value into cutter pointer data

evaluate h values at the dividing points between cutter regions of three different radii

decide the type of cutter

evaluate the cutter coefficients for first and third cutter radius regions

print all the cutter pointer data for checking

5.5 Tool axis motion during verification

During verification, an additional data structure called segment_structure is filled as each toolpath segment is processed. The data structure contains the three slope coefficients of the linear tool motion from the start point to the end point, the arc angle between the orientation of the start point and the end point, and the function coefficients

defined below for the tool axis unit vector N(t). As described in Chapter 2, the tool axis orientation is linearly interpolated along a great circle as shown below. Different tool axis orientation functions are possible, and would require code modifications to implement.

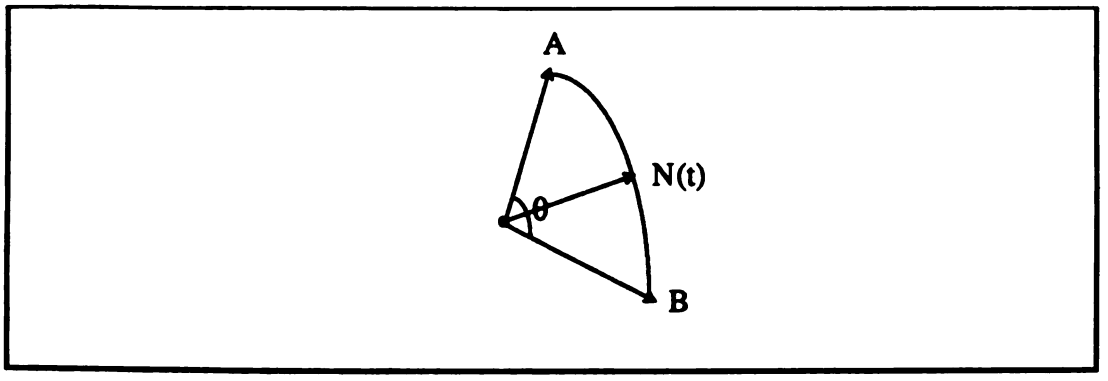

Figure 5.2 Circular interpolation between A and B in 3-D space

It is equivalent to solving the following three equations:

$$A \cdot N = |A||N|\cos(\theta t) \quad \text{---------- (1)}$$

$$N \cdot (A \times B) = 0 \quad \text{---------- (2)}$$

$$B \cdot N = |B||N|\cos(\theta (t-1)) \quad \text{---------- (3)}$$

where $0 \leq t \leq 1$,

N is the interpolation vector corresponding to motion on a great circle on a unit sphere.

The A unit vector is the previous tool axis and the B unit vector is the current tool axis

θ is arc angle between the A vector and the B vector.

Therefore, solving (1), (2) and (3), the tool axis orientation N(t) becomes three components $N_x(t)$, $N_y(t)$, $N_z(t)$, where each component can be expressed in terms of $\cos(\theta t)$ and $\cos(\theta(t-1))$. This is very easy to evaluate for any interpolation unit vector N(t) between $0 \leq t \leq 1$.

$$arc\_angle = \cos^{-1}(P\_orient \cdot P_{next\_orient})$$

```
if ( arc angle > epsilon )
  {
    evaluate N(t)
    Nx(t) = Nx(t)_coeff[0]*cos(θt) + Nx(t)_coeff[1]*cos(θ (t-1))
    Ny(t) = Ny(t)_coeff[0]*cos(θt) + Ny(t)_coeff[1]*cos(θ (t-1))
    Nz(t) = Nz(t)_coeff[0]*cos(θt) + Nz(t)_coeff[1]*cos(θ (t-1))
  }
else
  {
    Nx(t)_coeff[0] = Nx(t)_coeff[1] = 0.5*Ax[0]
    Ny(t)_coeff[0] = Ny(t)_coeff[1] = 0.5*Ay[1]
    Nz(t)_coeff[0] = Nz(t)_coeff[1] = 0.5*Az[2]
  }
```

5.6 Additional toolpath discretization during verification for multi-axis milling

In equation (3.2), $h_i(t) = [x_i - x(t)]N_X(t) + [y_i - y(t)]N_Y(t) + [z_i - z(t)]N_Z(t)$. For each surface point, there is a corresponding value $h_i(t)$ within $0 \leq t \leq 1$. The N(t) is expressed in terms of cos(θt) and cos(θ(t-1)). For multi-axis tool motion, the $h_i(t)$ may have more than two extrema within $0 \leq t \leq 1$. Therefore, evaluation of equation (3.3) becomes very complicated. Instead, dynamic discretization of the toolpath into shorter, constant-orientation segments is used for multi-axis tool motion. For three-axis tool motion, this additional discretization is not needed, since the tool axis does not change, so the number of minima in the interval $0 < t < 1$ is at most 1. For the multi-axis case, the number of additional segments to be produced is decided by the angle between orientation of the start point and the end point, relative to a maximum angle provided by the user.

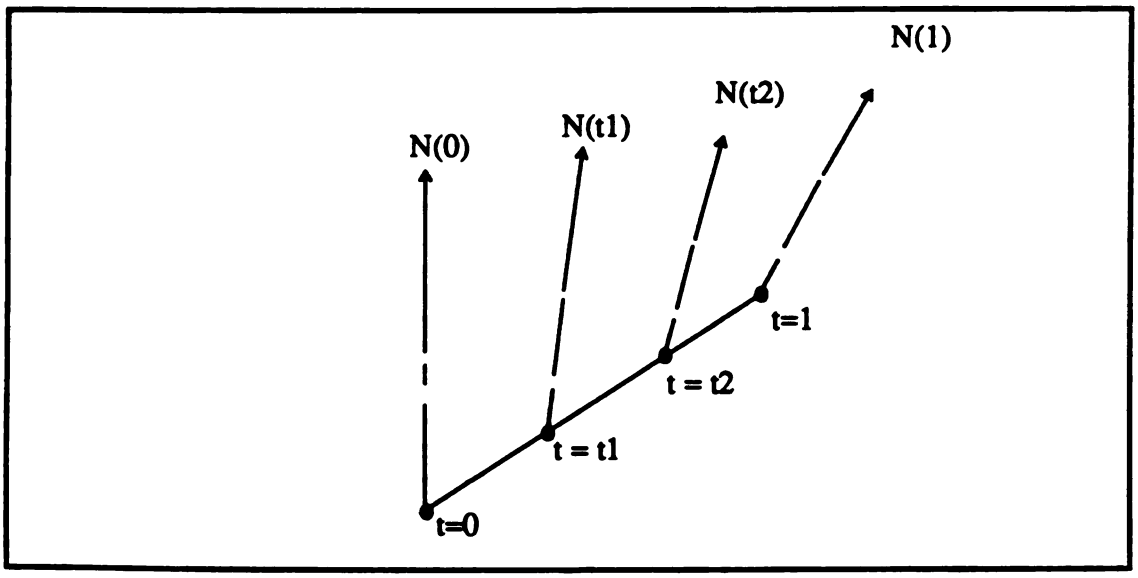

Figure 5.3 Toolpath discretization for multi-axis milling

```
if ( arc angle < epsilon )
number of segments =1
else
number of segments = arc_angle / user_specified_maximum_angle
store current toolpath data into temporary variables
for ( j = 1 , j <= number of segments, j ++)
{
 and cut value evaluation
}
restore the toolpath data values
```

5.7 Searching voxel space for a tool motion

A tool motion can be confined within some voxel volume, thus reducing useless calculations for surface points far away from the tool motion. The edge dimensions of a

voxel are usually selected to be similar to the length of the typical tool motion. After the surfaces are discretized into points, during which process extrema are tracked, all the surface points are transformed into a positive coordinate space and linked-listed within their appropriate voxels in that space. The voxel volume containing a particular tool motion can be evaluated according to equation (3.1). The mathematical concept can be expressed as shown. The deviation is caused by the angle change between the orientation of the start point and the end point.

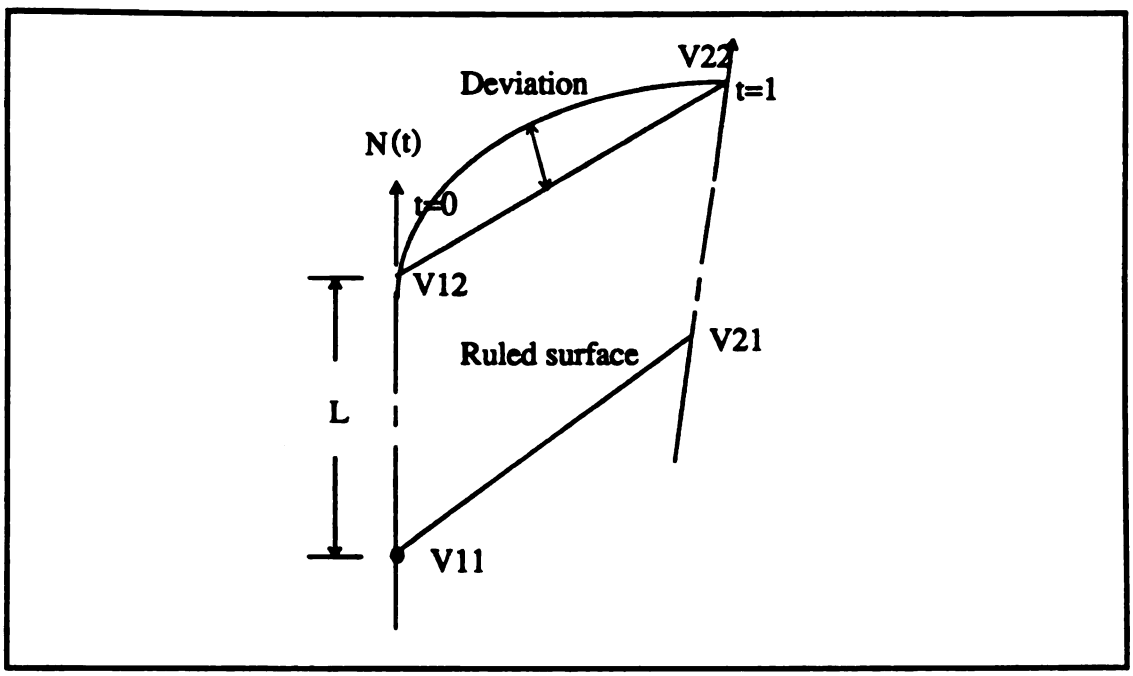

Figure 5.4 Space bound for a ruled surface

Therefore,

Maximum of a tool motion boundary = max (V11, V12, V21, V22) + R_{max} + Deviation

Minimum of a tool motion boundary = min (V11, V12, V21, V22) - R_{max} - Deviation

Where, V11, V12, V21, V22 are evaluated according to the ruled surface.

R_{max} is the maximum radius of the cutter plus range of interest R_{int}.

Deviation = $L(1-\cos\theta_l)$ and θ_l is the angle between the start point orientation and

end point orientation.

Pseudo code for the algorithm can be expressed as follows:

Let cutter_bound = max_cutter_radius + range_of_interest + Deviation

Let min_x, min_y, min_z = minimum x, y and z coordinates of a tool motion boundary, respectively.

Let max_x, max_y, max_z = maximum x, y and z coordinates of a tool motion boundary, respectively.

```
evaluate the maximum and minimum for each of x, y, and z for the ruled surface of the tool
 path segment
maximum coordinate of the tool motion = maximum coordinate of the ruled surface +
                                        cutter_bound
minimum coordinate of the tool motion = minimum coordinate of the ruled surface -
                                        cutter_bound
for ( i= min_x, i <= max_x, i++ )
 for ( j= min_y, j <= max_y, j++ )
  for ( k= min_z, k <= max_z, k++ )
      {
        convert the voxel coordinate into a corresponding voxel number (integer)
        set surface_point = first pointer of this voxel number
        while ( surface_point != NULL)
              {
                evaluate cut value of a surface point
                set surface_point = surface_point->neighbor
              }
      }
```

5.8 Minimum distance evaluation from the surface point

Minimum distance evaluation between a surface point and the ruled surface is used to decide whether or not the surface point is inside the envelope of the tool motion. The approximate cut value is also evaluated, according to the equation in section 3.3. Recall the equation -- it is similiar to an optimization problem in which an objective function is minimized subject to two constraints.

$$\text{Objective: minimize } [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t))$$

$$\text{Constraints: } 0 \leq h_i(t) \leq L$$

$$0 \leq t \leq 1$$

For this optimization problem, it is possible to solve the minimization problem without using a penalty function if the objective function is only a second-order polynomial equation and the constraints are only first order, as is the case in this problem. In fact, for the three-axis case, or with the linear approximation used in this work for solution of the five-axis case, we can solve the constraints first for t, and then minimize the objective function later. Let's discuss this problem by examining three-axis and five-axis milling as two different cases.

5.8.1 Three-Axis Case

In a three-axis milling machine, the N(t) is constant throughout the tool motion. Therefore, h(t) is a linear function if the toolpath motion is linear. The algorithm simply evaluates the start point, middle point and end point h(t) functions, then decides which R(h) function we need to use (*i.e.*, which regions of the cutter the point may be cut by). Alternatively, one could solve the h(t) constraint for the t values at which h(t) is a dividing point between cutter radius functions (i.e., the function R(h) changes shape). Using either technique, one can usually determine that the radius function R(h) need only be evaluated

for one or two of its three possible functions, because the possible intersections with the point are confined to those one or two regions of the cutter geometry. In the worst case, all three radius functions must be evaluated.

$$f(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \text{--------} (5.6.1)$$

$$h_i(t) = [x_i - x(t)]N_X + [y_i - y(t)]N_Y + [z_i - z(t)]N_Z$$

The minimization equation actually is only a second-order equation. To get the minimized value, it is simple to solve $f'(t) = 0$. Compare the f(0), f(minimized_t), f(1). which satisified the constraints on h(t). The minimized value of f(t) is therefore obtained without solving the penalty equation of the optimization problem.

The minimum distance from the surface point along the normal to the bottom (or with a similar argument, the top surface) of the tool motion can be solved by solving the constraints first, then minimizing the cut value s(t) later.

evaluate the coefficients of $[x_i - x(t)]$, $[y_i - y(t)]$, $[z_i - z(t)]$ *and* $h_i(t)$

evaluate the function coefficients of $[x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t)$

evaluate the approximate cut value of the start point and end point

set min_cut_value = min(start point cut_value, end point cut_value)

evaluate the h_i *value of the start point , middle point and end point*

decide which portions of the tool may cut deepest during this toolpath segment (i.e., which radius functions to evaluate)

evaluate the function coefficients of equation (5.6.1)

evaluate the first derivative of equation (5.6.1)

solve for the root of the above equation, t1

if (t1 is between 0 and 1 , and $h_i(t)$ *is between 0 and L)*

{

evaluate the approximate cut value: t_min_cut_value =

$sqrt\{[x_i - x(t\_min)]^2 + [y_i - y(t\_min)]^2 + [z_i - z(t\_min)]^2 - h_i^2(t\_min)\} - R(h_i(t\_min))$

set min_cut_value = min(min_cut_value, tmin_cut_value)

}

5.8.2 Five-Axis Case -- First Approach

For five-axis milling, one approach to evaluating the approximate cut_value is to use an iterative solution technique. The solution can be reached subject to a convergence condition. For five-axis tool motion, N(t) is not a constant any more. Direct solution of the h(t) and $f'(t) = 0$ equations is no longer easy. The pseudo code of a simple iterative technique is as follows :

set t_new = 0.5

if ((t_new - t_old) < epsilon)

{

evaluate the coefficients of $h_i(t)$ *according to N(t_new)*

evaluate the function coefficients of $[x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t)$

solve the first derivative of the above equation for t_min

let t_old = t_new

if $0 \le t \le 1$ *and t_min_cut_value = f(t_min) is the minumum value for function f(t)*

then t_new = t_min_cut_value

else if (t_min_cut_value > 1)

t_new = 1

else if (t_min_cut_value < 0)

t_new = 0

}

The proof of a sufficient condition under which the algorithm converges is based on the following equation:

$$f(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \text{ ---------- (5.6.2)}$$

$$h_i(t) = [x_i - x(t)]N_X(t) + [y_i - y(t)]N_Y(t) + [z_i - z(t)]N_Z(t)$$

$$= (a + b*t)(c*\cos\theta\, t + d*\cos(\theta\,(t-1)) \text{ ---------- (5.6.3)}$$

Let $\{t_i\}$ be the sequence of approximate solutions for t in the above algorithm.

For all t_i, let $f'(t) - f'(t^*) = M_i(t - t^*)$. Then, if $|M_i| < 1$, for all $i>1$, then the solution converges, where t^* is the solution of the equation.

Let's derive the converge condition based on the special case R(h(t)) = R

Therefore,

$$f(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2$$

Then $F(t) = f'(t) = 2(x_i - x(t))\,\dot{x}(t) + 2[y_i - y(t)]\,\dot{y}(t) + 2[z_i - z(t)]\,\dot{z}(t) - 2h_i(t)\dot{h}_i(t)$

Let t_0 be the first iteration value and t_1 and t_2, the next two values. We need to prove that the solution converges under some conditions. First, assume t_n is n^{th} iteration value and t_{n+1} is $n+1^{th}$ iteration value.

Then

$$F(t_n) = 2\ (x_i - x(t_n))\,\dot{x}(t) + 2[y_i - y(t_n)]\,\dot{y}(t) + 2[z_i - z(t_n)]\,\dot{z}(t) -$$

$$2(a + b\,t_n)(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))[b\,(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1)) +$$

$$(a+b\,t_n)\theta(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1))] = 0$$

and

$$F(t_{n+1}) = 2(\ x_i - x(t_{n+1}))\,\dot{x}(t) + 2[y_i - y(t_{n+1})]\,\dot{y}(t) + 2[z_i - z(t_{n+1})]\,\dot{z}(t) -$$

$$2(a + b\,t_{n+1})(c\cos\theta t_n + d\cos\theta(t_n - 1))\,[b\,(c\cos\theta t_n + d\cos\theta(t_n - 1)) +$$

$$(a+b\,t_{n+1})\theta(-c\sin\theta t_n - d\sin\theta(t_n - 1))] = 0$$

We need to derive the relationship between successive iteration values:

$$F(t_n) - F(t_{n+1}) = 2\,(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\ )(t_{n+1} - t_n) - 2(a + b\,t_n)(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))$$

$$[b\,(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1)) + (a+b\,t_n)\theta(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1))]$$

$$+ 2(a + b\,t_{n+1})(c\cos\theta t_n + d\cos\theta(t_n - 1))\,[b\,(c\cos\theta t_n +$$

$$d\cos\theta(t_n - 1)) + (a+b\,t_{n+1})\theta(-c\sin\theta t_n - d\sin\theta(t_n - 1))] = 0$$

Rearranging, the equation becomes:

$$2\,(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\ )(t_{n+1} - t_n) - 2b\,(a + b\,t_n)(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))^2$$

$$2\theta(a + b\,t_n)^2(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1)) +$$

$$2b(a + b\,t_{n+1})(c\cos\theta t_n + d\cos\theta(t_n - 1))^2 +$$

$$2\theta(a + b\,t_{n+1})^2(c\cos\theta t_n + d\cos\theta(t_n - 1))(-c\sin\theta t_n - d\sin\theta(t_n - 1)) = 0 \text{ ---- (5.6.2)}$$

Let's add and subtract two terms $2b(a + b\,t_n)(c\cos\theta t_n + d\cos\theta(t_n - 1))^2$ and

$$2\theta(a + b\,t_{n+1})^2\,(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1))$$

from equation (5.6.2)

Rearranging, the equation becomes

$$2(\dot{x}^2 + \dot{y}^2 + \dot{z}^2)(t_{n+1} - t_n) +$$

$$2b(a + b\,t_n)[(c\cos\theta t_n + d\cos\theta(t_n - 1))^2 - (c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))^2] +$$

$$2\theta((a + b\,t_{n+1})^2 - (a + b\,t_n)^2)(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1)(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1))$$

$$+ 2b^2(t_{n+1} - t_n)(c\cos\theta t_n + d\cos\theta(t_n - 1))^2 +$$

$$2\theta(a + b\,t_{n+1})^2((c\cos\theta t_n + d\cos\theta(t_n - 1))(-c\sin\theta t_n - d\sin\theta(t_n - 1)) -$$

$$(c\cos\theta t_{n-1} + d\cos\theta(t_{n-1} - 1))(-c\sin\theta t_{n-1} - d\sin\theta(t_{n-1} - 1)) = 0$$

Let $g(t) = (c\cos\theta t + d\cos\theta(t - 1))^2$, $(a + b\,t)^2$ or

$$(c\cos\theta t + d\cos\theta(t - 1))(-c\sin\theta t - d\sin\theta(t - 1))$$

Introduce the mean value equation $g(t_n) - g(t_{n-1}) = g'(t)(t_n - t_{n-1})$, where t is between t_n and t_{n-1}

Therefore,

$$(2(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) + 2b^2(c\cos\theta t_n + d\cos\theta(t_n - 1))^2)(t_{n+1} - t_n) -$$

$$2b\theta(a + b\,t_n)[2(c\cos\theta t + d\cos\theta(t - 1))(c\sin\theta t + d\sin\theta(t - 1))](t_n - t_{n-1}) +$$

$$2\theta * 2b(a + b\,t)(c\cos\theta t_n + d\cos\theta(t_n - 1))(-c\sin\theta t_n - d\sin\theta(t_n - 1))(t_{n+1} - t_n) -$$

$$2\theta^2(a + b\,t_{n+1})^2[(c\cos\theta t + d\cos\theta(t - 1))^2 - (c\sin\theta t + d\sin\theta(t - 1))^2](t_n - t_{n-1}) = 0$$

Rearrange the equation to obtain the final expression:

$$[2(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) + 2b^2(c\cos\theta t_n + d\cos\theta(t_n - 1))^2) +$$

$$4b\theta(a + b\,t)(c\cos\theta t_n + d\cos\theta(t_n - 1))(-c\sin\theta t_n - d\sin\theta(t_n - 1))](t_{n+1} - t_n)$$

$$= \{4b\theta(a + b\,t_n)[(c\cos\theta t + d\cos\theta(t - 1))(c\sin\theta t + d\sin\theta(t - 1))] +$$

$$2\theta^2(a + b\,t_{n+1})^2[(c\cos\theta t + d\cos\theta(t - 1))^2 - (c\sin\theta t + d\sin\theta(t - 1))^2]\}(t_n - t_{n-1})$$

If $(t_{n+1} - t_n) = M(t_n - t_{n-1})$ and $|M| < 1$, then the iteration converges. But the above equation is of that form, considering

$$M = \frac{K1}{K2}$$ and defining

$$K1 = \{4b\theta(a + b\,t_n)\,[\,(c\cos\theta t + d\cos\theta(t-1))\,(c\sin\theta t + d\sin\theta(t-1))\,] +$$
$$2\theta^2(a + b\,t_{n+1})^2[(c\cos\theta t + d\cos\theta(t-1))^2 - (c\sin\theta t + d\sin\theta(t-1))^2]\,\}$$

$$K2 = [2\,(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) + 2b^2(c\cos\theta t_n + d\cos\theta(t_n - 1))^2\,) +$$
$$4b\theta(a + b\,t)(c\cos\theta t_n + d\cos\theta(t_n - 1))(-c\sin\theta t_n - d\sin\theta(t_n - 1))]$$

Using upper limits for some trigonometric values in the above equation, we can evaluate an upper bound of M:

$$M \le 4*D*\theta*L*1 + 2*\theta^2*L^2/(2*D^2 - 4*D*\theta*L*1) < 1$$

Convergence condition : $D > 4.3\,|\theta L|$

Where, L = length of the cutter

D = distance between toolpath start point and toolpath end point

The above convergence condition is for R(h(t)) = R. If R(h) is different from R (i.e., other than a cylindrical cutter is used), one needs to add coefficient factors to the above derivative expression. The upper bound of M is a convergence condition for the algorithm to work, allowing solution of 5-axis problems without discretizing the toolpath into a series of short, 3-axis segments to approximate it. Whether or not this convergence condition holds is actually determined by the distance between successive CL points, the length of the cutter, and the rate at which the tool axis orientation changes. There are some situations in which the algorithm will never work properly. For instance, if D = 0 and the angle change is non-zero, then the convergence condition will not be satisified. It is necessary to check the inequality before using the algorithm. But even though the algorithm may converge, it is not the case that the point to which the solution converges is always the

correct solution. The iteration algorithm for equation (5.6.1) does not necessarily produce a global minimum or maximum. Therefore, subdivision of intervals with large angular changes of the tool axis, and using linear solutions to approximate the nonlinear case is necessary to avoid this problem, and it is the approach employed throughout the remainder of this dissertation and in the final software implemented.

5.8.3 Five-Axis Case -- Second Approach

Let's discuss next the situation if we try to do five-axis verification using a toolpath discretized into linear subsegments and calculating the cut value of the surface points based on the three-axis algorithm, but allowing the tool axis to have slightly different orientations at the start and end points of the subsegment. What is the maximum error of this algorithm? Suppose we discretized the toolpath to assure only a small angle change for each step. The angle between the toolpath start point and end point axis orientations is θ , and we call the start point tool axis orientation N(t=0) and end point tool axis orientation N(t=1). If we evaluate the cut value based on the tool axis orientation interpolated linearly to a middle point orientation N(t=0.5), the maximum error of this algorithm will be $L(1 - \cos(\theta/2)) + L \sin(\theta/2)$, where L is the length of the tool.

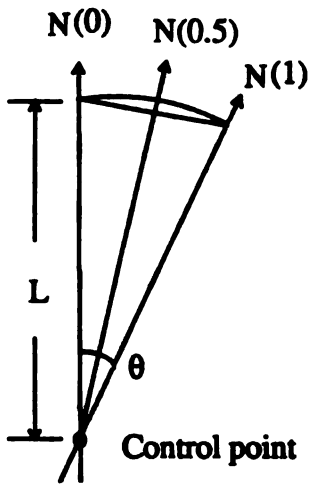

Figure 5.5 Tool axis approximation for the multi-axis case

For a discretized toolpath allowing 1 degree changes in axis orientation between segments, and a cutter of length 1 inch, the maximum error is 10^{-3} inch. This occurs at the top of the cutter (assuming the control point is at the bottom of the cutter). This error is a significant amount in comparison to tolerance specifications which are not uncommonly only fractions of a thousandth of an inch. If we discretize more finely, we can have smaller error. For example, if the toolpath were discretized to have only 0.01 degree steps, the maximum error would be reduced to 10^{-5}L. If one uses the tip of the ball end cutter to mill the surface, and the control point is at the center of the spherical cutter, then there is no error, so fine discretization is completely unnecessary. This is one reason some verification software only verifies toolpaths for ball end cutters, at least for 5-axis milling. But if one needs, for example, to use the side of the cutter to mill the surface in a multi-axis milling machine, the maximum error of this approach for practical levels of discretizations may be too large at the top of the cutter. Therefore, there is a need to use a different approach for a more general verification algorithm.

5.8.4 Five-Axis Case -- Third Approach

The most promising algorithm developed in this work also relies on discretizing multi-axis toolpath segments into several subsegments, so that each segment typically has only a relatively small change in tool axis orientation and cutter extension/retraction. We divide the toolpath segment into several segments, using an upper bound on the error caused by the tool axis orientation change and cutter extension/retraction. For each surface point, we also translate the control points of each subsegment by the amount $h(t = 0.5)$ which is the local coordinate of the surface point on the tool axis $N(t = 0.5)$ in the middle of the subsegment. This reduces the error due both to cutter extension or retraction along its axis and to change in orientation.

Assuming that we examine portions of the cutter with continuous slopes, the error in cut value (which can be seen as error due to angular change plus error due to extension

or retraction of the tool along its axis during the subsegment) has a maximum for this approach at $L(1 - \cos(\theta/2)) + h_{max}\sin(\theta/2)$, where h_{max} is the maximum change in h values of the local coordinate of a surface point on the cutter between two toolpath subsegment points (see Figure 5.6); it is the sum of the retraction/extension distance between adjacent subsegment control points and the sin of the axis angle change between them.

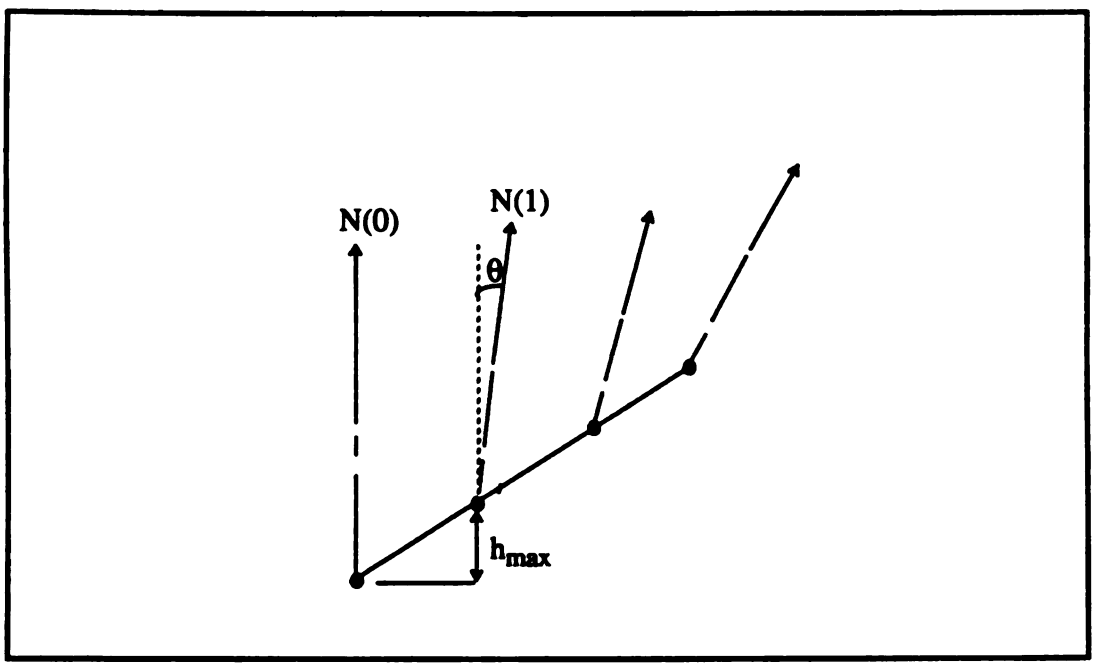

Figure 5.6 Tool retraction and change of orientation of discretized toolpath

This maximum error occurs when the portion of the tool furthest from the control point is cutting. The term $h_{max}\sin(\theta/2)$ replaces the term $L\sin(\theta/2)$ from the second approach described earlier. The maximum error is greatly reduced, since the control point is, in effect, translated for each surface point to the position on the cutter axis offering the least error for the surface point under consideration. For example, if the length of the cutter is 1 inch and the allowable h_{max} is about 0.1 inch (that is, the sum of extension/retraction (in inches) and the sin of the maximum change in angle is about 0.1), then the maximum

error for this approach is about 10^{-5} inches. This error is essentially independent of the cutter length, since the $L(1 - \cos(\theta/2))$ term is very small compared with $h_{max}\sin(\theta/2)$ for realistic situations with θ less than a few degrees. The pseudo code for this algorithm can be expressed as follows:

If it is a multi-axis tool motion, then use a user-specified allowable maximum error (allowable h_{max}) to determine a bound on tool extension/retraction and angular change for the subsegments in the interval $0 \leq t \leq 1$.

Subdivide the toolpath segment into n subsegments accordingly. This yields subsegments for which the maximum error is bounded at $L(1 - \cos(\theta/2)) + h_{max}\sin(\theta/2)$.

```
for ( j = 1 , j <= number of subsegments n, j ++)
  {
    evaluate discretized toolpath point and orientation
    perform voxel space search
    for every surface point in the voxel space
      {
          find the local coordinate h_i(t = 0.5) of that point on the tool axis N(t = 0.5) at
            the midpoint of the segment.
          translate the starting and ending control points for the segment along the tool
            axis by h_i(t = 0.5).
          evaluate approximate cut value according to the three-axis algorithm with the
            new control points and the tool orientation at N(t = 0.5)
      }
  }
```

5.9 The Use of Offset Surface Points for Tolerance Checking

If the approximate cut value is evaluated for the minimum distance between the surface point and the envelope of the tool motion, it provides only a rough test for the toolpath (so it is called "rough verify" in the software). Some undercut and overcut regions may still be undetected due to the discrepancy between using surface normals and distances of the ruled surface from the surface points, as shown in Figure 5.7. In Figure 5.8, the approximate cut value is less than the actual cut value value. If the OUTTOL is specified to be between the lengths $\overline{AB}$ and $\overline{AC}$, then using the minimum distance evaluator from the surface point, one gets a cut within tolerance. But, in fact, it is outside the tolerance region. In Figure 5.9, if the INTOL is specified between length $\overline{AE}$ and $\overline{AD}$, then one gets a cut within tolerance. However, in fact, it is a gouged region. To avoid this situation, the minimum distance evaluator is used instead for points offset from the surface by INTOL and OUTTOL. The approximate cut value in Section 4.4. describes this situation.

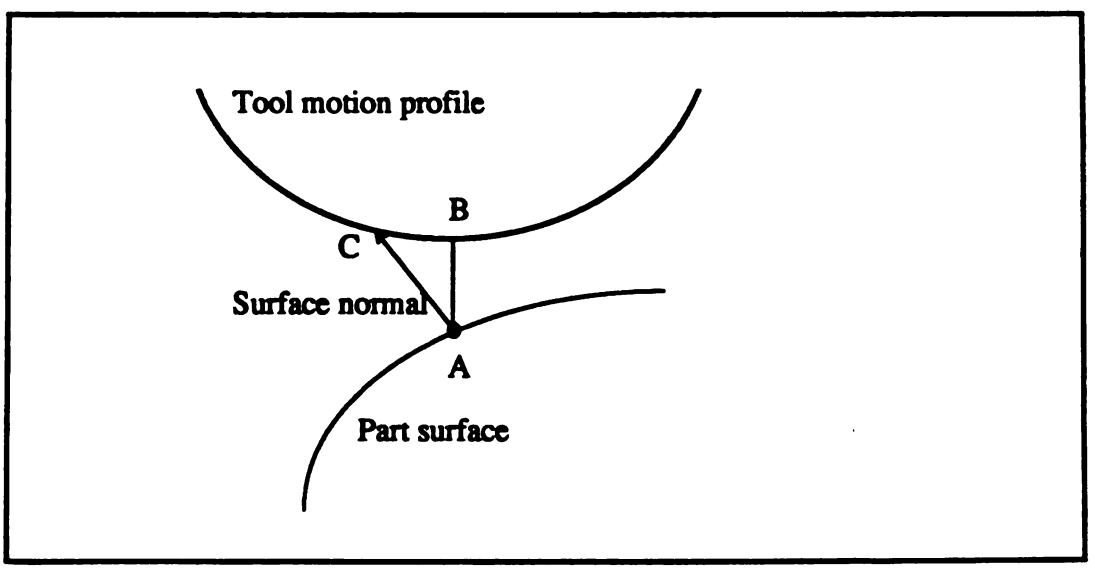

Figure 5.7 Undetected undercut region

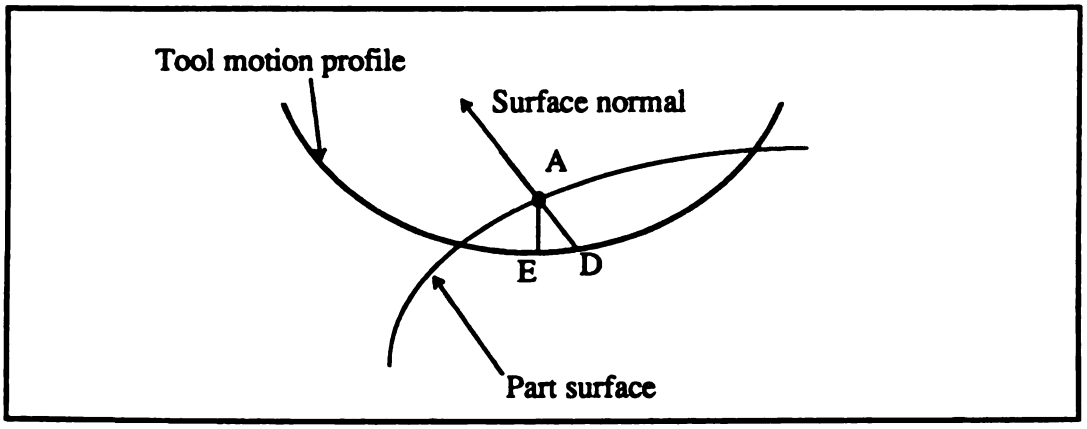

Figure 5.8 Undetected overcut region

```
INTOL and OUTTOL surface points are evaluated
for every surface point
 {
        evaluate approximate cut value of OUTTOL surface point
        evaluate approximate cut value of INTOL surface point
       If ( INTOL_cut_value >= 0 && OUTTOL_cut_value >= 0)
         {
            if (checking_edges && surface point is missed along normal to range of in-
              terest) set cut_value = 99
            else set cut_value = OUTTOL_cut_value + OUTTOL
         }
       else if ( INTOL_cut_value >= 0 && OUTTOL_cut_value =< 0)
            set cut_value = OUTTOL_cut_value + OUTTOL
       else if ( INTOL_cut_value =< 0 && OUTTOL_cut_value =< 0)
            set cut_value = INTOL_cut_value - INTOL
```

```
    else if ( INTOL_cut_value =< 0 && OUTTOL_cut_value >= 0)
     {
         evaluate approximate cut value of the surface point
         if ( surface_cut_value <= 0) set cut_value = surface_cut_value
         else set cut_value = 99
     }
}
```

The approximate cut_value = 99 means the surface point has (so far) always been missed by the tool motion. For surface points for which the surface normal vector is very different from the tool axis orientation, an edge checking process can be used to recognize that this point is missed by the tool motion as shown in Figure 5.9. For the "rough" verification, not using offset surface points, this edge checking would look along the normal to determine whether or not there is an actual intersection with the cutter envelope. But this edge checking requires a great deal of additional calculation effort for many surface points between the OUTTOL and the range of the interest. It is not active in the current code. There is one case in which the INTOL_cut_value is less than 0 and the OUTTOL_cut_value is larger than 0 (see Figure 5.10). In this case, the surface normal is nearly perpendicular to the swept volume of the tool motion. Whether or not this case is detected depends on whether or not the evaluation is made of the cutting of the INTOL point and the surface point when the OUTTOL point is not cut. If these (usually unnecessary) evaluations are also done, then the program can distinguish a gouge (the INTOL point and the surface point are cut) and a cut within tolerance (the INTOL point is not cut, but the surface point is) from the undercut usually indicated by not cutting the OUTTOL point. However, in the current software, this extra checking is not done.

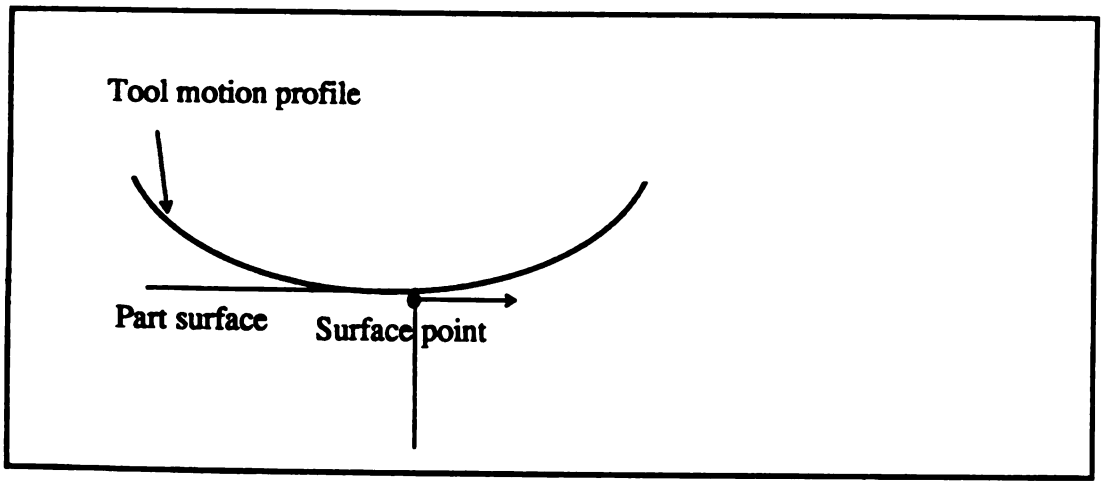

Figure 5.9 Checking edge by tool motion

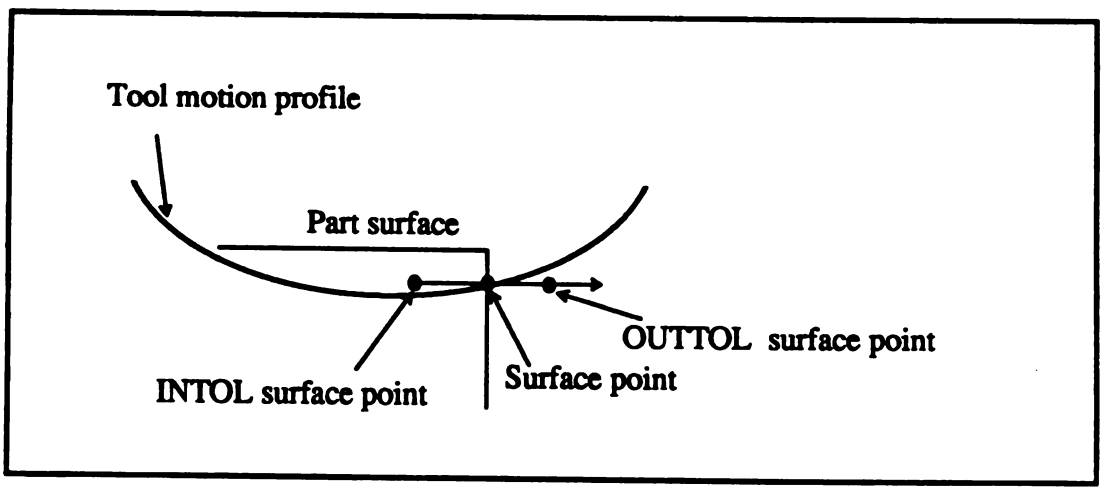

Figure 5.10 Part surface is cut by wrong tool motion

5.10 Circular or Contouring Motions

Although in this verification algorithm the trajectory of the control point in each segment is rendered by linear interpolation between the previous and current position, contouring motions and circularly interpolated motions could be treated easily by discretizing the toolpath into a number of subsegments, with the number dependent on allowable error bounds. For example, if the control point moves on a 1-inch-radius circular arc of 45 degrees from a previous position to the current position, then if the toolpath is discretized into 0.5 degree subsegments, the maximum error (chordal deviation) from the linear toolpath is about 10^{-5}inches. Similar techniques would suffice for parabolic motion, etc. The penalty for simulating additional subsegments is typically not unmanageably large, since circular motions on large arcs may not occur often in the toolpath. This approach eliminates the need to write additional evaluation routines to treat these cases.

5.11 Correctness test of the program

It is said that good programmers make errors. The software development based on this algorithm needs to tested to be error-free before it can be relied upon as a production tool. Some testing of the software has already been conducted using both industrially-supplied data and test parts produced by a solid modeling package. One process used for testing the NC verification program was to create channel-style solids with surfaces which match the theoretical shape of the tool swept volume for a simple toolpath, as shown in Figure 5.11. To easily visualize the tool motion and see the color image display, it is easiest to use half channels, since the algorithm uses the cutter symmetrically, anyway. Therefore, half channels can test the correctness of the software. Output of test runs made with half channel surfaces are shown in Chapter 6, together with other sample outputs.

A half channel was created with the I-DEAS solid modeler and written to a file as NURBS surfaces in IGES type 128 format. For this purpose, the untrimmed surfaces at

the ends of the channels needed to be removed. Several such test channels were created for testing 3-axis toolpaths. However, this modeling package was not used to create a swept volume representing a tool moving in 3-D space with a changing tool axis orientation. For this case, instead of the swept volume, one can generate a skin group volume by sweeping the tool profile along a generatrix. However, this skin group object is not equal to the swept volume of the tool motion since, for example, rotating the tool forward or backward causes different tool profiles to become the active cutting regions.

The tests of the software were divided into two major parts: (1) the half-channel tests, which let the tool move from the start point to end point with no tool axis change. This is a general three-axis tool motion. But the tool axis need not always point to the coordinate x-axis , y-axis or z -axis. The test was used not only to prove that the tool sweep is an exact fit to the half channel, but also, using different cutter diameters or offset tools, to evaluate gouge and undercut cut values to verify that the display is what we expected. As discussed in Chapter 6, the undercut and overcut regions were what we expected, so it increases our confidence that this software can verify the toolpath correctly for general three-axis tool motions.

(2) The next step was to use a tool profile to generate a half channel which only roughly approximates the shape of the tool swept volume, for a tool motion in five axes. To do this channel-creation operation by sweeping a single tool profile guarantees that there will be errors between the swept volume and the walls of the channel. However, such channels are still useful, because one can calculate the expected deviation at various points, and compare those values with the cut values calculated for those points by the software. After creation of the channel, it was rotated and translated it in 3-D space, to avoid accidental alignments with axes, which may tend to hide errors in the software. It then became a primitive model for five-axis tool motion, using a similarly translated and rotated toolpath. But the bottom of the tool collided with the half channel, as it should, when we

used the real tool to simulate it. This reflects the difference between the surface of the swept volume and the skin group (channel surface).

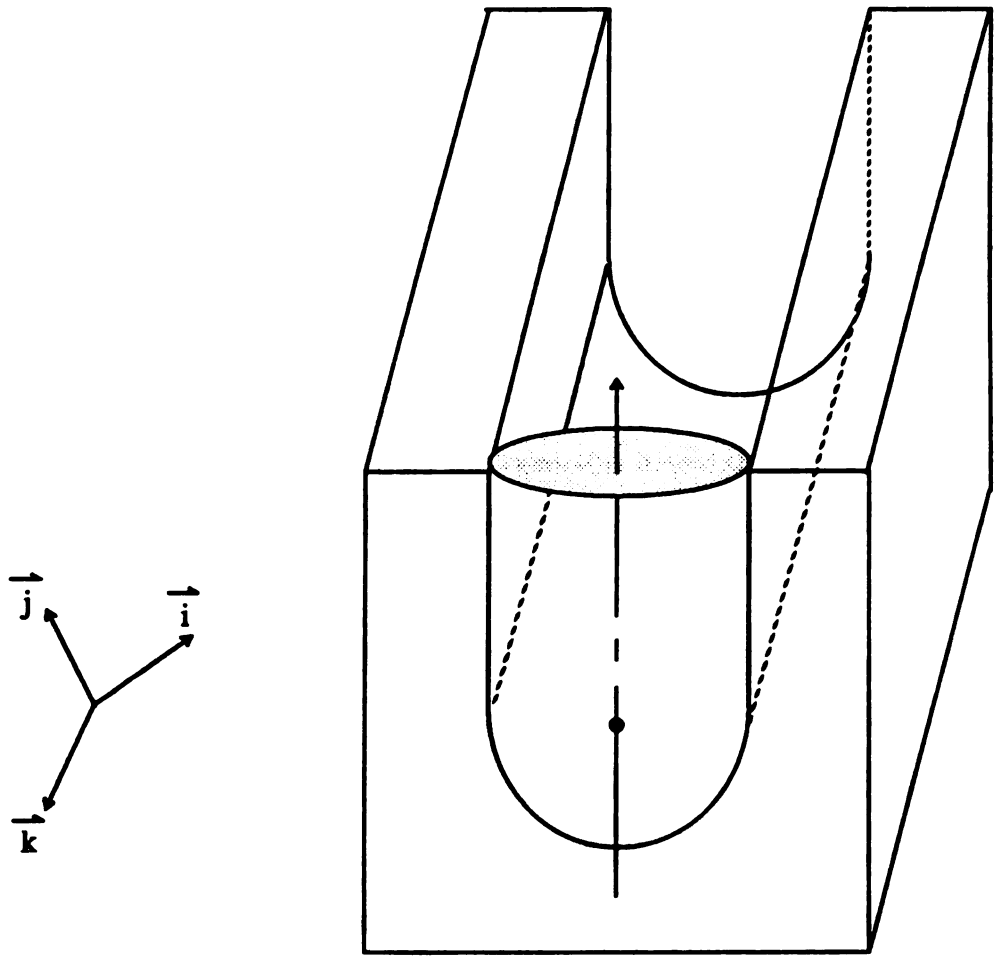

Figure 5.11 Solid model for test of three-axis milling process

The cut value after the tool axis was rotated was checked with the "query cut value" command, which prints the cut value of the surface point. The analytically calculated cut values matched the cut values evaluated by the program. The side surface of tool still fit the channel relatively closely, and was checked visually. Since the software passed these tests,

it is plausible that the software can verify more general five-axis tool motions correctly, since the basic theory appears to be correct and correctly implemented.

5.12 Efficiency Analysis of the NC Geometric Verification Algorithm

This section presents an efficiency analysis of the algorithm. The computational complexity or time complexity is measured based on order of complexity or "Big Oh" analysis. The result of such an analysis is a function called the time complexity of the algorithm, t(n), which is proportional to some function of the size of the input data "n". We define the worst case for complexity t(n) by an inequality $\forall\ n \geq n_0$, $t(n) \leq cf(n)$, where n and n_0 are natural numbers, c is a positive real constant, and f(n) is called the "growth rate". In other words, O(f(n)) is the set of all functions t(n) bounded above by a positive real number multiple of f(n), provided that n is sufficiently large (greater than some threshold n_0).

For the purpose of the analysis, the NC geometric verification algorithm may be depicted in "pseudo-code" as below. The algorithm is shown broken down into nested tasks. The time complexity of each of the tasks is denoted by the term "O(f(n))", where f(n) is the growth rate of each task. The term O(1) indicates that the task has constant order time complexity; *i.e.* , it is independent of input size. Each task in the algorithm will be examined individually to determine the growth rate of the algorithm as a whole.

For every CL-point from 1 to N

{

Calculate the parameters for the tool axis motion, implicitly defining a ruled surface. O(1)

Further discretize the toolpath segment if necessary, if the tool axis changes within the segment. O(1)

Determine which voxels may contain this toolpath segment. O(1)

For every surface point in every voxel possibly interfering with this toolpath seg-

ment

{

Simulate the motion along the toolpath and get the cut value for the surface point. O(1)

Update the cut_value in the data structure of POINT.

}

}

The function "calculate the parameters for the tool axis motion" involves calculations related to defining a ruled surface. The function reduces to a calculation of some constant, independent of any other structures; therefore it is of constant order time complexity.

The functions which subdivide the toolpath for multi-axis motions and which search voxel space are of constant order since they are independent of the number of CL points in the toolpath. The voxel search algorithm is actually also independent of the extent of the part surfaces loaded, since it is performed only in the local area immediately surrounding the tool swept volume for the motion being simulated. The main NC simulation function is to find the approximate cut value. In this function, the approximate cut value is evaluated according to the simplified optimization problem described above. For any tool motion, this is of constant order time complexity (it is independent of the number of other CL points) and can be bounded. Thus, since each of the tasks within the outer loop is of constant time complexity, the growth rate for the entire NC geometric verification algorithm is proportional to the number of times the statements in the outer loop are executed. Thus, it is of order O(N), where N is the number of the tool path points.

5.13 Computational Error

Some amount of error in the computations described here is evitable, since the they utilize floating point representations. For example, if a flat end cutter is milling along a

plate, a point at the tip of the cutter might be supposed to be evaluated as h(t) = 0, but in fact, be represented as $h(t) = -10^{-10}$. Depending on how the logic of the program is implemented, this might cause either a minor computational error, or a more important logical error. In some cases, interval calculations, rather than strict equalities or inequalities, can help to avoid logical errors. For example, for the situation just mentioned, for a negative h, we ordinarily would not invoke the evaluation function involving the cylindrical surface of the tool. Then this point could not be cut by the tool motion. Actually, it should probably be calculated as being cut within tolerance by this tool motion. To avoid this problem, it is necessary to perform interval calculations, defining an epsilon to set the boundary of the tool motion. Of course, the actual value of epsilon depends on whether float or double precision is used, on the magnitudes of the variables involved, etc.

5.14 Direct Dimensional NC Verification

In direct dimensional NC verification, the cut value is directly evaluated as described in Appendix B. The purpose of NC verification is not only to determine toolpath errors, but also to do so efficiently. The part of the equation shown below under the square root is not a second-order equation in the multi-axis milling case, even if the toolpath motion is linear, since H(t) depends implicitly on N(t), the axis orientation. Thus, if it were necessary to find the actual cut value by minimizing the full equation, the problem would be very complicated, multimodal, and could not be solved by simple interpolation.

$$s(t) = \frac{-H(t) \pm \sqrt{H^2(t) - 4*J(t)*P(t)}}{2*J(t)}$$

CHAPTER VI

RESULTS AND EXAMPLES OF PERFORMANCE OF THE ALGORITHMS

6.1 Application of the NC Verification

In Chapters Three, Four and Five, a new algorithm for multi-axis NC verification is presented. This chapter deals with the application of the software based on the algorithms to a series of test problems and more realistic models of manufactured part surfaces. For verification of NC milling, a color-shaded image output is displayed. First, the structure of the NC verification program is illustrated. Figure 6.1 presents the general ideas from the loading of the surfaces and toolpath data to final color image display. Tradeoffs between high-accuracy of the shaded color image and the conflicting goal of speedup of computation time are mentioned. Next, some testing of the software which has already been conducted using test solid model parts produced by a solid modeling package is discussed. This method was used for testing the program before it was tested on real industrial parts. Then some data from automotive applications for three-axis milling and additional applications for five-axis milling demonstrate how the algorithm could be applied in an industrial CAM environment. Examples are shown to illustrate that the parts can be displayed from abitrary viewpoints without needing to recalculate the verification.

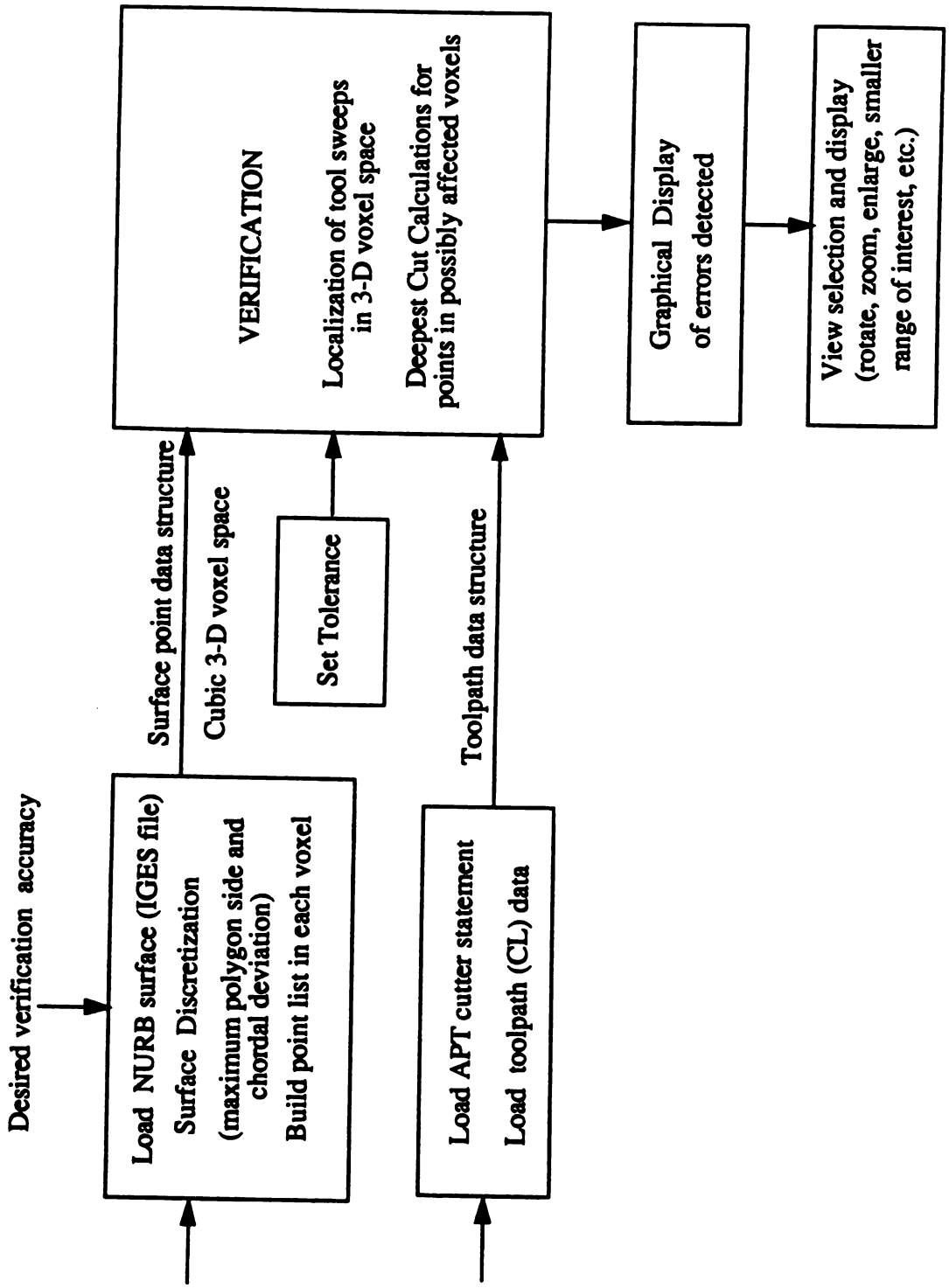

Figure 6.1 Overview of verification system

6.2 Solid Model Examples

The first step in testing the capability of the NC verification program was to create channel-style solids with surfaces which match the theoretical shape of the tool swept volume for a simple toolpath. A flat-end cutter and a fillet-end cutter example are shown in Figure 6.2 and Figure 6.3, respectively. The single toolpath was used with different cutter diameters or with offset tools, to produce desired gouge and undercut cut values, in order to verify that the display produced matched what was expected. There are also fillet-end cutter and flat-end cutter examples, which are shown in Figure 6.4 and Figure 6.5, respectively. Figure 6.4 displays the gouging produced when the fillet-end cutter diameter was increased by 0.0136 inch. Figure 6.5 shows the undercuts produced when the flat-end cutter diameter is decreased by 0.0136 inch. The correctness of these examples shows that the software meets at least minimum requirements for verifying three-axis motion.

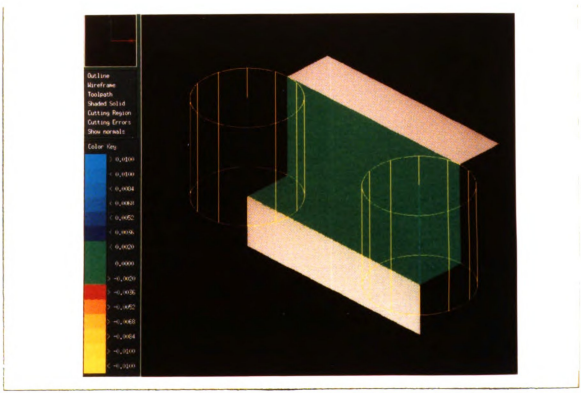

Figure 6.2 Fitting test of flat-end half channel

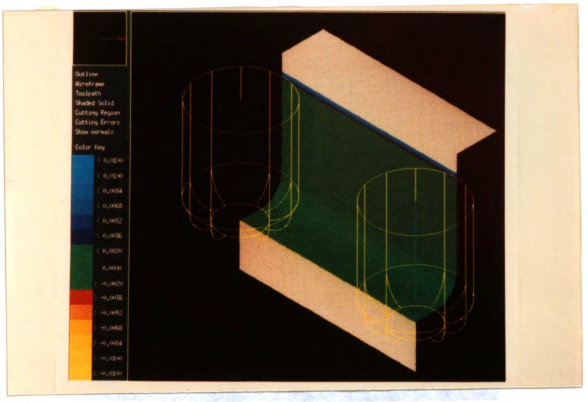

Figure 6.3 Fitting test of fillet-end half channel

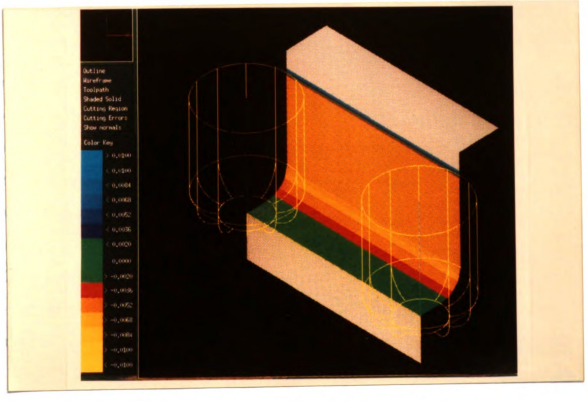

Figure 6.4 Overcut testing of fillet-end half channel

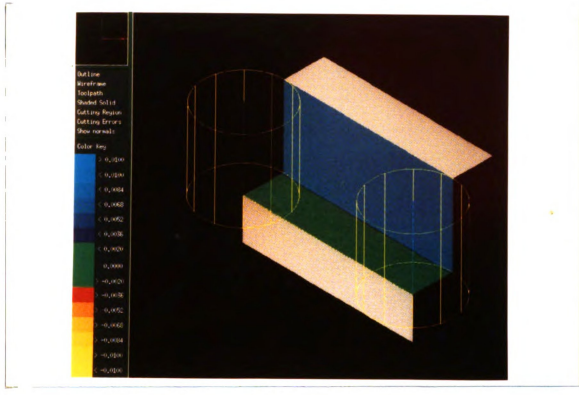

Figure 6.5 Undercut test of flat-end half channel

Testing the four- or five-axis tool motion is not as easy as testing three-axis tool motion. With the solid modeling software (I-DEAS^{TM}), it was difficult to create a channel with geometry exactly matching the boundaries of the tool swept volume. Instead, a single profile of the tool was swept through a 5-axis motion and skinned. Therefore, we are testing to see that the discrepancies found are what is expected, rather than checking for an exact match. One example using a ball-end cutter to sweep the side of a half channel is shown as in Figure 6.6. In this example, the axis of the cutter was gradually tilted forward 10 degrees during the motion. Because this was only a four-axis move, and was restricted to the plane of the motion, the fit is exact on the side surface, but not on the bottom. Another example also using a ball end cutter to sweep the side of a half channel, is shown as in Figure 6.7. In this case, the axis was tilted 10 degrees in the direction of motion and 10 degrees normal to that direction. In these multi-axis tool motions, the tool collides with

the half channel due to the differences between the surface of the swept volume and the skin group (channel surface).

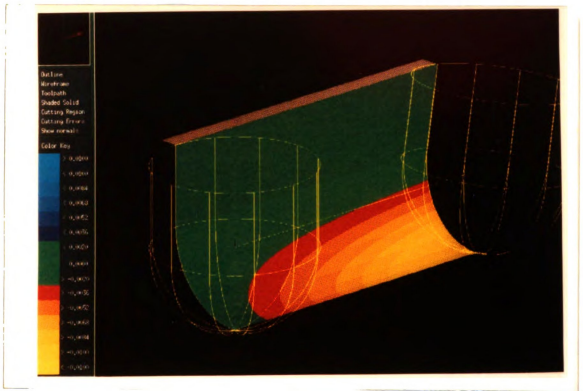

Figure 6.6 4-axis test of ball-end half channel

6.3 Application Examples

This section deals with the application of the algorithm to realistic industrial parts. Two final image displays are described. First, the two application examples are presented to demonstrate the capabilities of the three-axis NC verification algorithm. The first example manufactured part an automobile wheel (data furnished by CIMLINC). With discretization parameters set at: chordal deviation = 0.1 mm, max_s = 1.0mm, max_t = 1.0mm,

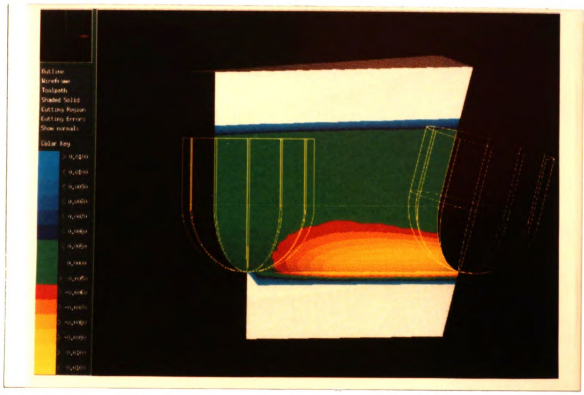

Figure 6.7 5-axis test of ball-end half channel

the verification system discretized the surface into 399,039 points. The toolpath consisted of 6621 CL points. The CPU time verifying the surface points along the normal direction for this case was 26 minutes and 45 seconds on a Sun SPARCstation 2. For this example, INTOL and OUTTOL were 0.2 mm, and the range of interest, R_{int}, was 1.0 mm. The output shows cusps (undercuts) above the wheel surface which slightly violate the specified outside tolerance, perhaps caused by the toolpath step size, as shown in Figure 6.8. To see the detail of the part and to avoid the aliasing of the color image, a zoomed picture for this example is necessary. The final verification image can be easily zoomed, rotated, or evaluated roughly for other tolerance limits without rerunning the whole verification program. Zooming is illustrated in Figure 6.9.

Figure 6.8 NC geometric verification of an automobile wheel

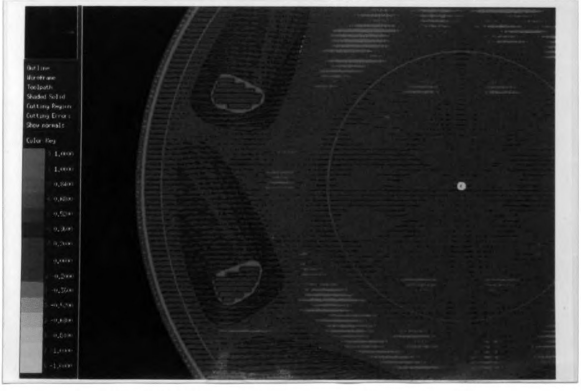

Figure 6.9 Zooming in on an automobile wheel surface

The second example of a manufactured part is a frequently employed benchmark surface set, also supplied by CIMLINC. For values of the discretization set at: chordal deviation = 0.01 mm, max_s = 0.5mm, max_t = 0.5mm, the verification system discretized the surface into 299, 962 points. The 3-axis toolpath consisted of 21,758 CL points. It is cut with a ball-end mill. The CPU time verifying the surface points along the normal direction for this case was 64 minutes and 2 seconds on a Sun SPARCstation 2. For this example, INTOL and OUTTOL were 0.03 mm, and the range of interest, R_{int}, was 0.15 mm. The output shows cusps (undercuts) and gouges (overcuts) in numerous places, using an example toolfile generated for test purposes, as shown in Figure 6.10. An actual part cut using that toolpath was examined visually, and it exhibited the same cusps in the same locations as are shown in that figure. The other side of this part is displayed, without further verification calculations, in Figure 6.11.

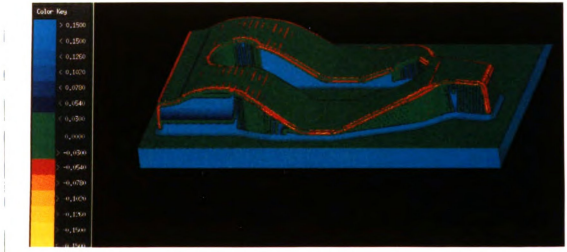

Figure 6.10 NC geometric verification of a benchmark part

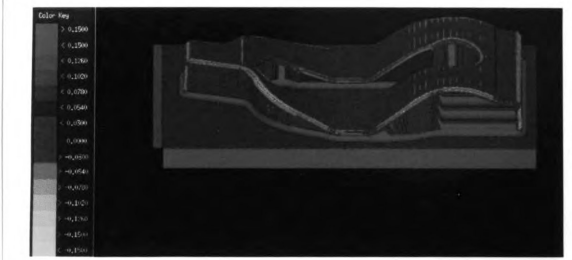

Figure 6.11 Different view of a benchmark surface

Two additional application examples are presented to demonstrate the capabilities of the multi-axis NC verification algorithm. The third example is a five-axis ball-end cutter milling operation on a turbine blade. Using a maximum chordal deviation of $5x10^{-5}$ in, the verification system discretized the surface into 9750 points. The toolpath consisted of 1742 CL points. The CPU time verifying this case was 6 minutes and 31 seconds on a Sun 4/ 110. The shaded image is shown in Figure 6.12. For this example, INTOL and OUTTOL were 0.001 in, and the range of interest, R_{int}, was 0.01 in. The output shows cusps (undercuts) violating the specified outside tolerance, indicating that the tool used to cut the surface was too small in radius or the stepover selected for toolpath generation was too large, for example.

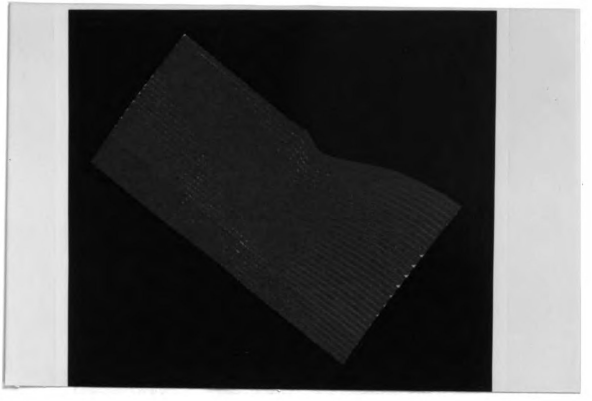

Figure 6.12 NC Verification of a Turbine Blade

The fourth example is a five-axis ball-end cutter milling operation on a sculptured surface. The distance evaluator between the surface point and boundary of the tool distinguishes whether the surface point is intersected by the cylindrical side surface or the bottom spherical surface of the tool. Using a maximum chordal deviation of 0.001mm, the verification system discretized the surface into 10345 points for verification. The toolpath consisted of 2512 CL points. The CPU time to verify this case was 8 minutes and 22 seconds on a Sun 4/110. The shaded image is shown in Figure 6.13. The errors shown are the result of deliberately choosing a tolerance tighter than that provided by the number of CL points used for cutting the surface. For this example, INTOL and OUTTOL were 0.05mm, and the range of interest, R_{int}, was 0.25mm. The Mach bands are due to the restricted color map available to such artificially pseudo-colored images on an 8-bit color display.

Figure 6.13 NC Verification of a Sculptured Surface

CHAPTER VII

SUMMARY AND CONCLUSIONS

7.1 Summary

The surface-based verification software developed by using this algorithm is useful for verifying multi-axis NC machining programs with APT cutters. The system first reads standard IGES files defining desired part geometry,and CLfiles defining the toolpath. After the surfaces are discretized, a voxel space is used to group surface points. Toolpath segments are also transformed into voxel space, localizing them in 3D working space, so that only the surface points possibly affected by a given tool motion will be used for further computation. The distance calculation between the surface points and tool swept volume can be done without explicited creating the surface boundary of the tool motion (the tool swept volume). The verification process is quite practical for verifying, for example, complex parts (hundreds of surfaces) of a size on the order of one meter square with toolpaths of several thousand CLpoints. The output display image uses color coding to show the areas cut within tolerance and out of tolerance in different colors, and runs on arbitrary X Window systems. Also it is view-independent; therefore, it allows the user to do detailed checking of the verification results without repeating intense calculations, which is an advantage of our view-independent, surface-based system. The algorithm is very accurate at the specified tolerance limits, and provides useful approximate depths of undercut and overcut outside the tolerance range. The results may be used to adjust the

toolpath based on this approximate cut value to eliminate overcut or undercut regions, or to generate additional toolpaths using smaller tools to complete milling of undercut regions not cuttable by the original tool.

7.2 Limitations of the Algorithm

The resolution of the surface discretization process is a very important factor in the accuracy of the final output. If the surface points are not dense enough, both cusps and gouges may be undetected by the verification algorithm. Too large a toolpath stepover will cause a cusp, which might be undetected as shown in Figure 7.1.

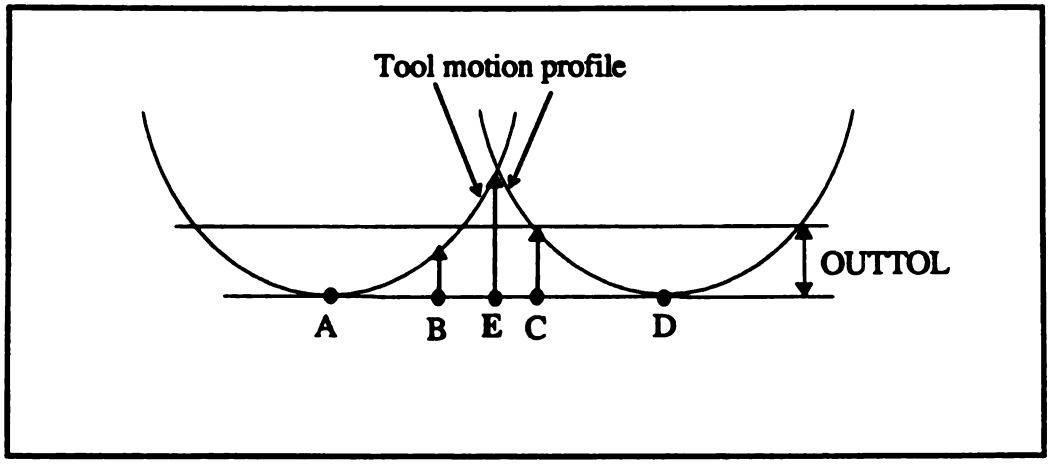

Figure 7.1 Undetected cusp error

Points B and C are cut within the tolerance; therefore, the interpolation algorithm is utilized between them. The shaded pixels between B and C are shown as cut within tolerance; but, in fact, there are some out-of-tolerance regions. This error can be solved if one adds some surface points between point B and point C; for example, if point E is created, then the undetected error is eliminated. A gouging error due to a small radius of curvature in the tool geometry is shown as in Figure 7.2. The tool curve protrudes into the surface, and an

undetected gouging area occurs between these two points. This also can be eliminated if one adds some points between these two surface points. Generally speaking, the denser the surface points, the better the results. But more surface points not only cause potential problems of virtual memory limitation, but also slow down the verification speed. Therefore, how to decide the maximum distance between points to efficiently eliminate undetected cusps and gouges is an important issue here. Although it has not been done here, it may be possible for a given discretization, to determine bounds on the magnitude of undetected errors, given adequate characterization of the surface curvature, tool shape, and required tolerances. This information, in turn, can be used to determine the level of discretization required to assure conservative operation of the verification software (i.e., no tolerance violations are missed, subject to certain conditions).

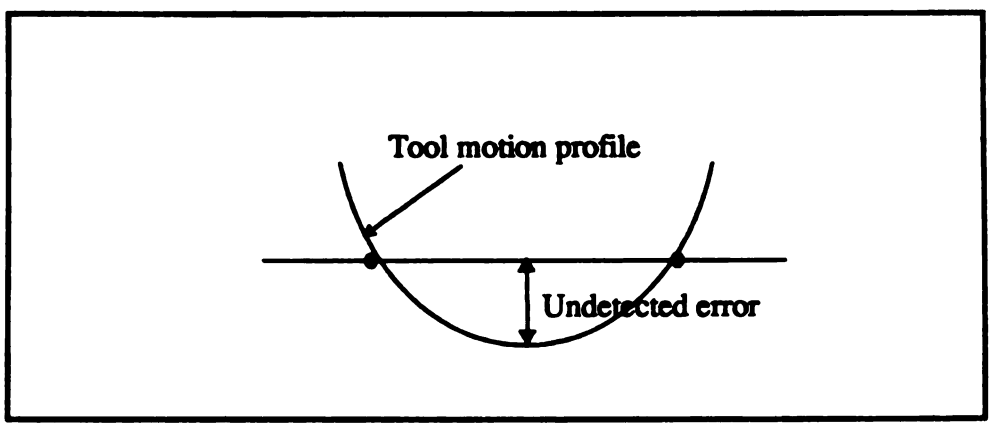

Figure 7.2 Undetected gouge error, due to tool geometry

7.3 Comparison with other NC verification systems

The algorithm provides 5-axis NC verification for a general APT cutter. A major difference between this algorithm and a solid-modeling-based system is that each surface point has a local coordinate function for each tool motion, thus reducing the effort needed to calculate the intersection of a surface point and a tool sweep model. The NC verification

system described here behaves with a time complexity of O(k), where k is the number of CL points. However, if the level of surface discretization required is also viewed as varying with the same parameters which determine the number k of CL points, then a higher than linear complexity results; in the current software, these are specified independently, so O(k) complexity results. In contrast, a solid modeling system needs to compute the boundary of each tool motion or combine the boundaries for all the tool motions, then find the possible intersection between the desired part surface and the boundary of the tool motion. This makes the problem very complicated and time-consuming.

Image-based systems use a Z-buffer to find the possible intersection between the desired surface and the tool swept volume. These image-based algorithms are faster than the classical solid modeling approach, but the results are view dependent, have limited resolution, and cannot calculate true positional tolerance (offset surface) violations, so they cannot totally replace the CSG or other methods. The data structure resulting from the algorithm in this paper is a surface-based description, not an image-resolution data structure. The final part at the completion of verification can readily be redisplayed from another viewpoint. The out-of-tolerance regions can be traced to specific toolpath segments for modification of the machining programs. User-variable in- and out-tolerances may be used for adjustment of toolpath segments. The system does provide a way to work cost-effectively in verification of NC programs.

7.4 Future Research

The mathematical basis presented in this dissertation for representing swept volume can also be used in other applications. Obviously, it is useful for interference checking of robot manipulators and their working environments. The voxel space concept can be used for rough interference detection of two or more machining process working in the same workspace. Then the mathematical model can be used for more precise intereference detection [47]. Future research goals include extending verification to include tool wear,

force calculation for tool deflection, and automatic generation of multi-axis toolpaths for general APT cutters. The generation of error-free multi-axis toolpaths for general APT cutter will be a very challenging problem, especially when one needs to use side surfaces of cutters to mill sculptured surfaces.

APPENDICES

APPENDIX A

DEFINITION OF THE APT CUTTER

The statement defining a cutter in the APT language usually has seven parameters and it is as follows (Figure A.1) [48]:

CUTTER/d, r, e, f, a, b, L

where

d = the diameter of the circle generated by the intersection of the end and the side surfaces of the cutter.

r = the radius of the corner circle.

e = the distance from the corner circle center to the cutter center line.

f = the distance from the corner circle center to the plane passing through the cutter end center and perpendicular to the cutter center line.

a = the angle (degrees) between the cutter end surface and the plane perpendicular to the cutter axis (it lies in the range of $0 \leq a < 90$)

b = the angle (degrees) between the cutter side surface and the cutter axis (it lies in the range of $-90 < b < 90$)

L= the cutter height measured from the control point of the cutter.

Note that usually the corner circle should be tangent to the end and side surface.

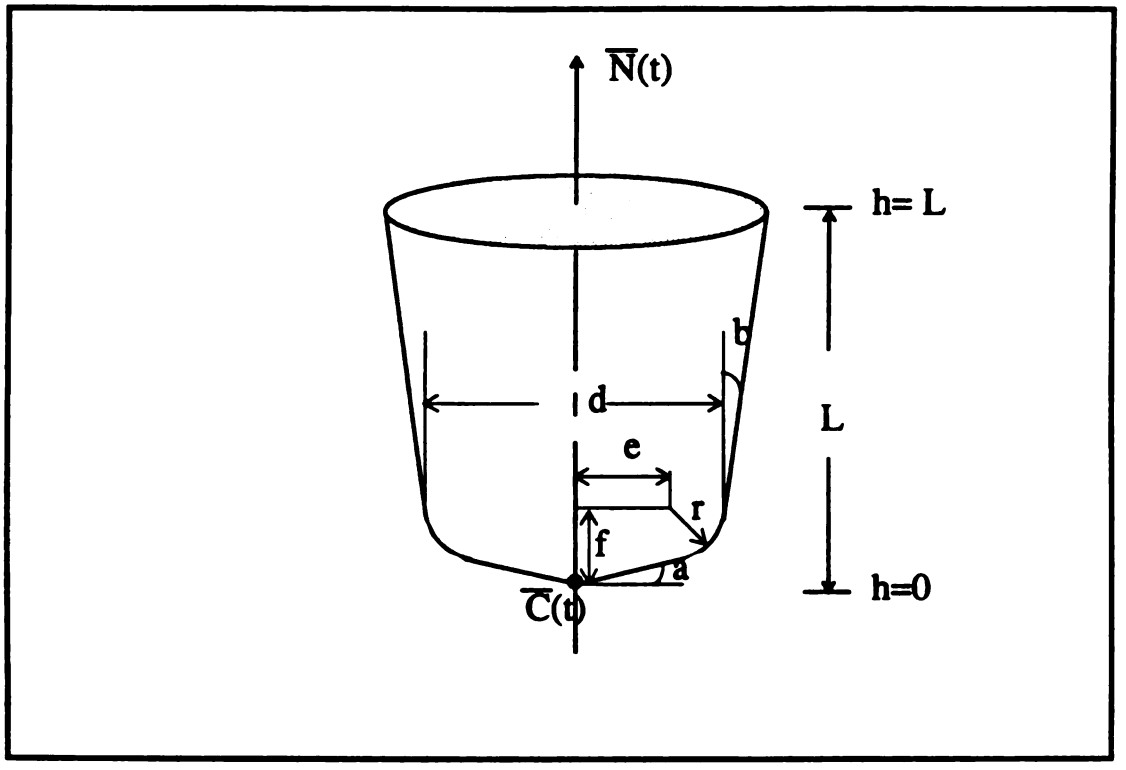

Figure A.1 Standard APT seven-parameter cutter

For this seven-parameter cutter, the radius of the cutter can be defined in terms of three linear functions $R_1(h)$, $R_2(h)$ and $R_3(h)$, and according to simple geometric analysis, can be expressed as a function of position (height) along the cutter axis from $\overline{C}(t)$ as follows:

For $0 \le h \le f - r*\cos(a)$, $\quad R_1(h) = h/\tan(a) \quad (a \neq 0)$

For $f - r*\cos(a) \le h \le f - r*\sin(b)$,

if $h > f$ then $h_1 = h-(f-r)$

if $h \le f$ then $h_1 = r-(h-f)$

$$R_2(h) = e + (2*h_1*r - h_1*h1)**0.5$$

For $f - r*\sin(b) \le h \le L$, $\quad R_3(h) = d/2 + (h - d/2*\tan(a))*\tan b$

APPENDIX B

INTERSECTION OF A VECTOR WITH A SWEPT VOLUME

Part 1: Intersection of a surface normal with a side surface of a swept volume

The intersection between a surface normal vector and a side surface (*i.e.* any surface but a flat bottom surface) of a tool swept volume is shown as in Figure B.1:

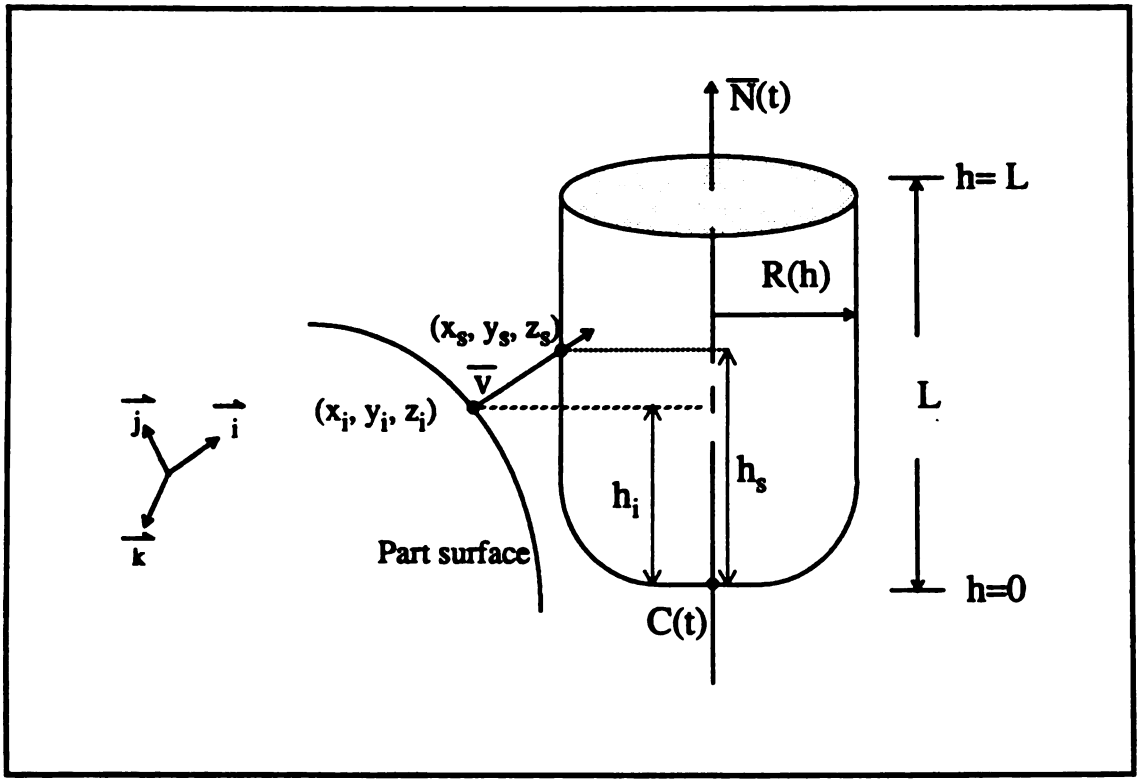

Figure B.1 Intersection of a surface normal vector with a side surface of a tool swept volume

Given a surface normal unit vector $\bar{v}$ and the coordinate (x_i, y_i, z_i)

$$\bar{v} = \alpha\vec{i} + \beta\vec{j} + \gamma\vec{k}$$

Assume that (x_s, y_s, z_s) is an intersection of with a side surface of the tool swept volume; it can be expressed, using the parametric form of the surface normal unit vector, as:

$$\frac{x_s - x_i}{\alpha} = \frac{y_s - y_i}{\beta} = \frac{z_s - z_i}{\gamma} = s$$

$$\text{or} \quad x_s = x_i + \alpha s \ , \quad y_s = y_i + \beta s \ , \quad z_s = z_i + \gamma s \qquad \text{(B.1)}$$

Since the surface normal vector $\bar{v}$ may intersect any part of the swept volume model, the first step in solving this problem is to determine which part of the model may be intersected by the surface vector. Let's first discuss the most general case treated in this work -- the intersection of the surface normal vector with the side surface of the general APT seven-parameter cutter.

For a given surface point (x_i, y_i, z_i), a surface normal unit vector $\bar{v}$ and a tool motion $(\bar{C}(t), \bar{N}(t))$, $0 \le t \le 1$, the minimum distance from the surface point along the normal to the envelope of the tool motion can be derived by the following expression:

The surface point (x_i, y_i, z_i) has local coordinates $(t, h_i(t))$. The intersection point (x_s, y_s, z_s) is a point on the surface of the moving rigid body model. If its corresponding h-axis value is $h_s(t)$, $0 \le h_s(t) \le L$, then we say that the local h coordinate of the point falls within the length bounds of the cutter. The minimum distance between the intersection point and the ruled surface is the length of vector joining the intersection point and the ruled surface point $\bar{r}(t, h_s(t))$, *i.e.*, $|(x_s, y_s, z_s) - \bar{r}(t, h_s(t))|$. Thus, whether an intersection point is on the envelope of the tool motion can be simplified to whether this intersection point is on a circle with center $\bar{r}(t, h_s(t)) = ([x(t) + h_s(t)N_x(t)], [y(t) +$

$h_s(t)N_Y(t)]$, $[z(t) + h_s(t)N_Z(t)]$) and radius $R(h_s(t))$.

The requirement that the intersection point (x_s, y_s, z_s) be on the circle can be written as:

$$[x_s - x(t) - h_s(t)N_X(t)]^2 + [y_s - y(t) - h_s(t)N_X(t)]^2 + [z_s - z(t) - h_s(t)N_Z(t)]^2 = R^2(h_s(t))$$

$$\text{or } [x_s - x(t)]^2 + [y_s - y(t)]^2 + [z_s - z(t)]^2 - h_s^2(t) = R^2(h_s(t)) \quad \textbf{(B.2)}$$

Recalling the expression for hi(t) from Equation 3.2, the relationship between $h_s(t)$ and $h_i(t)$ can be shown as follows:

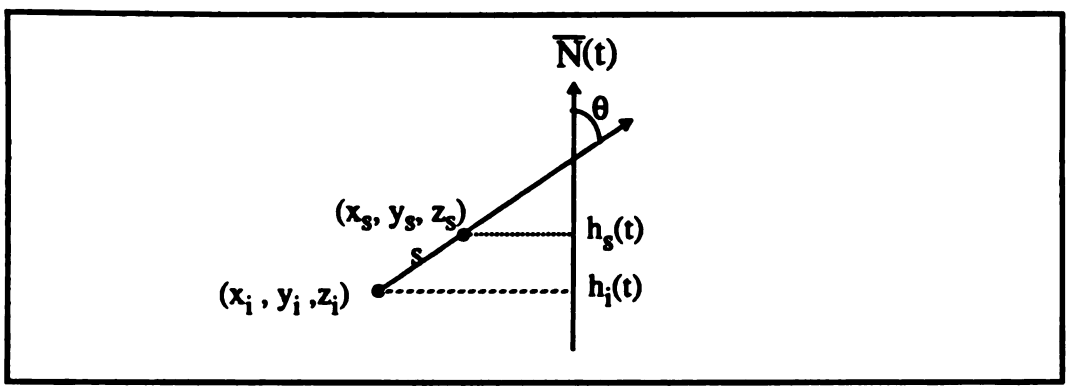

Figure B.2 Relationship between $h_s(t)$ and $h_i(t)$

From the above figure, we know that $h_s(t) = h_i(t) + s*\cos\theta$

$$\text{or } h_s(t) = h_i(t) + s[\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)] \quad \textbf{(B.3)}$$

Applying parametric forms **(B.1)** and **(B.2)** to **(B.3)**

$$[x_i + \alpha s - x(t)]^2 + [y_i + \beta s - y(t)]^2 + [z_i + \gamma s - z(t)]^2 - h_s^2(t) = R^2(h_s(t))$$

or, collecting coefficients of like powers of s,

$$J(t)*s^2 + H(t)*s + P(t) = 0, \quad \textbf{(B.4)}$$

where: $J(t) = 1 - [\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)]^2$

$$H(t) = 2\alpha[x_i - x(t)] + 2\beta[y_i - y(t)] + 2\gamma[z_i - z(t)] - 2h_i(t)[\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)]$$

$$P(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_s(t))$$

Then the cut value, that is, the minimum distance from the surface point along its normal unit vector to its intersection point with the moving solid model, can be obtained from the quadratic formula as:

$$s(t) = \frac{-H(t) \pm \sqrt{H^2(t) - 4*J(t)*P(t)}}{2*J(t)} \qquad \textbf{(B.5)}$$

The t value for the minimum s(t) can be obtained from $\frac{\partial s}{\partial t} = 0$ **(B.6)**

To speed up the program, one can compare the values s(0) and s(1) with the s(t) value from **(B.6)**. Thus, there is no need to get the $\frac{\partial^2 s}{\partial^2 t}$ in order to judge whether or not the t value represents a minimum or maximum s(t).

This quadratic in s is of higher order in t, and cannot be solved in closed form, even if a relatively simple form for the tool axis rotation functions $N_x(t)$, $N_y(t)$, and $N_z(t)$ is specified, so long as the tool axis is not held in a constant orientation (3-axis milling). Instead, we would typically employ a root finding algorithm if we were to implement this approach directly. The running time for a process based on such an algorithm would be quite long. We can restate the situation as a simple optimization problem as follows:

$$\text{Minimize: } s(t) = \frac{-H(t) \pm \sqrt{H^2(t) - 4*J(t)*P(t)}}{2*J(t)}$$
$$t \in R$$

subject to :

$$0 \leq t \leq 1$$

$$0 \leq h_s(t) \leq L$$

The minimum s value for $0 \leq t \leq 1$ which satisfies equation (B.4) is the cut value of the surface point. If $H^2(t) - 4J(t)*P(t) < 0$, the surface vector misses the side surfaces of the model. If two intersections with the side surface of the model are found, and $0 \leq h_s(t) \leq L$ for both solutions, then the closer of the two intersections defines the cut value. However, if no intersections with the side surface are found, or a single intersection with the side surface is found, and if the tool has a horizontal end surface, then the end surface must be tested for an intersection. That is, the intersection of the surface normal with a circle in the h = 0 plane normal to the cutter axis needs to be considered.

Part 2: Intersection of a surface normal with a circle in a plane

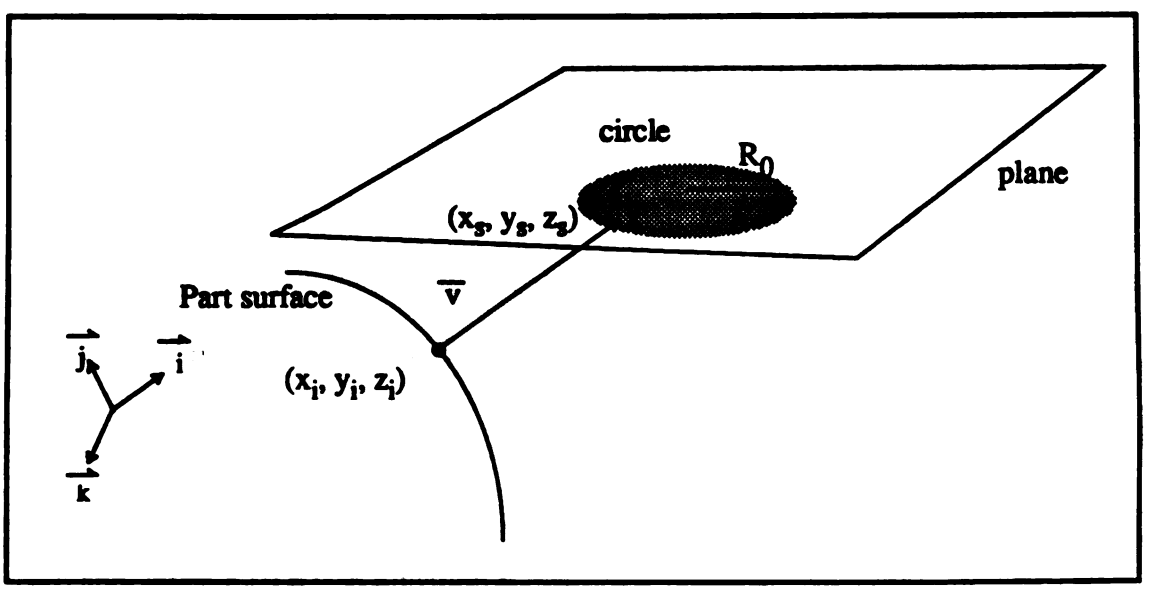

Figure B.3 Intersection of a surface normal vector with a circle in a plane

Given a surface normal unit vector $\bar{v}$ and the coordinate (x_i, y_i, z_i)

$$\bar{v} = \alpha\vec{i} + \beta\vec{j} + \gamma\vec{k}$$

Assume the intersection point is (x_s, y_s, z_s); it can be expressed, using the parametric form of the surface normal unit vector, as:

$$\frac{x_s - x_i}{\alpha} = \frac{y_s - y_i}{\beta} = \frac{z_s - z_i}{\gamma} = sc$$

$$\text{or} \quad x_s = x_i + \alpha s_c \ , \quad y_s = y_i + \beta s_c \ , \quad z_s = z_i + \gamma s_c \qquad \text{(B.7)}$$

For a given surface point (x_i, y_i, z_i), a surface normal unit vector $\bar{v}$ and a tool motion $(\bar{C}(t), \bar{N}(t))$, $0 \le t \le 1$, the minimum distance from the surface point along the normal to the bottom (or with a similar argument, the top surface) of the tool motion can be derived by the following expression:

The surface point (x_i, y_i, z_i) has local coordinate $(t, h_i(t))$ in the ruled surface $r(t,h(t))$. The minimum distance from the surface point along the normal to any intersection point on the plane of the bottom surface must have $h_s(t) = 0$ in equation (B.3). Thus this problem can be simplified to whether the intersection point (x_s, y_s, z_s) with this plane is a point within a circle with center $\bar{r}(t, 0) = (x(t), y(t), z(t))$ and radius R_0.

There are two conditions to assure that the intersection point is point within a circle in the $h_s(t) = 0$ plane at some time $0 < t < 1$:

$$[x_s - x(t)]^2 + [y_s - y(t)]^2 + [z_s - z(t)]^2 \le R^2_0 \ , \text{for some } 0 < t < 1, \qquad \text{(B.8)}$$

and, given $h_s(t) = 0$ in Equation B.3, $s_c(t) = -h_i(t) / [\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)]$

Or we can define the problem as a simple optimization problem as follows:

$$\text{minimize} : \underset{t \in R}{s_c}(t) = - h_i(t) / [\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)]$$

subject to :

$$0 \leq t \leq 1$$

$$[x_s - x(t)]^2 + [y_s - y(t)]^2 + [z_s - z(t)]^2 \leq R^2_o$$

Part 3: Example of a Ball-End Cutter

Let's consider an example of a surface normal vector intersection with the volume swept by a ball-end cutter, for a given definition of the control point of the tool motion as shown in Figure B.4.

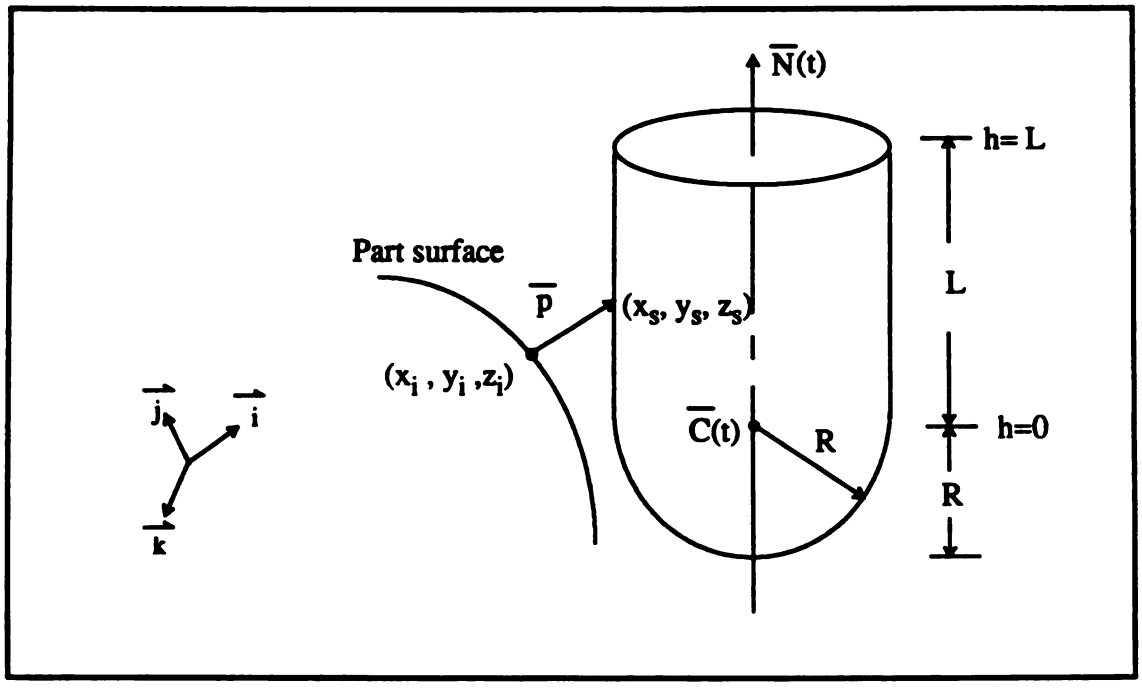

Figure B.4 Intersection of a surface vector with a ball-end cutter

The mathematical model for the ball-end cutter can be divided into a cylindrical part and a spherical part, if the control point is located at the center of the sphere. The next step in solving the problem is to determine which part of the cutter is being intersected by the surface vector. An easy way to solve the problem is to determine the t value for which $h_s(t)=0$ -- that is, the time (if any) within the sweep at which the cutter switches from cutting with the ball to cutting with the side, or vice-versa. Then the mathematical model for the cylindrical part has already been given in Part 1, and the mathematical model for the spherical part can be expressed as follows:

$$[x_s - x(t)]^2 + [y_s - y(t)]^2 + [z_s - z(t)]^2 = R^2 \qquad \textbf{(B.10)}$$

$$\text{and} \quad h_p = h_i(t) + s[\alpha N_X(t) + \beta N_Y(t) + \gamma N_Z(t)] \leq 0 \qquad \textbf{(B.11)}$$

Applying parametric form **(B.7)** to **(B.10)**

$$[x_i + \alpha s - x(t)]^2 + [y_i + \beta s - y(t)]^2 + [z_i + \gamma s - z(t)]^2 = R^2$$

$$\text{or,} \quad s^2 + H(t)*s + P(t) = 0, \qquad \textbf{(B.12)}$$

where:

$$H(t) = 2\alpha[x_i - x(t)] + 2\beta[y_i - y(t)] + 2\gamma[z_i - z(t)]$$

$$P(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - R^2$$

Or we can define the problem as the following:

$$\text{Minimize:} \underset{t \in R}{s(t)} = \frac{-H(t) \pm \sqrt{H^2(t) - 4*P(t)}}{2}$$

subject to :

$$0 \leq t \leq 1$$

$$-R \leq h_s(t) \leq 0$$

The minimum s value of equation **(B.12)** for $0 \leq t \leq 1$ which satisfies equation **(B.11)** is the cut value of the surface normal vector. If $H^2(t) - 4P(t) < 0$, the surface vector misses the model. If the surface vector intersects with both parts, the cut value for this toolpath is the minimum value of the two.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Wysocki, D. A., Oliver, J. H. and Goodman, E. D., "Gouge Detection Algorithm for Sculptured Surface NC Generation," Symposium on Computer-Aided Design and Manufacturing of Cutting and Forming/Forging Tools, 1989 ASME Winter Annual Meeting.

[2] Wysocki, D. A., Generation, Verification and Correction of Numerical Control Tool Paths," Master's thesis, Michigan State University, 1987.

[3] Choi, B. K. and Jun. C. S. "Ball-End Cutter Interference Avoidance in NC machining of sculptured surface," Computer-Aided Design, Vol. 21, No. 6, pp.371~378, 1989.

[4] Requicha, A.A.G. and Voelcker, H. B., "Solid Modeling : A Historical Summary and Contemporary Assessment," IEEE Computer Graphics and Applications, Vol. 2, No. 2, March, 1982.

[5] Requicha, A.A.G. and Voelcker, H. B., "Solid Modeling : Current Status and Research Directions," IEEE Computer Graphics and Applications, Vol.3, No.10, October, 1983.

[6] Gasale, M.S. and Stanton, E. L., "An Overview of Analytic Solid Modeling," IEEE Computer Graphics and Applications, Vol. 5, No. 2, February, 1985.

[7] Voelcker, H. B. and Requicha, A.A.G., "Geometric Modeling of Mechanical Parts and Process," Computer, Vol. 10, No. 12, December, 1977, pp.48~57.

[8] Voelcker, H. B. and Hunt, W.A., "The role of Solid Modelling in Machine-Process Modelling and NC verification," Proceedings of SAE 1981 International Congress and Exposition, Detroit, Michigan, February, 1981.

[9] Hunt, W.A. and Voelcker, H. B., "An Exploratory Study of Automatic Verification

of Programs for Numerically Controlled Machine Tools," Product Automation Project, Technical Memo No. 34, University of Rochester, 1982.

[10] Boyse, J. W. and Gilchrist, J. E., "GM Solid : Interactive Modeling for Design and Analysis of Solids," IEEE Computer Graphics and Applications, March, 1982, pp.27~40.

[11] R. Fridshal, K. P. Cheng, D. Duncan and W. Zucker, "Numerical Control Part Program Verification System," Proceeding Conference CAD/CAM Technology in Mechanical Engineering, MIT press, Cambridge, Massachusetts, 1982, pp.236~254.

[12] Okino, N., Kakazu, Y. and Kubo, H., "TIPS-I: Technical Information Processing System for Computer-Aided Design, Drawing and Manufacturing," Computer Languages for Numerical Control, Hatvany, J., ed., North-Holland, Amsterdam, 1973.

[13] Wang, W. P., "Geometric Modeling for Swept Volume of Moving Solids," IEEE Computer Graphics and Applications, December, 1986, pp.8~17.

[14] Hook, T. Van, "Real-Time Shaded NC Milling Display," Computer Graphics (Proceeding of SIGGRAPH), Vol. 20, No. 4, August 1986, pp.8~17.

[15] Wang, W. P., "Solid Modeling for Mold Design and Manufacture," Ph.D. Dissertation, Cornell University, 1984.

[16] Wang, W. P., "Three Dimensional Collision Avoidance in Production Automation," Computer Applications in Production Engineering Conference, Tokyo, 1989.

[17] Wang, W. P., and Wang, K. K. "Real Time Verification of Multiaxis NC programs with Raster Graphics," Proceedings of the IEEE International Conference on Robotics and Automation, April 7~10, 1986.

[18] Saito, T., and Takahashi, T., "Comprehensible Rendering of 3-D Shapes,"

Computer Graphics, Vol. 24, No. 4, (Proceeding of SIGGRAPH), pp. 197~206, 1990.

[19] Saito, T., and Takahashi, T., "NC Machining with G-buffer Method," Computer Graphics, Vol. 25, No. 4, (Proceeding of SIGGRAPH), pp. 207~216, 1991.

[20] Prun, J., "Manufacturing Software NC Troubleshooting," Computer-Aided Engineering, May, 1990.

[21] Chappel, I. T.,"The Use of Vectors to Simulate Material Removed by Numerically Controlled Milling," Computer-Aided Design, Vol. 15, No. 3, May 1983, pp.156~158.

[22] Bobrow, J. E., "NC Machine Tool Path Generation from CSG part representations," Computer Aided Design, Vol. 17, No. 2, 1985, pp.69~76.

[23] Duncan, J. P. and Mair, S. G."The Anti-interference Features of Polyhedral Machining," in Mcpherson (ed.) Advances in Computer-Aided Manufacture, North-Holand Publishing Company, Amsterdam, 1976.

[24] Oliver, J. H., "Graphical Verification of Numerically Controlled Milling Programs for Sculptured Surface parts," Ph.D. Dissertation, Michigan State University, 1986.

[25] Oliver, J. H. and Goodman, E. D., "Direct Dimensional NC Verification," Computer Aided Design, Vol. 22, No. 1, 1990, pp.3~10 .

[26] Drysdale, R. L., and Jerard, R. B., "Discrete Simulation of NC Machining," Proceeding of Annual ACM Conference Computational Geometry, 1987, pp.126~135.

[27] Jerard, R. B., Drysdale, R. L., Hauck, k., and Schaudt, B., "Method for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces," IEEE Computer Graphics & Applications, Vol. 9, No. 1, January 1989, pp.26~39.

[28] Jerard, R. B., Hussaini, S. Z., Drysdale, R. L., and Schaudt, B., "Approximate

Methods for Simulation and Verification of Numerical Controlled Machining Programs," Visual Computer, Vol. 5, No. 6, 1989, pp.329~348.

[29] CAM-I Inc. "APT4 Sculptured Surfaces Part Programmers Manual," January, 1985.

[30] Mortenson, M. E., "Geometric Modeling ," John Wiley & Sons Inc., 1985.

[31] Preparata, F. P. and Shamos, M. I.,"Computational Geometry : An Introduction," Springer-Verag New York Inc., 1985.

[32] Correns, C. I. Martin, "Numerical Evaluation of Nonuniform Rational B-spline surface," Technical Report, Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, 1989.

[33] Loney, G. C. and Ozsoy, T. M., "NC Machining of Free form Surface," Computer Aided Design, Vol. 19, No. 2, 1987, pp. 85~90.

[34] Pratt, M. J., "Smooth Parametric Surface Approximations to Discrete Data," Computer Aided Geometric Design, Vol. 2, 1985, pp.165~171.

[35] Barnhill, R. E., Farin, G., Jordan, M. and Piper, B. R., "Surface/surface Intersection," Computer Aided Geometric Design, Vol. 4, 1987, pp. 3~16.

[36] Kline, W. A., DeVor, W. A. and Shareef, I. A., "The Prediction of Surface Accuracy in End Milling," Transactions of the ASME, Vol. 104, August 1982. pp.272~278.

[37] Saeed, S. E. O., Pennington, A. D., and Dodsworth, J. R., " Offsetting in Geometric Modelling," Vol. 20, 1988.

[38] Anderson, R. O., "Detecting and Eliminating Collisions in NC Machining," Computer-Aided Design, Vol. 10, No. 4, July 1978, pp. 231~237.

[39] Blinn, J. F. , "Computer Display of Curved Surfaces," Ph.D. Dissertation, University of Utah, 1978.

[40] Burger, P. and Grillies, D., "Interactive Computer Graphics," Addison-Wesley

Publishing Company, 1989.

[41] Young, D. A., "X Window Systems Programming and Applications with Xt," Prentice-Hall Inc., New Jersey, 1989.

[42] Heller, D., "X Window User's Guide," O'Reilly & Associates Inc., 1990.

[43] Heller, D., "X Toolkit Intrinsics Programming Manual," O'Reilly & Associates Inc., 1990.

[44] Heller, D., "X Toolkit Intrinsics Reference Manual," O'Reilly & Associates Inc., 1990.

[45] Heller, D., "XView Programming Manual," O'Reilly & Associates Inc., 1990.

[46] Gries, D. "The Science of Programming," Springer-Verlag New York Inc., pp.48~57.

[47] Chang, K. and E. Goodman, "A Method for NC Toolpath Interference Detection for A Multi-Axis Milling System," Control/Manufacturing Symposium on ASME Winter Annual Conference, 1991.

[48] Chang, C. H. and Melkanoff, M. A.,"NC Machine Programming and Software Design," Prentice-Hall Inc., New Jersey, 1989.